

Lower bounds

Lecture 26

December 2, 2014

Sorting...

- ① n items: x_1, \dots, 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, \dots, 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, \dots, 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, \dots, 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 i th item in input to j th 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 i th item in input to j th 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 i th item in input to j th 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 i th item in input to j th 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

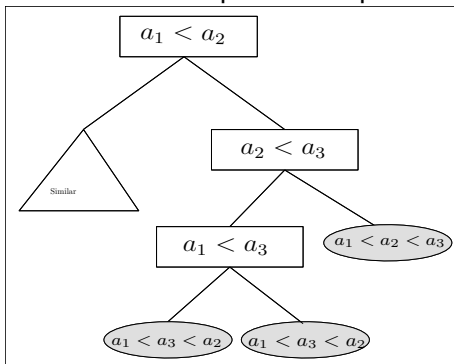
- ① sorting algorithm: a decision procedure.
- ② Each stage: has current collection of comparisons done.
- ③ ... need to decide which comparison to perform next.

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 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 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 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 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 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 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 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 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?

- ❶ $\pi = (3, 4, 1, 2)$ is permutation in $P(v)$.
- ❷ Formally $\pi : \llbracket n \rrbracket \rightarrow \llbracket n \rrbracket$ is a one-to-one function.
 $\llbracket n \rrbracket = \{1, \dots, n\}$
can be written as:
$$\pi = (3, 4, 1, 2) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix}$$
- ❸ Input is: x_1, x_2, x_3, x_4
- ❹ 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).

What are permutations?

- ① $\pi = (3, 4, 1, 2)$ is permutation in $P(v)$.
- ② Formally $\pi : \llbracket n \rrbracket \rightarrow \llbracket n \rrbracket$ is a one-to-one function.
 $\llbracket n \rrbracket = \{1, \dots, n\}$

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).

What are permutations?

- ① $\pi = (3, 4, 1, 2)$ is permutation in $P(v)$.
- ② Formally $\pi : \llbracket n \rrbracket \rightarrow \llbracket n \rrbracket$ is a one-to-one function.
 $\llbracket n \rrbracket = \{1, \dots, n\}$

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).

What are permutations?

- ① $\pi = (3, 4, 1, 2)$ is permutation in $P(v)$.
- ② Formally $\pi : \llbracket n \rrbracket \rightarrow \llbracket n \rrbracket$ is a one-to-one function.
 $\llbracket n \rrbracket = \{1, \dots, n\}$

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).

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

$$x_1 = \pi^{-1}(1) = 4 \dots x_i = \pi^{-1}(i) \dots$$

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
$$x_1 = \pi^{-1}(1) = 4 \dots x_i = \pi^{-1}(i) \dots$$

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
 $x_1 = \pi^{-1}(1) = 4 \dots x_i = \pi^{-1}(i) \dots$

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
$$x_1 = \pi^{-1}(1) = 4 \dots x_i = \pi^{-1}(i) \dots$$

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 T : the optimal decision tree.
- 6 $|P(r)| = n!$, where $r = \text{root}(T)$.

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 T : the optimal decision tree.
- 6 $|P(r)| = n!$, where $r = \text{root}(T)$.

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 T : the optimal decision tree.
- 6 $|P(r)| = n!$, where $r = \text{root}(T)$.

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 T : the optimal decision tree.
- 6 $|P(r)| = n!$, where $r = \text{root}(T)$.

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 T : the optimal decision tree.
- 6 $|P(r)| = n!$, where $r = \text{root}(T)$.

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 **T**: the optimal decision tree.
- 6 $|P(r)| = n!$, where $r = \text{root}(\mathbf{T})$.

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 \mathbf{T} : the optimal decision tree.
- 6 $|P(r)| = n!$, where $r = \text{root}(\mathbf{T})$.

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, \dots, 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, \dots, 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, \dots, 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, \dots, 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, \dots, 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, \dots, 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, \dots, 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, \dots, v_k : path taken by adversary.
- ⑧ Adversary input:
The input realizing the single permutation of $P(v_k)$.

Proof continued...

- ① Note, that

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

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

Proof continued...

- ① Note, that

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

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

Proof continued...

- ① Note, that


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

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

Proof continued...

- ① Note, that

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

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

Proof continued...

- ① Note, that


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

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

Proof continued...

- ① Note, that


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

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

Proof continued...

- ① Note, that


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

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

Proof continued...

- ① Note, that

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

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

Proof continued...

- ① Note, that

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

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

Part I

Uniqueness

Uniqueness

Problem

Given an input of n real numbers x_1, \dots, 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, \dots, 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, \dots, 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, \dots, 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, \dots, 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, \dots, 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.

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

Uniqueness lower bound

Proof similar but trickier.

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

Uniqueness lower bound

Proof similar but trickier.

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

Uniqueness lower bound

Proof similar but trickier.

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

Uniqueness lower bound

Proof similar but trickier.

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

Proof continued...

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

Proof continued...

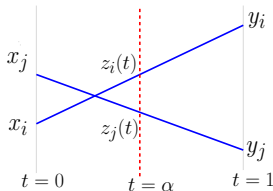
- 1 $Z(t) = (z_1(t), \dots, z_n(t))$ an input where $z_i(t) = tx_i + (1 - t)y_i$, for $t \in [0, 1]$.
- 2 $Z(0) = (x_1, \dots, x_n)$ and $Z(1) = (y_1, \dots, y_n)$.
- 3 Claim: $\forall t \in [0, 1]$ the input $Z(t)$ will arrive to the node v in T .

Proof continued...

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

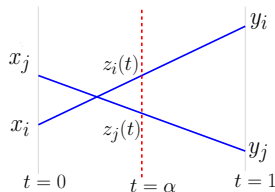
Proof of claim...

- 1 Assume false.
- 2 Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in \mathbf{T} .
- 3 Assume: compared the i th to j th input element, when paths diverted in \mathbf{T} .
- 4 I.e., Different path in \mathbf{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 \mathbf{T} .
- 7 Consider the functions $z_i(t)$ and $z_j(t)$:



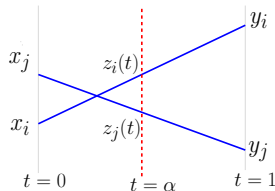
Proof of claim...

- 1 Assume false.
- 2 Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in \mathbf{T} .
- 3 Assume: compared the i th to j th input element, when paths diverted in \mathbf{T} .
- 4 I.e., Different path in \mathbf{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 \mathbf{T} .
- 7 Consider the functions $z_i(t)$ and $z_j(t)$:



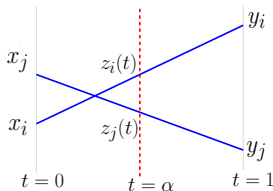
Proof of claim...

- 1 Assume false.
- 2 Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in \mathbf{T} .
- 3 Assume: compared the i th to j th input element, when paths diverted in \mathbf{T} .
- 4 I.e., Different path in \mathbf{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 \mathbf{T} .
- 7 Consider the functions $z_i(t)$ and $z_j(t)$:



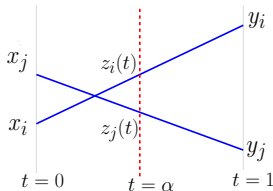
Proof of claim...

- 1 Assume false.
- 2 Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in \mathbf{T} .
- 3 Assume: compared the i th to j th input element, when paths diverted in \mathbf{T} .
- 4 I.e., Different path in \mathbf{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 \mathbf{T} .
- 7 Consider the functions $z_i(t)$ and $z_j(t)$:



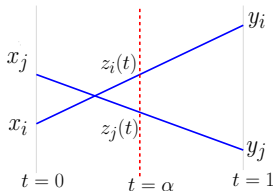
Proof of claim...

- 1 Assume false.
- 2 Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in \mathbf{T} .
- 3 Assume: compared the i th to j th input element, when paths diverted in \mathbf{T} .
- 4 I.e., Different path in \mathbf{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 \mathbf{T} .
- 7 Consider the functions $z_i(t)$ and $z_j(t)$:



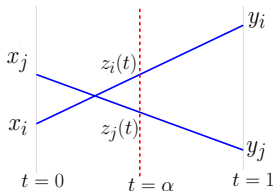
Proof of claim...

- 1 Assume false.
- 2 Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in \mathbf{T} .
- 3 Assume: compared the i th to j th input element, when paths diverted in \mathbf{T} .
- 4 I.e., Different path in \mathbf{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 \mathbf{T} .
- 7 Consider the functions $z_i(t)$ and $z_j(t)$:



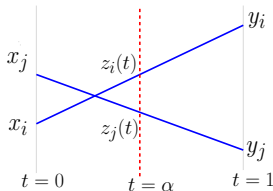
Proof of claim...

- 1 Assume false.
- 2 Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in \mathbf{T} .
- 3 Assume: compared the i th to j th input element, when paths diverted in \mathbf{T} .
- 4 I.e., Different path in \mathbf{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 \mathbf{T} .
- 7 Consider the functions $z_i(t)$ and $z_j(t)$:



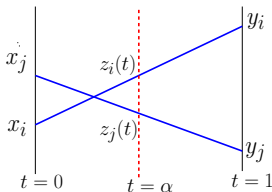
Proof of claim...

- 1 Assume false.
- 2 Assume for $t = \alpha \in [0, 1]$ the input $Z(t)$ did not get to v in \mathbf{T} .
- 3 Assume: compared the i th to j th input element, when paths diverted in \mathbf{T} .
- 4 I.e., Different path in \mathbf{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 \mathbf{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 \mathbf{T} .
- 3 Assume: compared the i th to j th input element, when paths diverted in \mathbf{T} .
- 4 I.e., Different path in \mathbf{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 \mathbf{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 $v \in T$. ■

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 $v \in T$. ■

Back to proof of Lemma...

① Recap:

- ① Recall: X, Y to different permutations that their distinct input arrives to the same node $v \in T$.
- ② Proved: $\forall t \in [0, 1]: Z(t) = (z_1(t), \dots, z_n(t))$ arrives to same node $v \in T$.

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 $v \in T$.
- ② Proved: $\forall t \in [0, 1]: Z(t) = (z_1(t), \dots, z_n(t))$ arrives to same node $v \in T$.

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 $v \in T$.
- ② Proved: $\forall t \in [0, 1]: Z(t) = (z_1(t), \dots, z_n(t))$ arrives to same node $v \in T$.

② However: There must be $\beta \in (0, 1)$ where $Z(\beta)$ has two numbers equal:

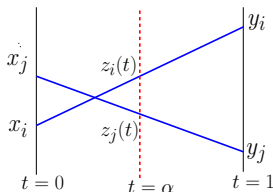
③ $Z(\beta)$: has a pair of numbers that are not unique.

Back to proof of Lemma...

① Recap:

- ① Recall: X, Y to different permutations that their distinct input arrives to the same node $v \in T$.
- ② Proved: $\forall t \in [0, 1]: Z(t) = (z_1(t), \dots, z_n(t))$ arrives to same node $v \in T$.

② However: There must be $\beta \in (0, 1)$ where $Z(\beta)$ has two numbers equal:



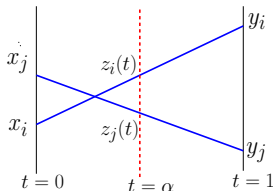
③ $Z(\beta)$: has a pair of numbers that are not unique.

Back to proof of Lemma...

① Recap:

- ① Recall: X, Y to different permutations that their distinct input arrives to the same node $v \in T$.
- ② Proved: $\forall t \in [0, 1]: Z(t) = (z_1(t), \dots, z_n(t))$ arrives to same node $v \in T$.

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.

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 . ■

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 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 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 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 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 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.

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.

Part II

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.

