Lower bounds

Lecture 22

November 12, 2015

22.1: Sorting

Sorting...

1. n items: $x_1, \ldots, x_n$.
2. Can be sorted in $O(n \log n)$ time.
3. Claim: $\Omega(n \log n)$ time to solve this.
4. Rules of engagement: What can an algorithm do???

Sorting...

1. n items: $x_1, \ldots, x_n$.
2. Can be sorted in $O(n \log n)$ time.
3. Claim: $\Omega(n \log n)$ time to solve this.
4. Rules of engagement: What can an algorithm do???

Sorting...

1. n items: $x_1, \ldots, x_n$.
2. Can be sorted in $O(n \log n)$ time.
3. Claim: $\Omega(n \log n)$ time to solve this.
4. Rules of engagement: What can an algorithm do???

Sorting...

1. n items: $x_1, \ldots, x_n$.
2. Can be sorted in $O(n \log n)$ time.
3. Claim: $\Omega(n \log n)$ time to solve this.
4. Rules of engagement: What can an algorithm do???

Comparison model

1. In the **comparison model**:
    1. Algorithm only allowed to compare two elements.
    2. **compare**(i, j): Compare ith item in input to jth item in input.
2. **Q:** # calls to **compare** a deterministic sorting algorithm has to perform?

Comparison model

1. In the **comparison model**:
    1. Algorithm only allowed to compare two elements.
    2. **compare**(i, j): Compare ith item in input to jth item in input.
2. **Q:** # calls to **compare** a deterministic sorting algorithm has to perform?

Comparison model

1. In the **comparison model**:
   1. Algorithm only allowed to compare two elements.
   2. **compare**(i, j): Compare ith item in input to jth item in input.
2. **Q:** # calls to **compare** a deterministic sorting algorithm has to perform?

Comparison model

1. In the **comparison model**:
   1. Algorithm only allowed to compare two elements.
   2. **compare**(i, j): Compare ith item in input to jth item in input.
2. **Q:** # calls to **compare** a deterministic sorting algorithm has to perform?

Decision tree for sorting

1. sorting algorithm: a decision procedure.
2. Each stage: has current collection of comparisons done.
3. ... need to decide which comparison to perform next.

Decision tree for sorting

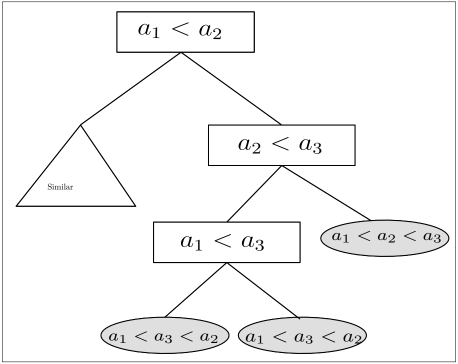
1. sorting algorithm: a decision procedure.
2. Each stage: has current collection of comparisons done.
3. ... need to decide which comparison to perform next.

Decision tree for sorting

1. sorting algorithm: a decision procedure.
2. Each stage: has current collection of comparisons done.
3. ... need to decide which comparison to perform next.

Decision tree for sorting

1. sorting algorithm: a decision procedure.
2. Each stage: has current collection of comparisons done.
3. ... need to decide which comparison to perform next.



Sorting algorithm...

1. sorting algorithm outputs a permutation.
2. ... order of the input elements so sorted.
3. Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
    1. Output: $1, 2, 3, 7, 19$.
    2. Output: x_3, x_5, x_2, x_1, x_4.
    3. Output: $\pi = (3, 5, 2, 1, 4)$
    4. Output as permutation: $\pi(1) = 3, \pi(2) = 5, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4$.
4. **Interpretation**: $x_{\pi(i)}$ is the ith smallest number in $x_1, \ldots, x_n$.
5. v: Node of decision tree.
   $P(v)$: A set of all permutations compatible with the set of comparisons from root to v.

Sorting algorithm...

1. sorting algorithm outputs a permutation.
2. ... order of the input elements so sorted.
3. Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
   1. Output: $1, 2, 3, 7, 19$.
   2. Output: x_3, x_5, x_2, x_1, x_4.
   3. Output: $\pi = (3, 5, 2, 1, 4)$
   4. Output as permutation:
      $\pi(1) = 3, \pi(2) = 5, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4$.
4. **Interpretation**: $x_{\pi(i)}$ is the ith smallest number in $x_1, \ldots, x_n$.
5. v: Node of decision tree.
   $P(v)$: A set of all permutations compatible with the set of comparisons from root to v.

Sorting algorithm...

1. sorting algorithm outputs a permutation.
2. ... order of the input elements so sorted.
3. Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
   1. Output: $1, 2, 3, 7, 19$.
   2. Output: x_3, x_5, x_2, x_1, x_4.
   3. Output: $\pi = (3, 5, 2, 1, 4)$
   4. Output as permutation: $\pi(1) = 3, \pi(2) = 5, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4$.
4. **Interpretation**: $x_{\pi(i)}$ is the ith smallest number in $x_1, \ldots, x_n$.
5. v: Node of decision tree.
   $P(v)$: A set of all permutations compatible with the set of comparisons from root to v.

Sorting algorithm...

1. sorting algorithm outputs a permutation.
2. ... order of the input elements so sorted.
3. Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
   1. Output: $1, 2, 3, 7, 19$.
   2. Output: x_3, x_5, x_2, x_1, x_4.
   3. Output: $\pi = (3, 5, 2, 1, 4)$
   4. Output as permutation:
      $\pi(1) = 3, \pi(2) = 5, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4$.
4. **Interpretation**: $x_{\pi(i)}$ is the ith smallest number in $x_1, \ldots, x_n$.
5. v: Node of decision tree.
   $P(v)$: A set of all permutations compatible with the set of comparisons from root to v.

Sorting algorithm...

1. sorting algorithm outputs a permutation.
2. ... order of the input elements so sorted.
3. Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
    1. Output: $1, 2, 3, 7, 19$.
    2. Output: x_3, x_5, x_2, x_1, x_4.
    3. Output: $\pi = (3, 5, 2, 1, 4)$
    4. Output as permutation: $\pi(1) = 3, \pi(2) = 5, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4$.
4. **Interpretation**: $x_{\pi(i)}$ is the ith smallest number in $x_1, \ldots, x_n$.
5. v: Node of decision tree.
   $P(v)$: A set of all permutations compatible with the set of comparisons from root to v.

Sorting algorithm...

1. sorting algorithm outputs a permutation.
2. ... order of the input elements so sorted.
3. Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
    1. Output: $1, 2, 3, 7, 19$.
    2. Output: x_3, x_5, x_2, x_1, x_4.
    3. Output: $\pi = (3, 5, 2, 1, 4)$
    4. Output as permutation: $\pi(1) = 3, \pi(2) = 5, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4$.
4. **Interpretation**: $x_{\pi(i)}$ is the ith smallest number in $x_1, \ldots, x_n$.
5. v: Node of decision tree. $P(v)$: A set of all permutations compatible with the set of comparisons from root to v.

Sorting algorithm...

1. sorting algorithm outputs a permutation.
2. ... order of the input elements so sorted.
3. Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
    1. Output: $1, 2, 3, 7, 19$.
    2. Output: x_3, x_5, x_2, x_1, x_4.
    3. Output: $\pi = (3, 5, 2, 1, 4)$
    4. Output as permutation: $\pi(1) = 3, \pi(2) = 5, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4$.
4. **Interpretation**: $x_{\pi(i)}$ is the ith smallest number in $x_1, \ldots, x_n$.
5. v: Node of decision tree. $P(v)$: A set of all permutations compatible with the set of comparisons from root to v.

Sorting algorithm...

1. sorting algorithm outputs a permutation.
2. ... order of the input elements so sorted.
3. Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
   1. Output: $1, 2, 3, 7, 19$.
   2. Output: x_3, x_5, x_2, x_1, x_4.
   3. Output: $\pi = (3, 5, 2, 1, 4)$
   4. Output as permutation: $\pi(1) = 3, \pi(2) = 5, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4$.
4. **Interpretation**: $x_{\pi(i)}$ is the ith smallest number in $x_1, \ldots, x_n$.
5. v: Node of decision tree. $P(v)$: A set of all permutations compatible with the set of comparisons from root to v.

1. sorting algorithm outputs a permutation.
2. ... order of the input elements so sorted.
3. Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
    1. Output: $1, 2, 3, 7, 19$.
    2. Output: x_3, x_5, x_2, x_1, x_4.
    3. Output: $\pi = (3, 5, 2, 1, 4)$
    4. Output as permutation: $\pi(1) = 3, \pi(2) = 5, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4$.
4. **Interpretation**: $x_{\pi(i)}$ is the ith smallest number in $x_1, \ldots, x_n$.
5. v: Node of decision tree.
   $P(v)$: A set of all permutations compatible with the set of comparisons from root to v.

What are permutations?

1. $\pi = (3, 4, 1, 2)$ is permutation in $P(v)$.
2. Formally $\pi : [\![n]\!] \to [\![n]\!]$ is a one-to-one function. $[\![n]\!] = \{1, \ldots, n\}$ can be written as: $\pi = (3, 4, 1, 2) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix}$
3. Input is: x_1, x_2, x_3, x_4
4. If arrived to v and $\pi \in P(v)$ then
$$x_3 < x_4 < x_1 < x_2.$$
a possible ordering (as far as what seen so far).
5.

What are permutations?

1. $\pi = (3, 4, 1, 2)$ is permutation in $P(v)$.
2. Formally $\pi : [\![n]\!] \to [\![n]\!]$ is a one-to-one function.
   $[\![n]\!] = \{1, \ldots, n\}$
   can be written as:
   $\pi = (3, 4, 1, 2) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix}$
3. Input is: x_1, x_2, x_3, x_4
4. If arrived to v and $\pi \in P(v)$ then
   $$x_3 < x_4 < x_1 < x_2.$$
   a possible ordering (as far as what seen so far).
5.

What are permutations?

1. $\pi = (3, 4, 1, 2)$ is permutation in $P(v)$.
2. Formally $\pi : [\![n]\!] \rightarrow [\![n]\!]$ is a one-to-one function.
   $[\![n]\!] = \{1, \ldots, n\}$
   can be written as:
   $\pi = (3, 4, 1, 2) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix}$
3. Input is: x_1, x_2, x_3, x_4
4. If arrived to v and $\pi \in P(v)$ then
   $$x_3 < x_4 < x_1 < x_2.$$
   a possible ordering (as far as what seen so far).
5.

What are permutations?

1. $\pi = (3,4,1,2)$ is permutation in $P(v)$.
2. Formally $\pi : [\![n]\!] \rightarrow [\![n]\!]$ is a one-to-one function.
   $[\![n]\!] = \{1,\ldots,n\}$
   can be written as:
   $\pi = (3,4,1,2) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix}$
3. Input is: x_1, x_2, x_3, x_4
4. If arrived to v and $\pi \in P(v)$ then
   $$x_3 < x_4 < x_1 < x_2.$$
   a possible ordering (as far as what seen so far).
5.

What are permutations?

1. $\pi = (3, 4, 1, 2)$ is permutation in $P(v)$.
2. Formally $\pi : [\![n]\!] \to [\![n]\!]$ is a one-to-one function.
   $[\![n]\!] = \{1, \ldots, n\}$
   can be written as:
   $\pi = (3, 4, 1, 2) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix}$
3. Input is: x_1, x_2, x_3, x_4
4. If arrived to v and $\pi \in P(v)$ then
   $$x_3 < x_4 < x_1 < x_2.$$
   a possible ordering (as far as what seen so far).
5.

Input realizing a permutation, by example

1. Let $\pi = (3, 4, 2, 1) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix}$
2. Then the input $\pi^{-1} = (3, 4, 1, 2) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{pmatrix}$
3. ... would generate this permutation.
4. Formally
   $x_1 = \pi^{-1}(1) = 4 \ldots x_i = \pi^{-1}(i) \ldots$

Input realizing a permutation, by example

1. Let $\pi = (3,4,2,1) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix}$
2. Then the input $\pi^{-1} = (3,4,1,2) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{pmatrix}$
3. ... would generate this permutation.
4. Formally
   $x_1 = \pi^{-1}(1) = 4 \ldots x_i = \pi^{-1}(i) \ldots$

Input realizing a permutation, by example

1. Let $\pi = (3, 4, 2, 1) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix}$
2. Then the input $\pi^{-1} = (3, 4, 1, 2) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{pmatrix}$
3. ... would generate this permutation.
4. Formally
   $x_1 = \pi^{-1}(1) = 4 \ldots x_i = \pi^{-1}(i) \ldots$

Input realizing a permutation, by example

1. Let $\pi = (3,4,2,1) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix}$
2. Then the input $\pi^{-1} = (3,4,1,2) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{pmatrix}$
3. ... would generate this permutation.
4. Formally
   $x_1 = \pi^{-1}(1) = 4 \ldots x_i = \pi^{-1}(i) \ldots$

Back to sorting...

1. v: a node in decision tree.
2. If $|P(v)| > 1$: more than one permutation associated with it...
3. algorithm must continue performing comparisons
4. ...otherwise, not know what to output...
5. **Q:** What is the worst running time of algorithm?
6. Answer: Longest path from root in the decision tree.
   ...because we count only comparisons!

Back to sorting...

1. v: a node in decision tree.
2. If $|P(v)| > 1$: more than one permutation associated with it...
3. algorithm must continue performing comparisons
4. ...otherwise, not know what to output...
5. **Q:** What is the worst running time of algorithm?
6. Answer: Longest path from root in the decision tree.
   ...because we count only comparisons!

Back to sorting...

1. v: a node in decision tree.
2. If $|P(v)| > 1$: more than one permutation associated with it...
3. algorithm must continue performing comparisons
4. ...otherwise, not know what to output...
5. **Q:** What is the worst running time of algorithm?
6. Answer: Longest path from root in the decision tree.
   ...because we count only comparisons!

Back to sorting...

1. v: a node in decision tree.
2. If $|P(v)| > 1$: more than one permutation associated with it...
3. algorithm must continue performing comparisons
4. ...otherwise, not know what to output...
5. **Q:** What is the worst running time of algorithm?
6. Answer: Longest path from root in the decision tree.
   ...because we count only comparisons!

Back to sorting...

1. v: a node in decision tree.
2. If $|P(v)| > 1$: more than one permutation associated with it...
3. algorithm must continue performing comparisons
4. ...otherwise, not know what to output...
5. **Q:** What is the worst running time of algorithm?
6. Answer: Longest path from root in the decision tree.
   ...because we count only comparisons!

Back to sorting...

1. v: a node in decision tree.
2. If $|P(v)| > 1$: more than one permutation associated with it...
3. algorithm must continue performing comparisons
4. ...otherwise, not know what to output...
5. **Q:** What is the worst running time of algorithm?
6. Answer: Longest path from root in the decision tree.
   ...because we count only comparisons!

Back to sorting...

1. v: a node in decision tree.
2. If $|P(v)| > 1$: more than one permutation associated with it...
3. algorithm must continue performing comparisons
4. ...otherwise, not know what to output...
5. **Q:** What is the worst running time of algorithm?
6. Answer: Longest path from root in the decision tree.
   ...because we count only comparisons!

Lower bound on sorting...

Lemma

Any deterministic sorting algorithm in the comparisons model, must perform $\Omega(n \log n)$ comparisons.

Proof

1. Algorithm in the comparison model $\equiv$ a decision tree.
2. Use an adversary argument.
3. Adversary pick the worse possible input for the algorithm.
4. Input is a permutation.
5. $\mathcal{T}$: the optimal decision tree.
6. $|P(r)| = n!$, where $r = \mathrm{root}(\mathcal{T})$.

Lower bound on sorting...

Lemma

Any deterministic sorting algorithm in the comparisons model, must perform $\Omega(n \log n)$ comparisons.

Proof

1. Algorithm in the comparison model $\equiv$ a decision tree.
2. Use an adversary argument.
3. Adversary pick the worse possible input for the algorithm.
4. Input is a permutation.
5. $\mathcal{T}$: the optimal decision tree.
6. $|P(r)| = n!$, where $r = \mathrm{root}(\mathcal{T})$.

Lower bound on sorting...

Lemma

Any deterministic sorting algorithm in the comparisons model, must perform $\Omega(n \log n)$ comparisons.

Proof

1. Algorithm in the comparison model $\equiv$ a decision tree.
2. Use an adversary argument.
3. Adversary pick the worse possible input for the algorithm.
4. Input is a permutation.
5. $\mathcal{T}$: the optimal decision tree.
6. $|P(r)| = n!$, where $r = \mathrm{root}(\mathcal{T})$.

Lower bound on sorting...

Lemma

Any deterministic sorting algorithm in the comparisons model, must perform $\Omega(n \log n)$ comparisons.

Proof

1. Algorithm in the comparison model $\equiv$ a decision tree.
2. Use an adversary argument.
3. Adversary pick the worse possible input for the algorithm.
4. Input is a permutation.
5. $\mathcal{T}$: the optimal decision tree.
6. $|P(r)| = n!$, where $r = \mathrm{root}(\mathcal{T})$.

Lower bound on sorting...

Lemma

Any deterministic sorting algorithm in the comparisons model, must perform $\Omega(n \log n)$ *comparisons.*

Proof

1. Algorithm in the comparison model $\equiv$ a decision tree.
2. Use an adversary argument.
3. Adversary pick the worse possible input for the algorithm.
4. Input is a permutation.
5. $\mathcal{T}$: the optimal decision tree.
6. $|P(r)| = n!$, where $r = \mathrm{root}(\mathcal{T})$.

Lower bound on sorting...

Lemma

Any deterministic sorting algorithm in the comparisons model, must perform $\Omega(n \log n)$ *comparisons.*

Proof

1. Algorithm in the comparison model $\equiv$ a decision tree.
2. Use an adversary argument.
3. Adversary pick the worse possible input for the algorithm.
4. Input is a permutation.
5. $\mathcal{T}$: the optimal decision tree.
6. $|P(r)| = n!$, where $r = \mathrm{root}(\mathcal{T})$.

Lower bound on sorting...

Lemma

Any deterministic sorting algorithm in the comparisons model, must perform $\Omega(n \log n)$ *comparisons.*

Proof

1. Algorithm in the comparison model $\equiv$ a decision tree.
2. Use an adversary argument.
3. Adversary pick the worse possible input for the algorithm.
4. Input is a permutation.
5. $\mathcal{T}$: the optimal decision tree.
6. $|P(r)| = n!$, where $r = \mathrm{root}(\mathcal{T})$.

Proof continued...

1. u, v: children of r.
2. Adversary: no commitment on which of the permutations of $P(r)$ it is using.
3. Algorithm perform compares x_i to x_j in root...
4. Adversary computes $P(u)$ and $P(v)$
   [Adversary has infinite computation power!]
5. Adversary goes to u if $|P(u)| \geq |P(v)|$, and to v otherwise.
6. Adversary traversal: always pick child with more permutations.
7. $v_1, \ldots, v_k$: path taken by adversary.
8. Adversary input:
   The input realizing the single permutation of $P(v_k)$.

Proof continued...

1. u, v: children of r.
2. Adversary: no commitment on which of the permutations of $P(r)$ it is using.
3. Algorithm perform compares x_i to x_j in root...
4. Adversary computes $P(u)$ and $P(v)$
   [Adversary has infinite computation power!]
5. Adversary goes to u if $|P(u)| \geq |P(v)|$, and to v otherwise.
6. Adversary traversal: always pick child with more permutations.
7. $v_1, \ldots, v_k$: path taken by adversary.
8. Adversary input:
   The input realizing the single permutation of $P(v_k)$.

Proof continued...

1. u, v: children of r.
2. Adversary: no commitment on which of the permutations of $P(r)$ it is using.
3. Algorithm perform compares x_i to x_j in root...
4. Adversary computes $P(u)$ and $P(v)$
   [Adversary has infinite computation power!]
5. Adversary goes to u if $|P(u)| \geq |P(v)|$, and to v otherwise.
6. Adversary traversal: always pick child with more permutations.
7. $v_1, \ldots, v_k$: path taken by adversary.
8. Adversary input:
   The input realizing the single permutation of $P(v_k)$.

Proof continued...

1. u, v: children of r.
2. Adversary: no commitment on which of the permutations of $P(r)$ it is using.
3. Algorithm perform compares x_i to x_j in root...
4. Adversary computes $P(u)$ and $P(v)$
   [Adversary has infinite computation power!]
5. Adversary goes to u if $|P(u)| \geq |P(v)|$, and to v otherwise.
6. Adversary traversal: always pick child with more permutations.
7. $v_1, \ldots, v_k$: path taken by adversary.
8. Adversary input:
   The input realizing the single permutation of $P(v_k)$.

Proof continued...

1. u, v: children of r.
2. Adversary: no commitment on which of the permutations of $P(r)$ it is using.
3. Algorithm perform compares x_i to x_j in root...
4. Adversary computes $P(u)$ and $P(v)$
   [Adversary has infinite computation power!]
5. Adversary goes to u if $|P(u)| \geq |P(v)|$, and to v otherwise.
6. Adversary traversal: always pick child with more permutations.
7. $v_1, \ldots, v_k$: path taken by adversary.
8. Adversary input:
   The input realizing the single permutation of $P(v_k)$.

Proof continued...

1. u, v: children of r.
2. Adversary: no commitment on which of the permutations of $P(r)$ it is using.
3. Algorithm perform compares x_i to x_j in root...
4. Adversary computes $P(u)$ and $P(v)$
   [Adversary has infinite computation power!]
5. Adversary goes to u if $|P(u)| \geq |P(v)|$, and to v otherwise.
6. Adversary traversal: always pick child with more permutations.
7. $v_1, \ldots, v_k$: path taken by adversary.
8. Adversary input:
   The input realizing the single permutation of $P(v_k)$.

Proof continued...

1. u, v: children of r.
2. Adversary: no commitment on which of the permutations of $P(r)$ it is using.
3. Algorithm perform compares x_i to x_j in root...
4. Adversary computes $P(u)$ and $P(v)$ [Adversary has infinite computation power!]
5. Adversary goes to u if $|P(u)| \geq |P(v)|$, and to v otherwise.
6. Adversary traversal: always pick child with more permutations.
7. $v_1, \ldots, v_k$: path taken by adversary.
8. Adversary input:
   The input realizing the single permutation of $P(v_k)$.

Proof continued...

1. u, v: children of r.
2. Adversary: no commitment on which of the permutations of $P(r)$ it is using.
3. Algorithm perform compares x_i to x_j in root...
4. Adversary computes $P(u)$ and $P(v)$
   [Adversary has infinite computation power!]
5. Adversary goes to u if $|P(u)| \geq |P(v)|$, and to v otherwise.
6. Adversary traversal: always pick child with more permutations.
7. $v_1, \ldots, v_k$: path taken by adversary.
8. Adversary input:
   The input realizing the single permutation of $P(v_k)$.

Proof continued...

1. Note, that

$$1 = |P(v_k)| \geq \frac{|P(v_{k-1})|}{2} \geq \ldots \geq \frac{|P(v_1)|}{2^{k-1}}.$$

2. $2^{k-1} \geq |P(v_1)| = n!$.
3. $k \geq \lg(n!) + 1 = \Omega(n \log n)$.
4. Depth of $\mathcal{T}$ is $\Omega(n \log n)$. ■

Proof continued...

1. Note, that

$$1 = |P(v_k)| \geq \frac{|P(v_{k-1})|}{2} \geq \ldots \geq \frac{|P(v_1)|}{2^{k-1}}.$$

2. $2^{k-1} \geq |P(v_1)| = n!$.
3. $k \geq \lg(n!) + 1 = \Omega(n \log n)$.
4. Depth of $\mathcal{T}$ is $\Omega(n \log n)$. ∎

Proof continued...

1. Note, that
$$1 = |P(v_k)| \geq \frac{|P(v_{k-1})|}{2} \geq \ldots \geq \frac{|P(v_1)|}{2^{k-1}}.$$
2. $2^{k-1} \geq |P(v_1)| = n!$.
3. $k \geq \lg(n!) + 1 = \Omega(n \log n)$.
4. Depth of $\mathcal{T}$ is $\Omega(n \log n)$. ■

Proof continued...

1. Note, that
$$1 = |P(v_k)| \geq \frac{|P(v_{k-1})|}{2} \geq \ldots \geq \frac{|P(v_1)|}{2^{k-1}}.$$
2. $2^{k-1} \geq |P(v_1)| = n!$.
3. $k \geq \lg(n!) + 1 = \Omega(n \log n)$.
4. Depth of $\mathcal{T}$ is $\Omega(n \log n)$. ∎

Proof continued...

1. Note, that

$$1 = |P(v_k)| \geq \frac{|P(v_{k-1})|}{2} \geq \ldots \geq \frac{|P(v_1)|}{2^{k-1}}.$$

2. $2^{k-1} \geq |P(v_1)| = n!$.
3. $k \geq \lg(n!) + 1 = \Omega(n \log n)$.
4. Depth of $\mathcal{T}$ is $\Omega(n \log n)$.

Proof continued...

1. Note, that

$$1 = |P(v_k)| \geq \frac{|P(v_{k-1})|}{2} \geq \ldots \geq \frac{|P(v_1)|}{2^{k-1}}.$$

2. $2^{k-1} \geq |P(v_1)| = n!$.
3. $k \geq \lg(n!) + 1 = \Omega(n \log n)$.
4. Depth of $\mathcal{T}$ is $\Omega(n \log n)$.

Proof continued...

1. Note, that

$$1 = |P(v_k)| \geq \frac{|P(v_{k-1})|}{2} \geq \ldots \geq \frac{|P(v_1)|}{2^{k-1}}.$$

2. $2^{k-1} \geq |P(v_1)| = n!$.
3. $k \geq \lg(n!) + 1 = \Omega(n \log n)$.
4. Depth of $\mathcal{T}$ is $\Omega(n \log n)$.

Proof continued...

1. Note, that

$$1 = |P(v_k)| \geq \frac{|P(v_{k-1})|}{2} \geq \ldots \geq \frac{|P(v_1)|}{2^{k-1}}.$$

2. $2^{k-1} \geq |P(v_1)| = n!$.
3. $k \geq \lg(n!) + 1 = \Omega(n \log n)$.
4. Depth of $\mathcal{T}$ is $\Omega(n \log n)$.

Proof continued...

1. Note, that

$$1 = |P(v_k)| \geq \frac{|P(v_{k-1})|}{2} \geq \ldots \geq \frac{|P(v_1)|}{2^{k-1}}.$$

2. $2^{k-1} \geq |P(v_1)| = n!$.
3. $k \geq \lg(n!) + 1 = \Omega(n \log n)$.
4. Depth of $\mathcal{T}$ is $\Omega(n \log n)$. ■

22.2: Uniqueness

22.2.1: Uniqueness

Uniqueness

Problem

Given an input of n real numbers $x_1, \ldots, x_n$. Decide if all the numbers are unique.

1. Intuitively: easier than sorting.
2. Can be solved in linear time!
3. ...but in a strange computation model.
4. Surprisingly...

Theorem

Any deterministic algorithm in the comparison model that solves Uniqueness, has $\Omega(n \log n)$ running time in the worst case.

5. Different models, different results.

Uniqueness

Problem

Given an input of n real numbers $x_1, \ldots, x_n$. Decide if all the numbers are unique.

1. Intuitively: easier than sorting.
2. Can be solved in linear time!
3. ...but in a strange computation model.
4. Surprisingly...

Theorem

Any deterministic algorithm in the comparison model that solves Uniqueness, has $\Omega(n \log n)$ running time in the worst case.

5. Different models, different results.

Uniqueness

Problem

Given an input of n real numbers $x_1, \ldots, x_n$. Decide if all the numbers are unique.

1. Intuitively: easier than sorting.
2. Can be solved in linear time!
3. ...but in a strange computation model.
4. Surprisingly...

Theorem

Any deterministic algorithm in the comparison model that solves Uniqueness, has $\Omega(n \log n)$ running time in the worst case.

5. Different models, different results.

Uniqueness

Problem

Given an input of n real numbers $x_1, \ldots, x_n$. Decide if all the numbers are unique.

1. Intuitively: easier than sorting.
2. Can be solved in linear time!
3. ...but in a strange computation model.
4. Surprisingly...

Theorem

Any deterministic algorithm in the comparison model that solves Uniqueness, has $\Omega(n \log n)$ running time in the worst case.

5. Different models, different results.

Uniqueness

Problem

Given an input of n real numbers $x_1, \ldots, x_n$. Decide if all the numbers are unique.

1. Intuitively: easier than sorting.
2. Can be solved in linear time!
3. ...but in a strange computation model.
4. Surprisingly...

Theorem

Any deterministic algorithm in the comparison model that solves Uniqueness, has $\Omega(n \log n)$ running time in the worst case.

5. Different models, different results.

Uniqueness

Problem

Given an input of n real numbers $x_1, \ldots, x_n$. Decide if all the numbers are unique.

1. Intuitively: easier than sorting.
2. Can be solved in linear time!
3. ...but in a strange computation model.
4. Surprisingly...

Theorem

Any deterministic algorithm in the comparison model that solves Uniqueness, has $\Omega(n \log n)$ running time in the worst case.

5. Different models, different results.

Uniqueness lower bound

Proof similar but trickier.

$\mathcal{T}$: decision tree (every node has three children).

Lemma

v: node in decision tree. If $P(v)$ contains more than one permutation, then there exists two inputs which arrive to v, where one is unique and other is not.

Proof

1. σ, σ': any two different permutations in $P(v)$.
2. $X = x_1, \ldots, x_n$ be an input realizing σ.
3. $Y = y_1, \ldots, y_n$: input realizing σ'.
4. Let $Z(t) = (z_1(t), \ldots, z_n(t))$ an input where $z_i(t) = tx_i + (1-t)y_i$, for $t \in [0, 1]$.

Uniqueness lower bound

Proof similar but trickier.
$\mathcal{T}$: decision tree (every node has three children).

Lemma

v: node in decision tree. If $P(v)$ contains more than one permutation, then there exists two inputs which arrive to v, where one is unique and other is not.

Proof

1. σ, σ': any two different permutations in $P(v)$.
2. $X = x_1, \ldots, x_n$ be an input realizing σ.
3. $Y = y_1, \ldots, y_n$: input realizing σ'.
4. Let $Z(t) = (z_1(t), \ldots, z_n(t))$ an input where $z_i(t) = tx_i + (1-t)y_i$, for $t \in [0, 1]$.

Uniqueness lower bound

Proof similar but trickier.

$\mathcal{T}$: decision tree (every node has three children).

Lemma

v: node in decision tree. If $P(v)$ contains more than one permutation, then there exists two inputs which arrive to v, where one is unique and other is not.

Proof

1. σ, σ': any two different permutations in $P(v)$.
2. $X = x_1, \ldots, x_n$ be an input realizing σ.
3. $Y = y_1, \ldots, y_n$: input realizing σ'.
4. Let $Z(t) = (z_1(t), \ldots, z_n(t))$ an input where $z_i(t) = tx_i + (1-t)y_i$, for $t \in [0, 1]$.

Uniqueness lower bound

Proof similar but trickier.

$\mathcal{T}$: decision tree (every node has three children).

Lemma

v: node in decision tree. If $P(v)$ contains more than one permutation, then there exists two inputs which arrive to v, where one is unique and other is not.

Proof

1. σ, σ': any two different permutations in $P(v)$.
2. $X = x_1, \ldots, x_n$ be an input realizing σ.
3. $Y = y_1, \ldots, y_n$: input realizing σ'.
4. Let $Z(t) = (z_1(t), \ldots, z_n(t))$ an input where $z_i(t) = tx_i + (1-t)y_i$, for $t \in [0, 1]$.

Uniqueness lower bound

Proof similar but trickier.

$\mathcal{T}$: decision tree (every node has three children).

Lemma

v: node in decision tree. If $P(v)$ contains more than one permutation, then there exists two inputs which arrive to v, where one is unique and other is not.

Proof

1. σ, σ': any two different permutations in $P(v)$.
2. $X = x_1, \ldots, x_n$ be an input realizing σ.
3. $Y = y_1, \ldots, y_n$: input realizing σ'.
4. Let $Z(t) = (z_1(t), \ldots, z_n(t))$ an input where $z_i(t) = tx_i + (1-t)y_i$, for $t \in [0, 1]$.

Proof continued...

1. $Z(t) = (z_1(t), \ldots, z_n(t))$ an input where $z_i(t) = tx_i + (1-t)y_i$, for $t \in [0,1]$.
2. $Z(0) = (x_1, \ldots, x_n)$ and $Z(1) = (y_1, \ldots, y_n)$.
3. Claim: $\forall t \in [0,1]$ the input $Z(t)$ will arrive to the node v in $\mathcal{T}$.

Proof continued...

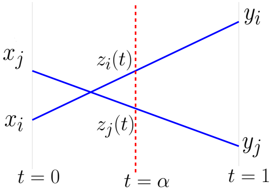
1. $Z(t) = (z_1(t), \ldots, z_n(t))$ an input where $z_i(t) = tx_i + (1-t)y_i$, for $t \in [0,1]$.
2. $Z(0) = (x_1, \ldots, x_n)$ and $Z(1) = (y_1, \ldots, y_n)$.
3. Claim: $\forall t \in [0,1]$ the input $Z(t)$ will arrive to the node v in $\mathcal{T}$.

Proof continued...

1. $Z(t) = (z_1(t), \ldots, z_n(t))$ an input where $z_i(t) = tx_i + (1-t)y_i$, for $t \in [0,1]$.
2. $Z(0) = (x_1, \ldots, x_n)$ and $Z(1) = (y_1, \ldots, y_n)$.
3. Claim: $\forall t \in [0,1]$ the input $Z(t)$ will arrive to the node v in $\mathfrak{T}$.

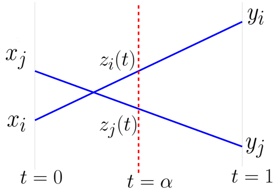
Proof of claim...

1. Assume false.
2. Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in $\mathcal{T}$.
3. Assume: compared the ith to jth input element, when paths diverted in $\mathcal{T}$.
4. I.e., Different path in $\mathcal{T}$ then the one for X and Y.
5. Claim: $x_i < x_j$ and $y_i > y_j$ or $x_i > x_j$ and $y_i < y_j$.
6. In either case X or Y will not arrive to v in $\mathcal{T}$.
7. Consider the functions $z_i(t)$ and $z_j(t)$:



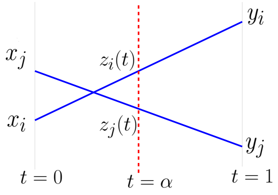
Proof of claim...

1. Assume false.
2. Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in $\mathcal{T}$.
3. Assume: compared the ith to jth input element, when paths diverted in $\mathcal{T}$.
4. I.e., Different path in $\mathcal{T}$ then the one for X and Y.
5. Claim: $x_i < x_j$ and $y_i > y_j$ or $x_i > x_j$ and $y_i < y_j$.
6. In either case X or Y will not arrive to v in $\mathcal{T}$.
7. Consider the functions $z_i(t)$ and $z_j(t)$:



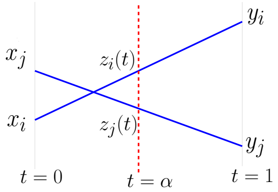
Proof of claim...

1. Assume false.
2. Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in $\mathcal{T}$.
3. Assume: compared the ith to jth input element, when paths diverted in $\mathcal{T}$.
4. I.e., Different path in $\mathcal{T}$ then the one for X and Y.
5. Claim: $x_i < x_j$ and $y_i > y_j$ or $x_i > x_j$ and $y_i < y_j$.
6. In either case X or Y will not arrive to v in $\mathcal{T}$.
7. Consider the functions $z_i(t)$ and $z_j(t)$:



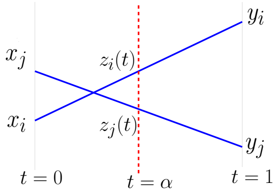
Proof of claim...

1. Assume false.
2. Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in $\mathcal{T}$.
3. Assume: compared the ith to jth input element, when paths diverted in $\mathcal{T}$.
4. I.e., Different path in $\mathcal{T}$ then the one for X and Y.
5. Claim: $x_i < x_j$ and $y_i > y_j$ or $x_i > x_j$ and $y_i < y_j$.
6. In either case X or Y will not arrive to v in $\mathcal{T}$.
7. Consider the functions $z_i(t)$ and $z_j(t)$:



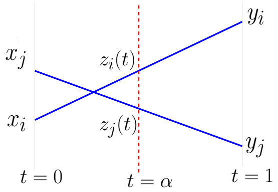
Proof of claim...

1. Assume false.
2. Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in $\mathcal{T}$.
3. Assume: compared the ith to jth input element, when paths diverted in $\mathcal{T}$.
4. I.e., Different path in $\mathcal{T}$ then the one for X and Y.
5. Claim: $x_i < x_j$ and $y_i > y_j$ or $x_i > x_j$ and $y_i < y_j$.
6. In either case X or Y will not arrive to v in $\mathcal{T}$.
7. Consider the functions $z_i(t)$ and $z_j(t)$:



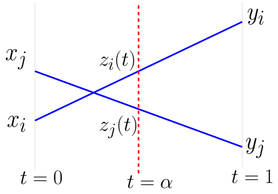
Proof of claim...

1. Assume false.
2. Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in $\mathcal{T}$.
3. Assume: compared the ith to jth input element, when paths diverted in $\mathcal{T}$.
4. I.e., Different path in $\mathcal{T}$ then the one for X and Y.
5. Claim: $x_i < x_j$ and $y_i > y_j$ or $x_i > x_j$ and $y_i < y_j$.
6. In either case X or Y will not arrive to v in $\mathcal{T}$.
7. Consider the functions $z_i(t)$ and $z_j(t)$:



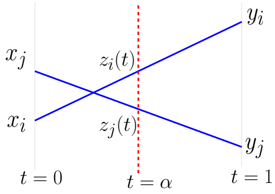
Proof of claim...

1. Assume false.
2. Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in $\mathcal{T}$.
3. Assume: compared the ith to jth input element, when paths diverted in $\mathcal{T}$.
4. I.e., Different path in $\mathcal{T}$ then the one for X and Y.
5. Claim: $x_i < x_j$ and $y_i > y_j$ or $x_i > x_j$ and $y_i < y_j$.
6. In either case X or Y will not arrive to v in $\mathcal{T}$.
7. Consider the functions $z_i(t)$ and $z_j(t)$:



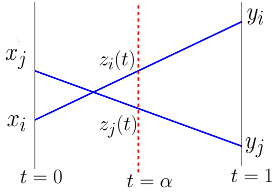
Proof of claim...

1. Assume false.
2. Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in $\mathfrak{T}$.
3. Assume: compared the ith to jth input element, when paths diverted in $\mathfrak{T}$.
4. I.e., Different path in $\mathfrak{T}$ then the one for X and Y.
5. Claim: $x_i < x_j$ and $y_i > y_j$ or $x_i > x_j$ and $y_i < y_j$.
6. In either case X or Y will not arrive to v in $\mathfrak{T}$.
7. Consider the functions $z_i(t)$ and $z_j(t)$:



Proof of claim...

1. Assume false.
2. Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in $\mathfrak{T}$.
3. Assume: compared the ith to jth input element, when paths diverted in $\mathfrak{T}$.
4. I.e., Different path in $\mathfrak{T}$ then the one for X and Y.
5. Claim: $x_i < x_j$ and $y_i > y_j$ or $x_i > x_j$ and $y_i < y_j$.
6. In either case X or Y will not arrive to v in $\mathfrak{T}$.
7. Consider the functions $z_i(t)$ and $z_j(t)$:



Proof of claim continued...

1. Ordering between $z_i(t)$ and $z_j(t)$ is either ordering between x_i and x_j or the ordering between y_i and y_j.
2. Conclusion: $\forall t$: inputs $Z(t)$ arrive to the same node $v \in \mathcal{T}$. ■

Proof of claim continued...

1. Ordering between $z_i(t)$ and $z_j(t)$ is either ordering between x_i and x_j or the ordering between y_i and y_j.
2. Conclusion: $\forall t$: inputs $Z(t)$ arrive to the same node $v \in \mathcal{T}$. ∎

Back to proof of Lemma...

1. Recap:
   1. Recall: X, Y to different permutations that their distinct input arrives to the same node $v \in \mathcal{T}$.
   2. Proved: $\forall t \in [0, 1]$: $Z(t) = (z_1(t), \ldots, z_n(t))$ arrives to same node $v \in \mathcal{T}$.
2. However: There must be $\beta \in (0, 1)$ where $Z(\beta)$ has two numbers equal:
3. $Z(\beta)$: has a pair of numbers that are not unique.

Back to proof of Lemma...

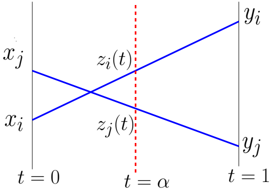
1. Recap:
   1. Recall: $\boldsymbol{X}, \boldsymbol{Y}$ to different permutations that their distinct input arrives to the same node $\boldsymbol{v} \in \mathfrak{T}$.
   2. Proved: $\forall t \in [0, 1]$: $Z(t) = (z_1(t), \ldots, z_n(t))$ arrives to same node $v \in \mathfrak{T}$.
2. However: There must be $\beta \in (0, 1)$ where $Z(\beta)$ has two numbers equal:
3. $Z(\beta)$: has a pair of numbers that are not unique.

Back to proof of Lemma...

1. Recap:
   1. Recall: $\boldsymbol{X}, \boldsymbol{Y}$ to different permutations that their distinct input arrives to the same node $\boldsymbol{v} \in \mathfrak{T}$.
   2. Proved: $\forall \boldsymbol{t} \in [\mathbf{0}, \mathbf{1}]$: $\boldsymbol{Z}(\boldsymbol{t}) = (\boldsymbol{z_1}(\boldsymbol{t}), \ldots, \boldsymbol{z_n}(\boldsymbol{t}))$ arrives to same node $\boldsymbol{v} \in \mathfrak{T}$.
2. However: There must be $\beta \in (0, 1)$ where $Z(\beta)$ has two numbers equal:
3. $Z(\beta)$: has a pair of numbers that are not unique.

Back to proof of Lemma...

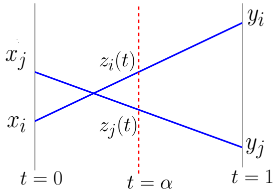
1. Recap:
   1. Recall: X, Y to different permutations that their distinct input arrives to the same node $v \in \mathcal{T}$.
   2. Proved: $\forall t \in [0, 1]$: $Z(t) = (z_1(t), \ldots, z_n(t))$ arrives to same node $v \in \mathcal{T}$.
2. However: There must be $\beta \in (0, 1)$ where $Z(\beta)$ has two numbers equal:



3. $Z(\beta)$: has a pair of numbers that are not unique.

Back to proof of Lemma...

1. Recap:
   1. Recall: $\boldsymbol{X}, \boldsymbol{Y}$ to different permutations that their distinct input arrives to the same node $\boldsymbol{v} \in \mathfrak{T}$.
   2. Proved: $\forall t \in [0, 1]$: $\boldsymbol{Z(t) = (z_1(t), \ldots, z_n(t))}$ arrives to same node $\boldsymbol{v} \in \mathfrak{T}$.
2. However: There must be $\boldsymbol{\beta} \in \boldsymbol{(0, 1)}$ where $\boldsymbol{Z(\beta)}$ has two numbers equal:



3. $\boldsymbol{Z(\beta)}$: has a pair of numbers that are not unique.

Proof of Lemma continued...

1. Done: Found inputs $Z(\mathbf{0})$ and $Z(\beta)$
2. such that one is unique and the other is not.
3. ... both arrive to v. ■

Proof of Lemma continued...

1. Done: Found inputs $Z(0)$ and $Z(\beta)$
2. such that one is unique and the other is not.
3. ... both arrive to v. ∎

Proof of Lemma continued...

1. Done: Found inputs $Z(0)$ and $Z(\beta)$
2. such that one is unique and the other is not.
3. ... both arrive to v. ■

Proof of Lemma continued...

1. Done: Found inputs $Z(0)$ and $Z(\beta)$
2. such that one is unique and the other is not.
3. ... both arrive to v. ■

Proof of Lemma continued...

1. Done: Found inputs $Z(0)$ and $Z(\beta)$
2. such that one is unique and the other is not.
3. ... both arrive to v. ■

Proved the following:

Lemma

v: node in decision tree. If $P(v)$ contains more than one permutation, then there exists two inputs which arrive to v, where one is unique and other is not.

Uniqueness takes $\Omega(n \log n)$ time

1. Apply the same argument as before.
2. If in the decision tree, the adversary arrived to a node...
3. containing more than one permutation, it continues into the child with more permutations.
4. As in the sorting argument, it follows that there exists a path in $\mathcal{T}$ of length $\Omega(n \log n)$.
5. We conclude:

Theorem

Solving **Uniqueness** *for a set of n real numbers takes $\Theta(n \log n)$ time in the comparison model.*

Uniqueness takes $\Omega(n \log n)$ time

1. Apply the same argument as before.
2. If in the decision tree, the adversary arrived to a node...
3. containing more than one permutation, it continues into the child with more permutations.
4. As in the sorting argument, it follows that there exists a path in $\mathcal{T}$ of length $\Omega(n \log n)$.
5. We conclude:

Theorem

Solving **Uniqueness** *for a set of n real numbers takes $\Theta(n \log n)$ time in the comparison model.*

Uniqueness takes $\Omega(n \log n)$ time

1. Apply the same argument as before.
2. If in the decision tree, the adversary arrived to a node...
3. containing more than one permutation, it continues into the child with more permutations.
4. As in the sorting argument, it follows that there exists a path in $\mathcal{T}$ of length $\Omega(n \log n)$.
5. We conclude:

Theorem

Solving **Uniqueness** *for a set of n real numbers takes $\Theta(n \log n)$ time in the comparison model.*

Uniqueness takes $\Omega(n \log n)$ time

1. Apply the same argument as before.
2. If in the decision tree, the adversary arrived to a node...
3. containing more than one permutation, it continues into the child with more permutations.
4. As in the sorting argument, it follows that there exists a path in $\mathcal{T}$ of length $\Omega(n \log n)$.
5. We conclude:

Theorem

Solving **Uniqueness** *for a set of n real numbers takes $\Theta(n \log n)$ time in the comparison model.*

Uniqueness takes $\Omega(n \log n)$ time

1. Apply the same argument as before.
2. If in the decision tree, the adversary arrived to a node...
3. containing more than one permutation, it continues into the child with more permutations.
4. As in the sorting argument, it follows that there exists a path in $\mathcal{T}$ of length $\Omega(n \log n)$.
5. We conclude:

Theorem

Solving **Uniqueness** *for a set of* n *real numbers takes* $\Theta(n \log n)$ *time in the comparison model.*

22.2.2: Algebraic tree model

Algebraic tree model

1. At each node, allowed to compute a polynomial, and ask for its sign at a certain point
2. Example: comparing x_i to x_j is equivalent to asking if the polynomial $x_i - x_j$ is positive/negative/zero).
3. One can prove things in this model, but it requires considerably stronger techniques.

Problem

(Degenerate points) Given a set P of n points in $\mathbb{R}^d$, deciding if there are $d+1$ points in P which are co-linear (all lying on a common plane).

4. Jeff Erickson and Raimund Seidel: Solving the degenerate points problem requires $\Omega(n^d)$ time in a "reasonable" model of computation.

Algebraic tree model

1. At each node, allowed to compute a polynomial, and ask for its sign at a certain point
2. Example: comparing x_i to x_j is equivalent to asking if the polynomial $x_i - x_j$ is positive/negative/zero).
3. One can prove things in this model, but it requires considerably stronger techniques.

Problem

(Degenerate points) Given a set P of n points in $\mathbb{R}^d$, deciding if there are $d+1$ points in P which are co-linear (all lying on a common plane).

4. Jeff Erickson and Raimund Seidel: Solving the degenerate points problem requires $\Omega(n^d)$ time in a "reasonable" model of computation.

Algebraic tree model

1. At each node, allowed to compute a polynomial, and ask for its sign at a certain point
2. Example: comparing x_i to x_j is equivalent to asking if the polynomial $x_i - x_j$ is positive/negative/zero).
3. One can prove things in this model, but it requires considerably stronger techniques.

Problem

(Degenerate points) Given a set P of n points in $\mathbb{R}^d$, deciding if there are $d+1$ points in P which are co-linear (all lying on a common plane).

4. Jeff Erickson and Raimund Seidel: Solving the degenerate points problem requires $\Omega(n^d)$ time in a "reasonable" model of computation.

Algebraic tree model

1. At each node, allowed to compute a polynomial, and ask for its sign at a certain point
2. Example: comparing x_i to x_j is equivalent to asking if the polynomial $x_i - x_j$ is positive/negative/zero).
3. One can prove things in this model, but it requires considerably stronger techniques.

Problem

(Degenerate points) Given a set P of n points in $\mathbb{R}^d$, deciding if there are $d+1$ points in P which are co-linear (all lying on a common plane).

4. Jeff Erickson and Raimund Seidel: Solving the degenerate points problem requires $\Omega(n^d)$ time in a "reasonable" model of computation.

22.3: 3Sum-Hard

22.3.1: 3Sum-Hard

3Sum-Hard

1. Consider the following problem:

Problem

(3SUM): Given three sets of numbers A, B, C *are there three numbers* $a \in A$, $b \in B$ *and* $c \in C$, *such that* $a + b = c$.

2. One can show...

Lemma

One can solve the $3SUM$ *problem in* $O(n^2)$ *time.*

Proof.

Exercise... □

3Sum-Hard

1. Consider the following problem:

Problem

(3SUM): Given three sets of numbers A, B, C are there three numbers $a \in A$, $b \in B$ and $c \in C$, such that $a + b = c$.

2. One can show...

Lemma

One can solve the $3SUM$ problem in $O(n^2)$ time.

Proof.

Exercise... □

3Sum-Hard continued

1. Somewhat surprisingly, no better solution is known.
2. Open Problem: Find a subquadratic algorithm for **3SUM**.
3. It is widely believed that no such algorithm exists.
4. There is a large collection problems that are 3SUM-Hard: if you solve them in subquadratic time, then you can solve 3SUM in subquadratic time.

3Sum-Hard continued

1. Somewhat surprisingly, no better solution is known.
2. Open Problem: Find a subquadratic algorithm for **3SUM**.
3. It is widely believed that no such algorithm exists.
4. There is a large collection problems that are 3SUM-Hard: if you solve them in subquadratic time, then you can solve 3SUM in subquadratic time.

3Sum-Hard continued

1. Somewhat surprisingly, no better solution is known.
2. Open Problem: Find a subquadratic algorithm for **3SUM**.
3. It is widely believed that no such algorithm exists.
4. There is a large collection problems that are 3SUM-Hard: if you solve them in subquadratic time, then you can solve 3SUM in subquadratic time.

3Sum-Hard continued

1. Somewhat surprisingly, no better solution is known.
2. Open Problem: Find a subquadratic algorithm for **3SUM**.
3. It is widely believed that no such algorithm exists.
4. There is a large collection problems that are 3SUM-Hard: if you solve them in subquadratic time, then you can solve 3SUM in subquadratic time.

3Sum-Hard continued

1. Somewhat surprisingly, no better solution is known.
2. Open Problem: Find a subquadratic algorithm for **3SUM**.
3. It is widely believed that no such algorithm exists.
4. There is a large collection problems that are 3SUM-Hard: if you solve them in subquadratic time, then you can solve 3SUM in subquadratic time.

3SUM-hard problems

1. Those problems include:
   1. For n points in the plane, is there three points that lie on the same line.
   2. Given a set of n triangles in the plane, do they cover the unit square
   3. Given two polygons P and Q can one translate P such that it is contained inside Q?
2. So, how does one prove that a problem is 3SUM hard?
3. Reductions.
4. Reductions must have subquadratic running time.
5. The details are interesting, but are omitted.

3SUM-hard problems

1. Those problems include:
    1. For n points in the plane, is there three points that lie on the same line.
    2. Given a set of n triangles in the plane, do they cover the unit square
    3. Given two polygons P and Q can one translate P such that it is contained inside Q?
2. So, how does one prove that a problem is 3SUM hard?
3. Reductions.
4. Reductions must have subquadratic running time.
5. The details are interesting, but are omitted.

3SUM-hard problems

1. Those problems include:
   1. For n points in the plane, is there three points that lie on the same line.
   2. Given a set of n triangles in the plane, do they cover the unit square
   3. Given two polygons P and Q can one translate P such that it is contained inside Q?
2. So, how does one prove that a problem is 3SUM hard?
3. Reductions.
4. Reductions must have subquadratic running time.
5. The details are interesting, but are omitted.

3SUM-hard problems

1. Those problems include:
   1. For n points in the plane, is there three points that lie on the same line.
   2. Given a set of n triangles in the plane, do they cover the unit square
   3. Given two polygons P and Q can one translate P such that it is contained inside Q?
2. So, how does one prove that a problem is 3SUM hard?
3. Reductions.
4. Reductions must have subquadratic running time.
5. The details are interesting, but are omitted.

3SUM-hard problems

1. Those problems include:
   1. For n points in the plane, is there three points that lie on the same line.
   2. Given a set of n triangles in the plane, do they cover the unit square
   3. Given two polygons P and Q can one translate P such that it is contained inside Q?
2. So, how does one prove that a problem is 3SUM hard?
3. Reductions.
4. Reductions must have subquadratic running time.
5. The details are interesting, but are omitted.

3SUM-hard problems

1. Those problems include:
   1. For n points in the plane, is there three points that lie on the same line.
   2. Given a set of n triangles in the plane, do they cover the unit square
   3. Given two polygons P and Q can one translate P such that it is contained inside Q?
2. So, how does one prove that a problem is 3SUM hard?
3. Reductions.
4. Reductions must have subquadratic running time.
5. The details are interesting, but are omitted.

3SUM-hard problems

1. Those problems include:
   1. For n points in the plane, is there three points that lie on the same line.
   2. Given a set of n triangles in the plane, do they cover the unit square
   3. Given two polygons P and Q can one translate P such that it is contained inside Q?
2. So, how does one prove that a problem is 3SUM hard?
3. Reductions.
4. Reductions must have subquadratic running time.
5. The details are interesting, but are omitted.

Notes

Notes

Notes

Notes