Lower bounds

Lecture 22
November 12, 2015
22.1: 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???
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???
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???
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???
In the comparison model:

1. Algorithm only allowed to compare two elements.
2. \texttt{compare}(i, j): Compare \textit{i}th item in input to \textit{j}th item in input.

Q: How many calls to \texttt{compare} a deterministic sorting algorithm has to perform?
In the **comparison model**: 

1. Algorithm only allowed to compare two elements.
2. \texttt{compare}\((i, j)\): Compare \(i\)th item in input to \(j\)th item in input.

Q: # calls to \texttt{compare} a deterministic sorting algorithm has to perform?
In the **comparison model**:

1. Algorithm only allowed to compare two elements.
2. `\texttt{compare}(i, j)`: Compare $i$th item in input to $j$th item in input.

Q: # calls to `compare` a deterministic sorting algorithm has to perform?
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. 

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

- $a_1 < a_2$
- $a_2 < a_3$
- $a_1 < a_3$
- $a_1 < a_3 < a_2$
- $a_1 < a_2 < a_3$
- Similar
sorting algorithm outputs a permutation.

... order of the input elements so sorted.

Example: Input \( x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2 \).

Output: \( 1, 2, 3, 7, 19 \).

Output: \( x_3, x_5, x_2, x_1, x_4 \).

Output: \( \pi = (3, 5, 2, 1, 4) \)

Output as permutation:
\[
\pi(1) = 3, \pi(2) = 5, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4.
\]

**Interpretation**: \( x_{\pi(i)} \) is the \( i \)th smallest number in \( x_1, \ldots, x_n \).

\( v \): 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:
      
      $\pi(1) = 3, \pi(2) = 5, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4$.
   5. **Interpretation**: $x_{\pi(i)}$ is the $i$th smallest number in $x_1, \ldots, x_n$.

4. $v$: Node of decision tree.

5. $P(v)$: A set of all permutations compatible with the set of comparisons from root to $v$. 
sorting algorithm outputs a permutation.

... order of the input elements so sorted.

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:
   $\pi(1) = 3, \pi(2) = 5, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4$.

**Interpretation**: $x_{\pi(i)}$ is the $i$th smallest number in $x_1, \ldots, x_n$.

$v$: 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 $i$th 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 $i$th 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 $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
   - Output: $1, 2, 3, 7, 19$.
   - Output: $x_3, x_5, x_2, x_1, x_4$.
   - Output: $\pi = (3, 5, 2, 1, 4)$
   - 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 $i$th 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 $v$. 

---

v: Node of decision tree.
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 $i$th 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 outputs a permutation.

... order of the input elements so sorted.

Example: Input \( x_1 = 7, x_2 = 3, x_3 = 1, x_4 = 19, x_5 = 2 \).

Output: \( 1, 2, 3, 7, 19 \).

Output: \( x_3, x_5, x_2, x_1, x_4 \).

Output: \( \pi = (3, 5, 2, 1, 4) \).

Output as permutation:
\[
\pi(1) = 3, \pi(2) = 5, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4.
\]

**Interpretation:** \( x_{\pi(i)} \) is the \( i \)th smallest number in \( x_1, \ldots, x_n \).

\( v \): Node of decision tree.

\( P(v) \): A set of all permutations compatible with the set of comparisons from root to \( v \).
sorting algorithm outputs a permutation.

... order of the input elements so sorted.

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:
   \[
   \pi(1) = 3, \pi(2) = 5, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4.
   \]

Interpretation: \( x_{\pi(\cdot)} \) is the \( \cdot \)th smallest number in \( x_1, \ldots, x_n \).

\( \nu \): Node of decision tree.

\( P(\nu) \): A set of all permutations compatible with the set of comparisons from root to \( \nu \).
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).
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).
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. $\{1, \ldots, n\}$ 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).
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. 

   Sariel (UIUC)  
   New CS473  
   Fall 2015
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
   \[ x_1 = \pi^{-1}(1) = 4 \ldots x_i = \pi^{-1}(i) \ldots \]
Let \( \pi = (3, 4, 2, 1) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix} \)

Then the input \( \pi^{-1} = (3, 4, 1, 2) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{pmatrix} \)

... would generate this permutation.

Formally
\[
x_1 = \pi^{-1}(1) = 4 \ldots x_i = \pi^{-1}(i) \ldots
\]
Let $\pi = (3, 4, 2, 1) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix}$

Then the input $\pi^{-1} = (3, 4, 1, 2) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{pmatrix}$

... would generate this permutation.

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
   
   $x_1 = \pi^{-1}(1) = 4 \ldots x_i = \pi^{-1}(i) \ldots$
$v$: a node in decision tree.

If $|P(v)| > 1$: more than one permutation associated with it...

algorithm must continue performing comparisons

...otherwise, not know what to output...

Q: What is the worst running time of algorithm?

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...
   - algorithm must continue performing comparisons
   - ...otherwise, not know what to output...
3. Q: What is the worst running time of algorithm?
4. Answer: Longest path from root in the decision tree.
   ...because we count only comparisons!
\( v \): a node in decision tree.

2. If \(|P(v)| > 1\): more than one permutation associated with it...

3. The algorithm must continue performing comparisons...

...otherwise, not know what to output...

4. \textbf{Q:} What is the worst running time of algorithm?

5. \textbf{Answer:} Longest path from root in the decision tree.

...because we count only comparisons!
\( v \): 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!
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...

$\mathbf{v}$: a node in decision tree.

2. If $|P(\mathbf{v})| > 1$: more than one permutation associated with it...

algorithm must continue performing comparisons

...otherwise, not know what to output...

Q: What is the worst running time of algorithm?

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!
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 $r = \text{root}(T)$. 
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 $r = \text{root}(\mathcal{T})$. 
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. $T$: 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. $\mathcal{T}$: the optimal decision tree.
6. $|P(r)| = n!$, where $r = \text{root}(\mathcal{T})$. 
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 $r = \text{root}(\mathcal{T})$. 
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 $r = \text{root}(T)$. 
**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 $r = \text{root}(\mathcal{T})$. 
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 \( P(v_k) \).
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 \( P(v_k) \).
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. \(k \geq \log(n!) + 1 = \Omega(n \log n)\).

4. Depth of \(T\) is \(\Omega(n \log n)\).
Proof continued...

1. Note, that

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

2. \(2^{k-1} \geq |P(v_1)| = n!.\)

3. \(k \geq \lg(n!) + 1 = \Omega(n \log n).\)

4. Depth of \(T\) is \(\Omega(n \log n).\)
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 \( T \) is \( \Omega(n \log n) \).
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 \( T \) is \( \Omega(n \log n) \).
Note, that

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

1. \[ 2^{k-1} \geq |P(v_1)| = n!. \]
2. \[ k \geq \log(n!) + 1 = \Omega(n \log n). \]
3. Depth of \( T \) is \( \Omega(n \log n) \).
Proof continued...

Note, that

\[ 1 = |P(v_k)| \geq \frac{|P(v_{k-1})|}{2} \geq \cdots \geq \frac{|P(v_1)|}{2^{k-1}}. \]

\[ 2^{k-1} \geq |P(v_1)| = n!. \]

\[ k \geq \lg(n!) + 1 = \Omega(n \log n). \]

Depth of \( \mathcal{I} \) is \( \Omega(n \log n) \).
Note, that

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

\[ 2^{k-1} \geq |P(v_1)| = n!. \]

\[ k \geq \lg(n!) + 1 = \Omega(n \log n). \]

Depth of \( T \) is \( \Omega(n \log n) \).
Note, that

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

\[ 2^{k-1} \geq |P(v_1)| = n!. \]

\[ k \geq \log(n!) + 1 = \Omega(n \log n). \]

Depth of \( \mathcal{J} \) 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. \[ k \geq \log(n!) + 1 = \Omega(n \log n). \]

4. Depth of \( \mathcal{T} \) is \( \Omega(n \log n) \).

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

Sariel (UIUC)  
New CS473  
Fall 2015
Proof of claim...

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

\[
\begin{align*}
  x_i &< x_j \\
  y_i &> y_j
\end{align*}
\]
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 \( i \)th to \( j \)th 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 \( z_j(t) \):

\[
\begin{align*}
  x_i &< x_j & y_i > y_j &\text{or} & x_i > x_j & y_i < y_j \\
  z_i(t) & & & & z_j(t) &
\end{align*}
\]

In either case, \( X \) or \( Y \) will not arrive to \( v \) in \( \mathcal{T} \).
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 $i$th to $j$th 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 $\mathcal{I}$.
3. Assume: compared the $i$th to $j$th input element, when paths diverted in $\mathcal{I}$.
4. I.e., Different path in $\mathcal{I}$ 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 $\mathcal{I}$.
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 $\mathcal{I}$.
3. Assume: compared the $i$th to $j$th input element, when paths diverted in $\mathcal{I}$.
4. I.e., Different path in $\mathcal{I}$ 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 $\mathcal{I}$.
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 $i$th to $j$th 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 $T$.
3. Assume: compared the $i$th to $j$th input element, when paths diverted in $T$.
4. I.e., Different path in $T$ 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 $T$.
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 $T$.
3. Assume: compared the $i$th to $j$th input element, when paths diverted in $T$.
4. I.e., Different path in $T$ 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 $T$.
7. Consider the functions $z_i(t)$ and $z_j(t)$:

![Diagram showing the functions $z_i(t)$ and $z_j(t)$ with $x_i$, $x_j$, $y_i$, $y_j$, and $t = 0$, $t = \alpha$, $t = 1$.]
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 $v \in T$. ■
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 \(v \in \mathcal{T}\).
3. However: There must be \(\beta \in (0, 1)\) where \(Z(\beta)\) has two numbers equal:

\[ Z(\beta) \] has a pair of numbers that are not unique.
Recap:

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

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

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

\(Z(\beta)\): has a pair of numbers that are not unique.
Recall: \( X, Y \) to different permutations that their distinct input arrives to the same node \( v \in \mathcal{I} \).

Proved: \( \forall t \in [0, 1]: Z(t) = (z_1(t), \ldots, z_n(t)) \) arrives to same node \( v \in \mathcal{I} \).

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

\[ Z(\beta) : \text{has a pair of numbers that are not unique.} \]
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 $v \in \mathcal{T}$.

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

\[ Z(\beta): \text{has a pair of numbers that are not unique.} \]
Proof of Lemma continued...

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

Done: Found inputs $Z(0)$ and $Z(\beta)$ such that one is unique and the other is not.

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

... both arrive to $v$. 

Proof of Lemma continued...
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 \( v \).
Done: Found inputs $Z(0)$ and $Z(\beta)$ such that one is unique and the other is not.

... 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.
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
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 $a + b = c$.

One can show...

**Lemma**

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

**Proof.**

Exercise...
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 $a + b = c$.

One can show...

**Lemma**

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

**Proof.**

Exercise...
Somewhat surprisingly, no better solution is known.

Open Problem: Find a subquadratic algorithm for $3\text{SUM}$.

It is widely believed that no such algorithm exists.

There is a large collection of problems that are $3\text{SUM}$-Hard: if you solve them in subquadratic time, then you can solve $3\text{SUM}$ in subquadratic time.
Somewhat surprisingly, no better solution is known.

Open Problem: Find a subquadratic algorithm for 3SUM.

It is widely believed that no such algorithm exists.

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.
Somewhat surprisingly, no better solution is known.

Open Problem: Find a subquadratic algorithm for 3SUM.

It is widely believed that no such algorithm exists.

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.
Somewhat surprisingly, no better solution is known.

Open Problem: Find a subquadratic algorithm for 3SUM.

It is widely believed that no such algorithm exists.

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

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 $Q$?

So, how does one prove that a problem is 3SUM hard?

Reductions.

Reductions must have subquadratic running time.

The details are interesting, but are omitted.
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 $Q$?

So, how does one prove that a problem is 3SUM hard?

Reductions.

Reductions must have subquadratic running time.

The details are interesting, but are omitted.
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 $Q$?

So, how does one prove that a problem is 3SUM hard?

- Reductions.
- Reductions must have subquadratic running time.
- The details are interesting, but are omitted.
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 \( Q \)?

So, how does one prove that a problem is 3SUM hard?

- Reductions.
- Reductions must have subquadratic running time.
- The details are interesting, but are omitted.
3SUM-hard problems

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 \( Q \)?

So, how does one prove that a problem is 3SUM hard?

Reductions.

Reductions must have subquadratic running time.
The details are interesting, but are omitted.
3SUM-hard problems

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

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 $Q$?

So, how does one prove that a problem is 3SUM hard?

Reductions.

Reductions must have subquadratic running time.

The details are interesting, but are omitted.