NEW CS 473: Theory II, Fall 2015

Lower bounds

Lecture 22 November 12, 2015

22.1: Sorting

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

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

- lacksquare 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.
- Rules of engagement: What can an algorithm do????

- lacksquare 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.
- Rules of engagement: What can an algorithm do????

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

- In the comparison model:
 - Algorithm only allowed to compare two elements.
- Q: # calls to compare a deterministic sorting algorithm has to perform?

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

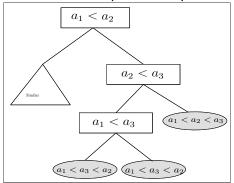
- In the comparison model:
 - Algorithm only allowed to compare two elements.
 - **2** compare (i, j): Compare ith item in input to jth item in input.
- Q: # calls to compare a deterministic sorting algorithm has to perform?

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

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

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

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



- sorting algorithm outputs a permutation.
- Order of the input elements so sorted.
- - Output: 1, 2, 3, 7, 19.
 - ② Output: x_3, x_5, x_2, x_1, x_4 .
 - **3** Output: $\pi = (3, 5, 2, 1, 4)$
 - 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.
- On 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.
 - ② Output: x_3, x_5, x_2, x_1, x_4 .
 - **3** 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.
- **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: $\pi = (3, 5, 2, 1, 4)$
 - Output as permutation:

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

- **Interpretation**: $x_{\pi(i)}$ is the ith smallest number in x_1, \ldots, x_n .
- $oldsymbol{v}$: 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.
- **3** Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
 - Output: 1, 2, 3, 7, 19.
 - ② Output: x_3, x_5, x_2, x_1, x_4 .
 - **3** Output: $\pi = (3, 5, 2, 1, 4)$
 - Output as permutation:

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

- **Interpretation**: $x_{\pi(i)}$ is the ith smallest number in x_1, \ldots, x_n .
- **1** v: Node of decision tree. P(v): A set of all permutations con
 - 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.
- **3** Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
 - Output: 1, 2, 3, 7, 19.
 - **2** Output: x_3, x_5, x_2, x_1, x_4 .
 - **3** Output: $\pi = (3, 5, 2, 1, 4)$
 - Output as permutation:

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

- **Interpretation**: $x_{\pi(i)}$ is the ith smallest number in x_1, \ldots, x_n .
- $v \colon \mathsf{Node} \ \mathsf{of} \ \mathsf{decision} \ \mathsf{tree}.$
 - 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.
- **3** Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
 - \bullet Output: 1, 2, 3, 7, 19.
 - **2** Output: x_3, x_5, x_2, x_1, x_4 .
 - **3** Output: $\pi = (3, 5, 2, 1, 4)$
 - Output as permutation:

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

- **Interpretation**: $x_{\pi(i)}$ is the ith smallest number in x_1, \ldots, x_n .
- $v \colon \mathsf{Node} \ \mathsf{of} \ \mathsf{decision} \ \mathsf{tree}.$
 - 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.
- **3** Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
 - \bullet Output: 1, 2, 3, 7, 19.
 - **2** Output: x_3, x_5, x_2, x_1, x_4 .
 - **3** Output: $\pi = (3, 5, 2, 1, 4)$
 - Output as permutation:

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

- **Interpretation**: $x_{\pi(i)}$ is the ith 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.
- **3** Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
 - \bullet Output: 1, 2, 3, 7, 19.
 - **2** Output: x_3, x_5, x_2, x_1, x_4 .
 - **3** Output: $\pi = (3, 5, 2, 1, 4)$
 - Output as permutation:

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

- **Interpretation**: $x_{\pi(i)}$ is the ith 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.
- **3** Example: Input $x_1 = 7, x_2 = 3, X_3 = 1, x_4 = 19, x_5 = 2$.
 - Output: 1, 2, 3, 7, 19.
 - ② Output: x_3, x_5, x_2, x_1, x_4 .
 - **3** Output: $\pi = (3, 5, 2, 1, 4)$
 - 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.

- \bullet $\pi = (3,4,1,2)$ is permutation in P(v).
- Formally $\pi : [n] \to [n]$ is a one-to-one function. $\pi = (3,4,1,2) = \left(\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{array} \right)$
- ullet If arrived to v and $\pi \in P(v)$ then

- \bullet $\pi=(3,4,1,2)$ is permutation in P(v).
- $\begin{array}{l} \textbf{@} \ \, \text{Formally} \ \pi: \llbracket n \rrbracket \to \llbracket n \rrbracket \ \text{is a one-to-one function.} \\ \llbracket n \rrbracket = \{1, \ldots, n\} \\ \text{can be written as:} \\ \pi = (3,4,1,2) = \left(\begin{array}{ccc} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{array} \right) \\ \end{array}$
- Input is: x_1, x_2, x_3, x_4
- ① 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).

- \bullet $\pi=(3,4,1,2)$ is permutation in P(v).
- $\begin{array}{l} \textbf{@} \ \, \text{Formally} \, \pi: \llbracket n \rrbracket \to \llbracket n \rrbracket \, \text{ is a one-to-one function.} \\ \llbracket n \rrbracket = \{1, \ldots, n\} \\ \text{can be written as:} \\ \pi = (3,4,1,2) = \left(\begin{array}{ccc} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{array} \right) \\ \end{array}$
- **3** Input is: x_1, x_2, x_3, x_4
- ① 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).

- \bullet $\pi=(3,4,1,2)$ is permutation in P(v).
- $lacksquare ext{Formally } \pi: \llbracket n
 bracket o \llbracket n
 bracket ext{is a one-to-one function.} \ \llbracket n
 bracket = \{1,\ldots,n\} \ ext{can be written as:} \ lacksquare 1 2 3 4 \ lacksquare$

$$\pi=(3,4,1,2)=\left(egin{array}{cccc} 1 & 2 & 3 & 4 \ 3 & 4 & 1 & 2 \end{array}
ight)$$

- lacksquare Input is: x_1, x_2, x_3, x_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).

- \bullet $\pi=(3,4,1,2)$ is permutation in P(v).
- $lacksquare ext{Formally } \pi: \llbracket n
 bracket o \llbracket n
 bracket ext{is a one-to-one function.} \ \llbracket n
 bracket = \{1,\ldots,n\} \ ext{can be written as:} \ egin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}$

$$\pi=(3,4,1,2)=\left(egin{array}{cccc}1&2&3&4\3&4&1&2\end{array}
ight)$$

- **3** Input is: x_1, x_2, x_3, x_4
- $oldsymbol{0}$ 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).

1 Let
$$\pi = (3,4,2,1) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix}$$

- ② Then the input $\pi^{-1}=(3,4,1,2)=\left(egin{array}{ccc}1&2&3&4\\4&3&1&2\end{array}
 ight)$
- ... would generate this permutation.
- $oldsymbol{0}$ Formally $x_1=\pi^{-1}(1)=4\,\ldots\,x_i=\pi^{-1}(i)\,\ldots$

1 Let
$$\pi = (3,4,2,1) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix}$$

- **3** Then the input $\pi^{-1} = (3,4,1,2) = \left(egin{array}{ccc} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{array} \right)$
- ... would generate this permutation
- $oldsymbol{\circ}$ Formally $x_1=\pi^{-1}(1)=4\ldots x_i=\pi^{-1}(i)\ldots$

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

• Let
$$\pi = (3,4,2,1) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix}$$

- **2** 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.
- $oldsymbol{\circ}$ Formally $x_1=\pi^{-1}(1)=4\,\ldots\,x_i=\pi^{-1}(i)\,\ldots$

- $\mathbf{0}$ \mathbf{v} : a node in decision tree.
- $ext{@}$ 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!

- $\mathbf{0}$ \mathbf{v} : a node in decision tree.
- ullet 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!

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

Sariel (UIUC) New CS473 9 Fall 2015 9 / 23

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...
- 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...
- Q: What is the worst running time of algorithm?
- 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.

- $exttt{ iny Algorithm}$ in the comparison model \equiv a decision tree.
- ② Use an adversary argument.
- Adversary pick the worse possible input for the algorithm.
- Input is a permutation.
- 5 T: the optimal decision tree.

Lemma

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

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

Lemma

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

- **1** Algorithm in the comparison model \equiv a decision tree.
- Use an adversary argument.
- Adversary pick the worse possible input for the algorithm.
- Input is a permutation.
- |P(r)| = n!, where $r = \operatorname{root}(\mathfrak{I}).$

Lemma

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

- **1** Algorithm in the comparison model \equiv a decision tree.
- Use an adversary argument.
- Adversary pick the worse possible input for the algorithm.
- Input is a permutation.
- 5 T: the optimal decision tree.

Lemma

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

- **1** Algorithm in the comparison model \equiv a decision tree.
- Use an adversary argument.
- Adversary pick the worse possible input for the algorithm.
- Input is a permutation.
- 5 T: the optimal decision tree.

Lemma

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

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

Lemma

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

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

- $\mathbf{0}$ \mathbf{u} , \mathbf{v} : children of \mathbf{r} .
- ② Adversary: no commitment on which of the permutations of P(r) it is using.
- Algorithm perform compares x_i to x_j in root...
- Adversary computes P(u) and P(v)[Adversary has infinite computation power!]
- ullet Adversary goes to u if $|P(u)| \geq |P(v)|$, and to v otherwise.
- Adversary traversal: always pick child with more permutations.
- $m{v}_1,\ldots,m{v}_k$: path taken by adversary.
- $^{\circ}$ Adversary input: The input realizing the single permutation of $P(v_k)$

- $\mathbf{0}$ \mathbf{u} , \mathbf{v} : children of \mathbf{r} .
- ② Adversary: no commitment on which of the permutations of P(r) it is using.
- $ext{ } ext{ } ext$
- Adversary computes P(u) and P(v)[Adversary has infinite computation power!]
- ullet Adversary goes to u if $|P(u)| \geq |P(v)|$, and to v otherwise.
- O Adversary traversal: always pick child with more permutations.
- v_1, \ldots, v_k : path taken by adversary.
- $^{\circ}$ Adversary input: The input realizing the single permutation of $P(v_k)$

- $\mathbf{0}$ \mathbf{u} , \mathbf{v} : children of \mathbf{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...
- Adversary computes P(u) and P(v)[Adversary has infinite computation power!]
- ullet Adversary goes to u if $|P(u)| \geq |P(v)|$, and to v otherwise.
- Adversary traversal: always pick child with more permutations.
- $oldsymbol{v}_1,\ldots,oldsymbol{v}_k$: path taken by adversary.
- $^{\odot}$ Adversary input: The input realizing the single permutation of $P(v_k)$

- $\mathbf{0}$ \mathbf{u} , \mathbf{v} : children of \mathbf{r} .
- ② Adversary: no commitment on which of the permutations of P(r) it is using.
- lacktriangledown Algorithm perform compares x_i to x_j in root...
- lacktriangledown Adversary computes $m{P}(u)$ and $m{P}(v)$ [Adversary has infinite computation power!]
- $exttt{ iny Adversary goes to } u$ if $|P(u)| \geq |P(v)|$, and to v otherwise.
- Adversary traversal: always pick child with more permutations.
- $m{0} \ v_1, \ldots, v_k$: path taken by adversary.
- Our Adversary input:
 The input realizing the single permutation of $P(v_k)$

- $\mathbf{0}$ \mathbf{u} , \mathbf{v} : children of \mathbf{r} .
- ② Adversary: no commitment on which of the permutations of $m{P(r)}$ it is using.
- **3** Algorithm perform compares x_i to x_j in root...
- 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.
- Adversary traversal: always pick child with more permutations.
- $oldsymbol{v}_1,\ldots,oldsymbol{v}_k$: path taken by adversary.
- $^{\odot}$ Adversary input: The input realizing the single permutation of $P(v_k)$

- $\mathbf{0}$ \mathbf{u} , \mathbf{v} : children of \mathbf{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...
- Adversary computes P(u) and P(v)[Adversary has infinite computation power!]
- **3** Adversary goes to u if $|P(u)| \geq |P(v)|$, and to v otherwise.
- Adversary traversal: always pick child with more permutations.
- $m{0} \ v_1, \ldots, v_k$: path taken by adversary.
- ullet Adversary input: The input realizing the single permutation of $P(v_k)$

- $\mathbf{0}$ \mathbf{u} , \mathbf{v} : children of \mathbf{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...
- lacktriangledown Adversary computes $m{P}(u)$ and $m{P}(v)$ [Adversary has infinite computation power!]
- lacktriangledown Adversary goes to u if $|P(u)| \geq |P(v)|$, and to v otherwise.
- Adversary traversal: always pick child with more permutations.
- v_1, \ldots, v_k : path taken by adversary.
- $^{\odot}$ Adversary input: The input realizing the single permutation of $P(v_k)$

- $\mathbf{0}$ \mathbf{u} , \mathbf{v} : children of \mathbf{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...
- lacktriangledown Adversary computes $m{P}(u)$ and $m{P}(v)$ [Adversary has infinite computation power!]
- lacktriangledown Adversary goes to u if $|P(u)| \geq |P(v)|$, and to v otherwise.
- Adversary traversal: always pick child with more permutations.
- v_1, \ldots, v_k : path taken by adversary.
- Adversary input:
 The input realizing the single permutation of $P(v_k)$.

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

- $2^{k-1} \ge |P(v_1)| = n!.$
- Depth of \mathfrak{T} is $\Omega(n \log n)$.



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

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



Note, that

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

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

New CS473 Fall 2015 12 / 23

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

- $2^{k-1} \ge |P(v_1)| = n!.$
- Depth of \mathfrak{T} is $\Omega(n \log n)$.

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

- $2^{k-1} \ge |P(v_1)| = n!.$
- Depth of \mathfrak{T} is $\Omega(n \log n)$.

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

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

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

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

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

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

Note, that

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

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

New CS473 Fall 2015 12 / 23

22.2: Uniqueness

$22.2.1: \ \mathsf{Uniqueness}$

Problem

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

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

Theorem

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

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!
- ...but in a strange computation model.
- Surprisingly...

Theorem

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

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!
- ...but in a strange computation model.
- Surprisingly...

Theorem

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

Problem

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

- Intuitively: easier than sorting.
- Can be solved in linear time!
- ...but in a strange computation model.
- Surprisingly...

Theorem

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

Problem

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

- Intuitively: easier than sorting.
- Can be solved in linear time!
- ...but in a strange computation model.
- Surprisingly...

Theorem

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

Problem

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

- Intuitively: easier than sorting.
- Can be solved in linear time!
- ...but in a strange computation model.
- Surprisingly...

Theorem

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

Uniqueness lower bound

Proof similar but trickier.

T: decision tree (every node has three children).

Lemma

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

- ① σ , σ' : any two different permutations in P(v).
- $X = x_1, \ldots, x_n$ be an input realizing σ .
- $Y = y_1, \ldots, y_n$: input realizing σ' .
- $lacksymbol{\circ}$ 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 similar but trickier.

T: decision tree (every node has three children).

Lemma

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

- $oldsymbol{0}$ σ , σ' : any two different permutations in P(v).
- ② $X=x_1,\ldots,x_n$ be an input realizing σ .
- $Y = y_1, \ldots, y_n$: input realizing σ' .
- $lacksymbol{\circ}$ 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 similar but trickier.

T: decision tree (every node has three children).

Lemma

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

- **①** σ , σ' : any two different permutations in P(v).
- $X = x_1, \ldots, x_n$ be an input realizing σ .
- $Y = y_1, \ldots, y_n$: input realizing σ' .
- ① 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 similar but trickier.

T: decision tree (every node has three children).

Lemma

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

- $oldsymbol{0}$ σ , σ' : any two different permutations in P(v).
- $X = x_1, \ldots, x_n$ be an input realizing σ .
- $lackbr{0} Y = y_1, \ldots, y_n$: input realizing σ' .
- $lacksymbol{\circ}$ 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 similar but trickier.

T: decision tree (every node has three children).

Lemma

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

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

Proof continued...

- $oldsymbol{0} Z(t) = (z_1(t), \ldots, z_n(t))$ 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 $Z(1) = (y_1, \ldots, y_n)$.
- ullet Claim: $orall t \in [0,1]$ the input Z(t) will arrive to the node v in ${\mathfrak T}$.

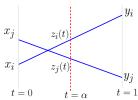
Proof continued...

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

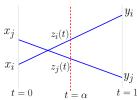
Proof continued...

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

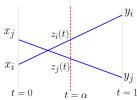
- Assume false.
- ② Assume for $t=lpha\in[0,1]$ the input Z(t) did not get to v in $\mathfrak{T}.$
- \odot Assume: compared the *i*th to *j*th input element, when paths diverted in \Im .
- lacksquare I.e., Different path in $oldsymbol{\Im}$ then the one for X and Y
- lacksquare Claim: $x_i < x_j$ and $y_i > y_j$ or $x_i > x_j$ and $y_i < y_j$.
- \bullet In either case X or Y will not arrive to v in \mathfrak{T} .
- O Consider the functions $z_i(t)$ and $z_j(t)$:



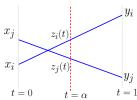
- Assume false.
- ② Assume for $t=lpha\in[0,1]$ the input Z(t) did not get to v in $\mathfrak{T}.$
- **a** Assume: compared the ith to jth input element, when paths diverted in \mathfrak{T} .
- lacksquare I.e., Different path in $oldsymbol{\Im}$ then the one for $oldsymbol{X}$ and $oldsymbol{Y}$.
- lacksquare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



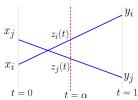
- Assume false.
- ② Assume for $t=lpha\in[0,1]$ the input Z(t) did not get to v in $\mathfrak{T}.$
- **a** Assume: compared the ith to jth input element, when paths diverted in \mathfrak{T} .
- lacksquare I.e., Different path in $oldsymbol{\Im}$ then the one for $oldsymbol{X}$ and $oldsymbol{Y}$.
- lacksquare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



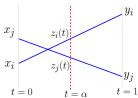
- Assume false.
- ② Assume for $t=lpha\in[0,1]$ the input Z(t) did not get to v in \Im .
- 3 Assume: compared the ith to jth input element, when paths diverted in \mathfrak{T} .
- lacksquare I.e., Different path in $\mathfrak T$ then the one for X and Y.
- lacksquare Claim: $x_i < x_j$ and $y_i > y_j$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



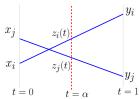
- Assume false.
- ② Assume for $t=lpha\in[0,1]$ the input Z(t) did not get to v in $\mathfrak{T}.$
- **3** Assume: compared the ith to jth input element, when paths diverted in \mathfrak{T} .
- ${ t @}$ l.e., Different path in ${ t T}$ then the one for X and Y.
- ullet Claim: $x_i < x_j$ and $y_i > y_j$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



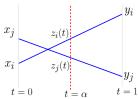
- Assume false.
- ② Assume for $t=lpha\in[0,1]$ the input Z(t) did not get to v in $\mathfrak{T}.$
- **3** Assume: compared the ith to jth input element, when paths diverted in \mathfrak{T} .
- lacktriangle I.e., Different path in ${\mathfrak T}$ then the one for $oldsymbol{X}$ and $oldsymbol{Y}$.
-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



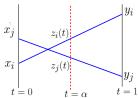
- Assume false.
- ② Assume for $t=lpha\in[0,1]$ the input Z(t) did not get to v in ${\mathfrak T}.$
- **3** Assume: compared the ith to jth input element, when paths diverted in \mathfrak{T} .
- lacktriangle I.e., Different path in $oldsymbol{\mathfrak{T}}$ then the one for $oldsymbol{X}$ and $oldsymbol{Y}$.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



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



- Assume false.
- ② Assume for $t=lpha\in[0,1]$ the input Z(t) did not get to v in $\mathfrak{T}.$
- **3** Assume: compared the ith to jth input element, when paths diverted in \mathfrak{T} .
- lacktriangle I.e., Different path in $oldsymbol{\mathfrak{T}}$ then the one for $oldsymbol{X}$ and $oldsymbol{Y}$.
- lacksquare Claim: $x_i < x_j$ and $y_i > y_j$ or $x_i > x_j$ and $y_i < y_j$.
- **1** In either case X or Y will not arrive to v in \mathfrak{T} .
- **O** 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 $v \in \mathfrak{I}$.

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 $v \in \mathfrak{T}$.

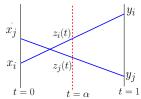
Recap:

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

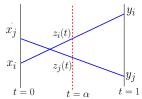
- Recap:
 - Recall: X, Y to different permutations that their distinct input arrives to the same node $v \in \mathcal{T}$.
 - ② Proved: $\forall t \in [0,1]$: $Z(t) = (z_1(t), \dots, z_n(t))$ arrives to same node $v \in \mathcal{T}$.
- Mowever: There must be $\beta \in (0,1)$ where $Z(\beta)$ has two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

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

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



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



- ① Done: Found inputs Z(0) and Z(eta)
- such that one is unique and the other is not.
- $ext{ } ext{ } ext$

- lacksquare Done: Found inputs Z(0) and Z(eta)
- Such that one is unique and the other is not.
- \odot ... both arrive to $oldsymbol{v}$

- **1** Done: Found inputs Z(0) and $Z(\beta)$
- such that one is unique and the other is not.
- \odot ... both arrive to $oldsymbol{v}$

- **1** Done: Found inputs Z(0) and $Z(\beta)$
- such that one is unique and the other is not.
- ullet ... both arrive to $oldsymbol{v}$.

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

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

Theorem

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

Theorem

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

Theorem

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

Theorem

- Apply the same argument as before.
- If in the decision tree, the adversary arrived to a node...
- Ocontaining 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 \mathfrak{T} of length $\Omega(n \log n)$.
- We conclude:

Theorem

22.2.2: Algebraic tree model

Algebraic tree model

- 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).
- 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).

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

Algebraic tree model

- 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).
- 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).

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

Algebraic tree model

- 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).
- 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).

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

Algebraic tree model

- 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).
- 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).

3 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

Sariel (UIUC) New CS473 25 Fall 2015 25 / 23

22.3.1: 3Sum-Hard

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..

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...

- Somewhat surprisingly, no better solution is known.
- ② 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.

- 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.

- 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.

- Those problems include:
 - For n points in the plane, is there three points that lie on the same line.
 - f 2 Given a set of m n triangles in the plane, do they cover the unit square
 - 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:
 - For n points in the plane, is there three points that lie on the same line.
 - $oldsymbol{0}$ Given a set of n triangles in the plane, do they cover the unit square
 - 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:
 - For n points in the plane, is there three points that lie on the same line.
 - f 2 Given a set of n triangles in the plane, do they cover the unit square
 - $footnote{\circ}$ 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:
 - For n points in the plane, is there three points that lie on the same line.
 - Given a set of n triangles in the plane, do they cover the unit square
 - ullet 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:
 - For n points in the plane, is there three points that lie on the same line.
 - Given a set of n triangles in the plane, do they cover the unit square
 - 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:
 - For n points in the plane, is there three points that lie on the same line.
 - Given a set of n triangles in the plane, do they cover the unit square
 - ullet 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:
 - For n points in the plane, is there three points that lie on the same line.
 - f 2 Given a set of n triangles in the plane, do they cover the unit square
 - 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.

Sariel (UIUC) New CS473 30 Fall 2015 30 / 23

Sariel (UIUC) New CS473 31 Fall 2015 31 / 23

Sariel (UIUC) New CS473 32 Fall 2015 32 / 23

Sariel (UIUC) New CS473 33 Fall 2015 33 / 23