Karatsuba's Algorithm and Linear Time Selection

Lecture 11

We will learn

1. Last lecture
   1. How to think about recursion as a design paradigm
   2. How to analyze running time recurrences
2. More complicated recursion in action
   1. Fast multiplication (Karatsuba's Algorithm)
   2. Linear Time Selection

Recursion

Another way to think about it

Reduce a problem to *smaller* instances of the *same* problem.

Reduction

Reduction = Delegation

- Solve a problem using elementary operations + call a bunch of subroutines
- Subroutines = Black boxes

Example of Reduction: Distinct Elements Problem

Problem Given an array A of n integers, are there any *duplicates* in A?

Example of Reduction: Distinct Elements Problem

Problem Given an array A of n integers, are there any *duplicates* in A?

Naive algorithm:

```
DistinctElements(A[1..n])
    for i = 1 to n − 1 do
        for j = i + 1 to n do
            if (A[i] = A[j])
                return YES
    return NO
```

Example of Reduction: Distinct Elements Problem

Problem: Given an array A of n integers, are there any *duplicates* in A?

Naive algorithm:

```
DistinctElements(A[1..n])
    for i = 1 to n − 1 do
        for j = i + 1 to n do
            if (A[i] = A[j])
                return YES
    return NO
```

Running time:

Example of Reduction: Distinct Elements Problem

Problem Given an array A of n integers, are there any *duplicates* in A?

Naive algorithm:

```
DistinctElements(A[1..n])
    for i = 1 to n − 1 do
        for j = i + 1 to n do
            if (A[i] = A[j])
                return YES
    return NO
```

Running time: $O(n^2)$

Reduction to Sorting

```
DistinctElements(A[1..n])
    Sort A
    for i = 1 to n − 1 do
        if (A[i] = A[i + 1]) then
            return YES
    return NO
```

Reduction to Sorting

```
DistinctElements(A[1..n])
    Sort A
    for i = 1 to n − 1 do
        if (A[i] = A[i + 1]) then
            return YES
    return NO
```

Running time: $O(n)$ plus time to sort an array of n numbers

Important point: algorithm uses sorting as a *black box*

Recursion

It requires discipline to delegate

It is important to think of the recursive calls as black boxes, that is, subroutines taken care of by the recursion fairy.

Recursion

It requires discipline to delegate

It is important to think of the recursive calls as black boxes, that is, subroutines taken care of by the recursion fairy.

```
MergeSort(A[1..n]):
  if n > 1
      m ← ⌊n/2⌋
      MergeSort(A[1..m])
      MergeSort(A[m+1..n])
      Merge(A[1..n], m)
```

Solving Recurrences

Two general methods:

1. Guess and Verify
2. Recursion tree method: At every level of recursion, how much *non-recursive* work you are doing.

Solving Recurrences

Two general methods:

1. Guess and Verify
2. Recursion tree method: At every level of recursion, how much *non-recursive* work you are doing.
    1. Merge Sort: same amount of work at every level

Solving Recurrences

Two general methods:

1. Guess and Verify
2. Recursion tree method: At every level of recursion, how much *non-recursive* work you are doing.
    1. Merge Sort: same amount of work at every level
    2. Increasing geometric series: count number of leaves. (Fast multiplication)
    3. Decreasing geometric series: summable, first level dominates. (Selection)

Part I

Fast Multiplication

Multiplying Numbers

Problem Given two n-digit numbers x and y, compute their product.

Grade School Multiplication

Compute "partial product" by multiplying each digit of y with x and adding the partial products.

$$
\begin{array}{r}
3141 \\
\times 2718 \\
\hline
25128 \\
3141 \\
21987 \\
6282 \\
\hline
8537238
\end{array}
$$

Time Analysis of Grade School Multiplication

1. Each partial product: $\Theta(n)$
2. Number of partial products: $\Theta(n)$
3. Addition of partial products: $\Theta(n^2)$
4. Total time: $\Theta(n^2)$

Divide and Conquer

Assume n is a power of 2 for simplicity and numbers are in decimal.

Split each number into two numbers with equal number of digits

1. $x = x_{n-1}x_{n-2}\ldots x_0$ and $y = y_{n-1}y_{n-2}\ldots y_0$
2. $x = x_{n-1}\ldots x_{n/2}0\ldots 0 + x_{n/2-1}\ldots x_0$
3. $x = 10^{n/2}x_L + x_R$ where $x_L = x_{n-1}\ldots x_{n/2}$ and $x_R = x_{n/2-1}\ldots x_0$
4. Similarly $y = 10^{n/2}y_L + y_R$ where $y_L = y_{n-1}\ldots y_{n/2}$ and $y_R = y_{n/2-1}\ldots y_0$

Example

$$
\begin{aligned}
1234 \times 5678 &= (100 \times 12 + 34) \times (100 \times 56 + 78) \\
&= 10000 \times 12 \times 56 \\
&\quad +100 \times (12 \times 78 + 34 \times 56) \\
&\quad +34 \times 78
\end{aligned}
$$

Divide and Conquer

Assume n is a power of 2 for simplicity and numbers are in decimal.

1. $x = x_{n-1}x_{n-2}\ldots x_0$ and $y = y_{n-1}y_{n-2}\ldots y_0$
2. $x = 10^{n/2}x_L + x_R$ where $x_L = x_{n-1}\ldots x_{n/2}$ and $x_R = x_{n/2-1}\ldots x_0$
3. $y = 10^{n/2}y_L + y_R$ where $y_L = y_{n-1}\ldots y_{n/2}$ and $y_R = y_{n/2-1}\ldots y_0$

Therefore

$$\begin{aligned} xy &= (10^{n/2}x_L + x_R)(10^{n/2}y_L + y_R) \\ &= 10^n x_L y_L + 10^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

Time Analysis

$$
\begin{aligned}
xy &= (10^{n/2}x_L + x_R)(10^{n/2}y_L + y_R) \\
&= 10^n x_L y_L + 10^{n/2}(x_L y_R + x_R y_L) + x_R y_R
\end{aligned}
$$

4 recursive multiplications of size $n/2$ plus 3 additions and left shifts (adding enough 0's to the right)

Time Analysis

$$\begin{aligned} xy &= (10^{n/2} x_L + x_R)(10^{n/2} y_L + y_R) \\ &= 10^n x_L y_L + 10^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

4 recursive multiplications of size $n/2$ plus 3 additions and left shifts (adding enough 0's to the right)

$$T(n) = 4T(n/2) + O(n) \qquad T(1) = O(1)$$

Time Analysis

$$\begin{aligned} xy &= (10^{n/2}x_L + x_R)(10^{n/2}y_L + y_R) \\ &= 10^n x_L y_L + 10^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

4 recursive multiplications of size $n/2$ plus 3 additions and left shifts (adding enough 0's to the right)

$$T(n) = 4T(n/2) + O(n) \qquad T(1) = O(1)$$

$T(n) = \Theta(n^2)$. No better than grade school multiplication!

Recursion Tree

A Trick of Gauss

Carl Friedrich Gauss: 1777–1855 "Prince of Mathematicians"

Observation: Multiply two complex numbers: $(a + bi)$ and $(c + di)$

$$(a + bi)(c + di) = ac - bd + (ad + bc)i$$

A Trick of Gauss

Carl Friedrich Gauss: 1777–1855 "Prince of Mathematicians"

Observation: Multiply two complex numbers: $(a + bi)$ and $(c + di)$

$$(a + bi)(c + di) = ac - bd + (ad + bc)i$$

How many multiplications do we need?

A Trick of Gauss

Carl Friedrich Gauss: 1777–1855 "Prince of Mathematicians"

Observation: Multiply two complex numbers: $(a + bi)$ and $(c + di)$

$$(a + bi)(c + di) = ac - bd + (ad + bc)i$$

How many multiplications do we need?

Only 3! If we do extra additions and subtractions.
Compute $ac, bd, (a + b)(c + d)$. Then
$(ad + bc) = (a + b)(c + d) - ac - bd$

Improving the Running Time

$$
\begin{aligned}
xy &= (10^{n/2}x_L + x_R)(10^{n/2}y_L + y_R) \\
&= 10^n x_L y_L + 10^{n/2}(x_L y_R + x_R y_L) + x_R y_R
\end{aligned}
$$

Gauss trick: $x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$

Improving the Running Time

$$\begin{aligned} xy &= (10^{n/2}x_L + x_R)(10^{n/2}y_L + y_R) \\ &= 10^n x_L y_L + 10^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

Gauss trick: $x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$

Recursively compute only $x_L y_L, x_R y_R, (x_L + x_R)(y_L + y_R)$.

Improving the Running Time

$$\begin{aligned} xy &= (10^{n/2}x_L + x_R)(10^{n/2}y_L + y_R) \\ &= 10^n x_L y_L + 10^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

Gauss trick: $x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$

Recursively compute only $x_L y_L, x_R y_R, (x_L + x_R)(y_L + y_R)$.

Time Analysis

Running time is given by

$$T(n) = 3T(n/2) + O(n) \qquad T(1) = O(1)$$

which means

Improving the Running Time

$$
\begin{aligned}
xy &= (10^{n/2}x_L + x_R)(10^{n/2}y_L + y_R) \\
&= 10^n x_L y_L + 10^{n/2}(x_L y_R + x_R y_L) + x_R y_R
\end{aligned}
$$

Gauss trick: $x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$

Recursively compute only $x_L y_L, x_R y_R, (x_L + x_R)(y_L + y_R)$.

Time Analysis

Running time is given by

$$T(n) = 3T(n/2) + O(n) \qquad\qquad T(1) = O(1)$$

which means $T(n) = O(n^{\log_2 3}) = O(n^{1.585})$

Analyzing the Recurrences

1. Basic divide and conquer: $T(n) = 4T(n/2) + O(n)$, $T(1) = 1$. **Claim:** $T(n) = \Theta(n^2)$.
2. Saving a multiplication: $T(n) = 3T(n/2) + O(n)$, $T(1) = 1$. **Claim:** $T(n) = \Theta(n^{\log_2 3})$

Analyzing the Recurrences

1. Basic divide and conquer: $T(n) = 4T(n/2) + O(n)$, $T(1) = 1$. **Claim:** $T(n) = \Theta(n^2)$.
2. Saving a multiplication: $T(n) = 3T(n/2) + O(n)$, $T(1) = 1$. **Claim:** $T(n) = \Theta(n^{\log_2 3})$

Use recursion tree method:

1. In both cases, depth of recursion $L = \log n$.
2. Work at depth i is $4^i n/2^i$ and $3^i n/2^i$ respectively: number of children at depth i times the work at each child
3. Total work is therefore $n \sum_{i=0}^{L} 2^i$ and $n \sum_{i=0}^{L} (3/2)^i$ respectively.

Recursion tree analysis

Part II

Selecting in Unsorted Lists

Rank of element in an array

A: an unsorted array of n integers

Definition

For $1 \le j \le n$, element of rank j is the j'th smallest element in A.

Unsorted array	16	14	34	20	12	5	3	19	11
Ranks	6	5	9	8	4	2	1	7	3

Sort of array	3	5	11	12	14	16	19	20	34

Problem - Selection

Input Unsorted array A of n integers **and** integer j

Goal Find the jth smallest number in A (*rank* j number)

Median: $j = \lfloor (n+1)/2 \rfloor$

Problem - Selection

Input Unsorted array A of n integers **and** integer j

Goal Find the jth smallest number in A (*rank* j number)

Median: $j = \lfloor (n+1)/2 \rfloor$

Simplifying assumption for sake of notation: elements of A are distinct

Algorithm I

1. Sort the elements in A
2. Pick jth element in sorted order

Time taken $= O(n \log n)$

Algorithm I

1. Sort the elements in A
2. Pick jth element in sorted order

Time taken $= O(n \log n)$

Do we need to sort? Is there an $O(n)$ time algorithm?

Algorithm II. One-armed Quick Sort

$\underline{\text{QuickSort}(A[1..n]):}$
if $(n > 1)$
    Choose a pivot element $A[p]$
    $r \leftarrow \text{Partition}(A, p)$
    $\text{QuickSort}(A[1..r-1])$ ⟨⟨*Recurse!*⟩⟩
    $\text{QuickSort}(A[r+1..n])$ ⟨⟨*Recurse!*⟩⟩

Algorithm II. One-armed Quick Sort

```
QuickSelect(A[1 .. n], k):
  if n = 1
      return A[1]
  else
      Choose a pivot element A[p]
      r ← Partition(A[1 .. n], p)

      if k < r
          return QuickSelect(A[1 .. r − 1], k)
      else if k > r
          return QuickSelect(A[r + 1 .. n], k − r)
      else
          return A[r]
```

Running Time Analysis

1. Partitioning step: $O(n)$ time to scan A
2. $$T(n) = \max_{1 \leq k \leq n} max(T(k-1), T(n-k)) + O(n)$$

   In the worst case $T(n) = T(n-1) + O(n)$, which means $T(n) = O(n^2)$. Happens if array is already sorted and pivot is always first element.

A Better Pivot

Suppose pivot is the ℓth smallest element where $n/4 \leq \ell \leq 3n/4$.
That is pivot is *approximately* in the middle of A
Then $n/4 \leq |A_{\text{less}}| \leq n/2$ *and* $n/2 \leq |A_{\text{greater}}| \leq 3n/4$. If we apply recursion,

A Better Pivot

Suppose pivot is the ℓth smallest element where $n/4 \leq \ell \leq 3n/4$.
That is pivot is *approximately* in the middle of A
Then $n/4 \leq |A_{\text{less}}| \leq n/2$ *and* $n/2 \leq |A_{\text{greater}}| \leq 3n/4$. If we apply recursion,

$$T(n) \leq T(3n/4) + O(n)$$

Implies $T(n) = O(n)$!

A Better Pivot

Suppose pivot is the ℓth smallest element where $n/4 \leq \ell \leq 3n/4$.
That is pivot is *approximately* in the middle of A
Then $n/4 \leq |A_{\text{less}}| \leq n/2$ *and* $n/2 \leq |A_{\text{greater}}| \leq 3n/4$. If we apply recursion,

$$T(n) \leq T(3n/4) + O(n)$$

Implies $T(n) = O(n)$!

How do we find such a pivot?

A Better Pivot

Suppose pivot is the ℓth smallest element where $n/4 \leq \ell \leq 3n/4$.
That is pivot is *approximately* in the middle of A
Then $n/4 \leq |A_{\text{less}}| \leq n/2$ *and* $n/2 \leq |A_{\text{greater}}| \leq 3n/4$. If we apply recursion,

$$T(n) \leq T(3n/4) + O(n)$$

Implies $T(n) = O(n)$!

How do we find such a pivot? Randomly?

A Better Pivot

Suppose pivot is the ℓth smallest element where $n/4 \leq \ell \leq 3n/4$.
That is pivot is *approximately* in the middle of A
Then $n/4 \leq |A_{\text{less}}| \leq n/2$ *and* $n/2 \leq |A_{\text{greater}}| \leq 3n/4$. If we apply recursion,

$$T(n) \leq T(3n/4) + O(n)$$

Implies $T(n) = O(n)$!

How do we find such a pivot? Randomly? In fact works!
Analysis a little bit later.

A Better Pivot

Suppose pivot is the ℓth smallest element where $n/4 \leq \ell \leq 3n/4$.
That is pivot is *approximately* in the middle of A
Then $n/4 \leq |A_{\text{less}}| \leq n/2$ *and* $n/2 \leq |A_{\text{greater}}| \leq 3n/4$. If we apply recursion,

$$T(n) \leq T(3n/4) + O(n)$$

Implies $T(n) = O(n)$!

How do we find such a pivot? Randomly? In fact works!
Analysis a little bit later.

Can we choose pivot deterministically?

Divide and Conquer Approach

A game of medians

Idea

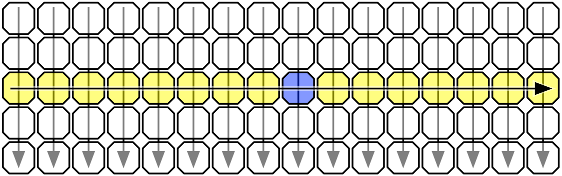
1. Break input A into many subarrays: $L_1, \ldots L_k$.
2. Find median m_i in each subarray L_i.
3. Find the median x of the medians $m_1, \ldots, m_k$.
4. Intuition: The median x should be close to being a good median of all the numbers in A.
5. Use x as pivot in previous algorithm.

Example

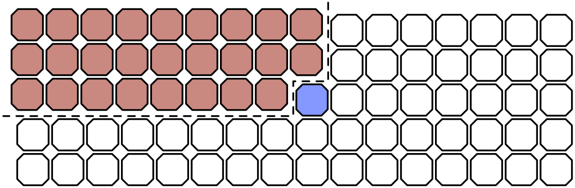
11	7	3	42	174	310	1	92	87	12	19	15

Example

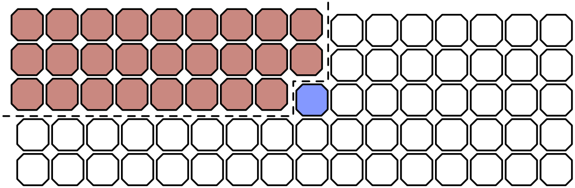
11	7	3	42	174	310	1	92	87	12	19	15



Median of median



Median of median



Lemma

Median of $\boldsymbol{B}$ *is an* approximate *median of* $\boldsymbol{A}$. *That is, if* $\boldsymbol{b}$ *is used as a pivot to partition* $\boldsymbol{A}$, *then* $|\boldsymbol{A}_{greater}| \leq \mathbf{7}\boldsymbol{n}/\mathbf{10}$.

Algorithm for Selection

A storm of medians

```
select(A, j):
    Form lists L_1, L_2, ..., L_⌈n/5⌉ where L_i = {A[5i − 4], ..., A[5i]}
    Find median b_i of each L_i using brute-force
    Find median b of B = {b_1, b_2, ..., b_⌈n/5⌉}
    Partition A into A_less and A_greater using b as pivot
    if (|A_less|) = j return b
    else if (|A_less|) > j)
        return select(A_less, j)
    else
        return select(A_greater, j − |A_less|)
```

Algorithm for Selection

A storm of medians

```
select(A, j):
    Form lists L_1, L_2, ..., L_⌈n/5⌉ where L_i = {A[5i − 4], ..., A[5i]}
    Find median b_i of each L_i using brute-force
    Find median b of B = {b_1, b_2, ..., b_⌈n/5⌉}
    Partition A into A_less and A_greater using b as pivot
    if (|A_less|) = j return b
    else if (|A_less|) > j)
        return select(A_less, j)
    else
        return select(A_greater, j − |A_less|)
```

How do we find median of B?

Algorithm for Selection

A storm of medians

```
select(A, j):
    Form lists L_1, L_2, ..., L_⌈n/5⌉ where L_i = {A[5i − 4], ..., A[5i]}
    Find median b_i of each L_i using brute-force
    Find median b of B = {b_1, b_2, ..., b_⌈n/5⌉}
    Partition A into A_less and A_greater using b as pivot
    if (|A_less|) = j return b
    else if (|A_less|) > j)
        return select(A_less, j)
    else
        return select(A_greater, j − |A_less|)
```

How do we find median of B? Recursively!

Algorithm for Selection

A storm of medians

select(A, j):
  Form lists $L_1, L_2, \ldots, L_{\lceil n/5 \rceil}$ where $L_i = \{A[5i-4], \ldots, A[5i]\}$
  Find median b_i of each L_i using brute-force
  $B = [b_1, b_2, \ldots, b_{\lceil n/5 \rceil}]$
  $b =$ select(B, $\lceil n/10 \rceil$)
  Partition A into $A_{\texttt{less}}$ and $A_{\texttt{greater}}$ using b as pivot
  if ($|A_{\texttt{less}}|) = j$ return b
  else if ($|A_{\texttt{less}}|) > j$)
    return select($A_{\texttt{less}}$, j)
  else
    return select($A_{\texttt{greater}}$, $j - |A_{\texttt{less}}|$)

Running time of deterministic median selection

A dance with recurrences

$$T(n) \le T(\lceil n/5 \rceil) + \max\{T(|A_{\text{less}}|), T(|A_{\text{greater}}|)\} + O(n)$$

Running time of deterministic median selection

A dance with recurrences

$$T(n) \leq T(\lceil n/5 \rceil) + \max\{T(|A_{\text{less}}|), T(|A_{\text{greater}}|)\} + O(n)$$

From Lemma,

$$T(n) \leq T(\lceil n/5 \rceil) + T(\lceil 7n/10 \rceil) + O(n)$$

and

$$T(n) = O(1) \qquad n < 10$$

Recursion Tree

Why 5? How about 3?