Part I

Recurrences
Two general methods:

- Recursion tree method: need to do sums
  - elementary methods, geometric series
  - integration
- Guess and Verify
  - guessing involves intuition, experience and trial & error
  - verification is via induction
Consider $T(n) = 2T(n/2) + n/\log n$. 

Construct recursion tree, and observe pattern.
Consider $T(n) = 2T(n/2) + n/\log n$.

Construct recursion tree, and observe pattern.

\[
\frac{n}{\log n} + \frac{n}{\log n - 1} + \frac{n}{\log n - 2} + \frac{n}{\log n - 3} + \cdots + \frac{n}{1} \\
n \left( \frac{1}{\log n} + \frac{1}{\log n - 1} + \cdots + \frac{1}{\log n - 2^k} \right) \\
\leq \Theta(n \log \log n)
\]
Consider $T(n) = 2T(n/2) + n/\log n$.

Construct recursion tree, and observe pattern. $i$th level has $2^i$ nodes, and problem size at each node is $n/2^i$ and hence work at each node is $\frac{n}{2^i}/\log \frac{n}{2^i}$.

Summing over all levels:

$$T(n) = \sum_{i=0}^{\log n - 1} 2^i \log \left(\frac{n}{2^i}\right) = \sum_{i=1}^{\log n} n \log n - i \log \frac{n}{2^i} = n \sum_{j=1}^{\log n} \frac{1}{j} \log(n) = \Theta(n \log \log n).$$
Consider $T(n) = 2T(n/2) + n/\log n$.

Construct recursion tree, and observe pattern. $i$th level has $2^i$ nodes, and problem size at each node is $n/2^i$ and hence work at each node is $n/2^i/\log n/2^i$.

Summing over all levels

$$T(n) = \sum_{i=0}^{\log n-1} 2^i \left[ \frac{(n/2^i)}{\log(n/2^i)} \right] = \sum_{i=0}^{\log n-1} \frac{n}{\log n - i}$$

$$= n \sum_{j=1}^{\log n} \frac{1}{j} = nH_{\log n} = \Theta(n\log \log n)$$
Recurrence: Example II

Consider $T(n) = T(\sqrt{n}) + 1$.

$$\frac{1}{n^{2^L}} \approx 1$$
Consider $T(n) = T(\sqrt{n}) + 1$.

What is the depth of recursion? $\sqrt{n}, \sqrt[4]{n}, \sqrt[8]{n}, \ldots, O(1)$
Consider $T(n) = T(\sqrt{n}) + 1$.

What is the depth of recursion? $\sqrt{n}, \sqrt{\sqrt{n}}, \sqrt{\sqrt{\sqrt{n}}}, \ldots, O(1)$

Number of levels: $n^{2^{-L}} = 2$ means $L = \log \log n$
Recurrence: Example II

- Consider $T(n) = T(\sqrt{n}) + 1$.
- What is the depth of recursion? $\sqrt{n}, \sqrt[4]{n}, \sqrt[8]{n}, \ldots, O(1)$
- Number of levels: $n^{2^{-L}} = 2$ means $L = \log \log n$
- Number of children at each level is 1, work at each node is 1
Consider $T(n) = T(\sqrt{n}) + 1$.

What is the depth of recursion? $\sqrt{n}, \sqrt{\sqrt{n}}, \sqrt{\sqrt{\sqrt{n}}}, \ldots, O(1)$

Number of levels: $n^{2^{-L}} = 2$ means $L = \log \log n$

Number of children at each level is 1, work at each node is 1

Thus, $T(n) = \sum_{i=0}^{L} 1 = \Theta(L) = \Theta(\log \log n)$. 
Consider $T(n) = \sqrt{n}T(\sqrt{n}) + n$. 

Using recursion trees: number of levels $L = \log \log n$. 

Work at each level? Root is $n$, next level is $\sqrt{n} \times \sqrt{n} = n$, so on. Can check that each level is $n$. 

Thus, $T(n) = \Theta(n \log \log n)$. 
Consider $T(n) = \sqrt{n}T(\sqrt{n}) + n$.

Using recursion trees: number of levels $L = \log \log n$.
Consider $T(n) = \sqrt{n}T(\sqrt{n}) + n$.

Using recursion trees: number of levels $L = \log \log n$.

Work at each level? Root is $n$, next level is $\sqrt{n} \times \sqrt{n} = n$, so on. Can check that each level is $n$. 

Thus, $T(n) = \Theta(n \log \log n)$. 

Recurrence: Example III
Consider $T(n) = \sqrt{n}T(\sqrt{n}) + n$.

Using recursion trees: number of levels $L = \log \log n$

Work at each level? Root is $n$, next level is $\sqrt{n} \times \sqrt{n} = n$, so on. Can check that each level is $n$.

Thus, $T(n) = \Theta(n \log \log n)$
Consider $T(n) = T(n/4) + T(3n/4) + n$. 

Using recursion tree, we observe the tree has leaves at different levels (a lop-sided tree).

Total work in any level is at most $n$. Total work in any level without leaves is exactly $n$.

Highest leaf is at level $\log_4 n$ and lowest leaf is at level $\log_4 \frac{n}{3}$.

Thus, $n \log_4 n \leq T(n) \leq n \log_4 \frac{n}{3}$, which means $T(n) = \Theta(n \log n)$. 

Chekuri

CS473
Recurrence: Example IV

- Consider $T(n) = T(n/4) + T(3n/4) + n$.
- Using recursion tree, we observe the tree has leaves at different levels (a lop-sided tree).
Consider $T(n) = T(n/4) + T(3n/4) + n$.

Using recursion tree, we observe the tree has leaves at different levels (a *lop-sided* tree).

Total work in any level is at most $n$. Total work in any level without leaves is exactly $n$. 

Thus, $n \log_4 n \leq T(n) \leq n \log_4 \frac{4}{3} n$, which means $T(n) = \Theta(n \log n)$. 

Consider $T(n) = T(n/4) + T(3n/4) + n$.

Using recursion tree, we observe the tree has leaves at different levels (a *lop-sided* tree).

Total work in any level is at most $n$. Total work in any level without leaves is exactly $n$.

Highest leaf is at level $\log_4 n$ and lowest leaf is at level $\log_4/3 n$.
Recurrence: Example IV

- Consider $T(n) = T(n/4) + T(3n/4) + n$.
- Using recursion tree, we observe the tree has leaves at different levels (a lop-sided tree).
- Total work in any level is at most $n$. Total work in any level without leaves is exactly $n$.
- Highest leaf is at level $\log_4 n$ and lowest leaf is at level $\log_{4/3} n$.
- Thus, $n \log_4 n \leq T(n) \leq n \log_{4/3} n$, which means $T(n) = \Theta(n \log n)$.
Part II

Closest Pair
Input  Given a set $S$ of $n$ points on the plane

Goal  Find $p, q \in S$ such that $d(p, q)$ is minimum
Closest Pair

**Input**  Given a set $S$ of $n$ points on the plane

**Goal**  Find $p, q \in S$ such that $d(p, q)$ is minimum
Applications

- Basic primitive used in graphics, vision, molecular modelling
- Ideas used in solving nearest neighbor, voronoi diagrams, euclidean MST
Algorithm

Algorithm: Brute Force

Compute distance between every pair of points and find minimum.

Takes $O(n^2)$ time.

Can we do better?
Algorithm

- Compute distance between every pair of points and find minimum
- Takes $O(n^2)$ time
Algorithm: Brute Force

- Compute distance between every pair of points and find minimum
- Takes $O(n^2)$ time
- Can we do better?
Closest Pair: 1-d case

**Input**  Given a set $S$ of $n$ points on a line

**Goal**  Find $p, q \in S$ such that $d(p, q)$ is minimum
Closest Pair: 1-d case

**Input**  Given a set $S$ of $n$ points on a line

**Goal**  Find $p, q \in S$ such that $d(p, q)$ is minimum

**Algorithm**

1. Sort points based on coordinate
2. Compute the distance between successive points, keeping track of the closest pair.

Running time $O(n \log n)$

Can we do this in better running time?
Can reduce Distinct Elements Problem (see lecture 1) to this problem in $O(n)$ time. Do you see how?
Generalizing 1-d case

Can we generalize 1-d algorithm to 2-d?

Sort according to $x$ or $y$-coordinate??
Generalizing 1-d case

Can we generalize 1-d algorithm to 2-d?

Sort according to $x$ or $y$-coordinate??

No easy generalization.
First Attempt

Divide and Conquer I

1. Partition into 4 quadrants of roughly equal size.
2. Find closest pair in each quadrant recursively.
3. Combine solutions.

Diagram:

- Partition the space into four quadrants.
- Find the closest pairs in each quadrant recursively.
- Combine the solutions from each quadrant.

Legend:

- Points in each quadrant indicate the distribution of points for closest pair finding.
First Attempt

Divide and Conquer I

1. Partition into 4 quadrants of roughly equal size. Not always!
2. Find closest pair in each quadrant recursively
3. Combine solutions
Divide and Conquer II

1. Divide the set of points into two equal parts via vertical line
New Algorithm

Divide and Conquer II

1. Divide the set of points into two equal parts via vertical line
2. Find closest pair in each half recursively
New Algorithm

Divide and Conquer II

1. Divide the set of points into two equal parts via vertical line
2. Find closest pair in each half recursively
3. Find closest pair with one point in each half
New Algorithm

Divide and Conquer II

1. Divide the set of points into two equal parts via vertical line
2. Find closest pair in each half recursively
3. Find closest pair with one point in each half
4. Return the best pair among the above 3 solutions
New Algorithm

**Divide and Conquer II**

1. Divide the set of points into two equal parts via vertical line
2. Find closest pair in each half recursively
3. Find closest pair with one point in each half
4. Return the best pair among the above 3 solutions
New Algorithm

Divide and Conquer II

1. Divide the set of points into two equal parts via vertical line
2. Find closest pair in each half recursively
3. Find closest pair with one point in each half
4. Return the best pair among the above 3 solutions
New Algorithm

**Divide and Conquer II**

1. Divide the set of points into two equal parts via vertical line
2. Find closest pair in each half recursively
3. Find closest pair with one point in each half
4. Return the best pair among the above 3 solutions

- Sort points based on $x$-coordinate and pick the median. Time $= O(n \log n)$
New Algorithm

Divide and Conquer II

1. Divide the set of points into two equal parts via vertical line
2. Find closest pair in each half recursively
3. Find closest pair with one point in each half
4. Return the best pair among the above 3 solutions

- Sort points based on $x$-coordinate and pick the median. Time $= O(n \log n)$
New Algorithm

Divide and Conquer II

1. Divide the set of points into two equal parts via vertical line
2. Find closest pair in each half recursively
3. Find closest pair with one point in each half
4. Return the best pair among the above 3 solutions

- Sort points based on $x$-coordinate and pick the median. Time $= O(n \log n)$
- How to find closest pair with points in different halves? $O(n^2)$ is trivial. Better?
Combining Partial Solutions

- Does it take $O(n^2)$ to combine solutions?
- Let $\delta$ be the distance between closest pairs, where both points belong to the same half.
Combining Partial Solutions

- Let \( \delta \) be the distance between closest pairs, where both points belong to the same half.
- Need to consider points within \( \delta \) of dividing line.
Sparsity of Band

Divide the band into square boxes of size $\frac{\delta}{2}$
Sparsity of Band

Divide the band into square boxes of size $\delta/2$

**Lemma**

*Each box has at most one point*
Sparsity of Band

Divide the band into square boxes of size \( \delta/2 \)

**Lemma**

*Each box has at most one point*

**Proof.**

If not, then there are a pair of points (both belonging to one half) that are at most \( \sqrt{2}\delta/2 < \delta \) apart!
Searching within the Band

**Lemma**

Suppose $a, b$ are at distance less than $\delta$ in the band. Then $a, b$ have at most two rows of boxes between them.
Searching within the Band

Lemma

Suppose $a, b$ are at distance less than $\delta$ in the band. Then $a, b$ have at most two rows of boxes between them.

Proof.

Each row of boxes has height $\delta/2$. If more than two rows then distance between $a, b$ greater than $\delta$. 

\[ \text{Proof.} \]
Searching within the Band

**Corollary**

Order points according to their $y$-coordinate. If $p, q$ are such that $d(p, q) < \delta$ then $p$ and $q$ are within 12 positions in the sorted list.

**Proof.**
Corollary

Order points according to their y-coordinate. If p, q are such that \( d(p, q) < \delta \) then p and q are within 12 positions in the sorted list.

Proof.
- Suppose not. Let p and q have at least 11 points between them in the sorted order.
Searching within the Band

Corollary

Order points according to their y-coordinate. If \( p, q \) are such that \( d(p, q) < \delta \) then \( p \) and \( q \) are within 12 positions in the sorted list.

Proof.

- Suppose not. Let \( p \) and \( q \) have at least 11 points between them in the sorted order.
- \( p \) and \( q \) are at least two rows apart in grid because each box has at most one point.
Searching within the Band

**Corollary**

*Order points according to their y-coordinate. If \( p, q \) are such that \( d(p, q) < \delta \) then \( p \) and \( q \) are within 12 positions in the sorted list.*

**Proof.**

- Suppose not. Let \( p \) and \( q \) have at least 11 points between them in the sorted order.
- \( p \) and \( q \) are at least two rows apart in grid because each box has at most one point.
- \( d(p, q) > 2(\delta/2) = \delta! \)
The Algorithm

1. Find vertical line $L$ that splits the points into equal halves
2. Compute closest pair in the left half; let the distance be $\delta_1$
3. Compute closest pair in right half; let the distance be $\delta_2$
4. $\delta = \min(\delta_1, \delta_2)$
5. Delete points further than $\delta$ from $L$
6. Sort remaining points based on $y$-coordinate into an array $A$
7. for $i = 1$ to $|A| - 1$ do
   for $j = i + 1$ to $\min\{i + 11, |A|\}$ do
     If ($\text{dist}(A[i], A[j]) < \delta$) update $\delta$ and closest pair
The Algorithm

1. **Find vertical line** $L$ **that splits the points into equal halves**
2. Compute closest pair in the left half; let the distance be $\delta_1$
3. Compute closest pair in right half; let the distance be $\delta_2$
4. $\delta = \min(\delta_1, \delta_2)$
5. Delete points further than $\delta$ from $L$
6. Sort remaining points based on $y$-coordinate into an array $A$
7. for $i = 1$ to $|A| - 1$ do
   for $j = i + 1$ to $\min\{i + 11, |A|\}$ do
     If $(\text{dist}(A[i], A[j]) < \delta)$ update $\delta$ and closest pair
The Algorithm

1. Find vertical line $L$ that splits the points into equal halves
2. Compute closest pair in the left half; let the distance be $\delta_1$
3. Compute closest pair in right half; let the distance be $\delta_2$
4. $\delta = \min(\delta_1, \delta_2)$
5. Delete points further than $\delta$ from $L$
6. Sort remaining points based on $y$-coordinate into an array $A$
7. for $i = 1$ to $|A| - 1$ do
   for $j = i + 1$ to $\min\{i + 11, |A|\}$ do
     If ($dist(A[i], A[j]) < \delta$) update $\delta$ and closest pair

- Step 1, involves sorting and scanning. Takes $O(n \log n)$ time.
The Algorithm

1. Find vertical line $L$ that splits the points into equal halves
2. Compute closest pair in the left half; let the distance be $\delta_1$
3. Compute closest pair in right half; let the distance be $\delta_2$
4. $\delta = \min(\delta_1, \delta_2)$
5. **Delete points further than $\delta$ from $L$**
6. Sort remaining points based on $y$-coordinate into an array $A$
7. for $i = 1$ to $|A| - 1$ do
   for $j = i + 1$ to $\min\{i + 11, |A|\}$ do
     If ($\text{dist}(A[i], A[j]) < \delta$) update $\delta$ and closest pair

- Step 1, involves sorting and scanning. Takes $O(n \log n)$ time.
The Algorithm

1. Find vertical line $L$ that splits the points into equal halves
2. Compute closest pair in the left half; let the distance be $\delta_1$
3. Compute closest pair in right half; let the distance be $\delta_2$
4. $\delta = \min(\delta_1, \delta_2)$
5. Delete points further than $\delta$ from $L$
6. Sort remaining points based on $y$-coordinate into an array $A$
7. for $i = 1$ to $|A| - 1$ do
   for $j = i + 1$ to $\min\{i + 11, |A|\}$ do
      If $(\text{dist}(A[i], A[j]) < \delta)$ update $\delta$ and closest pair

- Step 1, involves sorting and scanning. Takes $O(n \log n)$ time.
- Step 5 takes $O(n)$ time
The Algorithm

1. Find vertical line $L$ that splits the points into equal halves
2. Compute closest pair in the left half; let the distance be $\delta_1$
3. Compute closest pair in right half; let the distance be $\delta_2$
4. $\delta = \min(\delta_1, \delta_2)$
5. Delete points further than $\delta$ from $L$
6. Sort remaining points based on $y$-coordinate into an array $A$
7. for $i = 1$ to $|A| - 1$ do
   for $j = i + 1$ to $\min\{i + 11, |A|\}$ do
      If ($\text{dist}(A[i], A[j]) < \delta$) update $\delta$ and closest pair

- Step 1, involves sorting and scanning. Takes $O(n \log n)$ time.
- Step 5 takes $O(n)$ time
The Algorithm

1. Find vertical line $L$ that splits the points into equal halves
2. Compute closest pair in the left half; let the distance be $\delta_1$
3. Compute closest pair in right half; let the distance be $\delta_2$
4. $\delta = \min(\delta_1, \delta_2)$
5. Delete points further than $\delta$ from $L$
6. Sort remaining points based on $y$-coordinate into an array $A$
7. for $i = 1$ to $|A| - 1$ do
   for $j = i + 1$ to $\min\{i + 11, |A|\}$ do
      If $(\text{dist}(A[i], A[j]) < \delta)$ update $\delta$ and closest pair

- Step 1, involves sorting and scanning. Takes $O(n \log n)$ time.
- Step 5 takes $O(n)$ time
- Step 6 takes $O(n \log n)$ time
The Algorithm

1. Find vertical line $L$ that splits the points into equal halves
2. Compute closest pair in the left half; let the distance be $\delta_1$
3. Compute closest pair in right half; let the distance be $\delta_2$
4. $\delta = \min(\delta_1, \delta_2)$
5. Delete points further than $\delta$ from $L$
6. Sort remaining points based on $y$-coordinate into an array $A$
7. for $i = 1$ to $|A| - 1$ do
   for $j = i + 1$ to min{$i + 11, |A|$} do
      If (dist($A[i], A[j]$) < $\delta$) update $\delta$ and closest pair

- Step 1, involves sorting and scanning. Takes $O(n \log n)$ time.
- Step 5 takes $O(n)$ time.
- Step 6 takes $O(n \log n)$ time.
The Algorithm

1. Find vertical line $L$ that splits the points into equal halves
2. Compute closest pair in the left half; let the distance be $\delta_1$
3. Compute closest pair in right half; let the distance be $\delta_2$
4. $\delta = \min(\delta_1, \delta_2)$
5. Delete points further than $\delta$ from $L$
6. Sort remaining points based on $y$-coordinate into an array $A$
7. for $i = 1$ to $|A| - 1$ do
   for $j = i + 1$ to $\min\{i + 11, |A|\}$ do
     If ($\text{dist}(A[i], A[j]) < \delta$) update $\delta$ and closest pair

- Step 1, involves sorting and scanning. Takes $O(n \log n)$ time.
- Step 5 takes $O(n)$ time
- Step 6 takes $O(n \log n)$ time
- Step 7 takes $O(n)$ time
Running Time

The running time of the algorithm is given by

\[ T(n) \leq 2T(n/2) + O(n \log n) \]
Running Time

The running time of the algorithm is given by

\[ T(n) \leq 2T(n/2) + O(n \log n) \]

Thus, \( T(n) = O(n \log^2 n) \).
The running time of the algorithm is given by

\[ T(n) \leq 2T(n/2) + O(n \log n) \]

Thus, \( T(n) = O(n \log^2 n) \).

**Improved Algorithm**

Avoid repeated sorting of points in band: two options

- Sort all points by \( y \)-coordinate and store the list. In conquer step use this to avoid sorting
- Each recursive call returns a list of points sorted by their \( y \)-coordinates. Merge in conquer step in linear time.

Analysis: \( T(n) \leq 2T(n/2) + O(n) = O(n \log n) \)
Part III

Selecting in Unsorted Lists
**Quick Sort**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pick a pivot element from array</td>
</tr>
<tr>
<td>2</td>
<td>Split array into 3 subarrays: those smaller than pivot, those larger than pivot, and the pivot itself.</td>
</tr>
<tr>
<td>3</td>
<td>Recursively sort the subarrays, and concatenate them.</td>
</tr>
</tbody>
</table>
Quick Sort

Quick Sort[Hoare]

1. Pick a pivot element from array
2. Split array into 3 subarrays: those smaller than pivot, those larger than pivot, and the pivot itself.
3. Recursively sort the subarrays, and concatenate them.

Example:
- array: 16, 12, 14, 20, 5, 3, 18, 19, 1
- pivot: 16
- split into 12, 14, 5, 3, 1 and 20, 19, 18 and recursively sort
- put them together with pivot in middle
Quick Sort

Quick Sort[Hoare]

1. Pick a pivot element from array
2. Split array into 3 subarrays: those smaller than pivot, those larger than pivot, and the pivot itself. Linear scan of array does it. Time is $O(n)$
3. Recursively sort the subarrays, and concatenate them.

Example:

- array: 16, 12, 14, 20, 5, 3, 18, 19, 1
- pivot: 16
- split into 12, 14, 5, 3, 1 and 20, 19, 18 and recursively sort
- put them together with pivot in middle
Quick Sort

Quick Sort[Hoare]

1. Pick a pivot element from array
2. Split array into 3 subarrays: those smaller than pivot, those larger than pivot, and the pivot itself. Linear scan of array does it. Time is $O(n)$
3. Recursively sort the subarrays, and concatenate them.

Example:

- array: 16, 12, 14, 20, 5, 3, 18, 19, 1
- pivot: 16
- split into 12, 14, 5, 3, 1 and 20, 19, 18 and recursively sort
- put them together with pivot in middle
Let $k$ be the rank of the chosen pivot. Then,

$$T(n) = T(k - 1) + T(n - k) + O(n)$$
Time Analysis

- Let \( k \) be the rank of the chosen pivot. Then,
  \[
  T(n) = T(k - 1) + T(n - k) + O(n)
  \]
- If \( k = \lceil n/2 \rceil \) then
  \[
  T(n) = T(\lceil n/2 \rceil - 1) + T(\lfloor n/2 \rfloor) + O(n) \leq 2T(n/2) + O(n).
  \]
  Then, \( T(n) = O(n \log n) \).
Time Analysis

- Let $k$ be the rank of the chosen pivot. Then,
  \[ T(n) = T(k - 1) + T(n - k) + O(n) \]
- If $k = \lceil n/2 \rceil$ then
  \[ T(n) = T(\lceil n/2 \rceil - 1) + T(\lfloor n/2 \rfloor) + O(n) \leq 2T(n/2) + O(n). \]
  Then, $T(n) = O(n \log n)$.
  - Theoretically, median can be found in linear time.
Time Analysis

- Let \( k \) be the rank of the chosen pivot. Then,
  \[ T(n) = T(k - 1) + T(n - k) + O(n) \]

- If \( k = \lceil n/2 \rceil \) then
  \[ T(n) = T(\lceil n/2 \rceil - 1) + T(\lfloor n/2 \rfloor) + O(n) \leq 2T(n/2) + O(n). \]
  Then, \( T(n) = O(n \log n) \).

- Theoretically, median can be found in linear time.

- Typically, pivot is the first or last element of array. Then,
  \[ T(n) = \max_{1 \leq k \leq n} (T(k - 1) + T(n - k) + O(n)) \]

In the worst case \( T(n) = T(n - 1) + O(n) \), which means \( T(n) = O(n^2) \). Happens if array is already sorted and pivot is always first element.
Selection

Input: Unsorted array $A$ of $n$ integers

Goal: Find the $j$’th smallest number in $A$ (*rank* $j$ number)

Example

$A = \{4, 6, 2, 1, 5, 8, 7\}$ and $j = 4$. The $j$th smallest element is 5.

Median: $j = \lfloor n/2 \rfloor \leq (n+1)/2$
Algorithm I

1. Sort the elements in $A$
2. Pick $j$th element in sorted order

Time taken $= O(n \log n)$
Algorithm I

1. Sort the elements in $A$
2. Pick $j$th element in sorted order

Time taken $= O(n \log n)$

Do we need to sort? Is there an $O(n)$ time algorithm?
Algorithm II

If $j$ is small or $n - j$ is small then

- Find $j$ smallest/largest elements in $A$ in $O(jn)$ time. (How?)
- Time to find median is $O(n^2)$. 

Divide and Conquer Approach

1. Pick a pivot element \( a \) from \( A \)
2. Partition \( A \) based on \( a \).
   \[ A_{\text{less}} = \{ x \in A \mid x \leq a \} \quad \text{and} \quad A_{\text{greater}} = \{ x \in A \mid x > a \} \]
3. \( |A_{\text{less}}| = j \): return \( a \)
4. \( |A_{\text{less}}| > j \): recursively find \( j \)th smallest element in \( A_{\text{less}} \)
5. \( |A_{\text{less}}| < j \): recursively find \( k \)th smallest element in \( A_{\text{greater}} \)
   where \( k = j - |A_{\text{less}}| \).

Assumption: elements in \( A \) are distinct
Exercise: modify algorithm if \( A \) has duplicates
Time Analysis

- Partitioning step: $O(n)$ time to scan $A$
- How do we choose pivot? Recursive running time?
Time Analysis

- Partitioning step: \( O(n) \) time to scan \( A \)
- How do we choose pivot? Recursive running time?

Suppose we always choose pivot to be \( A[1] \).
Time Analysis

- Partitioning step: $O(n)$ time to scan $A$
- How do we choose pivot? Recursive running time?

Suppose we always choose pivot to be $A[1]$. Say $A$ is sorted in increasing order and $j = n$. Exercise: show that algorithm takes $\Omega(n^2)$ time.
A Better Pivot

Suppose pivot is the $\ell$’th smallest element where $n/4 \leq \ell \leq 3n/4$. That is pivot is \textit{approximately} in the middle of $A$.

Then $n/4 \leq |A_{\text{less}}| \leq 3n/4$ \textit{and} $n/4 \leq |A_{\text{greater}}| \leq 3n/4$. If we apply recursion,
A Better Pivot

Suppose pivot is the $\ell$’th smallest element where $n/4 \leq \ell \leq 3n/4$. That is pivot is \textit{approximately} in the middle of $A$

Then $n/4 \leq |A_{\text{less}}| \leq 3n/4$ and $n/4 \leq |A_{\text{greater}}| \leq 3n/4$. If we apply recursion,

\[
T(n) \leq T(3n/4) + O(n)
\]

Implies $T(n) = O(n)$!
A Better Pivot

Suppose pivot is the $\ell$’th smallest element where $n/4 \leq \ell \leq 3n/4$. That is pivot is \textit{approximately} in the middle of $A$.

Then $n/4 \leq |A_{\text{less}}| \leq 3n/4$ and $n/4 \leq |A_{\text{greater}}| \leq 3n/4$. If we apply recursion,

$$T(n) \leq T(3n/4) + O(n)$$

Implies $T(n) = O(n)$!

How do we find such a pivot?
A Better Pivot

Suppose pivot is the $\ell$’th smallest element where $n/4 \leq \ell \leq 3n/4$. That is pivot is approximately in the middle of $A$

Then $n/4 \leq |A_{\text{less}}| \leq 3n/4$ and $n/4 \leq |A_{\text{greater}}| \leq 3n/4$. If we apply recursion,

$$T(n) \leq T(3n/4) + O(n)$$

Implies $T(n) = O(n)$!

How do we find such a pivot? Randomly?
A Better Pivot

Suppose pivot is the $\ell$’th smallest element where $n/4 \leq \ell \leq 3n/4$. That is pivot is *approximately* in the middle of $A$.

Then $n/4 \leq |A_{\text{less}}| \leq 3n/4$ and $n/4 \leq |A_{\text{greater}}| \leq 3n/4$. If we apply recursion,

$$T(n) \leq T(3n/4) + O(n)$$

Implies $T(n) = O(n)$!

How do we find such a pivot? Randomly? In fact works! Analysis a little bit later.
A Better Pivot

Suppose pivot is the $\ell$'th smallest element where $n/4 \leq \ell \leq 3n/4$. That is pivot is \textit{approximately} in the middle of $A$.

Then $n/4 \leq |A_{\less}| \leq 3n/4$ and $n/4 \leq |A_{\greater}| \leq 3n/4$. If we apply recursion,

$$T(n) \leq T(3n/4) + O(n)$$

Implies $T(n) = O(n)$!

How do we find such a pivot? Randomly? In fact works! Analysis a little bit later.

Can we choose pivot deterministically?
Choosing the pivot

1. Partition array $A$ into $\lceil n/5 \rceil$ lists of 5 items each.
   
   $L_1 = \{A[1], A[2], \ldots, A[5]\}$, $L_2 = \{A[6], \ldots, A[10]\}$, $L_3 = \{A[11], \ldots, A[15]\}$, ...
   $L_i = \{A[5i+1], \ldots, A[5i-4]\}$, $L_{\lceil n/5 \rceil} = \{A[5\lceil n/5 \rceil - 4], \ldots, A[n]\}$.

2. For each $i$ find median $b_i$ of $L_i$ using brute-force in $O(1)$ time.
   Total $O(n)$ time

3. Let $B = \{b_1, b_2, \ldots, b_{\lceil n/5 \rceil}\}$

4. Find median $b$ of $B$
Choosing the pivot

1. Partition array $A$ into $\lceil n/5 \rceil$ lists of 5 items each.
   
   $L_1 = \{A[1], A[2], \ldots, A[5]\}$, $L_2 = \{A[6], \ldots, A[10]\}$, \ldots,
   
   $L_i = \{A[5i + 1], \ldots, A[5i - 4]\}$, \ldots,
   
   $L_{\lceil n/5 \rceil} = \{A[5 \lceil n/5 \rceil - 4, \ldots, A[n]\}$.

2. For each $i$ find median $b_i$ of $L_i$ using brute-force in $O(1)$ time.
   
   Total $O(n)$ time

3. Let $B = \{b_1, b_2, \ldots, b_{\lceil n/5 \rceil}\}$

4. Find median $b$ of $B$

---

Lemma

*Median of $B$ is an approximate median of $A$. That is, if $b$ is used a pivot to partition $A$, then $|A_{\text{less}}| \leq 7n/10 + 6$ and $|A_{\text{greater}}| \leq 7n/10 + 6$.**
Algorithm for Selection

SELECT(A, j):

Form lists \( L_1, L_2, \ldots, L_{\lceil n/5 \rceil} \) where \( L_i = \{A[5i - 4], \ldots, A[5i]\} \)

Find median \( b_i \) of each \( L_i \) using brute-force

Find median \( b \) of \( B = \{b_1, b_2, \ldots, b_{\lceil n/5 \rceil}\} \)

Partition \( A \) into \( A_{\text{less}} \) and \( A_{\text{greater}} \) using \( b \) as pivot

If \( |A_{\text{less}}| = j \) return \( b \)

Else if \( |A_{\text{less}}| > j \)

\[ \text{return SELECT}(A_{\text{less}}, j) \]

Else

\[ \text{return SELECT}(A_{\text{greater}}, j - |A_{\text{less}}|) \]
Algorithm for Selection

SELECT($A$, $j$):
  Form lists $L_1, L_2, \ldots, L_{\lceil n/5 \rceil}$ where $L_i = \{A[5i-4], \ldots, A[5i]\}$
  Find median $b_i$ of each $L_i$ using brute-force
  Find median $b$ of $B = \{b_1, b_2, \ldots, b_{\lceil n/5 \rceil} \}$
  Partition $A$ into $A_{\text{less}}$ and $A_{\text{greater}}$ using $b$ as pivot
  If ($|A_{\text{less}}| = j$) return $b$
  Else if ($|A_{\text{less}}| > j$)
    return SELECT($A_{\text{less}}$, $j$)
  Else
    return SELECT($A_{\text{greater}}$, $j - |A_{\text{less}}|$)

How do we find median of $B$?

Recursively!
Algorithm for Selection

SELECT(A, j):
Form lists \( L_1, L_2, \ldots, L_{\lceil n/5 \rceil} \) where \( L_i = \{A[5i - 4], \ldots, A[5i]\} \)
Find median \( b_i \) of each \( L_i \) using brute-force
Find median \( b \) of \( B = \{b_1, b_2, \ldots, b_{\lceil n/5 \rceil}\} \)
Partition \( A \) into \( A_{\text{less}} \) and \( A_{\text{greater}} \) using \( b \) as pivot
If \(|A_{\text{less}}| = j\) return \( b \)
Else if \(|A_{\text{less}}| > j\) return \( \text{SELECT}(A_{\text{less}}, j) \)
Else return \( \text{SELECT}(A_{\text{greater}}, j - |A_{\text{less}}|) \)

How do we find median of \( B \)? Recursively!
Recursive algorithm for Selection

SELECT($A$, $j$):
Form lists $L_1, L_2, \ldots, L_{\lceil n/5 \rceil}$ where $L_i = \{A[5i - 4], \ldots, A[5i]\}$
Find median $b_i$ of each $L_i$ using brute-force
$B$ is the array of $b_1, b_2, \ldots, b_{\lceil n/5 \rceil}$.
$b = \text{SELECT}(B, \lceil n/10 \rceil)$
Partition $A$ into $A_{\text{less}}$ and $A_{\text{greater}}$ using $b$ as pivot
If ($|A_{\text{less}}| = j$) return $b$
Else if ($|A_{\text{less}}| > j$) return $\text{SELECT}(A_{\text{less}}, j)$
Else
\quad return $\text{SELECT}(A_{\text{greater}}, j - |A_{\text{less}}|)$
Running time

\[
T(n) = T(\lceil n/5 \rceil) + \max\{T(|A_{\text{less}}|), T(|A_{\text{greater}}|)\} + O(n)
\]
Running time

\[ T(n) = T(\lceil n/5 \rceil) + \max\{ T(|A_{\text{less}}|), T(|A_{\text{greater}}|) \} + O(n) \]

From Lemma,

\[ T(n) \leq T(\lceil n/5 \rceil) + T(\lfloor 7n/10 + 6 \rfloor) + O(n) \]

and

\[ T(1) = 1 \]
Running time

\[ T(n) = T\left(\left\lceil \frac{n}{5} \right\rceil \right) + \max\{ T(|A_{\text{less}}|), T(|A_{\text{greater}}|) \} + O(n) \]

From Lemma,

\[ T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil \right) + T\left(\left\lfloor \frac{7n}{10} + 6 \right\rfloor \right) + O(n) \]

and

\[ T(1) = 1 \]

Exercise: show that \( T(n) = O(n) \)
Median of Medians: Proof of Lemma

**Proposition**

There are at least $3n/10 - 6$ elements greater than the median of medians $b$.

**Figure:** Shaded elements are all greater than $b$
Median of Medians: Proof of Lemma

**Proposition**

There are at least $3n/10 - 6$ elements greater than the median of medians $b$.

**Proof.**

At least half of the $\lceil n/5 \rceil$ groups have at least 3 elements larger than $b$, except for last group and the group containing $b$. So $b$ is less than

$$3(\lceil (1/2) \lceil n/5 \rceil \rceil - 2) \geq 3n/10 - 6$$
Median of Medians: Proof of Lemma

**Proposition**

There are at least $3n/10 - 6$ elements greater than the median of medians $b$.

**Corollary**

$|A_{less}| \leq 7n/10 + 6$.

Via symmetric argument,

**Corollary**

$|A_{greater}| \leq 7n/10 + 6$. 
Questions to ponder

- Why did we choose lists of size 5? Will lists of size 3 work?
- Write a recurrence to analyze the algorithm’s running time if we choose a list of size $k$. 
Median of Medians Algorithm

Due to:

Median of Medians Algorithm

Due to:


How many Turing Award winners in the author list?
Due to:


How many Turing Award winners in the author list? All except Vaughn Pratt!
Takeaway Points

- Recursion tree method and guess and verify are the most reliable methods to analyze recursions in algorithms.
- Recursive algorithms naturally lead to recurrences.
- Some times one can look for certain type of recursive algorithms (reverse engineering) by understanding recurrences and their behaviour.