

cs473: Algorithms

Lecture 6: Dynamic Programming

Michael A. Forbes

University of Illinois at Urbana-Champaign

September 11, 2019

logistics:

logistics:

- pset2 due R5

logistics:

- pset2 due R5 — can submit in *groups* of ≤ 3

logistics:

- pset2 due R5 — can submit in *groups* of ≤ 3

last lecture:

logistics:

- pset2 due R5 — can submit in *groups* of ≤ 3

last lecture:

- shortest paths

logistics:

- pset2 due R5 — can submit in *groups* of ≤ 3

last lecture:

- shortest paths
 - with negative lengths

logistics:

- pset2 due R5 — can submit in *groups* of ≤ 3

last lecture:

- shortest paths
 - with negative lengths — Bellman-Ford in $O(mn)$ time

logistics:

- pset2 due R5 — can submit in *groups* of ≤ 3

last lecture:

- shortest paths
 - with negative lengths — Bellman-Ford in $O(mn)$ time
 - all-pairs

logistics:

- pset2 due R5 — can submit in *groups* of ≤ 3

last lecture:

- shortest paths
 - with negative lengths — Bellman-Ford in $O(mn)$ time
 - all-pairs — Floyd-Warshall in $O(n^3)$ time

logistics:

- pset2 due R5 — can submit in *groups* of ≤ 3

last lecture:

- shortest paths
 - with negative lengths — Bellman-Ford in $O(mn)$ time
 - all-pairs — Floyd-Warshall in $O(n^3)$ time

today:

logistics:

- pset2 due R5 — can submit in *groups* of ≤ 3

last lecture:

- shortest paths
 - with negative lengths — Bellman-Ford in $O(mn)$ time
 - all-pairs — Floyd-Warshall in $O(n^3)$ time

today:

- dynamic programming *optimized*

logistics:

- pset2 due R5 — can submit in *groups* of ≤ 3

last lecture:

- shortest paths
 - with negative lengths — Bellman-Ford in $O(mn)$ time
 - all-pairs — Floyd-Warshall in $O(n^3)$ time

today:

- dynamic programming *optimized*
 - edit distance

logistics:

- pset2 due R5 — can submit in *groups* of ≤ 3

last lecture:

- shortest paths
 - with negative lengths — Bellman-Ford in $O(mn)$ time
 - all-pairs — Floyd-Warshall in $O(n^3)$ time

today:

- dynamic programming *optimized*
 - edit distance
 - longest increasing subsequence

dynamic programming:

- develop recursive algorithm
- understand structure of subproblems
 - names of subproblems
 - number of subproblems
 - dependency graph amongst subproblems
- memoize (implicitly, or explicitly)
- analysis (time, space)
- further optimization

remarks:

- memoizing a recursive algorithm does not necessarily lead to an efficient algorithm (e.g., knapsack problem) — you need the *right* recursion
- recognizing that dynamic programming applies to a problem can be non-obvious

Definition

Let $x, y \in \Sigma^*$ be two strings over the alphabet Σ . The **edit distance** between x and y is the minimum number of insertions, deletions and substitutions required to transform x into y .

Example

money \rightarrow boney \rightarrow bonee \rightarrow bona \rightarrow bo_a \rightarrow boba \implies edit distance ≤ 5

remarks:

- edit distance ≤ 4
- intermediate strings can be arbitrary in Σ^*

Edit Distance (II)

Definition

Let $x, y \in \Sigma^*$ be two strings over the alphabet Σ . An **alignment** is a sequence M of pairs of indices (i, j) such that

- an index could be empty, such as $(, 4)$ or $(5,)$
- each index appears exactly once per coordinate
- no crossings: for $(i, j), (i', j') \in M$ either $i < i'$ and $j < j'$, or $i > i'$ and $j > j'$

The **cost** of an alignment is the number of pairs (i, j) where $x_i \neq y_j$.

Example

mon ey

bo ba

$M = \{(1, 1), (2, 2), (3,), (, 3), (4, 4), (5,)\}$, cost 5

Edit Distance (III)

question: given two strings $x, y \in \Sigma^*$, compute their edit distance

Lemma

The edit distance between two strings $x, y \in \Sigma^$ is the minimum cost of an alignment.*

Proof.

Exercise. □

question: given two strings $x, y \in \Sigma^*$, compute the minimum cost of an alignment

remarks:

- can *also* ask to compute the alignment itself
- widely solved in practice, e.g., the BLAST heuristic for DNA edit distance

Edit Distance (IV)

Lemma

Let $x, y \in \Sigma^*$ be strings, and $a, b \in \Sigma$ be symbols. Then

$$\text{dist}(x \circ a, y \circ b) = \min \begin{cases} \text{dist}(x, y) + \mathbb{1}[a \neq b] \\ \text{dist}(x, y \circ b) + 1 \\ \text{dist}(x \circ a, y) + 1 \end{cases} .$$

Proof.

In an optimal alignment from $x \circ a$ to $y \circ b$, either:

- a aligns to b , with cost $\mathbb{1}[a \neq b]$
- a is deleted, with cost 1
- b is deleted, with cost 1



Edit Distance (V)

Edit Distance (V)

iterative algorithm:

Edit Distance (V)

iterative algorithm:

```
dist( $x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )  
  for  $0 \leq i \leq n$   
     $d[i][0] = i$   
  for  $0 \leq j \leq m$   
     $d[0][j] = j$   
  for  $0 \leq i \leq n$   
    for  $0 \leq j \leq m$   
       $d[i][j] = \min \begin{cases} d[i-1][j-1] + \mathbb{1}[x_i \neq y_j] \\ d[i-1][j] + 1 \\ d[i][j-1] + 1 \end{cases}$   
  return  $d[n][m]$ 
```

Edit Distance (V)

iterative algorithm:

```
dist( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  for  $0 \leq i \leq n$   
     $d[i][0] = i$   
  for  $0 \leq j \leq m$   
     $d[0][j] = j$   
  for  $0 \leq i \leq n$   
    for  $0 \leq j \leq m$   
       $d[i][j] = \min \begin{cases} d[i-1][j-1] + \mathbb{1}[x_i \neq y_j] \\ d[i-1][j] + 1 \\ d[i][j-1] + 1 \end{cases}$   
  return  $d[n][m]$ 
```

correctness:

Edit Distance (V)

iterative algorithm:

```
dist( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  for  $0 \leq i \leq n$   
     $d[i][0] = i$   
  for  $0 \leq j \leq m$   
     $d[0][j] = j$   
  for  $0 \leq i \leq n$   
    for  $0 \leq j \leq m$   
       $d[i][j] = \min \begin{cases} d[i-1][j-1] + \mathbb{1}[x_i \neq y_j] \\ d[i-1][j] + 1 \\ d[i][j-1] + 1 \end{cases}$   
  return  $d[n][m]$ 
```

correctness: clear

Edit Distance (V)

iterative algorithm:

```
dist( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  for  $0 \leq i \leq n$   
     $d[i][0] = i$   
  for  $0 \leq j \leq m$   
     $d[0][j] = j$   
  for  $0 \leq i \leq n$   
    for  $0 \leq j \leq m$   
       $d[i][j] = \min \begin{cases} d[i-1][j-1] + \mathbb{1}[x_i \neq y_j] \\ d[i-1][j] + 1 \\ d[i][j-1] + 1 \end{cases}$   
  return  $d[n][m]$ 
```

correctness: clear

complexity:

Edit Distance (V)

iterative algorithm:

```
dist( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  for  $0 \leq i \leq n$   
     $d[i][0] = i$   
  for  $0 \leq j \leq m$   
     $d[0][j] = j$   
  for  $0 \leq i \leq n$   
    for  $0 \leq j \leq m$   
       $d[i][j] = \min \begin{cases} d[i-1][j-1] + \mathbb{1}[x_i \neq y_j] \\ d[i-1][j] + 1 \\ d[i][j-1] + 1 \end{cases}$   
  return  $d[n][m]$ 
```

correctness: clear

complexity:

- $O(nm)$ time

Edit Distance (V)

iterative algorithm:

```
dist( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  for  $0 \leq i \leq n$   
     $d[i][0] = i$   
  for  $0 \leq j \leq m$   
     $d[0][j] = j$   
  for  $0 \leq i \leq n$   
    for  $0 \leq j \leq m$   
       $d[i][j] = \min \begin{cases} d[i-1][j-1] + \mathbb{1}[x_i \neq y_j] \\ d[i-1][j] + 1 \\ d[i][j-1] + 1 \end{cases}$   
  return  $d[n][m]$ 
```

correctness: clear

complexity:

- $O(nm)$ time
- space

Edit Distance (V)

iterative algorithm:

```
dist( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  for  $0 \leq i \leq n$   
     $d[i][0] = i$   
  for  $0 \leq j \leq m$   
     $d[0][j] = j$   
  for  $0 \leq i \leq n$   
    for  $0 \leq j \leq m$   
       $d[i][j] = \min \begin{cases} d[i-1][j-1] + \mathbb{1}[x_i \neq y_j] \\ d[i-1][j] + 1 \\ d[i][j-1] + 1 \end{cases}$   
  return  $d[n][m]$ 
```

correctness: clear

complexity:

- $O(nm)$ time
- space
 - clearly $O(nm)$

Edit Distance (V)

iterative algorithm:

```
dist( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  for  $0 \leq i \leq n$   
     $d[i][0] = i$   
  for  $0 \leq j \leq m$   
     $d[0][j] = j$   
  for  $0 \leq i \leq n$   
    for  $0 \leq j \leq m$   
       $d[i][j] = \min \begin{cases} d[i-1][j-1] + \mathbb{1}[x_i \neq y_j] \\ d[i-1][j] + 1 \\ d[i][j-1] + 1 \end{cases}$   
  return  $d[n][m]$ 
```

correctness: clear

complexity:

- $O(nm)$ time
- space
 - clearly $O(nm)$
 - *better:*

Edit Distance (V)

iterative algorithm:

```
dist( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  for  $0 \leq i \leq n$   
     $d[i][0] = i$   
  for  $0 \leq j \leq m$   
     $d[0][j] = j$   
  for  $0 \leq i \leq n$   
    for  $0 \leq j \leq m$   
       $d[i][j] = \min \begin{cases} d[i-1][j-1] + \mathbb{1}[x_i \neq y_j] \\ d[i-1][j] + 1 \\ d[i][j-1] + 1 \end{cases}$   
  return  $d[n][m]$ 
```

correctness: clear

complexity:

- $O(nm)$ time
- space
 - clearly $O(nm)$
 - *better:* only store $d[\text{cur}][\cdot]$ and $d[\text{prev}][\cdot]$

Edit Distance (V)

iterative algorithm:

```
dist( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  for  $0 \leq i \leq n$   
     $d[i][0] = i$   
  for  $0 \leq j \leq m$   
     $d[0][j] = j$   
  for  $0 \leq i \leq n$   
    for  $0 \leq j \leq m$   
       $d[i][j] = \min \begin{cases} d[i-1][j-1] + \mathbb{1}[x_i \neq y_j] \\ d[i-1][j] + 1 \\ d[i][j-1] + 1 \end{cases}$   
  return  $d[n][m]$ 
```

correctness: clear

complexity:

- $O(nm)$ time
- space
 - clearly $O(nm)$
 - *better:* only store $d[\text{cur}][\cdot]$ and $d[\text{prev}][\cdot] \implies O(m)$

Edit Distance (V)

iterative algorithm:

```
dist( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  for  $0 \leq i \leq n$   
     $d[i][0] = i$   
  for  $0 \leq j \leq m$   
     $d[0][j] = j$   
  for  $0 \leq i \leq n$   
    for  $0 \leq j \leq m$   
       $d[i][j] = \min \begin{cases} d[i-1][j-1] + \mathbb{1}[x_i \neq y_j] \\ d[i-1][j] + 1 \\ d[i][j-1] + 1 \end{cases}$   
  return  $d[n][m]$ 
```

correctness: clear

complexity:

- $O(nm)$ time
- space
 - clearly $O(nm)$
 - *better:* only store $d[\text{cur}][\cdot]$ and $d[\text{prev}][\cdot] \implies O(m)$

question:

Edit Distance (V)

iterative algorithm:

```
dist( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  for  $0 \leq i \leq n$   
     $d[i][0] = i$   
  for  $0 \leq j \leq m$   
     $d[0][j] = j$   
  for  $0 \leq i \leq n$   
    for  $0 \leq j \leq m$   
       $d[i][j] = \min \begin{cases} d[i-1][j-1] + \mathbb{1}[x_i \neq y_j] \\ d[i-1][j] + 1 \\ d[i][j-1] + 1 \end{cases}$   
  return  $d[n][m]$ 
```

correctness: clear

complexity:

- $O(nm)$ time
- space
 - clearly $O(nm)$
 - *better:* only store $d[\text{cur}][\cdot]$ and $d[\text{prev}][\cdot] \implies O(m)$

question: are we done?

Edit Distance (VI)

Corollary

Given two strings $x, y \in \Sigma^$, can compute the minimum cost alignment in $O(nm)$ -time and $O(nm)$ -space.*

Proof.

Exercise. *Hint: follow how each subproblem was solved.*

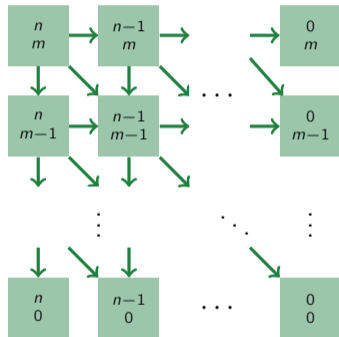


Edit Distance (VII)

dependency graph:

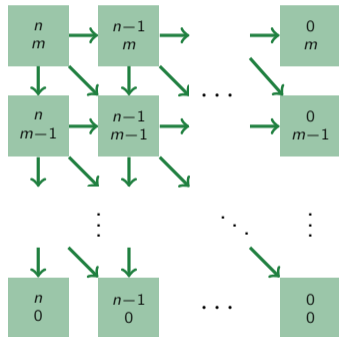
Edit Distance (VII)

dependency graph:



Edit Distance (VII)

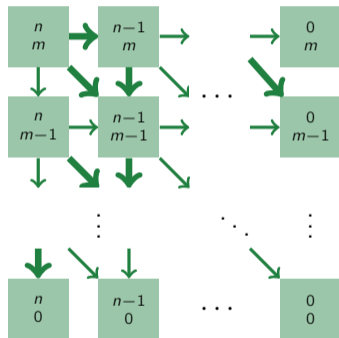
dependency graph:



computing the alignment:

Edit Distance (VII)

dependency graph:

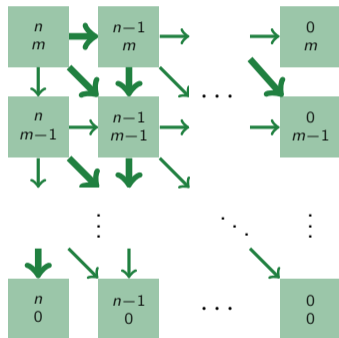


computing the alignment:

- *how* update rule is computed yields a pointer for each (i, j)

Edit Distance (VII)

dependency graph:

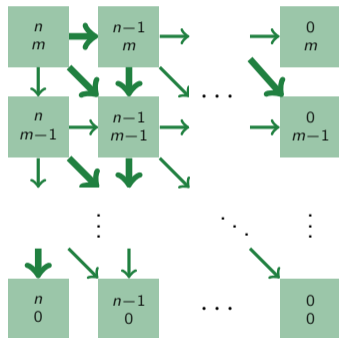


computing the alignment:

- *how* update rule is computed yields a pointer for each (i, j)
- one pointer per optimal choice

Edit Distance (VII)

dependency graph:

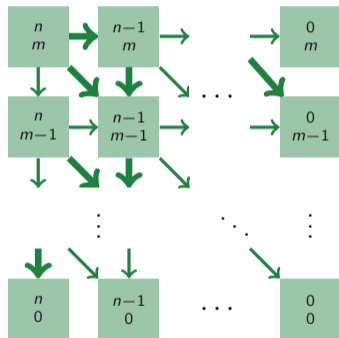


computing the alignment:

- *how* update rule is computed yields a pointer for each (i, j)
- one pointer per optimal choice — multiple pointers are possible

Edit Distance (VII)

dependency graph:

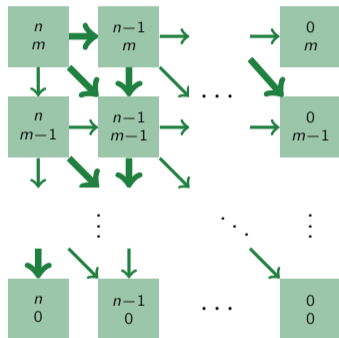


computing the alignment:

- *how* update rule is computed yields a pointer for each (i, j)
- one pointer per optimal choice — multiple pointers are possible
- *any* path from (n, m) to boundary

Edit Distance (VII)

dependency graph:

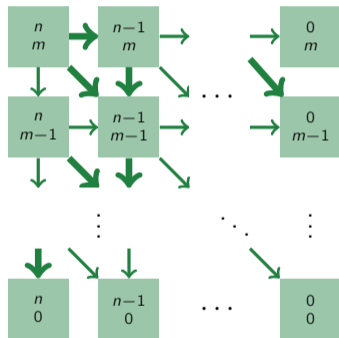


computing the alignment:

- *how* update rule is computed yields a pointer for each (i, j)
- one pointer per optimal choice — multiple pointers are possible
- *any* path from (n, m) to boundary yields optimal alignment

Edit Distance (VII)

dependency graph:

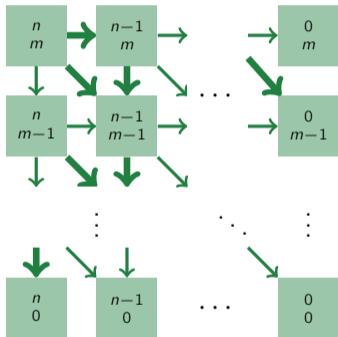


computing the alignment:

- *how* update rule is computed yields a pointer for each (i, j)
- one pointer per optimal choice — multiple pointers are possible
- *any* path from (n, m) to boundary yields optimal alignment
- compute path via graph search

Edit Distance (VII)

dependency graph:



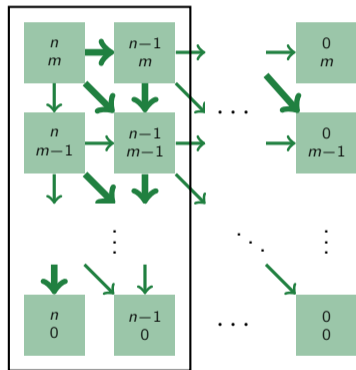
computing the alignment:

- *how* update rule is computed yields a pointer for each (i, j)
- one pointer per optimal choice — multiple pointers are possible
- *any* path from (n, m) to boundary yields optimal alignment
- compute path via graph search

saving space:

Edit Distance (VII)

dependency graph:



computing the alignment:

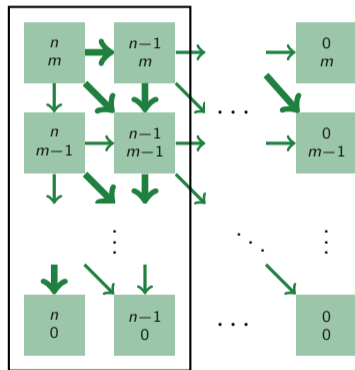
- *how* update rule is computed yields a pointer for each (i, j)
- one pointer per optimal choice — multiple pointers are possible
- *any* path from (n, m) to boundary yields optimal alignment
- compute path via graph search

saving space:

- only keep most recent two columns

Edit Distance (VII)

dependency graph:



computing the alignment:

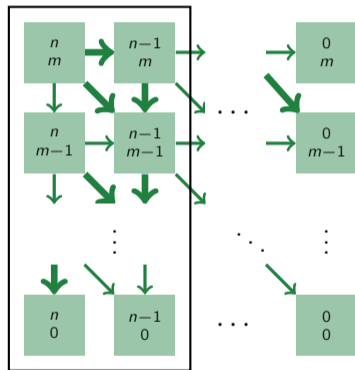
- how update rule is computed yields a pointer for each (i, j)
- one pointer per optimal choice — multiple pointers are possible
- any path from (n, m) to boundary yields optimal alignment
- compute path via graph search

saving space:

- only keep most recent two columns
- ⇒ we lost the pointers!

Edit Distance (VII)

dependency graph:



computing the alignment:

- how update rule is computed yields a pointer for each (i, j)
- one pointer per optimal choice — multiple pointers are possible
- any path from (n, m) to boundary yields optimal alignment
- compute path via graph search

saving space:

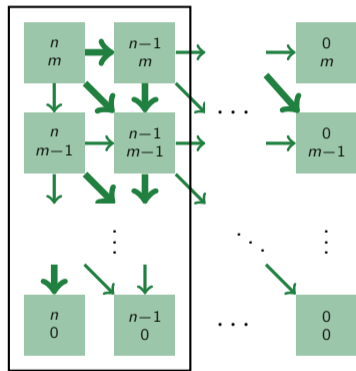
- only keep most recent two columns

⇒ we lost the pointers!

question:

Edit Distance (VII)

dependency graph:



computing the alignment:

- how update rule is computed yields a pointer for each (i, j)
- one pointer per optimal choice — multiple pointers are possible
- any path from (n, m) to boundary yields optimal alignment
- compute path via graph search

saving space:

- only keep most recent two columns

⇒ we lost the pointers!

question: compute the alignment in $O(n + m)$ space?

Lemma

Lemma

Let $x, y \in \Sigma^$ be strings,*

Lemma

Let $x, y \in \Sigma^$ be strings, with $n = |x|$ and $m = |y|$.*

Lemma

Let $x, y \in \Sigma^$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,*

Lemma

Let $x, y \in \Sigma^$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,*

$$\text{dist}(x, y) =$$

Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m}$$

Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m} \{ \text{dist}(x_{\leq i}, y_{\leq j}) +$$

Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m} \{ \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j}) \} .$$

Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m} \{ \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j}) \} .$$

Proof.

Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m} \{ \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j}) \} .$$

Proof.

\leq :

Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m} \{ \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j}) \} .$$

Proof.

\leq : Fix j .

Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m} \{ \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j}) \} .$$

Proof.

\leq : Fix j . Let A_{\leq} and $A_{>}$ be alignments

Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m} \{ \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j}) \} .$$

Proof.

\leq : Fix j . Let A_{\leq} and $A_{>}$ be alignments respectively between $x_{\leq i}, y_{\leq j}$ and $x_{> i}, y_{> j}$,

Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m} \{ \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j}) \} .$$

Proof.

\leq : Fix j . Let A_{\leq} and $A_{>}$ be alignments respectively between $x_{\leq i}, y_{\leq j}$ and $x_{> i}, y_{> j}$, with respective costs $\text{dist}(x_{\leq i}, y_{\leq j})$ and $\text{dist}(x_{> i}, y_{> j})$.

Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m} \{ \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j}) \} .$$

Proof.

\leq : Fix j . Let A_{\leq} and $A_{>}$ be alignments respectively between $x_{\leq i}, y_{\leq j}$ and $x_{> i}, y_{> j}$, with respective costs $\text{dist}(x_{\leq i}, y_{\leq j})$ and $\text{dist}(x_{> i}, y_{> j})$. Then $A_{\leq} \circ A_{>}$ is an alignment between x and y of cost $\text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})$.

Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m} \{ \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j}) \}.$$

Proof.

\leq : Fix j . Let A_{\leq} and $A_{>}$ be alignments respectively between $x_{\leq i}, y_{\leq j}$ and $x_{> i}, y_{> j}$, with respective costs $\text{dist}(x_{\leq i}, y_{\leq j})$ and $\text{dist}(x_{> i}, y_{> j})$. Then $A_{\leq} \circ A_{>}$ is an alignment between x and y of cost $\text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})$.

$\underline{\underline{=}}$:

Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m} \{ \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j}) \} .$$

Proof.

\leq : Fix j . Let A_{\leq} and $A_{>}$ be alignments respectively between $x_{\leq i}, y_{\leq j}$ and $x_{> i}, y_{> j}$, with respective costs $\text{dist}(x_{\leq i}, y_{\leq j})$ and $\text{dist}(x_{> i}, y_{> j})$. Then $A_{\leq} \circ A_{>}$ is an alignment between x and y of cost $\text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})$.

\geq : Any alignment A between x and y will align $x_{\leq i}$ to some prefix $y_{\leq j}$ of y in an alignment A_{\leq} ,

Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m} \{ \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j}) \} .$$

Proof.

\leq : Fix j . Let A_{\leq} and $A_{>}$ be alignments respectively between $x_{\leq i}, y_{\leq j}$ and $x_{> i}, y_{> j}$, with respective costs $\text{dist}(x_{\leq i}, y_{\leq j})$ and $\text{dist}(x_{> i}, y_{> j})$. Then $A_{\leq} \circ A_{>}$ is an alignment between x and y of cost $\text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})$.

\geq : Any alignment A between x and y will align $x_{\leq i}$ to some prefix $y_{\leq j}$ of y in an alignment A_{\leq} , and align $x_{> i}$ to the suffix $y_{> j}$ in an alignment $A_{>}$,

Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m} \{ \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j}) \} .$$

Proof.

\leq : Fix j . Let A_{\leq} and $A_{>}$ be alignments respectively between $x_{\leq i}, y_{\leq j}$ and $x_{> i}, y_{> j}$, with respective costs $\text{dist}(x_{\leq i}, y_{\leq j})$ and $\text{dist}(x_{> i}, y_{> j})$. Then $A_{\leq} \circ A_{>}$ is an alignment between x and y of cost $\text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})$.

\geq : Any alignment A between x and y will align $x_{\leq i}$ to some prefix $y_{\leq j}$ of y in an alignment A_{\leq} , and align $x_{> i}$ to the suffix $y_{> j}$ in an alignment $A_{>}$, and hence for this j we have $\text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})$. \square

Edit Distance, Better (II)

Edit Distance, Better (II)

Definition

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings,

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$.

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment.

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) =$$

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = j :$$

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \arg \min_{j : \text{dist}(x_{\leq i}, y_{\leq j}) = \text{dist}(x, y)}$$

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) +$$

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})$$

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark:

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark: previous lemma asserts such a j exists

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark: previous lemma asserts such a j exists

$$\text{meet}(i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m)$$

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark: previous lemma asserts such a j exists

$$\begin{aligned} \text{meet}(i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m) \\ d = \text{dist}(x, y) \end{aligned}$$

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark: previous lemma asserts such a j exists

$\text{meet}(i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m)$
 $d = \text{dist}(x, y)$
for $1 \leq j \leq m$

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
```

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )  
   $d = \text{dist}(x, y)$   
  for  $1 \leq j \leq m$   
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$   
  for  $1 \leq j \leq m$ 
```

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )  
   $d = \text{dist}(x, y)$   
  for  $1 \leq j \leq m$   
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$   
  for  $1 \leq j \leq m$   
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
```

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
  for  $1 \leq j \leq m$ 
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
   $j^* = \min\{j : d =$ 
```

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
  for  $1 \leq j \leq m$ 
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
   $j^* = \min\{j : d = d_{\leq}[j] +$ 
```

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\} .$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
  for  $1 \leq j \leq m$ 
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
   $j^* = \min\{j : d = d_{\leq}[j] + d_{>}[j]\}$ 
```

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\} .$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
  for  $1 \leq j \leq m$ 
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
   $j^* = \min\{j : d = d_{\leq}[j] + d_{>}[j]\}$ 
  output  $j^*$ 
```

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\} .$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
  for  $1 \leq j \leq m$ 
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
   $j^* = \min\{j : d = d_{\leq}[j] + d_{>}[j]\}$ 
  output  $j^*$ 
```


Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
  for  $1 \leq j \leq m$ 
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
   $j^* = \min\{j : d = d_{\leq}[j] + d_{>}[j]\}$ 
  output  $j^*$ 
```

correctness:

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
  for  $1 \leq j \leq m$ 
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
   $j^* = \min\{j : d = d_{\leq}[j] + d_{>}[j]\}$ 
  output  $j^*$ 
```

correctness: clear

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\} .$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
  for  $1 \leq j \leq m$ 
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
   $j^* = \min\{j : d = d_{\leq}[j] + d_{>}[j]\}$ 
  output  $j^*$ 
```

correctness: clear

complexity:

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
  for  $1 \leq j \leq m$ 
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
   $j^* = \min\{j : d = d_{\leq}[j] + d_{>}[j]\}$ 
  output  $j^*$ 
```

correctness: clear

complexity:

- $\text{dist}(x_{\leq i}, y)$ already computes $\text{dist}(x_{\leq i}, y_{\leq j})$ for all j

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
  for  $1 \leq j \leq m$ 
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
   $j^* = \min\{j : d = d_{\leq}[j] + d_{>}[j]\}$ 
  output  $j^*$ 
```

correctness: clear

complexity:

- $\text{dist}(x_{\leq i}, y)$ already computes $\text{dist}(x_{\leq i}, y_{\leq j})$ for all j
- $O(nm)$ time,

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
  for  $1 \leq j \leq m$ 
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
   $j^* = \min\{j : d = d_{\leq}[j] + d_{>}[j]\}$ 
  output  $j^*$ 
```

correctness: clear

complexity:

- $\text{dist}(x_{\leq i}, y)$ already computes $\text{dist}(x_{\leq i}, y_{\leq j})$ for all j
- $O(nm)$ time, $O(m)$ space

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\} .$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
  for  $1 \leq j \leq m$ 
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
   $j^* = \min\{j : d = d_{\leq}[j] + d_{>}[j]\}$ 
  output  $j^*$ 
```

correctness: clear

complexity:

- $\text{dist}(x_{\leq i}, y)$ already computes $\text{dist}(x_{\leq i}, y_{\leq j})$ for all j
 - $O(nm)$ time, $O(m)$ space
- $\text{dist}(x_{> i}, y)$ already computes $\text{dist}(x_{> i}, y_{> j})$ for all j

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\} .$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
  for  $1 \leq j \leq m$ 
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
   $j^* = \min\{j : d = d_{\leq}[j] + d_{>}[j]\}$ 
  output  $j^*$ 
```

correctness: clear

complexity:

- $\text{dist}(x_{\leq i}, y)$ already computes $\text{dist}(x_{\leq i}, y_{\leq j})$ for all j
 - $O(nm)$ time, $O(m)$ space
- $\text{dist}(\text{rev}(x_{> i}), \text{rev}(y))$ already computes $\text{dist}(x_{> i}, y_{> j})$ for all j

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\} .$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
   $d = \text{dist}(x, y)$   
  for  $1 \leq j \leq m$   
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$   
  for  $1 \leq j \leq m$   
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$   
   $j^* = \min\{j : d = d_{\leq}[j] + d_{>}[j]\}$   
  output  $j^*$ 
```

correctness: clear

complexity:

- $\text{dist}(x_{\leq i}, y)$ already computes $\text{dist}(x_{\leq i}, y_{\leq j})$ for all j
 - $O(nm)$ time, $O(m)$ space
- $\text{dist}(\text{rev}(x_{> i}), \text{rev}(y))$ already computes $\text{dist}(x_{> i}, y_{> j})$ for all j
 - $\implies O(nm)$ time,

Edit Distance, Better (II)

Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min\{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\} .$$

remark: previous lemma asserts such a j exists

```
meet( $i, x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )
   $d = \text{dist}(x, y)$ 
  for  $1 \leq j \leq m$ 
     $d_{\leq}[j] = \text{dist}(x_{\leq i}, y_{\leq j})$ 
  for  $1 \leq j \leq m$ 
     $d_{>}[j] = \text{dist}(x_{> i}, y_{> j})$ 
   $j^* = \min\{j : d = d_{\leq}[j] + d_{>}[j]\}$ 
  output  $j^*$ 
```

correctness: clear

complexity:

- $\text{dist}(x_{\leq i}, y)$ already computes $\text{dist}(x_{\leq i}, y_{\leq j})$ for all j
 - $O(nm)$ time, $O(m)$ space
- $\text{dist}(\text{rev}(x_{> i}), \text{rev}(y))$ already computes $\text{dist}(x_{> i}, y_{> j})$ for all j
 - $\implies O(nm)$ time, $O(m)$ space

Edit Distance, Better (III)

divide and conqueror:

divide and conqueror:

$\text{dist-align}(x_1x_2 \cdots x_n, y_1y_2 \cdots y_m)$

Edit Distance, Better (III)

divide and conqueror:

`dist-align($x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$)`

`if $n = 1$`

Edit Distance, Better (III)

divide and conqueror:

`dist-align($x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$)`

if $n = 1$

 use `dist(x, y)`

divide and conqueror:

`dist-align($x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$)`

if $n = 1$

 use `dist(x, y)`

if $m = 1$

 use `dist(x, y)`

divide and conqueror:

`dist-align($x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$)`

if $n = 1$

 use `dist(x, y)`

if $m = 1$

 use `dist(x, y)`

$j = \text{meet}(n - 1, x, y)$

divide and conqueror:

```
dist-align( $x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n - 1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$ 
```

divide and conqueror:

```
dist-align( $x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n - 1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$ 
```

divide and conqueror:

```
dist-align( $x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n - 1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

divide and conqueror:

```
dist-align( $x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n - 1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

Edit Distance, Better (III)

divide and conqueror:

```
dist-align( $x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n - 1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness:

divide and conqueror:

```
dist-align( $x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n - 1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

divide and conqueror:

```
dist-align( $x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n - 1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

divide and conqueror:

```
dist-align( $x_1x_2 \cdots x_n, y_1y_2 \cdots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n - 1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n - 1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
  if  $n = 1$ 
    use dist( $x, y$ )
  if  $m = 1$ 
    use dist( $x, y$ )
   $j = \text{meet}(n - 1, x, y)$ 
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$ 
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$ 
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time,

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
  if  $n = 1$ 
    use dist( $x, y$ )
  if  $m = 1$ 
    use dist( $x, y$ )
   $j = \text{meet}(n - 1, x, y)$ 
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$ 
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$ 
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1)$ space

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n - 1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
  if  $n = 1$ 
    use dist( $x, y$ )
  if  $m = 1$ 
    use dist( $x, y$ )
   $j = \text{meet}(n - 1, x, y)$ 
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$ 
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$ 
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time,

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n - 1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
  if  $n = 1$ 
    use dist( $x, y$ )
  if  $m = 1$ 
    use dist( $x, y$ )
   $j = \text{meet}(n-1, x, y)$ 
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$ 
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$ 
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n - 1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time,

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n-1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
  if  $n = 1$ 
    use dist( $x, y$ )
  if  $m = 1$ 
    use dist( $x, y$ )
   $j = \text{meet}(n-1, x, y)$ 
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$ 
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$ 
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence:

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
  if  $n = 1$ 
    use dist( $x, y$ )
  if  $m = 1$ 
    use dist( $x, y$ )
   $j = \text{meet}(n-1, x, y)$ 
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$ 
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$ 
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence: space can be *reused*

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n-1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence: space can be *reused*
 - $S(n, m) \leq$

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
  if  $n = 1$ 
    use dist( $x, y$ )
  if  $m = 1$ 
    use dist( $x, y$ )
   $j = \text{meet}(n-1, x, y)$ 
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$ 
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$ 
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence: space can be *reused*
 - $S(n, m) \leq \max\{O(n + m),$

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
  if  $n = 1$ 
    use dist( $x, y$ )
  if  $m = 1$ 
    use dist( $x, y$ )
   $j = \text{meet}(n-1, x, y)$ 
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$ 
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$ 
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence: space can be *reused*
 - $S(n, m) \leq \max\{O(n + m), S(n - 1, m)\}$,

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n - 1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence: space can be *reused*
 - $S(n, m) \leq \max\{O(n + m), S(n - 1, m), S(1, m)\}$

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
  if  $n = 1$ 
    use dist( $x, y$ )
  if  $m = 1$ 
    use dist( $x, y$ )
   $j = \text{meet}(n-1, x, y)$ 
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$ 
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$ 
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence: space can be *reused*
 - $S(n, m) \leq \max\{O(n + m), S(n - 1, m), S(1, m)\}$
 - $\Rightarrow S(n, m) \leq O(n + m)$

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n-1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence: space can be *reused*
 - $S(n, m) \leq \max\{O(n + m), S(n - 1, m), S(1, m)\}$
 $\implies S(n, m) \leq O(n + m)$
- time recurrence

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
  if  $n = 1$ 
    use dist( $x, y$ )
  if  $m = 1$ 
    use dist( $x, y$ )
   $j = \text{meet}(n-1, x, y)$ 
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$ 
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$ 
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence: space can be *reused*
 - $S(n, m) \leq \max\{O(n + m), S(n - 1, m), S(1, m)\}$
 - ⇒ $S(n, m) \leq O(n + m)$
- time recurrence
 - $T(n, m) \leq$

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
  if  $n = 1$ 
    use dist( $x, y$ )
  if  $m = 1$ 
    use dist( $x, y$ )
   $j = \text{meet}(n-1, x, y)$ 
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$ 
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$ 
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence: space can be *reused*
 - $S(n, m) \leq \max\{O(n + m), S(n - 1, m), S(1, m)\}$
 - ⇒ $S(n, m) \leq O(n + m)$
- time recurrence
 - $T(n, m) \leq O(nm) +$

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n-1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence: space can be *reused*
 - $S(n, m) \leq \max\{O(n + m), S(n - 1, m), S(1, m)\}$
 - $\implies S(n, m) \leq O(n + m)$
- time recurrence
 - $T(n, m) \leq O(nm) + T(n - 1, m) +$

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n-1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence: space can be *reused*
 - $S(n, m) \leq \max\{O(n + m), S(n - 1, m), S(1, m)\}$
 - $\implies S(n, m) \leq O(n + m)$
- time recurrence
 - $T(n, m) \leq O(nm) + T(n - 1, m) + T(1, m)$

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n-1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence: space can be *reused*
 - $S(n, m) \leq \max\{O(n + m), S(n - 1, m), S(1, m)\}$
 - $\implies S(n, m) \leq O(n + m)$
- time recurrence
 - $T(n, m) \leq O(nm) + T(n - 1, m) + T(1, m)$
 - $\implies T(n, m) \leq O(n^2m)$

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )  
  if  $n = 1$   
    use dist( $x, y$ )  
  if  $m = 1$   
    use dist( $x, y$ )  
   $j = \text{meet}(n-1, x, y)$   
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$   
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$   
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence: space can be *reused*
 - $S(n, m) \leq \max\{O(n + m), S(n - 1, m), S(1, m)\}$
 $\implies S(n, m) \leq O(n + m)$
- time recurrence
 - $T(n, m) \leq O(nm) + T(n - 1, m) + T(1, m)$
 $\implies T(n, m) \leq O(n^2m)$

question:

divide and conqueror:

```
dist-align( $x_1x_2 \dots x_n, y_1y_2 \dots y_m$ )
  if  $n = 1$ 
    use dist( $x, y$ )
  if  $m = 1$ 
    use dist( $x, y$ )
   $j = \text{meet}(n-1, x, y)$ 
   $A_{\leq} = \text{dist-align}(x_{\leq n-1}, y_{\leq j})$ 
   $A_{>} = \text{dist-align}(x_{> n-1}, y_{> j})$ 
  return  $A_{\leq} \circ A_{>}$ 
```

correctness: clear

complexity:

- alignment takes $O(n + m)$ space
- base cases
 - $O(m)$ time, $O(1) + O(n + m)$ space
 - $O(n)$ time, $O(n + m)$ space
- $\text{meet}_{n-1}(x, y)$
 - $O(nm)$ time, $O(n + m)$ space
- space recurrence: space can be *reused*
 - $S(n, m) \leq \max\{O(n + m), S(n - 1, m), S(1, m)\}$
 - ⇒ $S(n, m) \leq O(n + m)$
- time recurrence
 - $T(n, m) \leq O(nm) + T(n - 1, m) + T(1, m)$
 - ⇒ $T(n, m) \leq O(n^2m)$

question: can we do better?

Edit Distance, Better (IV)

Edit Distance, Better (IV)

divide and conqueror:

`dist-align'(x1x2⋯xn,y1y2⋯ym)`

`if n = 1`

`use dist(x,y)`

`if m = 1`

`use dist(x,y)`

`j = meet(`

Edit Distance, Better (IV)

divide and conqueror:

$\text{dist-align}'(x_1x_2 \cdots x_n, y_1y_2 \cdots y_m)$

if $n = 1$

use $\text{dist}(x, y)$

if $m = 1$

use $\text{dist}(x, y)$

$j = \text{meet}(\lfloor \frac{n}{2} \rfloor, x, y)$

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)  
  if n = 1  
    use dist(x,y)  
  if m = 1  
    use dist(x,y)  
  j = meet(⌊ $\frac{n}{2}$ ⌋,x,y)  
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋,y≤j)
```

Edit Distance, Better (IV)

divide and conqueror:

$\text{dist-align}'(x_1x_2\cdots x_n, y_1y_2\cdots y_m)$

if $n = 1$

use $\text{dist}(x, y)$

if $m = 1$

use $\text{dist}(x, y)$

$j = \text{meet}(\lfloor \frac{n}{2} \rfloor, x, y)$

$A_{\leq} = \text{dist-align}'(x_{\leq \lfloor \frac{n}{2} \rfloor}, y_{\leq j})$

$A_{>} = \text{dist-align}'(x_{> \lfloor \frac{n}{2} \rfloor}, y_{> j})$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)  
  if n = 1  
    use dist(x,y)  
  if m = 1  
    use dist(x,y)  
  j = meet(⌊ $\frac{n}{2}$ ⌋,x,y)  
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋,y≤j)  
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋,y>j)  
  return A≤ ◦ A>
```

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)  
  if n = 1  
    use dist(x,y)  
  if m = 1  
    use dist(x,y)  
  j = meet(⌊ $\frac{n}{2}$ ⌋, x, y)  
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)  
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)  
  return A≤ ◦ A>
```


Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)  
  if n = 1  
    use dist(x,y)  
  if m = 1  
    use dist(x,y)  
  j = meet(⌊ $\frac{n}{2}$ ⌋, x, y)  
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)  
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)  
  return A≤ ◦ A>
```

correctness: clear

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)  
  if n = 1  
    use dist(x,y)  
  if m = 1  
    use dist(x,y)  
  j = meet(⌊ $\frac{n}{2}$ ⌋, x, y)  
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)  
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)  
  return A≤ ◦ A>
```

complexity:

correctness: clear

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)  
  if n = 1  
    use dist(x,y)  
  if m = 1  
    use dist(x,y)  
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)  
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)  
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)  
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n + m)$ time, $O(n + m)$ space

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)  
  if n = 1  
    use dist(x,y)  
  if m = 1  
    use dist(x,y)  
  j = meet(⌊ $\frac{n}{2}$ ⌋, x, y)  
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)  
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)  
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n + m)$ time, $O(n + m)$ space
- meet_{⌊ $\frac{n}{2}$ ⌋}(x, y):

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)  
  if n = 1  
    use dist(x,y)  
  if m = 1  
    use dist(x,y)  
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)  
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)  
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)  
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n + m)$ time, $O(n + m)$ space
- meet_{⌊ $\frac{n}{2}$ ⌋}(x,y): $O(nm)$ time,

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)  
  if n = 1  
    use dist(x,y)  
  if m = 1  
    use dist(x,y)  
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)  
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋,y≤j)  
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋,y>j)  
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n + m)$ time, $O(n + m)$ space
- meet_{⌊ $\frac{n}{2}$ ⌋}(x,y): $O(nm)$ time, $O(n + m)$ space

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)  
  if n = 1  
    use dist(x,y)  
  if m = 1  
    use dist(x,y)  
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)  
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋,y≤j)  
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋,y>j)  
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n + m)$ time, $O(n + m)$ space
- meet_{⌊ $\frac{n}{2}$ ⌋}(x,y): $O(nm)$ time, $O(n + m)$ space
- space recurrence

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)  
  if n = 1  
    use dist(x,y)  
  if m = 1  
    use dist(x,y)  
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)  
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)  
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)  
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n + m)$ time, $O(n + m)$ space
- meet_{⌊ $\frac{n}{2}$ ⌋}(x,y): $O(nm)$ time, $O(n + m)$ space
- space recurrence
 - $S(n, m) \leq$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n + m)$ time, $O(n + m)$ space
- meet_{⌊ $\frac{n}{2}$ ⌋}(x,y): $O(nm)$ time, $O(n + m)$ space
- space recurrence
 - $S(n, m) \leq \max\{O(n+m),$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)  
  if n = 1  
    use dist(x,y)  
  if m = 1  
    use dist(x,y)  
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)  
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)  
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)  
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n + m)$ time, $O(n + m)$ space
- meet_{⌊ $\frac{n}{2}$ ⌋}(x,y): $O(nm)$ time, $O(n + m)$ space
- space recurrence
 - $S(n, m) \leq \max\{O(n + m), S(\lfloor \frac{n}{2} \rfloor, m)\}$,

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n + m)$ time, $O(n + m)$ space
- meet_{⌊ $\frac{n}{2}$ ⌋}(x,y): $O(nm)$ time, $O(n + m)$ space
- space recurrence
 - $S(n, m) \leq \max\{O(n + m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n + m)$ time, $O(n + m)$ space
- meet_{⌊ $\frac{n}{2}$ ⌋}(x,y): $O(nm)$ time, $O(n + m)$ space
- space recurrence
 - $S(n, m) \leq \max\{O(n + m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n, m) \leq O(n + m)$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)  
  if n = 1  
    use dist(x,y)  
  if m = 1  
    use dist(x,y)  
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)  
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)  
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)  
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n+m)$ time, $O(n+m)$ space
- meet_{⌊ $\frac{n}{2}$ ⌋}(x,y): $O(nm)$ time, $O(n+m)$ space
- space recurrence
 - $S(n,m) \leq \max\{O(n+m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n,m) \leq O(n+m)$
- time recurrence

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n+m)$ time, $O(n+m)$ space
- meet_{⌊ $\frac{n}{2}$ ⌋}(x,y): $O(nm)$ time, $O(n+m)$ space
- space recurrence
 - $S(n,m) \leq \max\{O(n+m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n,m) \leq O(n+m)$
- time recurrence
 - $T(n,m) \leq$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n + m)$ time, $O(n + m)$ space
- meet_{⌊ $\frac{n}{2}$ ⌋}(x,y): $O(nm)$ time, $O(n + m)$ space
- space recurrence
 - $S(n, m) \leq \max\{O(n+m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n, m) \leq O(n + m)$
- time recurrence
 - $T(n, m) \leq O(nm) +$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋,x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋,y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋,y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n+m)$ time, $O(n+m)$ space
- meet $_{\lfloor \frac{n}{2} \rfloor}(x,y)$: $O(nm)$ time, $O(n+m)$ space
- space recurrence
 - $S(n,m) \leq \max\{O(n+m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n,m) \leq O(n+m)$
- time recurrence
 - $T(n,m) \leq O(nm) + T(\lfloor \frac{n}{2} \rfloor, j) +$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n+m)$ time, $O(n+m)$ space
- meet $_{\lfloor \frac{n}{2} \rfloor}(x,y)$: $O(nm)$ time, $O(n+m)$ space
- space recurrence
 - $S(n,m) \leq \max\{O(n+m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n,m) \leq O(n+m)$
- time recurrence
 - $T(n,m) \leq O(nm) + T(\lfloor \frac{n}{2} \rfloor, j) + T(n - \lfloor \frac{n}{2} \rfloor, m - j)$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n+m)$ time, $O(n+m)$ space
- meet $_{\lfloor \frac{n}{2} \rfloor}(x,y)$: $O(nm)$ time, $O(n+m)$ space
- space recurrence
 - $S(n,m) \leq \max\{O(n+m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n,m) \leq O(n+m)$
- time recurrence
 - $T(n,m) \leq O(nm) + T(\lfloor \frac{n}{2} \rfloor, j) + T(n - \lfloor \frac{n}{2} \rfloor, m - j)$
 - guess $T(n,m) \leq \alpha \cdot nm$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n+m)$ time, $O(n+m)$ space
- meet $_{\lfloor \frac{n}{2} \rfloor}(x,y)$: $O(nm)$ time, $O(n+m)$ space
- space recurrence
 - $S(n,m) \leq \max\{O(n+m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n,m) \leq O(n+m)$
- time recurrence
 - $T(n,m) \leq O(nm) + T(\lfloor \frac{n}{2} \rfloor, j) + T(n - \lfloor \frac{n}{2} \rfloor, m - j)$
 - guess $T(n,m) \leq \alpha \cdot nm$
 - $T(n,m) \lesssim$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n+m)$ time, $O(n+m)$ space
- meet $_{\lfloor \frac{n}{2} \rfloor}(x,y)$: $O(nm)$ time, $O(n+m)$ space
- space recurrence
 - $S(n,m) \leq \max\{O(n+m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n,m) \leq O(n+m)$
- time recurrence
 - $T(n,m) \leq O(nm) + T(\lfloor \frac{n}{2} \rfloor, j) + T(n - \lfloor \frac{n}{2} \rfloor, m - j)$
 - guess $T(n,m) \leq \alpha \cdot nm$
 - $T(n,m) \lesssim \beta \cdot nm +$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋,x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋,y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋,y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n+m)$ time, $O(n+m)$ space
- meet $_{\lfloor \frac{n}{2} \rfloor}(x,y)$: $O(nm)$ time, $O(n+m)$ space
- space recurrence
 - $S(n,m) \leq \max\{O(n+m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n,m) \leq O(n+m)$
- time recurrence
 - $T(n,m) \leq O(nm) + T(\lfloor \frac{n}{2} \rfloor, j) + T(n - \lfloor \frac{n}{2} \rfloor, m - j)$
 - guess $T(n,m) \leq \alpha \cdot nm$
 - $T(n,m) \lesssim \beta \cdot nm + \alpha \cdot \frac{n}{2} \cdot j +$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n+m)$ time, $O(n+m)$ space
- meet $_{\lfloor \frac{n}{2} \rfloor}(x,y)$: $O(nm)$ time, $O(n+m)$ space
- space recurrence
 - $S(n,m) \leq \max\{O(n+m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n,m) \leq O(n+m)$
- time recurrence
 - $T(n,m) \leq O(nm) + T(\lfloor \frac{n}{2} \rfloor, j) + T(n - \lfloor \frac{n}{2} \rfloor, m - j)$
 - guess $T(n,m) \leq \alpha \cdot nm$
 - $T(n,m) \lesssim \beta \cdot nm + \alpha \cdot \frac{n}{2} \cdot j + \alpha \cdot \frac{n}{2} \cdot (m - j)$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n + m)$ time, $O(n + m)$ space
- meet $_{\lfloor \frac{n}{2} \rfloor}(x, y)$: $O(nm)$ time, $O(n + m)$ space
- space recurrence
 - $S(n, m) \leq \max\{O(n+m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n, m) \leq O(n + m)$
- time recurrence
 - $T(n, m) \leq O(nm) + T(\lfloor \frac{n}{2} \rfloor, j) + T(n - \lfloor \frac{n}{2} \rfloor, m - j)$
 - guess $T(n, m) \leq \alpha \cdot nm$
 - $T(n, m) \lesssim \beta \cdot nm + \alpha \cdot \frac{n}{2} \cdot j + \alpha \cdot \frac{n}{2} \cdot (m - j) = (\beta + \frac{\alpha}{2})nm$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n+m)$ time, $O(n+m)$ space
- meet $_{\lfloor \frac{n}{2} \rfloor}(x,y)$: $O(nm)$ time, $O(n+m)$ space
- space recurrence
 - $S(n,m) \leq \max\{O(n+m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n,m) \leq O(n+m)$
- time recurrence
 - $T(n,m) \leq O(nm) + T(\lfloor \frac{n}{2} \rfloor, j) + T(n - \lfloor \frac{n}{2} \rfloor, m - j)$
 - guess $T(n,m) \leq \alpha \cdot nm$
 - $T(n,m) \lesssim \beta \cdot nm + \alpha \cdot \frac{n}{2} \cdot j + \alpha \cdot \frac{n}{2} \cdot (m - j) = (\beta + \frac{\alpha}{2})nm$
 - ⇒ valid as long as $\alpha \geq 2\beta$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n+m)$ time, $O(n+m)$ space
- meet $_{\lfloor \frac{n}{2} \rfloor}(x,y)$: $O(nm)$ time, $O(n+m)$ space
- space recurrence
 - $S(n,m) \leq \max\{O(n+m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n,m) \leq O(n+m)$
- time recurrence
 - $T(n,m) \leq O(nm) + T(\lfloor \frac{n}{2} \rfloor, j) + T(n - \lfloor \frac{n}{2} \rfloor, m - j)$
 - guess $T(n,m) \leq \alpha \cdot nm$
 - $T(n,m) \lesssim \beta \cdot nm + \alpha \cdot \frac{n}{2} \cdot j + \alpha \cdot \frac{n}{2} \cdot (m - j) = (\beta + \frac{\alpha}{2})nm$
 - ⇒ valid as long as $\alpha \geq 2\beta$
 - ⇒ $T(n,m) \leq O(nm)$

Edit Distance, Better (IV)

divide and conqueror:

```
dist-align'(x1x2⋯xn,y1y2⋯ym)
  if n = 1
    use dist(x,y)
  if m = 1
    use dist(x,y)
  j = meet(⌊ $\frac{n}{2}$ ⌋, x,y)
  A≤ = dist-align'(x≤⌊ $\frac{n}{2}$ ⌋, y≤j)
  A> = dist-align'(x>⌊ $\frac{n}{2}$ ⌋, y>j)
  return A≤ ◦ A>
```

correctness: clear

complexity:

- base cases: $O(n+m)$ time, $O(n+m)$ space
- meet $_{\lfloor \frac{n}{2} \rfloor}(x,y)$: $O(nm)$ time, $O(n+m)$ space
- space recurrence
 - $S(n,m) \leq \max\{O(n+m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}$
 - ⇒ $S(n,m) \leq O(n+m)$
- time recurrence
 - $T(n,m) \leq O(nm) + T(\lfloor \frac{n}{2} \rfloor, j) + T(n - \lfloor \frac{n}{2} \rfloor, m - j)$
 - guess $T(n,m) \leq \alpha \cdot nm$
 - $T(n,m) \lesssim \beta \cdot nm + \alpha \cdot \frac{n}{2} \cdot j + \alpha \cdot \frac{n}{2} \cdot (m - j) = (\beta + \frac{\alpha}{2})nm$
 - ⇒ valid as long as $\alpha \geq 2\beta$
 - ⇒ $T(n,m) \leq O(nm)$

⇒ computing actual alignment in $O(nm)$ -time and $O(n+m)$ -space.

Longest Increasing Subsequence

Longest Increasing Subsequence

Definition

Longest Increasing Subsequence

Definition

A **sequence** of integers,

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n ,

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n , is an ordered list $a_1, a_2, \dots, a_n \in \mathbb{Z}$.

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n , is an ordered list $a_1, a_2, \dots, a_n \in \mathbb{Z}$. The sequence is **increasing** if $a_1 < a_2 < \dots < a_n$.

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n , is an ordered list $a_1, a_2, \dots, a_n \in \mathbb{Z}$. The sequence is **increasing** if $a_1 < a_2 < \dots < a_n$.

A **subsequence** of a_1, a_2, \dots, a_n

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n , is an ordered list $a_1, a_2, \dots, a_n \in \mathbb{Z}$. The sequence is **increasing** if $a_1 < a_2 < \dots < a_n$.

A **subsequence** of a_1, a_2, \dots, a_n is any sequence of the form $a_{i_1}, a_{i_2}, \dots, a_{i_m}$,

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n , is an ordered list $a_1, a_2, \dots, a_n \in \mathbb{Z}$. The sequence is **increasing** if $a_1 < a_2 < \dots < a_n$.

A **subsequence** of a_1, a_2, \dots, a_n is any sequence of the form $a_{i_1}, a_{i_2}, \dots, a_{i_m}$, where $1 \leq i_1 < \dots < i_m \leq n$.

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n , is an ordered list $a_1, a_2, \dots, a_n \in \mathbb{Z}$. The sequence is **increasing** if $a_1 < a_2 < \dots < a_n$.

A **subsequence** of a_1, a_2, \dots, a_n is any sequence of the form $a_{i_1}, a_{i_2}, \dots, a_{i_m}$, where $1 \leq i_1 < \dots < i_m \leq n$. The subsequence is **increasing (IS)** if $a_{i_1} < \dots < a_{i_m}$.

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n , is an ordered list $a_1, a_2, \dots, a_n \in \mathbb{Z}$. The sequence is **increasing** if $a_1 < a_2 < \dots < a_n$.

A **subsequence** of a_1, a_2, \dots, a_n is any sequence of the form $a_{i_1}, a_{i_2}, \dots, a_{i_m}$, where $1 \leq i_1 < \dots < i_m \leq n$. The subsequence is **increasing (IS)** if $a_{i_1} < \dots < a_{i_m}$.

Example

- 02139947200854008540943059472061801

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n , is an ordered list $a_1, a_2, \dots, a_n \in \mathbb{Z}$. The sequence is **increasing** if $a_1 < a_2 < \dots < a_n$.

A **subsequence** of a_1, a_2, \dots, a_n is any sequence of the form $a_{i_1}, a_{i_2}, \dots, a_{i_m}$, where $1 \leq i_1 < \dots < i_m \leq n$. The subsequence is **increasing (IS)** if $a_{i_1} < \dots < a_{i_m}$.

Example

- 02139947200854008540943059472061801 — sequence

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n , is an ordered list $a_1, a_2, \dots, a_n \in \mathbb{Z}$. The sequence is **increasing** if $a_1 < a_2 < \dots < a_n$.

A **subsequence** of a_1, a_2, \dots, a_n is any sequence of the form $a_{i_1}, a_{i_2}, \dots, a_{i_m}$, where $1 \leq i_1 < \dots < i_m \leq n$. The subsequence is **increasing (IS)** if $a_{i_1} < \dots < a_{i_m}$.

Example

- 02139947200854008540943059472061801 — sequence
- 02139947200854008540943059472061801

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n , is an ordered list $a_1, a_2, \dots, a_n \in \mathbb{Z}$. The sequence is **increasing** if $a_1 < a_2 < \dots < a_n$.

A **subsequence** of a_1, a_2, \dots, a_n is any sequence of the form $a_{i_1}, a_{i_2}, \dots, a_{i_m}$, where $1 \leq i_1 < \dots < i_m \leq n$. The subsequence is **increasing (IS)** if $a_{i_1} < \dots < a_{i_m}$.

Example

- 02139947200854008540943059472061801 — sequence
- 02139947200854008540943059472061801 — *subsequence*

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n , is an ordered list $a_1, a_2, \dots, a_n \in \mathbb{Z}$. The sequence is **increasing** if $a_1 < a_2 < \dots < a_n$.

A **subsequence** of a_1, a_2, \dots, a_n is any sequence of the form $a_{i_1}, a_{i_2}, \dots, a_{i_m}$, where $1 \leq i_1 < \dots < i_m \leq n$. The subsequence is **increasing (IS)** if $a_{i_1} < \dots < a_{i_m}$.

Example

- 02139947200854008540943059472061801 — sequence
- 02139947200854008540943059472061801 — *subsequence*
- 02139947200854008540943059472061801

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n , is an ordered list $a_1, a_2, \dots, a_n \in \mathbb{Z}$. The sequence is **increasing** if $a_1 < a_2 < \dots < a_n$.

A **subsequence** of a_1, a_2, \dots, a_n is any sequence of the form $a_{i_1}, a_{i_2}, \dots, a_{i_m}$, where $1 \leq i_1 < \dots < i_m \leq n$. The subsequence is **increasing (IS)** if $a_{i_1} < \dots < a_{i_m}$.

Example

- 02139947200854008540943059472061801 — sequence
- 02139947200854008540943059472061801 — *subsequence*
- 02139947200854008540943059472061801 — *increasing subsequence*

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n , is an ordered list $a_1, a_2, \dots, a_n \in \mathbb{Z}$. The sequence is **increasing** if $a_1 < a_2 < \dots < a_n$.

A **subsequence** of a_1, a_2, \dots, a_n is any sequence of the form $a_{i_1}, a_{i_2}, \dots, a_{i_m}$, where $1 \leq i_1 < \dots < i_m \leq n$. The subsequence is **increasing (IS)** if $a_{i_1} < \dots < a_{i_m}$.

Example

- 02139947200854008540943059472061801 — sequence
- 02139947200854008540943059472061801 — *subsequence*
- 02139947200854008540943059472061801 — *increasing subsequence*
- 02139947200854008540943059472061801

Longest Increasing Subsequence

Definition

A **sequence** of integers, of **length** n , is an ordered list $a_1, a_2, \dots, a_n \in \mathbb{Z}$. The sequence is **increasing** if $a_1 < a_2 < \dots < a_n$.

A **subsequence** of a_1, a_2, \dots, a_n is any sequence of the form $a_{i_1}, a_{i_2}, \dots, a_{i_m}$, where $1 \leq i_1 < \dots < i_m \leq n$. The subsequence is **increasing (IS)** if $a_{i_1} < \dots < a_{i_m}$.

Example

- 02139947200854008540943059472061801 — sequence
- 02139947200854008540943059472061801 — *subsequence*
- 02139947200854008540943059472061801 — *increasing subsequence*
- 02139947200854008540943059472061801 — *longer increasing subsequence*

Longest Increasing Subsequence (II)

Longest Increasing Subsequence (II)

Definition

Longest Increasing Subsequence (II)

Definition

The **longest increasing subsequence problem (LIS)** is to,

Longest Increasing Subsequence (II)

Definition

The **longest increasing subsequence problem (LIS)** is to, given a sequence $a_1, a_2, \dots, a_n \in \mathbb{Z}$,

Longest Increasing Subsequence (II)

Definition

The **longest increasing subsequence problem (LIS)** is to, given a sequence $a_1, a_2, \dots, a_n \in \mathbb{Z}$, compute the (length of) the longest increasing subsequence.

Longest Increasing Subsequence (II)

Definition

The **longest increasing subsequence problem (LIS)** is to, given a sequence $a_1, a_2, \dots, a_n \in \mathbb{Z}$, compute the (length of) the longest increasing subsequence.

goal:

Longest Increasing Subsequence (II)

Definition

The **longest increasing subsequence problem (LIS)** is to, given a sequence $a_1, a_2, \dots, a_n \in \mathbb{Z}$, compute the (length of) the longest increasing subsequence.

goal: solve with dynamic programming

Longest Increasing Subsequence (II)

Definition

The **longest increasing subsequence problem (LIS)** is to, given a sequence $a_1, a_2, \dots, a_n \in \mathbb{Z}$, compute the (length of) the longest increasing subsequence.

goal: solve with dynamic programming

- identify subproblems

Longest Increasing Subsequence (II)

Definition

The **longest increasing subsequence problem (LIS)** is to, given a sequence $a_1, a_2, \dots, a_n \in \mathbb{Z}$, compute the (length of) the longest increasing subsequence.

goal: solve with dynamic programming

- identify subproblems
- develop recursion

Longest Increasing Subsequence (II)

Definition

The **longest increasing subsequence problem (LIS)** is to, given a sequence $a_1, a_2, \dots, a_n \in \mathbb{Z}$, compute the (length of) the longest increasing subsequence.

goal: solve with dynamic programming

- identify subproblems
- develop recursion
- memoize

Longest Increasing Subsequence (II)

Definition

The **longest increasing subsequence problem (LIS)** is to, given a sequence $a_1, a_2, \dots, a_n \in \mathbb{Z}$, compute the (length of) the longest increasing subsequence.

goal: solve with dynamic programming

- identify subproblems
- develop recursion
- memoize
- analyze

Longest Increasing Subsequence (II)

Definition

The **longest increasing subsequence problem (LIS)** is to, given a sequence $a_1, a_2, \dots, a_n \in \mathbb{Z}$, compute the (length of) the longest increasing subsequence.

goal: solve with dynamic programming

- identify subproblems
- develop recursion
- memoize
- analyze
- optimize *time*

Longest Increasing Subsequence (II)

Definition

The **longest increasing subsequence problem (LIS)** is to, given a sequence $a_1, a_2, \dots, a_n \in \mathbb{Z}$, compute the (length of) the longest increasing subsequence.

goal: solve with dynamic programming

- identify subproblems
- develop recursion
- memoize
- analyze
- optimize *time*

remark:

Longest Increasing Subsequence (II)

Definition

The **longest increasing subsequence problem (LIS)** is to, given a sequence $a_1, a_2, \dots, a_n \in \mathbb{Z}$, compute the (length of) the longest increasing subsequence.

goal: solve with dynamic programming

- identify subproblems
- develop recursion
- memoize
- analyze
- optimize *time*

remark: without loss of generality the a_i are *distinct*,

Longest Increasing Subsequence (II)

Definition

The **longest increasing subsequence problem (LIS)** is to, given a sequence $a_1, a_2, \dots, a_n \in \mathbb{Z}$, compute the (length of) the longest increasing subsequence.

goal: solve with dynamic programming

- identify subproblems
- develop recursion
- memoize
- analyze
- optimize *time*

remark: without loss of generality the a_i are *distinct*, up to a cost of $\Theta(n \log n)$ in runtime

Longest Increasing Subsequence (II)

Definition

The **longest increasing subsequence problem (LIS)** is to, given a sequence $a_1, a_2, \dots, a_n \in \mathbb{Z}$, compute the (length of) the longest increasing subsequence.

goal: solve with dynamic programming

- identify subproblems
- develop recursion
- memoize
- analyze
- optimize *time*

remark: without loss of generality the a_i are *distinct*, up to a cost of $\Theta(n \log n)$ in runtime (**exercise**)

Longest Increasing Subsequence (III)

Longest Increasing Subsequence (III)

Lemma

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$,

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence.

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n .

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

$$\mathbf{1} \text{ LIS}(a_1, a_2, \dots, a_n) =$$

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

$$\mathbf{1} \quad \text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n}$$

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

$$\mathbf{1} \quad \text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*($$

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

$$\mathbf{1} \quad \text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i).$$

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

1 $\text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i)$.

2 $\text{LIS}^*(a_1, a_2, \dots, a_n) =$

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

- 1 $\text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i)$.
- 2 $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n}$

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

$$1 \quad \text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i).$$

$$2 \quad \text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 +$$

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

$$1 \quad \text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i).$$

$$2 \quad \text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_i)\},$$

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

$$1 \quad \text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i).$$

$$2 \quad \text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_i), 1\}.$$

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

$$1 \quad \text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i).$$

$$2 \quad \text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_i), 1\}.$$

Proof.

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

$$1 \quad \text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i).$$

$$2 \quad \text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_i), 1\}.$$

Proof.

1 Clear.

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

- 1 $\text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i)$.
- 2 $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_i), 1\}$.

Proof.

- 1 Clear.
- 2 For i with $a_i < a_n$,

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

- 1 $\text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i)$.
- 2 $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_i), 1\}$.

Proof.

- 1 Clear.
- 2 For i with $a_i < a_n$, an IS^{*} $a_{i_1} < \dots < a_{i_{m-1}} < a_{i_m}$ of $\bar{a}_{\leq i}$

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

- 1 $\text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i)$.
- 2 $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_i), 1\}$.

Proof.

- 1 Clear.
- 2 For i with $a_i < a_n$, an IS^{*} $a_{i_1} < \dots < a_{i_{m-1}} < a_{i_m=i}$ of $\bar{a}_{\leq i}$

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

- 1 $\text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i)$.
- 2 $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_i), 1\}$.

Proof.

- 1 Clear.
- 2 For i with $a_i < a_n$, an IS^* $a_{i_1} < \dots < a_{i_{m-1}} < a_{i_m=i}$ of $\bar{a}_{\leq i}$ can append a_n

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

- 1 $\text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i)$.
- 2 $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_i), 1\}$.

Proof.

- 1 Clear.
- 2 For i with $a_i < a_n$, an IS^* $a_{i_1} < \dots < a_{i_{m-1}} < a_{i_m=i}$ of $\bar{a}_{\leq i}$ can append a_n to yield an IS^* $a_{i_1} < \dots < a_{i_{m-1}} < a_i < a_n$ of \bar{a} ,

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

- 1 $\text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i)$.
- 2 $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_i), 1\}$.

Proof.

- 1 Clear.
- 2 For i with $a_i < a_n$, an IS^* $a_{i_1} < \dots < a_{i_{m-1}} < a_{i_m=i}$ of $\bar{a}_{\leq i}$ can append a_n to yield an IS^* $a_{i_1} < \dots < a_{i_{m-1}} < a_i < a_n$ of \bar{a} , and every IS^* of \bar{a} can be decomposed this way,

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

- 1 $\text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i)$.
- 2 $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_i), 1\}$.

Proof.

- 1 Clear.
- 2 For i with $a_i < a_n$, an IS^* $a_{i_1} < \dots < a_{i_{m-1}} < a_{i_m=i}$ of $\bar{a}_{\leq i}$ can append a_n to yield an IS^* $a_{i_1} < \dots < a_{i_{m-1}} < a_i < a_n$ of \bar{a} , and every IS^* of \bar{a} can be decomposed this way, or by taking the singleton sequence a_n .

Longest Increasing Subsequence (III)

Lemma

For a sequence $\bar{a} = a_1, a_2, \dots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that **contains the last element** a_n . Then

- 1 $\text{LIS}(a_1, a_2, \dots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \dots, a_i)$.
- 2 $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_i), 1\}$.

Proof.

- 1 Clear.
- 2 For i with $a_i < a_n$, an IS^* $a_{i_1} < \dots < a_{i_{m-1}} < a_{i_m=i}$ of $\bar{a}_{\leq i}$ can append a_n to yield an IS^* $a_{i_1} < \dots < a_{i_{m-1}} < a_i < a_n$ of \bar{a} , and every IS^* of \bar{a} can be decomposed this way, or by taking the singleton sequence a_n . Now take maximums. \square

Longest Increasing Subsequence (IV)

Longest Increasing Subsequence (IV)

Lemma

Define $\text{LIS}^(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element a_n . Then $\text{LIS}^*(a_1, a_2, \dots, a_n) =$*

Longest Increasing Subsequence (IV)

Lemma

Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element a_n . Then $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n}$

Longest Increasing Subsequence (IV)

Lemma

Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element a_n . Then $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 +$

Longest Increasing Subsequence (IV)

Lemma

Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element a_n . Then $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(\bar{a}_{\leq i})\}$,

Longest Increasing Subsequence (IV)

Lemma

Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element a_n . Then $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(\bar{a}_{\leq i}), 1\}$.

Longest Increasing Subsequence (IV)

Lemma

Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element a_n . Then $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(\bar{a}_{\leq i}), 1\}$.

Example

02139947200854008540943059472061801

Longest Increasing Subsequence (IV)

Lemma

Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element a_n . Then $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(\bar{a}_{\leq i}), 1\}$.

Example

02139947200854008540943059472061801

1 02139947200854008540943059472061801

Longest Increasing Subsequence (IV)

Lemma

Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element a_n . Then $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(\bar{a}_{\leq i}), 1\}$.

Example

02139947200854008540943059472061801

1 02139947200854008540943059472061801 — $\text{LIS}^*(a_1) = 1$

Longest Increasing Subsequence (IV)

Lemma

Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element a_n . Then $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(\bar{a}_{\leq i}), 1\}$.

Example

02139947200854008540943059472061801

1 02139947200854008540943059472061801 — $\text{LIS}^*(a_1) = 1$

2 02139947200854008540943059472061801

Longest Increasing Subsequence (IV)

Lemma

Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element a_n . Then $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(\bar{a}_{\leq i}), 1\}$.

Example

02139947200854008540943059472061801

1 02139947200854008540943059472061801 — $\text{LIS}^*(a_1) = 1$

2 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, a_2) = 2$

Longest Increasing Subsequence (IV)

Lemma

Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element a_n . Then $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(\bar{a}_{\leq i}), 1\}$.

Example

02139947200854008540943059472061801

- 1 02139947200854008540943059472061801 — $\text{LIS}^*(a_1) = 1$
- 2 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, a_2) = 2$
- 3 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, \dots, a_3) = 2$

Longest Increasing Subsequence (IV)

Lemma

Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element a_n . Then $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(\bar{a}_{\leq i}), 1\}$.

Example

02139947200854008540943059472061801

- 1 02139947200854008540943059472061801 — $\text{LIS}^*(a_1) = 1$
- 2 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, a_2) = 2$
- 3 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, \dots, a_3) = 2$
- 4 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, \dots, a_4) = 3$

Longest Increasing Subsequence (IV)

Lemma

Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element a_n . Then $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(\bar{a}_{\leq i}), 1\}$.

Example

02139947200854008540943059472061801

- 1 02139947200854008540943059472061801 — $\text{LIS}^*(a_1) = 1$
- 2 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, a_2) = 2$
- 3 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, \dots, a_3) = 2$
- 4 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, \dots, a_4) = 3$
- 5 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, \dots, a_5) = 4$

Longest Increasing Subsequence (IV)

Lemma

Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element a_n . Then $\text{LIS}^*(a_1, a_2, \dots, a_n) = \max_{i: a_i < a_n} \{1 + \text{LIS}^*(\bar{a}_{\leq i}), 1\}$.

Example

02139947200854008540943059472061801

- 1 02139947200854008540943059472061801 — $\text{LIS}^*(a_1) = 1$
- 2 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, a_2) = 2$
- 3 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, \dots, a_3) = 2$
- 4 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, \dots, a_4) = 3$
- 5 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, \dots, a_5) = 4$
- 6 02139947200854008540943059472061801 — $\text{LIS}^*(a_1, \dots, a_6) = 4$

Longest Increasing Subsequence (V)

Longest Increasing Subsequence (V)

iterative algorithm:

Longest Increasing Subsequence (V)

iterative algorithm:

$\text{LIS}(a_1, a_2, \dots, a_n)$:

Longest Increasing Subsequence (V)

iterative algorithm:

LIS(a_1, a_2, \dots, a_n):
 for $1 \leq i \leq n$

Longest Increasing Subsequence (V)

iterative algorithm:

LIS(a_1, a_2, \dots, a_n):

for $1 \leq i \leq n$

$L^*[i] = 1$

Longest Increasing Subsequence (V)

iterative algorithm:

LIS(a_1, a_2, \dots, a_n):

for $1 \leq i \leq n$

$L^*[i] = 1$

$L = 0$

Longest Increasing Subsequence (V)

iterative algorithm:

LIS(a_1, a_2, \dots, a_n):

for $1 \leq i \leq n$

$L^*[i] = 1$

$L = 0$

for $1 \leq i \leq n$

Longest Increasing Subsequence (V)

iterative algorithm:

LIS(a_1, a_2, \dots, a_n):

for $1 \leq i \leq n$

$L^*[i] = 1$

$L = 0$

for $1 \leq i \leq n$

for $1 \leq j < i$

Longest Increasing Subsequence (V)

iterative algorithm:

LIS(a_1, a_2, \dots, a_n):

for $1 \leq i \leq n$

$L^*[i] = 1$

$L = 0$

for $1 \leq i \leq n$

for $1 \leq j < i$

if $a_j < a_i$

Longest Increasing Subsequence (V)

iterative algorithm:

LIS(a_1, a_2, \dots, a_n):

for $1 \leq i \leq n$

$L^*[i] = 1$

$L = 0$

for $1 \leq i \leq n$

for $1 \leq j < i$

if $a_j < a_i$

$L^*[i] = \max\{L^*[i], 1 + L^*[j]\}$

Longest Increasing Subsequence (V)

iterative algorithm:

LIS(a_1, a_2, \dots, a_n):

for $1 \leq i \leq n$

$L^*[i] = 1$

$L = 0$

for $1 \leq i \leq n$

for $1 \leq j < i$

if $a_j < a_i$

$L^*[i] = \max\{L^*[i], 1 + L^*[j]\}$

$L = \max\{L, L^*[i]\}$

Longest Increasing Subsequence (V)

iterative algorithm:

```
LIS( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq i \leq n$   
     $L^*[i] = 1$   
   $L = 0$   
  for  $1 \leq i \leq n$   
    for  $1 \leq j < i$   
      if  $a_j < a_i$   
         $L^*[i] = \max\{L^*[i], 1 + L^*[j]\}$   
   $L = \max\{L, L^*[i]\}$   
  return  $L$ 
```


Longest Increasing Subsequence (V)

iterative algorithm:

```
LIS( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq i \leq n$   
     $L^*[i] = 1$   
   $L = 0$   
  for  $1 \leq i \leq n$   
    for  $1 \leq j < i$   
      if  $a_j < a_i$   
         $L^*[i] = \max\{L^*[i], 1 + L^*[j]\}$   
   $L = \max\{L, L^*[i]\}$   
  return  $L$ 
```

Longest Increasing Subsequence (V)

iterative algorithm:

```
LIS( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq i \leq n$   
     $L^*[i] = 1$   
   $L = 0$   
  for  $1 \leq i \leq n$   
    for  $1 \leq j < i$   
      if  $a_j < a_i$   
         $L^*[i] = \max\{L^*[i], 1 + L^*[j]\}$   
   $L = \max\{L, L^*[i]\}$   
  return  $L$ 
```

correctness:

Longest Increasing Subsequence (V)

iterative algorithm:

```
LIS( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq i \leq n$   
     $L^*[i] = 1$   
   $L = 0$   
  for  $1 \leq i \leq n$   
    for  $1 \leq j < i$   
      if  $a_j < a_i$   
         $L^*[i] = \max\{L^*[i], 1 + L^*[j]\}$   
   $L = \max\{L, L^*[i]\}$   
  return  $L$ 
```

correctness: clear

Longest Increasing Subsequence (V)

iterative algorithm:

```
LIS( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq i \leq n$   
     $L^*[i] = 1$   
   $L = 0$   
  for  $1 \leq i \leq n$   
    for  $1 \leq j < i$   
      if  $a_j < a_i$   
         $L^*[i] = \max\{L^*[i], 1 + L^*[j]\}$   
   $L = \max\{L, L^*[i]\}$   
  return  $L$ 
```

correctness: clear

complexity:

Longest Increasing Subsequence (V)

iterative algorithm:

```
LIS( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq i \leq n$   
     $L^*[i] = 1$   
   $L = 0$   
  for  $1 \leq i \leq n$   
    for  $1 \leq j < i$   
      if  $a_j < a_i$   
         $L^*[i] = \max\{L^*[i], 1 + L^*[j]\}$   
   $L = \max\{L, L^*[i]\}$   
  return  $L$ 
```

correctness: clear

complexity:

- $O(n)$ space

Longest Increasing Subsequence (V)

iterative algorithm:

```
LIS( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq i \leq n$   
     $L^*[i] = 1$   
   $L = 0$   
  for  $1 \leq i \leq n$   
    for  $1 \leq j < i$   
      if  $a_j < a_i$   
         $L^*[i] = \max\{L^*[i], 1 + L^*[j]\}$   
   $L = \max\{L, L^*[i]\}$   
  return  $L$ 
```

correctness: clear

complexity:

- $O(n)$ space
- $O(n^2)$ time

Longest Increasing Subsequence (V)

iterative algorithm:

```
LIS( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq i \leq n$   
     $L^*[i] = 1$   
   $L = 0$   
  for  $1 \leq i \leq n$   
    for  $1 \leq j < i$   
      if  $a_j < a_i$   
         $L^*[i] = \max\{L^*[i], 1 + L^*[j]\}$   
   $L = \max\{L, L^*[i]\}$   
  return  $L$ 
```

correctness: clear

complexity:

- $O(n)$ space
- $O(n^2)$ time — *do better?*

Longest Increasing Subsequence, Faster

Longest Increasing Subsequence, Faster

$$\text{LIS}^*(a_1, a_2, \dots, a_i) = \max_{j: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_j), 1\} .$$

Longest Increasing Subsequence, Faster

$$\text{LIS}^*(a_1, a_2, \dots, a_i) = \max_{j: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_j), 1\} .$$

This recursive step does too much

Longest Increasing Subsequence, Faster

$$\text{LIS}^*(a_1, a_2, \dots, a_i) = \max_{j: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_j), 1\} .$$

This recursive step does too much — all (a_j, a_i) are compared!

Longest Increasing Subsequence, Faster

$$\text{LIS}^*(a_1, a_2, \dots, a_i) = \max_{j: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_j), 1\} .$$

This recursive step does too much — all (a_j, a_i) are compared! Use sorting?

Longest Increasing Subsequence, Faster

$$\text{LIS}^*(a_1, a_2, \dots, a_i) = \max_{j: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_j), 1\} .$$

This recursive step does too much — all (a_j, a_i) are compared! Use sorting?

idea:

Longest Increasing Subsequence, Faster

$$\text{LIS}^*(a_1, a_2, \dots, a_i) = \max_{j: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_j), 1\} .$$

This recursive step does too much — all (a_j, a_i) are compared! Use sorting?

idea: define subproblem based on *length* of increasing subsequences

Longest Increasing Subsequence, Faster

$$\text{LIS}^*(a_1, a_2, \dots, a_i) = \max_{j: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_j), 1\} .$$

This recursive step does too much — all (a_j, a_i) are compared! Use sorting?

idea: define subproblem based on *length* of increasing subsequences

Definition

Longest Increasing Subsequence, Faster

$$\text{LIS}^*(a_1, a_2, \dots, a_i) = \max_{j: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_j), 1\} .$$

This recursive step does too much — all (a_j, a_i) are compared! Use sorting?

idea: define subproblem based on *length* of increasing subsequences

Definition

For sequence a_1, a_2, \dots, a_n ,

Longest Increasing Subsequence, Faster

$$\text{LIS}^*(a_1, a_2, \dots, a_i) = \max_{j: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_j), 1\} .$$

This recursive step does too much — all (a_j, a_i) are compared! Use sorting?

idea: define subproblem based on *length* of increasing subsequences

Definition

For sequence a_1, a_2, \dots, a_n , define the **end of increasing subsequence** $\text{EIS}(\ell, \bar{a})$

Longest Increasing Subsequence, Faster

$$\text{LIS}^*(a_1, a_2, \dots, a_i) = \max_{j: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_j), 1\} .$$

This recursive step does too much — all (a_j, a_i) are compared! Use sorting?

idea: define subproblem based on *length* of increasing subsequences

Definition

For sequence a_1, a_2, \dots, a_n , define the **end of increasing subsequence** $\text{EIS}(\ell, \bar{a})$ to be the minimum a_j such that there is an increasing sequence of length ℓ that terminates at a_j ,

Longest Increasing Subsequence, Faster

$$\text{LIS}^*(a_1, a_2, \dots, a_i) = \max_{j: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_j), 1\} .$$

This recursive step does too much — all (a_j, a_i) are compared! Use sorting?

idea: define subproblem based on *length* of increasing subsequences

Definition

For sequence a_1, a_2, \dots, a_n , define the **end of increasing subsequence** $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i , that is,

$$\text{EIS}(\ell, \bar{a}) := \min_{i: a_{i_1} < a_{i_2} < \dots < a_{i_\ell}} a_i .$$

Longest Increasing Subsequence, Faster

$$\text{LIS}^*(a_1, a_2, \dots, a_i) = \max_{j: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_j), 1\} .$$

This recursive step does too much — all (a_j, a_i) are compared! Use sorting?

idea: define subproblem based on *length* of increasing subsequences

Definition

For sequence a_1, a_2, \dots, a_n , define the **end of increasing subsequence** $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i , that is,

$$\text{EIS}(\ell, \bar{a}) := \min_{i: a_{i_1} < a_{i_2} < \dots < a_{i_\ell} = i} a_i .$$

Longest Increasing Subsequence, Faster

$$\text{LIS}^*(a_1, a_2, \dots, a_i) = \max_{j: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_j), 1\} .$$

This recursive step does too much — all (a_j, a_i) are compared! Use sorting?

idea: define subproblem based on *length* of increasing subsequences

Definition

For sequence a_1, a_2, \dots, a_n , define the **end of increasing subsequence** $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i , that is,

$$\text{EIS}(\ell, \bar{a}) := \min_{i: a_{i_1} < a_{i_2} < \dots < a_{i_\ell} = i} a_i .$$

$\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Longest Increasing Subsequence, Faster

$$\text{LIS}^*(a_1, a_2, \dots, a_i) = \max_{j: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \dots, a_j), 1\} .$$

This recursive step does too much — all (a_j, a_i) are compared! Use sorting?

idea: define subproblem based on *length* of increasing subsequences

Definition

For sequence a_1, a_2, \dots, a_n , define the **end of increasing subsequence** $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i , that is,

$$\text{EIS}(\ell, \bar{a}) := \min_{i: a_{i_1} < a_{i_2} < \dots < a_{i_\ell} = i} a_i .$$

$\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

intuition: prefer the '*smallest*' IS of each size

Longest Increasing Subsequence, Faster (II)

Longest Increasing Subsequence, Faster (II)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_j such that there is an increasing sequence of length ℓ that terminates at a_j .

Longest Increasing Subsequence, Faster (II)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Longest Increasing Subsequence, Faster (II)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

Longest Increasing Subsequence, Faster (II)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

$$\text{LIS}(\bar{a}) = \max_{\ell: \text{EIS}(\ell, \bar{a}) < \infty} \ell.$$

Longest Increasing Subsequence, Faster (II)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

$$\text{LIS}(\bar{a}) = \max_{\ell: \text{EIS}(\ell, \bar{a}) < \infty} \ell.$$

Proof.

Longest Increasing Subsequence, Faster (II)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

$$\text{LIS}(\bar{a}) = \max_{\ell: \text{EIS}(\ell, \bar{a}) < \infty} \ell.$$

Proof.

Clear. □

Longest Increasing Subsequence, Faster (III)

Longest Increasing Subsequence, Faster (III)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Longest Increasing Subsequence, Faster (III)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

Longest Increasing Subsequence, Faster (III)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

For sequence a_1, a_2, \dots, a_n ,

Longest Increasing Subsequence, Faster (III)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

For sequence a_1, a_2, \dots, a_n , $\text{EIS}(\ell, \bar{a}) < \text{EIS}(\ell + 1, \bar{a})$,

Longest Increasing Subsequence, Faster (III)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

For sequence a_1, a_2, \dots, a_n , $\text{EIS}(\ell, \bar{a}) < \text{EIS}(\ell + 1, \bar{a})$, for all ℓ .

Longest Increasing Subsequence, Faster (III)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

For sequence a_1, a_2, \dots, a_n , $\text{EIS}(\ell, \bar{a}) < \text{EIS}(\ell + 1, \bar{a})$, for all ℓ . That is, $\text{EIS}(\cdot, \bar{a})$ is a strictly sorted sequence.

Longest Increasing Subsequence, Faster (III)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

For sequence a_1, a_2, \dots, a_n , $\text{EIS}(\ell, \bar{a}) < \text{EIS}(\ell + 1, \bar{a})$, for all ℓ . That is, $\text{EIS}(\cdot, \bar{a})$ is a strictly sorted sequence.

Proof.

Longest Increasing Subsequence, Faster (III)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

For sequence a_1, a_2, \dots, a_n , $\text{EIS}(\ell, \bar{a}) < \text{EIS}(\ell + 1, \bar{a})$, for all ℓ . That is, $\text{EIS}(\cdot, \bar{a})$ is a strictly sorted sequence.

Proof.

Let $a_{i_1} < a_{i_2} < \dots < a_{i_\ell}$ be a witness for $\text{EIS}(\ell, \bar{a}) = a_{i_\ell}$,

Longest Increasing Subsequence, Faster (III)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

For sequence a_1, a_2, \dots, a_n , $\text{EIS}(\ell, \bar{a}) < \text{EIS}(\ell + 1, \bar{a})$, for all ℓ . That is, $\text{EIS}(\cdot, \bar{a})$ is a strictly sorted sequence.

Proof.

Let $a_{i_1} < a_{i_2} < \dots < a_{i_\ell}$ be a witness for $\text{EIS}(\ell, \bar{a}) = a_{i_\ell}$, and let $a_{i'_1} < a_{i'_2} < \dots < a_{i'_\ell} < a_{i'_{\ell+1}}$ be a witness for $\text{EIS}(\ell + 1, \bar{a}) = a_{i'_{\ell+1}}$.

Longest Increasing Subsequence, Faster (III)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

For sequence a_1, a_2, \dots, a_n , $\text{EIS}(\ell, \bar{a}) < \text{EIS}(\ell + 1, \bar{a})$, for all ℓ . That is, $\text{EIS}(\cdot, \bar{a})$ is a strictly sorted sequence.

Proof.

Let $a_{i_1} < a_{i_2} < \dots < a_{i_\ell}$ be a witness for $\text{EIS}(\ell, \bar{a}) = a_{i_\ell}$, and let $a_{i'_1} < a_{i'_2} < \dots < a_{i'_\ell} < a_{i'_{\ell+1}}$ be a witness for $\text{EIS}(\ell + 1, \bar{a}) = a_{i'_{\ell+1}}$. Then as $a_{i'_1} < a_{i'_2} < \dots < a_{i'_\ell}$ is length- ℓ increasing sequence

Longest Increasing Subsequence, Faster (III)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

For sequence a_1, a_2, \dots, a_n , $\text{EIS}(\ell, \bar{a}) < \text{EIS}(\ell + 1, \bar{a})$, for all ℓ . That is, $\text{EIS}(\cdot, \bar{a})$ is a strictly sorted sequence.

Proof.

Let $a_{i_1} < a_{i_2} < \dots < a_{i_\ell}$ be a witness for $\text{EIS}(\ell, \bar{a}) = a_{i_\ell}$, and let $a_{i'_1} < a_{i'_2} < \dots < a_{i'_\ell} < a_{i'_{\ell+1}}$ be a witness for $\text{EIS}(\ell + 1, \bar{a}) = a_{i'_{\ell+1}}$. Then as $a_{i'_1} < a_{i'_2} < \dots < a_{i'_\ell}$ is length- ℓ increasing sequence we have that $\text{EIS}(\ell, \bar{a}) \leq a_{i'_\ell}$

Longest Increasing Subsequence, Faster (III)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

For sequence a_1, a_2, \dots, a_n , $\text{EIS}(\ell, \bar{a}) < \text{EIS}(\ell + 1, \bar{a})$, for all ℓ . That is, $\text{EIS}(\cdot, \bar{a})$ is a strictly sorted sequence.

Proof.

Let $a_{i_1} < a_{i_2} < \dots < a_{i_\ell}$ be a witness for $\text{EIS}(\ell, \bar{a}) = a_{i_\ell}$, and let $a_{i'_1} < a_{i'_2} < \dots < a_{i'_\ell} < a_{i'_{\ell+1}}$ be a witness for $\text{EIS}(\ell + 1, \bar{a}) = a_{i'_{\ell+1}}$. Then as $a_{i'_1} < a_{i'_2} < \dots < a_{i'_\ell}$ is length- ℓ increasing sequence we have that $\text{EIS}(\ell, \bar{a}) \leq a_{i'_\ell} < a_{i'_{\ell+1}}$

Longest Increasing Subsequence, Faster (III)

Definition

For sequence a_1, a_2, \dots, a_n , define $\text{EIS}(\ell, \bar{a})$ to be the minimum a_i such that there is an increasing sequence of length ℓ that terminates at a_i . $\text{EIS}(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

Lemma

For sequence a_1, a_2, \dots, a_n , $\text{EIS}(\ell, \bar{a}) < \text{EIS}(\ell + 1, \bar{a})$, for all ℓ . That is, $\text{EIS}(\cdot, \bar{a})$ is a strictly sorted sequence.

Proof.

Let $a_{i_1} < a_{i_2} < \dots < a_{i_\ell}$ be a witness for $\text{EIS}(\ell, \bar{a}) = a_{i_\ell}$, and let $a_{i'_1} < a_{i'_2} < \dots < a_{i'_\ell} < a_{i'_{\ell+1}}$ be a witness for $\text{EIS}(\ell + 1, \bar{a}) = a_{i'_{\ell+1}}$. Then as $a_{i'_1} < a_{i'_2} < \dots < a_{i'_\ell}$ is length- ℓ increasing sequence we have that $\text{EIS}(\ell, \bar{a}) \leq a_{i'_\ell} < a_{i'_{\ell+1}} = \text{EIS}(\ell + 1, \bar{a})$. □

Longest Increasing Subsequence, Faster (IV)

Longest Increasing Subsequence, Faster (IV)

Lemma

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$$

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$$

$$\mathbf{1} \text{ EIS}(\ell, \bar{a}),$$

Lemma

$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$

- 1 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell, \bar{a}) < a_{n+1}$

Lemma

$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$

- 1 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell, \bar{a}) < a_{n+1}$
- 2 $\text{EIS}(\ell, \bar{a})$,

Lemma

$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$

- 1 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell, \bar{a}) < a_{n+1}$
- 2 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell - 1, \bar{a}) > a_{n+1}$

Lemma

$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$

- 1 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell, \bar{a}) < a_{n+1}$
- 2 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell - 1, \bar{a}) > a_{n+1}$
- 3 a_{n+1} ,

Lemma

$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$

- 1 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell, \bar{a}) < a_{n+1}$
- 2 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell - 1, \bar{a}) > a_{n+1}$
- 3 a_{n+1} , if $\text{EIS}(\ell, \bar{a}) > a_{n+1}$

Lemma

$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$

- 1 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell, \bar{a}) < a_{n+1}$
- 2 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell - 1, \bar{a}) > a_{n+1}$
- 3 a_{n+1} , if $\text{EIS}(\ell, \bar{a}) > a_{n+1}$ **and** $\text{EIS}(\ell - 1, \bar{a}) < a_{n+1}$

Longest Increasing Subsequence, Faster (IV)

Lemma

$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$

- 1 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell, \bar{a}) < a_{n+1}$
- 2 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell - 1, \bar{a}) > a_{n+1}$
- 3 a_{n+1} , if $\text{EIS}(\ell, \bar{a}) > a_{n+1}$ **and** $\text{EIS}(\ell - 1, \bar{a}) < a_{n+1}$

Proof.

Longest Increasing Subsequence, Faster (IV)

Lemma

$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$

- 1 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell, \bar{a}) < a_{n+1}$
- 2 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell - 1, \bar{a}) > a_{n+1}$
- 3 a_{n+1} , if $\text{EIS}(\ell, \bar{a}) > a_{n+1}$ **and** $\text{EIS}(\ell - 1, \bar{a}) < a_{n+1}$

Proof.

- 1 Clear.

Longest Increasing Subsequence, Faster (IV)

Lemma

$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$

- 1 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell, \bar{a}) < a_{n+1}$
- 2 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell - 1, \bar{a}) > a_{n+1}$
- 3 a_{n+1} , if $\text{EIS}(\ell, \bar{a}) > a_{n+1}$ **and** $\text{EIS}(\ell - 1, \bar{a}) < a_{n+1}$

Proof.

- 1 Clear.
- 2 Clear.

Longest Increasing Subsequence, Faster (IV)

Lemma

$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$

- 1 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell, \bar{a}) < a_{n+1}$
- 2 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell - 1, \bar{a}) > a_{n+1}$
- 3 a_{n+1} , if $\text{EIS}(\ell, \bar{a}) > a_{n+1}$ **and** $\text{EIS}(\ell - 1, \bar{a}) < a_{n+1}$

Proof.

- 1 Clear.
- 2 Clear.
- 3 Exists increasing sequence of length ℓ terminating at a_{n+1}

Longest Increasing Subsequence, Faster (IV)

Lemma

$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$

- 1 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell, \bar{a}) < a_{n+1}$
- 2 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell - 1, \bar{a}) > a_{n+1}$
- 3 a_{n+1} , if $\text{EIS}(\ell, \bar{a}) > a_{n+1}$ **and** $\text{EIS}(\ell - 1, \bar{a}) < a_{n+1}$

Proof.

- 1 Clear.
- 2 Clear.
- 3 Exists increasing sequence of length ℓ terminating at a_{n+1}
iff exists increasing sequence of length $\ell - 1$ terminating at $a_i < a_{n+1}$,

Longest Increasing Subsequence, Faster (IV)

Lemma

$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$

- 1 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell, \bar{a}) < a_{n+1}$
- 2 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell - 1, \bar{a}) > a_{n+1}$
- 3 a_{n+1} , if $\text{EIS}(\ell, \bar{a}) > a_{n+1}$ **and** $\text{EIS}(\ell - 1, \bar{a}) < a_{n+1}$

Proof.

- 1 Clear.
- 2 Clear.
- 3 Exists increasing sequence of length ℓ terminating at a_{n+1}
iff exists increasing sequence of length $\ell - 1$ terminating at $a_i < a_{n+1}$, for some i

Longest Increasing Subsequence, Faster (IV)

Lemma

$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) =$

- 1 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell, \bar{a}) < a_{n+1}$
- 2 $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell - 1, \bar{a}) > a_{n+1}$
- 3 a_{n+1} , if $\text{EIS}(\ell, \bar{a}) > a_{n+1}$ **and** $\text{EIS}(\ell - 1, \bar{a}) < a_{n+1}$

Proof.

- 1 Clear.
- 2 Clear.
- 3 Exists increasing sequence of length ℓ terminating at a_{n+1}
 - iff exists increasing sequence of length $\ell - 1$ terminating at $a_i < a_{n+1}$, for some i
 - iff exists increasing sequence of length $\ell - 1$ terminating at $\text{EIS}(\ell - 1, \bar{a}) < a_{n+1}$ \square

Longest Increasing Subsequence, Faster (V)

Longest Increasing Subsequence, Faster (V)

Lemma

For a fixed \bar{a} , $\text{EIS}(\ell, \bar{a})$ strictly increases with ℓ .

Longest Increasing Subsequence, Faster (V)

Lemma

For a fixed \bar{a} , $\text{EIS}(\ell, \bar{a})$ strictly increases with ℓ .

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1}))$$

Longest Increasing Subsequence, Faster (V)

Lemma

For a fixed \bar{a} , $\text{EIS}(\ell, \bar{a})$ strictly increases with ℓ .

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) = \begin{cases} a_{n+1} \end{cases}$$

Longest Increasing Subsequence, Faster (V)

Lemma

For a fixed \bar{a} , $\text{EIS}(\ell, \bar{a})$ strictly increases with ℓ .

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) = \begin{cases} a_{n+1} & \text{EIS}(\ell - 1, \bar{a}) < a_{n+1} < \text{EIS}(\ell, \bar{a}) \end{cases}$$

Longest Increasing Subsequence, Faster (V)

Lemma

For a fixed \bar{a} , $\text{EIS}(\ell, \bar{a})$ strictly increases with ℓ .

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) = \begin{cases} a_{n+1} & \text{EIS}(\ell - 1, \bar{a}) < a_{n+1} < \text{EIS}(\ell, \bar{a}) \\ \text{EIS}(\ell, \bar{a}) & \text{else} \end{cases}$$

Longest Increasing Subsequence, Faster (V)

Lemma

For a fixed \bar{a} , $\text{EIS}(\ell, \bar{a})$ strictly increases with ℓ .

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) = \begin{cases} a_{n+1} & \text{EIS}(\ell - 1, \bar{a}) < a_{n+1} < \text{EIS}(\ell, \bar{a}) \\ \text{EIS}(\ell, \bar{a}) & \text{else} \end{cases}$$

Corollary

Longest Increasing Subsequence, Faster (V)

Lemma

For a fixed \bar{a} , $\text{EIS}(\ell, \bar{a})$ strictly increases with ℓ .

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) = \begin{cases} a_{n+1} & \text{EIS}(\ell - 1, \bar{a}) < a_{n+1} < \text{EIS}(\ell, \bar{a}) \\ \text{EIS}(\ell, \bar{a}) & \text{else} \end{cases}$$

Corollary

- $\text{EIS}(\ell, (\bar{a}, a_{n+1})) \neq \text{EIS}(\ell, \bar{a})$

Longest Increasing Subsequence, Faster (V)

Lemma

For a fixed \bar{a} , $\text{EIS}(\ell, \bar{a})$ strictly increases with ℓ .

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) = \begin{cases} a_{n+1} & \text{EIS}(\ell - 1, \bar{a}) < a_{n+1} < \text{EIS}(\ell, \bar{a}) \\ \text{EIS}(\ell, \bar{a}) & \text{else} \end{cases}$$

Corollary

- $\text{EIS}(\ell, (\bar{a}, a_{n+1})) \neq \text{EIS}(\ell, \bar{a})$ for **exactly one** value of ℓ

Longest Increasing Subsequence, Faster (V)

Lemma

For a fixed \bar{a} , $\text{EIS}(\ell, \bar{a})$ strictly increases with ℓ .

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) = \begin{cases} a_{n+1} & \text{EIS}(\ell - 1, \bar{a}) < a_{n+1} < \text{EIS}(\ell, \bar{a}) \\ \text{EIS}(\ell, \bar{a}) & \text{else} \end{cases}$$

Corollary

- $\text{EIS}(\ell, (\bar{a}, a_{n+1})) \neq \text{EIS}(\ell, \bar{a})$ for **exactly one** value of ℓ
- This value of ℓ can be found by binary search.

Longest Increasing Subsequence, Faster (V)

Lemma

For a fixed \bar{a} , $\text{EIS}(\ell, \bar{a})$ strictly increases with ℓ .

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) = \begin{cases} a_{n+1} & \text{EIS}(\ell - 1, \bar{a}) < a_{n+1} < \text{EIS}(\ell, \bar{a}) \\ \text{EIS}(\ell, \bar{a}) & \text{else} \end{cases}$$

Corollary

- $\text{EIS}(\ell, (\bar{a}, a_{n+1})) \neq \text{EIS}(\ell, \bar{a})$ for **exactly one** value of ℓ
- This value of ℓ can be found by binary search.

remarks:

Longest Increasing Subsequence, Faster (V)

Lemma

For a fixed \bar{a} , $\text{EIS}(\ell, \bar{a})$ strictly increases with ℓ .

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) = \begin{cases} a_{n+1} & \text{EIS}(\ell - 1, \bar{a}) < a_{n+1} < \text{EIS}(\ell, \bar{a}) \\ \text{EIS}(\ell, \bar{a}) & \text{else} \end{cases}$$

Corollary

- $\text{EIS}(\ell, (\bar{a}, a_{n+1})) \neq \text{EIS}(\ell, \bar{a})$ for **exactly one** value of ℓ
- This value of ℓ can be found by binary search.

remarks:

- uses *distinctness* of the a_i

Longest Increasing Subsequence, Faster (V)

Lemma

For a fixed \bar{a} , $\text{EIS}(\ell, \bar{a})$ strictly increases with ℓ .

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) = \begin{cases} a_{n+1} & \text{EIS}(\ell - 1, \bar{a}) < a_{n+1} < \text{EIS}(\ell, \bar{a}) \\ \text{EIS}(\ell, \bar{a}) & \text{else} \end{cases}$$

Corollary

- $\text{EIS}(\ell, (\bar{a}, a_{n+1})) \neq \text{EIS}(\ell, \bar{a})$ for **exactly one** value of ℓ
- This value of ℓ can be found by binary search.

remarks:

- uses *distinctness* of the a_i
- boundary cases need attention,

Longest Increasing Subsequence, Faster (V)

Lemma

For a fixed \bar{a} , $\text{EIS}(\ell, \bar{a})$ strictly increases with ℓ .

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) = \begin{cases} a_{n+1} & \text{EIS}(\ell - 1, \bar{a}) < a_{n+1} < \text{EIS}(\ell, \bar{a}) \\ \text{EIS}(\ell, \bar{a}) & \text{else} \end{cases}$$

Corollary

- $\text{EIS}(\ell, (\bar{a}, a_{n+1})) \neq \text{EIS}(\ell, \bar{a})$ for **exactly one** value of ℓ
- This value of ℓ can be found by binary search.

remarks:

- uses *distinctness* of the a_i
- boundary cases need attention, e.g., $\text{EIS}(\ell, \bar{a}) = \infty$,

Longest Increasing Subsequence, Faster (V)

Lemma

For a fixed \bar{a} , $\text{EIS}(\ell, \bar{a})$ strictly increases with ℓ .

Lemma

$$\text{EIS}(\ell, (a_1, \dots, a_n, a_{n+1})) = \begin{cases} a_{n+1} & \text{EIS}(\ell - 1, \bar{a}) < a_{n+1} < \text{EIS}(\ell, \bar{a}) \\ \text{EIS}(\ell, \bar{a}) & \text{else} \end{cases}$$

Corollary

- $\text{EIS}(\ell, (\bar{a}, a_{n+1})) \neq \text{EIS}(\ell, \bar{a})$ for **exactly one** value of ℓ
- This value of ℓ can be found by binary search.

remarks:

- uses *distinctness* of the a_i
- boundary cases need attention, e.g., $\text{EIS}(\ell, \bar{a}) = \infty$, or $\ell - 1 = 0$

Longest Increasing Subsequence, Faster (VI)

Longest Increasing Subsequence, Faster (VI)

LIS' (a_1, a_2, \dots, a_n):

Longest Increasing Subsequence, Faster (VI)

LIS' (a_1, a_2, \dots, a_n):
 for $1 \leq \ell \leq n$

Longest Increasing Subsequence, Faster (VI)

```
LIS'(a1, a2, ..., an):  
  for 1 ≤ ℓ ≤ n  
    E[ℓ] = ∞
```

Longest Increasing Subsequence, Faster (VI)

LIS' (a_1, a_2, \dots, a_n):

for $1 \leq \ell \leq n$

$E[\ell] = \infty$

for $1 \leq i \leq n$

Longest Increasing Subsequence, Faster (VI)

LIS' (a_1, a_2, \dots, a_n):

for $1 \leq \ell \leq n$

$E[\ell] = \infty$

for $1 \leq i \leq n$

$\ell = \min\{k : E[k] > a_i\}$

Longest Increasing Subsequence, Faster (VI)

LIS' (a_1, a_2, \dots, a_n):

for $1 \leq \ell \leq n$

$E[\ell] = \infty$

for $1 \leq i \leq n$

$\ell = \min\{k : E[k] > a_i\}$

$E[\ell] = a_i$

Longest Increasing Subsequence, Faster (VI)

LIS' (a_1, a_2, \dots, a_n):

for $1 \leq \ell \leq n$

$E[\ell] = \infty$

for $1 \leq i \leq n$

$\ell = \min\{k : E[k] > a_i\}$

$E[\ell] = a_i$

for $1 \leq i \leq n$

Longest Increasing Subsequence, Faster (VI)

LIS' (a_1, a_2, \dots, a_n):

for $1 \leq \ell \leq n$

$E[\ell] = \infty$

for $1 \leq i \leq n$

$\ell = \min\{k : E[k] > a_i\}$

$E[\ell] = a_i$

for $1 \leq i \leq n$

if $E[i] < \infty$

Longest Increasing Subsequence, Faster (VI)

LIS' (a_1, a_2, \dots, a_n):

for $1 \leq \ell \leq n$

$E[\ell] = \infty$

for $1 \leq i \leq n$

$\ell = \min\{k : E[k] > a_i\}$

$E[\ell] = a_i$

for $1 \leq i \leq n$

if $E[i] < \infty$

$L = i$

Longest Increasing Subsequence, Faster (VI)

```
LIS' ( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq \ell \leq n$   
     $E[\ell] = \infty$   
  for  $1 \leq i \leq n$   
     $\ell = \min\{k : E[k] > a_i\}$   
     $E[\ell] = a_i$   
  for  $1 \leq i \leq n$   
    if  $E[i] < \infty$   
       $L = i$   
  return  $L$ 
```

Longest Increasing Subsequence, Faster (VI)

```
LIS' ( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq \ell \leq n$   
     $E[\ell] = \infty$   
  for  $1 \leq i \leq n$   
     $\ell = \min\{k : E[k] > a_i\}$   
     $E[\ell] = a_i$   
  for  $1 \leq i \leq n$   
    if  $E[i] < \infty$   
       $L = i$   
  return  $L$ 
```

Longest Increasing Subsequence, Faster (VI)

```
LIS' ( $a_1, a_2, \dots, a_n$ ):
```

```
  for  $1 \leq \ell \leq n$ 
```

```
     $E[\ell] = \infty$ 
```

```
  for  $1 \leq i \leq n$ 
```

```
     $\ell = \min\{k : E[k] > a_i\}$ 
```

```
     $E[\ell] = a_i$ 
```

```
  for  $1 \leq i \leq n$ 
```

```
    if  $E[i] < \infty$ 
```

```
       $L = i$ 
```

```
  return  $L$ 
```

correctness:

Longest Increasing Subsequence, Faster (VI)

```
LIS' ( $a_1, a_2, \dots, a_n$ ):
```

```
  for  $1 \leq \ell \leq n$ 
```

```
     $E[\ell] = \infty$ 
```

```
  for  $1 \leq i \leq n$ 
```

```
     $\ell = \min\{k : E[k] > a_i\}$ 
```

```
     $E[\ell] = a_i$ 
```

```
  for  $1 \leq i \leq n$ 
```

```
    if  $E[i] < \infty$ 
```

```
       $L = i$ 
```

```
  return  $L$ 
```

correctness: clear

Longest Increasing Subsequence, Faster (VI)

```
LIS' ( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq \ell \leq n$   
     $E[\ell] = \infty$   
  for  $1 \leq i \leq n$   
     $\ell = \min\{k : E[k] > a_i\}$   
     $E[\ell] = a_i$   
  for  $1 \leq i \leq n$   
    if  $E[i] < \infty$   
       $L = i$   
  return  $L$ 
```

correctness: clear

complexity:

Longest Increasing Subsequence, Faster (VI)

```
LIS'(a1, a2, ..., an):  
  for 1 ≤ ℓ ≤ n  
    E[ℓ] = ∞  
  for 1 ≤ i ≤ n  
    ℓ = min{k : E[k] > ai}  
    E[ℓ] = ai  
  for 1 ≤ i ≤ n  
    if E[i] < ∞  
      L = i  
  return L
```

correctness: clear

complexity:

- $O(n)$ space

Longest Increasing Subsequence, Faster (VI)

```
LIS' ( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq \ell \leq n$   
     $E[\ell] = \infty$   
  for  $1 \leq i \leq n$   
     $\ell = \min\{k : E[k] > a_i\}$   
     $E[\ell] = a_i$   
  for  $1 \leq i \leq n$   
    if  $E[i] < \infty$   
       $L = i$   
  return  $L$ 
```

correctness: clear

complexity:

- $O(n)$ space
- time

Longest Increasing Subsequence, Faster (VI)

```
LIS' ( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq \ell \leq n$   
     $E[\ell] = \infty$   
  for  $1 \leq i \leq n$   
     $\ell = \min\{k : E[k] > a_i\}$   
     $E[\ell] = a_i$   
  for  $1 \leq i \leq n$   
    if  $E[i] < \infty$   
       $L = i$   
  return  $L$ 
```

correctness: clear

complexity:

- $O(n)$ space
- time
 - $E[\cdot]$ remains sorted throughout

Longest Increasing Subsequence, Faster (VI)

```
LIS' ( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq \ell \leq n$   
     $E[\ell] = \infty$   
  for  $1 \leq i \leq n$   
     $\ell = \min\{k : E[k] > a_i\}$   
     $E[\ell] = a_i$   
  for  $1 \leq i \leq n$   
    if  $E[i] < \infty$   
       $L = i$   
  return  $L$ 
```

correctness: clear

complexity:

- $O(n)$ space
 - time
 - $E[\cdot]$ remains sorted throughout
- $\implies O(\log n)$ time to compute $\min\{k : E[k] > a_i\}$

Longest Increasing Subsequence, Faster (VI)

```
LIS' ( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq \ell \leq n$   
     $E[\ell] = \infty$   
  for  $1 \leq i \leq n$   
     $\ell = \min\{k : E[k] > a_i\}$   
     $E[\ell] = a_i$   
  for  $1 \leq i \leq n$   
    if  $E[i] < \infty$   
       $L = i$   
  return  $L$ 
```

correctness: clear

complexity:

- $O(n)$ space
- time
 - $E[\cdot]$ remains sorted throughout
 - ⇒ $O(\log n)$ time to compute $\min\{k : E[k] > a_i\}$
 - ⇒ $O(n \log n)$ total runtime

Longest Increasing Subsequence, Faster (VI)

```
LIS' ( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq \ell \leq n$   
     $E[\ell] = \infty$   
  for  $1 \leq i \leq n$   
     $\ell = \min\{k : E[k] > a_i\}$   
     $E[\ell] = a_i$   
  for  $1 \leq i \leq n$   
    if  $E[i] < \infty$   
       $L = i$   
  return  $L$ 
```

correctness: clear

complexity:

- $O(n)$ space
- time
 - $E[\cdot]$ remains sorted throughout
 - $\implies O(\log n)$ time to compute $\min\{k : E[k] > a_i\}$
 - $\implies O(n \log n)$ total runtime

remarks:

Longest Increasing Subsequence, Faster (VI)

```
LIS' ( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq \ell \leq n$   
     $E[\ell] = \infty$   
  for  $1 \leq i \leq n$   
     $\ell = \min\{k : E[k] > a_i\}$   
     $E[\ell] = a_i$   
  for  $1 \leq i \leq n$   
    if  $E[i] < \infty$   
       $L = i$   
  return  $L$ 
```

correctness: clear

complexity:

- $O(n)$ space
- time
 - $E[\cdot]$ remains sorted throughout
 - $\implies O(\log n)$ time to compute $\min\{k : E[k] > a_i\}$
 - $\implies O(n \log n)$ total runtime

remarks:

- making a_i distinct costs $\Theta(n \log n)$ extra time

Longest Increasing Subsequence, Faster (VI)

```
LIS' ( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq \ell \leq n$   
     $E[\ell] = \infty$   
  for  $1 \leq i \leq n$   
     $\ell = \min\{k : E[k] > a_i\}$   
     $E[\ell] = a_i$   
  for  $1 \leq i \leq n$   
    if  $E[i] < \infty$   
       $L = i$   
  return  $L$ 
```

correctness: clear

complexity:

- $O(n)$ space
- time
 - $E[\cdot]$ remains sorted throughout
 - $\implies O(\log n)$ time to compute $\min\{k : E[k] > a_i\}$
 - $\implies O(n \log n)$ total runtime

remarks:

- making a_i distinct costs $\Theta(n \log n)$ extra time
- can compute actual subsequence in same time bound,

Longest Increasing Subsequence, Faster (VI)

```
LIS' ( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq \ell \leq n$   
     $E[\ell] = \infty$   
  for  $1 \leq i \leq n$   
     $\ell = \min\{k : E[k] > a_i\}$   
     $E[\ell] = a_i$   
  for  $1 \leq i \leq n$   
    if  $E[i] < \infty$   
       $L = i$   
  return  $L$ 
```

correctness: clear

complexity:

- $O(n)$ space
- time
 - $E[\cdot]$ remains sorted throughout
 - $\implies O(\log n)$ time to compute $\min\{k : E[k] > a_i\}$
 - $\implies O(n \log n)$ total runtime

remarks:

- making a_i distinct costs $\Theta(n \log n)$ extra time
- can compute actual subsequence in same time bound, using back pointers

Longest Increasing Subsequence, Faster (VI)

```
LIS' ( $a_1, a_2, \dots, a_n$ ):  
  for  $1 \leq \ell \leq n$   
     $E[\ell] = \infty$   
  for  $1 \leq i \leq n$   
     $\ell = \min\{k : E[k] > a_i\}$   
     $E[\ell] = a_i$   
  for  $1 \leq i \leq n$   
    if  $E[i] < \infty$   
       $L = i$   
  return  $L$ 
```

correctness: clear

complexity:

- $O(n)$ space
- time
 - $E[\cdot]$ remains sorted throughout
 - $\implies O(\log n)$ time to compute $\min\{k : E[k] > a_i\}$
 - $\implies O(n \log n)$ total runtime

remarks:

- making a_i distinct costs $\Theta(n \log n)$ extra time
- can compute actual subsequence in same time bound, using back pointers (**exercise**)

today:

- dynamic programming *optimized*
 - edit distance
 - longest increasing subsequence

next lecture:

- randomized algorithms

logistics:

- pset2 due R5 — can submit in *groups* of ≤ 3

- 1 Title
- 2 Overview
- 3 Dynamic Programming
- 4 Edit Distance
- 5 Edit Distance (II)
- 6 Edit Distance (III)
- 7 Edit Distance (IV)
- 8 Edit Distance (V)
- 9 Edit Distance (VI)
- 10 Edit Distance (VII)
- 11 Edit Distance, Better
- 12 Edit Distance, Better (II)
- 13 Edit Distance, Better (III)
- 14 Edit Distance, Better (IV)
- 15 Longest Increasing Subsequence
- 16 Longest Increasing Subsequence (II)
- 17 Longest Increasing Subsequence (III)
- 18 Longest Increasing Subsequence (IV)
- 19 Longest Increasing Subsequence (V)
- 20 Longest Increasing Subsequence, Faster
- 21 Longest Increasing Subsequence, Faster (II)
- 22 Longest Increasing Subsequence, Faster (III)
- 23 Longest Increasing Subsequence, Faster (IV)
- 24 Longest Increasing Subsequence, Faster (V)
- 25 Longest Increasing Subsequence, Faster (VI)
- 26 Overview (II)