## CS 573: Algorithms, Fall 2014

# Lower bounds

Lecture 26 December 2, 2014

- $\bullet$  *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????

- 0 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????

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

CS573 Fall 2014 2 / 25

- 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.
- 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?

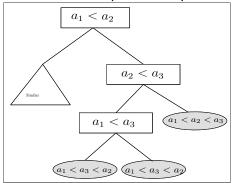
- In the comparison model:
  - Algorithm only allowed to compare two elements.
  - **2** compare (i, j): Compare ith item in input to jth item in input.
- Q: # calls to compare a deterministic sorting algorithm has to perform?

- 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.
- 2 Each stage: has current collection of comparisons done.
- ... need to decide which comparison to perform next.



- sorting algorithm outputs a permutation.
- - Output: 1, 2, 3, 7, 19.

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

CS573 Fall 2014 5 / 25

- sorting algorithm outputs a permutation.
- Order of the input elements so sorted.
- - 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:

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

- 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:

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

- 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:

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

- 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.
- 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.
  - **2** Output:  $x_3, x_5, x_2, x_1, x_4$ .
  - **3** Output:  $\pi = (3, 5, 2, 1, 4)$
  - Output as permutation:

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

- 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:

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

- 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.
- **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:

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

- 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:

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

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

CS573 6 Fall 2014 6 / 25

- $oldsymbol{0}$   $\pi=(3,4,1,2)$  is permutation in P(v).
- $\begin{array}{l} \textbf{@} \ \, \text{Formally} \, \pi: \llbracket n \rrbracket \to \llbracket n \rrbracket \, \text{ is a one-to-one function.} \\ \llbracket n \rrbracket = \{1, \ldots, n\} \\ \text{can be written as:} \\ \pi = (3,4,1,2) = \left( \begin{array}{ccc} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{array} \right) \\ \end{array}$
- $\bigcirc$  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).
- ② Formally  $\pi : [n] \to [n]$  is a one-to-one function.  $\llbracket n \rrbracket = \{1, \ldots, n\}$ can be written as:  $\pi = (3,4,1,2) = \left( egin{array}{cccc} 1 & 2 & 3 & 4 \ 3 & 4 & 1 & 2 \end{array} 
  ight)$
- **3** Input is:  $x_1, x_2, x_3, x_4$
- ullet If arrived to v and  $\pi \in P(v)$  then

Sariel (UIUC) CS573 6 Fall 2014 6 / 25

- $\bullet$   $\pi=(3,4,1,2)$  is permutation in P(v).

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

- **3** Input is:  $x_1, x_2, x_3, x_4$
- 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.
- lacksquare 9 Formally  $x_1=\pi^{-1}(1)=4\ldots\,x_i=\pi^{-1}(i)\ldots$

- **1** Let  $\pi = (3,4,2,1) = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix}$
- **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.
- $egin{aligned} egin{aligned} & \mathsf{Formally} \ x_1 = \pi^{-1}(1) = 4 \, \ldots \, x_i = \pi^{-1}(i) \, \ldots \end{aligned}$

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

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

- **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!

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

- 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!

- 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 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.

### Proof

- ullet 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.
- T: the optimal decision tree.

#### Lemma

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

### Proof

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

#### 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.
- Use an adversary argument.
- Adversary pick the worse possible input for the algorithm.
- Input is a permutation.
- T: the optimal decision tree.
- $|P(r)| = n!, ext{ where } r = ext{root}(\mathsf{T}).$

#### Lemma

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

### Proof

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

#### 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.
- 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  $r = \text{root}(\mathsf{T}).$

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

#### 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.
- Use an adversary argument.
- Adversary pick the worse possible input for the algorithm.
- Input is a permutation.
- T: the optimal decision tree.
- |P(r)| = n!, where  $r = \text{root}(\mathsf{T}).$

- $\mathbf{0}$   $\mathbf{u}$ ,  $\mathbf{v}$ : children of  $\mathbf{r}$ .
- ② Adversary: no commitment on which of the permutations of P(r) it is using.
- ullet 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.
- $v_1,\ldots,v_k$ : path taken by adversary.
- 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.
- ullet 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.
- $v_1, \ldots, v_k$ : path taken by adversary.
- ${ t 0}$  Adversary input: The input realizing the single permutation of  ${m P}(v_k)$

- $\mathbf{0}$   $\mathbf{u}$ ,  $\mathbf{v}$ : 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.
- $v_1,\ldots,v_k$ : path taken by adversary.
- ${ t 0}$  Adversary input: The input realizing the single permutation of  ${m P}(v_k)$

- $\mathbf{0}$   $\mathbf{u}$ ,  $\mathbf{v}$ : 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.
- $v_1,\ldots,v_k$ : path taken by adversary.
- ${ t @}$  Adversary input: The input realizing the single permutation of  ${m P}(v_k)$

- $\mathbf{0}$   $\mathbf{u}$ ,  $\mathbf{v}$ : 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!]
- **5** 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  $P(v_k)$

- $\mathbf{0}$   $\mathbf{u}$ ,  $\mathbf{v}$ : 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!]
- **5** 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.
- ${ t @}$  Adversary input: The input realizing the single permutation of  ${m P}(v_k)$

- $\mathbf{0}$   $\mathbf{u}$ ,  $\mathbf{v}$ : 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!]
- **5** 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  $P(v_k)$ .

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

Note, that

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

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

Note, that

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

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

Note, that

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

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

Note, that

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

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

Note, that

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

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

Note, that

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

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

Note, that

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

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

Note, that

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

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

Note, that

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

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

# Part I

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.
- 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.

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

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

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

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

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

## Proof continued...

- $oldsymbol{0} Z(t) = (z_1(t), \ldots, z_n(t))$  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  ${\sf T}.$

Sariel (UIUC) CS573 15 Fall 2014 15 / 25

## Proof continued...

- $z_i(t) = tx_i + (1-t)y_i$ , for  $t \in [0,1]$ .
- $Z(0) = (x_1, \ldots, x_n)$  and  $Z(1) = (y_1, \ldots, y_n)$ .
- Olaim:  $\forall t \in [0,1]$  the input Z(t) will arrive to the node v in

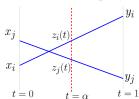
CS573 Fall 2014 15 / 25

## Proof continued...

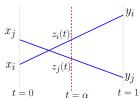
- $oldsymbol{Z}(t)=(z_1(t),\ldots,z_n(t))$  an input where  $z_i(t)=tx_i+(1-t)y_i$ , for  $t\in[0,1]$ .
- **3** Claim:  $\forall t \in [0,1]$  the input Z(t) will arrive to the node v in  $\mathsf{T}.$

Sariel (UIUC) CS573 15 Fall 2014 15 / 25

- Assume false
- ② Assume for  $t=lpha\in[0,1]$  the input Z(t) did not get to v in  $\mathsf{T}.$
- Assume: compared the ith to jth input element, when paths diverted in T.
- ① I.e., Different path in  ${f T}$  then the one for  ${f X}$  and  ${f Y}$ .
- 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 T.
- O Consider the functions  $z_i(t)$  and  $z_j(t)$ :

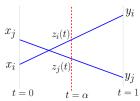


- Assume false.
- ② Assume for  $t=lpha\in[0,1]$  the input Z(t) did not get to v in  $\mathsf{T}.$
- Assume: compared the ith to jth input element, when paths diverted in T.
- lacksquare I.e., Different path in lacksquare then the one for  $oldsymbol{X}$  and  $oldsymbol{Y}$ .
- lacksquare Claim:  $x_i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



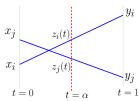
Sariel (UIUC) CS573 16 Fall 2014 16 / 25

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

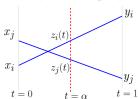


Sariel (UIUC) CS573 16 Fall 2014 16 / 25

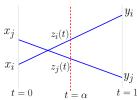
- Assume false.
- ② Assume for  $t=lpha\in[0,1]$  the input Z(t) did not get to v in  ${\sf T}.$
- Assume: compared the ith to jth input element, when paths diverted in T.
- lacksquare I.e., Different path in lacksquare then the one for X and Y.
- lacksquare Claim:  $x_i < x_j$  and  $y_i > y_j$  or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



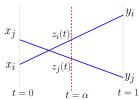
- Assume false.
- ② Assume for  $t=lpha\in[0,1]$  the input Z(t) did not get to v in  ${\sf T}.$
- **3** Assume: compared the ith to jth input element, when paths diverted in T.
- lacktriangle I.e., Different path in lacktriangle then the one for  $oldsymbol{X}$  and  $oldsymbol{Y}$ .
- $exttt{ iny Claim:}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



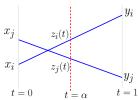
- Assume false.
- ② Assume for  $t=lpha\in[0,1]$  the input Z(t) did not get to v in  ${\sf T}.$
- Assume: compared the ith to jth input element, when paths diverted in T.
- **1.** I.e., Different path in **T** then the one for X and Y.
- $exttt{ iny Claim:} \; x_i < x_j \; ext{and} \; y_i > y_j \; ext{or}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



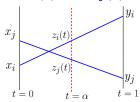
- Assume false.
- ② Assume for  $t=lpha\in[0,1]$  the input Z(t) did not get to v in  ${\sf T}.$
- Assume: compared the ith to jth input element, when paths diverted in T.
- lacktriangle I.e., Different path in lacktriangle then the one for  $oldsymbol{X}$  and  $oldsymbol{Y}$ .
- **3** Claim:  $x_i < x_j$  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  $\mathsf{T}$
- 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  ${\sf T}.$
- **3** Assume: compared the ith to jth input element, when paths diverted in T.
- lacktriangle I.e., Different path in lacktriangle 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 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  ${\sf T}.$
- **3** Assume: compared the ith to jth input element, when paths diverted in T.
- lacktriangle I.e., Different path in lacktriangle 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 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  $v \in \mathsf{T}$ .

### Proof of claim continued...

- 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  $v \in \mathsf{T}$ .

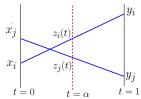
### Recap:

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

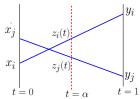
- Recap:
  - **Q** Recall: X, Y to different permutations that their distinct input arrives to the same node  $v \in T$ .
  - ② Proved:  $\forall t \in [0,1]$ :  $Z(t) = (z_1(t), \ldots, z_n(t))$  arrives to same node  $v \in \mathsf{T}$ .
- ② However: There must be  $\beta \in (0,1)$  where  $Z(\beta)$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



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

- ① Done: Found inputs Z(0) and Z(eta)
- Such that one is unique and the other is not.
- $\bigcirc$  ... both arrive to v.

Sariel (UIUC) CS573 19 Fall 2014 19 / 25

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

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

Sariel (UIUC) CS573 19 Fall 2014 19 / 25

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

Sariel (UIUC) CS573 19 Fall 2014 19 / 25

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

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.

Sariel (UIUC) CS573 Fall 2014 19 / 25

- 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.
- ① As in the sorting argument, it follows that there exists a path in **T** of length  $\Omega(n \log n)$ .
- We conclude:

#### Theorem

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

- 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 **T** of length  $\Omega(n \log n)$ .
- We conclude:

#### **Theorem**

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

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

#### **Theorem**

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

- 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 **T** of length  $\Omega(n \log n)$ .
- We conclude:

#### Theorem

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

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

#### **Theorem**

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

- 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.

- 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).

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

- 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.

# Part II

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.

② 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.
- 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.
- 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.

- 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
  - **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:
  - 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
  - f a Given two polygons m P and m Q can one translate m P such that it is contained inside m 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
  - **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:
  - 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
  - **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:
  - 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
  - **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:
  - 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
  - **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:
  - 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
  - **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.

Sariel (UIUC) CS573 26 Fall 2014 26 / 25

Sariel (UIUC) CS573 27 Fall 2014 27 / 2

Sariel (UIUC) CS573 28 Fall 2014 28 / 28

Sariel (UIUC) CS573 29 Fall 2014 29 / 25