

Lecture 5: Recursive Algorithms: The Median; Randomized Selection; Deterministic Selection

1 The Median Problem

- How do we find the **median** of a set $S = \{x_1, x_2, \dots, x_n\}$?
 - We could sort the list and then output the $\lceil \frac{n}{2} \rceil$ th number.
 - Using **Mergesort**, or otherwise, this will take time $O(n \cdot \log n)$.
- Is there a **faster** way to find the median?
 - We only need the median number. Sorting all numbers seems overkill. We cannot do better than $O(n)$ since we need to read n numbers in linear time.

2 The Selection Problem

- How do we find the k^{th} smallest number in $S = \{x_1, x_2, \dots, x_n\}$?
 - Again, we could **sort** the list and then output the k^{th} number.
 - Using **Mergesort**, this takes time $O(n \cdot \log n)$.
- We can do this much faster using recursion...

The Selection Algorithm

Select(S, k)

If $|S| = 1$ **then** output x_1

Else

Set $S_L = \{x_i \in S : x_i < x_1\}$

Set $S_R = \{x_i \in S \setminus x_1 : x_i \geq x_1\}$

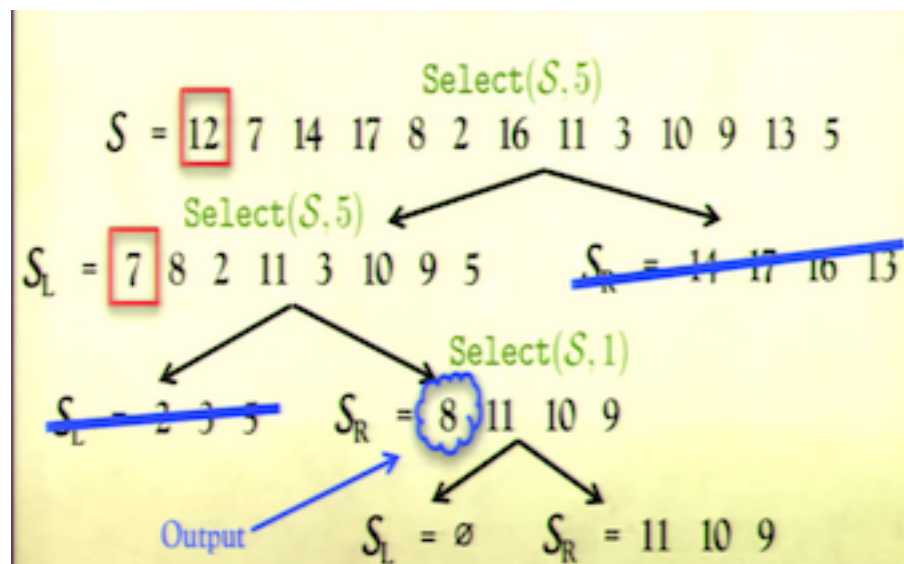
If $|S_L| = k - 1$ **then** output x_1

If $|S_L| > k - 1$ **then** output **Select**(S_L, k)

If $|S_L| < k - 1$ **then** output **Select**($S_R, k - 1 - |S_L|$)

Note:

1. If $|S_L| = k - 1$, then x_1 is the k^{th} smallest number.
2. On the other hand, suppose smallest S_L contains at least $(k - 1)$ elements, which means it gets k elements, in other words, k^{th} smallest element is actually in S_L . The k^{th} smallest element of S must also be the k^{th} smallest element of S_L .
3. S_L union x_1 has the most $(k - 1)$ elements, which means the k^{th} smallest element must be in the set S_R . Since everything in S_R is at least bigger than in x_1 , so that it is also bigger than in S_L , to find the k^{th} smallest element, we need $(k - 1 - |S_L|)$.



- How many comparisons $T(n)$ does this recursive selection algorithm make?

The Selection Algorithm

$$T(n) = n - 1 + T(\max\{|S_L|, |S_R|\})$$

$$\text{Select}(S, k) = n - 1 + T(n - 1)$$

If $|S| = 1$ then output x_1

Else

Set $S_L = \{x_i \in S : x_i < x_1\}$

Set $S_R = \{x_i \in S \setminus x_1 : x_i \geq x_1\}$ } $n-1$ comparisons

If $|S_L| = k - 1$ then output x_1

If $|S_L| > k - 1$ then output $\text{Select}(S_L, k)$

If $|S_L| < k - 1$ then output $\text{Select}(S_R, k - 1 - |S_L|)$

Worst case: $|S_L|$ or $|S_R| = n - 1$

- So in the **worst case** we have is:

$$\begin{aligned}
 T(n) &= (n - 1) + T(n - 1) \\
 &= (n - 1) + (n - 2) + T(n - 2) \\
 &\quad \cdot \\
 &\quad \cdot \\
 &\quad \cdot \\
 &= (n - 1) + (n - 2) + (n - 3) + \dots + 2 + 1 \\
 &= \frac{1}{2}n(n - 1) \\
 &= \Omega(n^2)
 \end{aligned}$$

- This is terrible - sorting would have been faster!
- How to fix it? Use Balanced Pivots!
 - The problem is the algorithm repeatedly **pivots** on the first number in the current list.
 - But if we are unlucky this pivot could be very **unbalanced**. That is:

$$\begin{aligned}
 \max\{|S_L|, |S_R|\} &\approx n \\
 \min\{|S_L|, |S_R|\} &\approx 0
 \end{aligned}$$

- What we would like is to select a pivot that is **balanced**. That is:
 $\max\{|S_L|, |S_R|\} \approx \frac{n}{2}$
 $\min\{|S_L|, |S_R|\} \approx \frac{n}{2}$
- Randomization
 - The current algorithm is **deterministic** in the choice of the pivot.
 - To fix the problem we consider a **randomized** implementation.
 - Do not pivot deterministically on x_1 .
 - Instead choose the pivot at random from $S = \{x_1, x_2, \dots, x_n\}$

Randomized Selection

RandSelect(S, k)

If $|S| = 1$ **then** output x_1

Else

Pick a random pivot $x_\tau \in \{x_1, x_2, \dots, x_n\}$

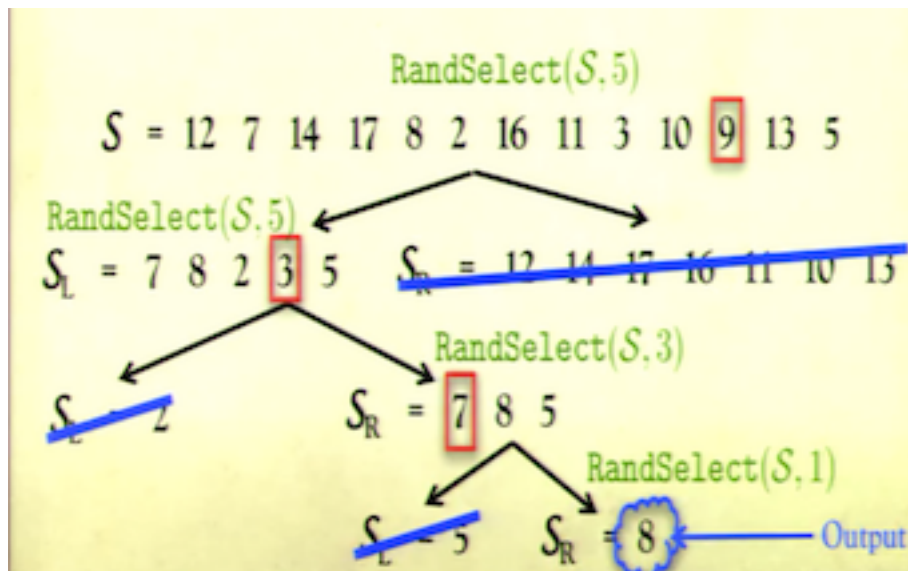
Set $S_L = \{x_i \in S : x_i < x_\tau\}$

Set $S_R = \{x_i \in S \setminus x_\tau : x_i \geq x_\tau\}$

If $|S_L| = k - 1$ **then** output x_τ

If $|S_L| > k - 1$ **then** output **RandSelect**(S_L, k)

If $|S_L| < k - 1$ **then** output **RandSelect**($S_R, k - 1 - |S_L|$)

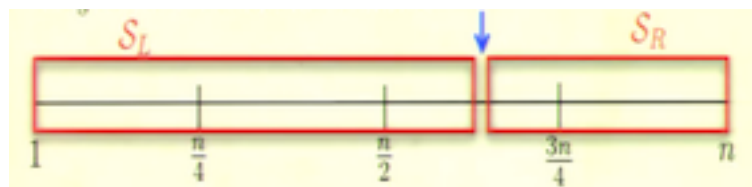


- Good vs Bad Pivots

- Imagine all the numbers in the **sorted** order.



- With probability $\frac{1}{2}$ the pivot x_τ lies between the 1st and 3rd quartiles.
 → We say that such a pivot is **good**.
 → We say that such a pivot is **bad**.
- **Key Observation:** If the pivot is good then
 $\max\{|S_L|, |S_R|\} \leq \frac{3}{4} \cdot n$



- Expected Runtime

- In the worst case, the *randomized algorithm* will pick the worst pivot! The probability of its happening is very small.
- So, for randomized algorithms, we are always interested in the **expected runtime** $\bar{T}(n) = E(T(n))$, not the worst case run time.
- Using our observation, we then have that:

$$\bar{T}(n) \leq \frac{1}{2} \cdot \bar{T}\left(\frac{3n}{4}\right) + \frac{1}{2} \cdot \bar{T}(n) + O(n)$$

Note:

1. The first term: the probability of $\frac{1}{2}$ comes from I make a good pivot.
 If I make a good pivot, I know both of my subsets will have at most the size of $\frac{3n}{4}$.

2. The second term: If I get a bad pivot, the size of the next problem might be $(n - 1)$, which is $\bar{T}(n)$ in the worst case.

$$\bar{T}(n) \leq \frac{1}{2} \cdot \bar{T}\left(\frac{3n}{4}\right) + \frac{1}{2} \cdot \bar{T}(n) + O(n)$$

$$\Rightarrow \frac{1}{2} \cdot \bar{T}(n) \leq \frac{1}{2} \cdot \bar{T}\left(\frac{3n}{4}\right) + O(n)$$

$$\Rightarrow \bar{T}(n) \leq \bar{T}\left(\frac{3n}{4}\right) + O(n)$$

- Apply the Master Theorem

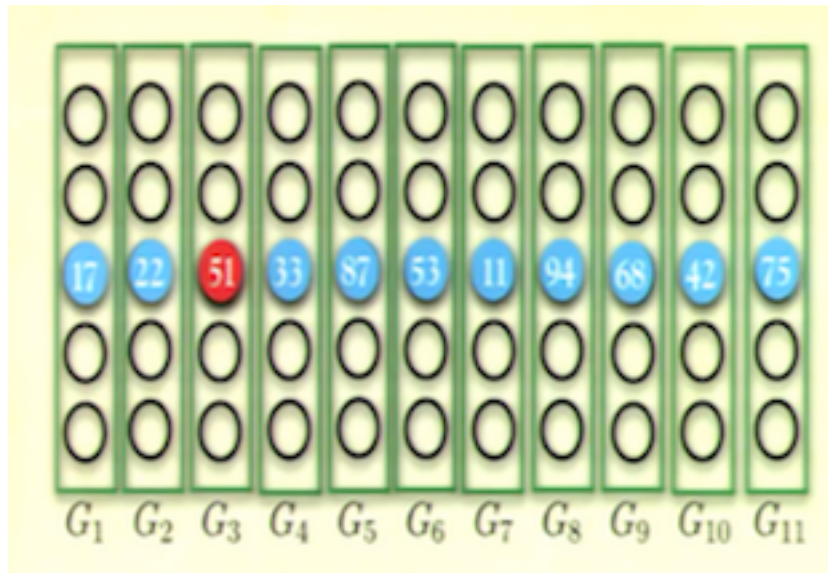
→ $a = 1$, $b = \frac{4}{3}$, and $d = 1$

→ This is Case 1 of the Master Theorem.

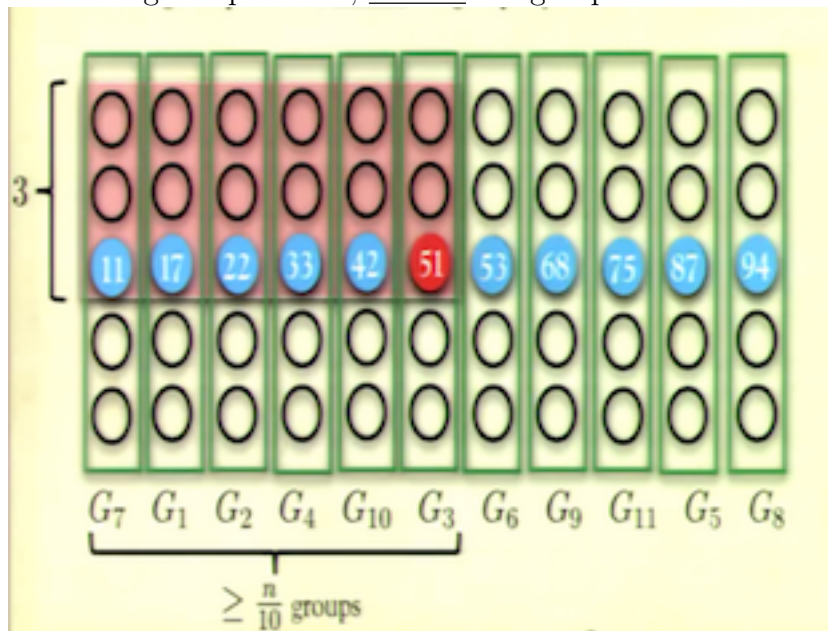
→ Runtime = $O(n^d) = O(n)$

3 Deterministic Selection

- So we have a **linear time** randomized algorithm for the selection problem.
- Is there a linear time deterministic algorithm?
- To do this, what we need is a deterministic method to find a **good pivot**.
- The idea is to find "the median of the medians."
- Divide $S = \{x_1, x_2, \dots, x_n\}$ into groups of cardinality five:
 $G_1 = \{x_1, \dots, x_5\}, G_2 = \{x_6, \dots, x_{10}\}, \dots, G_{\frac{n}{5}} = \{x_{n-4}, \dots, x_n\}$
- Now sort each group and let z_i be the median of the group G_i



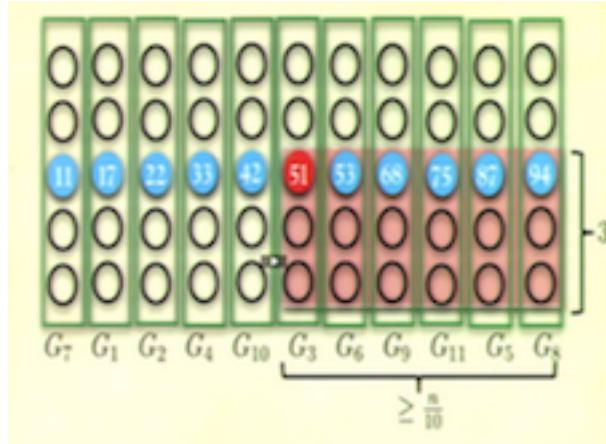
- Let **m** be the **median** of $Z = \{z_1, z_2, \dots, z_{\frac{n}{5}}\}$
- As a thought experiment, reorder the groups their median values:



- The **median of the medians** is greater than at least $\frac{3n}{10} - 1$ numbers in S
- $$\Rightarrow |S_R| = |\{x_i \in S \setminus m : x_i \geq m\}| \leq \frac{7}{10} \cdot n$$

- There are at least $\frac{3n}{10} - 1$ numbers in S that are at least as big as m .

$$\Rightarrow |S_L| = |\{x_i \in S : x_i < m\}| \leq \frac{7}{10} \cdot n$$



- Thus, using m as a **pivot** we have:

$$\Rightarrow \max\{|S_L|, |S_R|\} \leq \frac{7}{10} \cdot n$$

- Thus the median of the medians is a **good** pivot.
- But how actually do we find the median of the medians?
 - Using the same **deterministic recursive algorithm**!

;))

Deterministic Selection**DetSelect**(S, k)**If** $|S| = 1$ **then** output x_1 **Else**Partition S into $\lceil \frac{n}{5} \rceil$ groups of 5.**For** $j = \{1, 2, \dots, \frac{n}{5}\}$ Let z_j be the median of group G_j **Let** $Z = \{z_1, z_2, \dots, z_{\lceil \frac{n}{5} \rceil}\}$ Set $m \leftarrow \mathbf{DetSelect}(Z, \lceil \frac{n}{10} \rceil)$ Set $S_L = \{x_i \in S : x_i < m\}$ Set $S_R = \{x_i \in S \setminus \{m\} : x_i \geq m\}$ **If** $|S_L| = k - 1$ **then** output m **If** $|S_L| > k - 1$ **then** output **DetSelect**(S_L, k)**If** $|S_L| < k - 1$ **then** output **DetSelect**($S_R, k - 1 - |S_L|$)

- Thus, using m as a **pivot** we have:

$$\Rightarrow \max\{|S_L|, |S_R|\} \leq \frac{7}{10} \cdot n$$

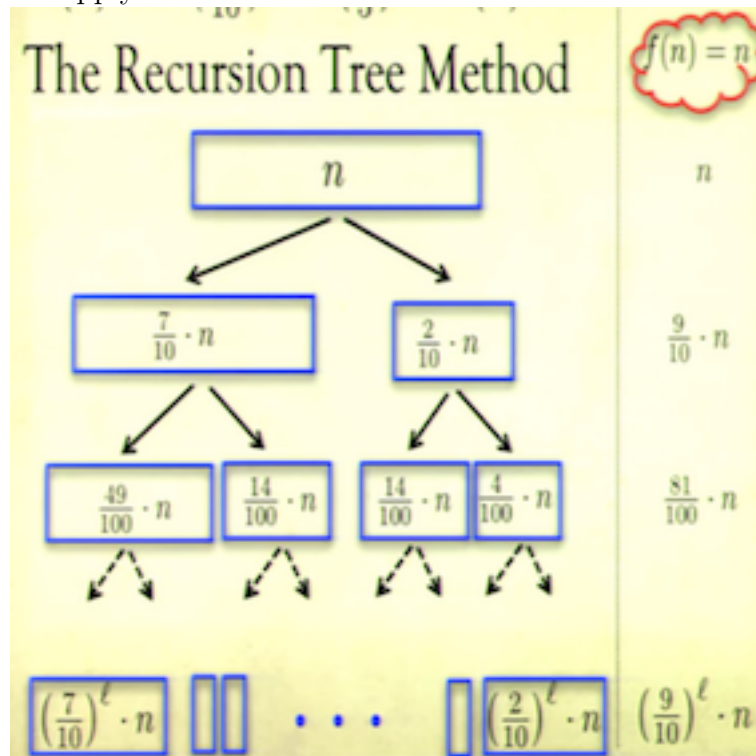
- The recursive formula for the running time is then:

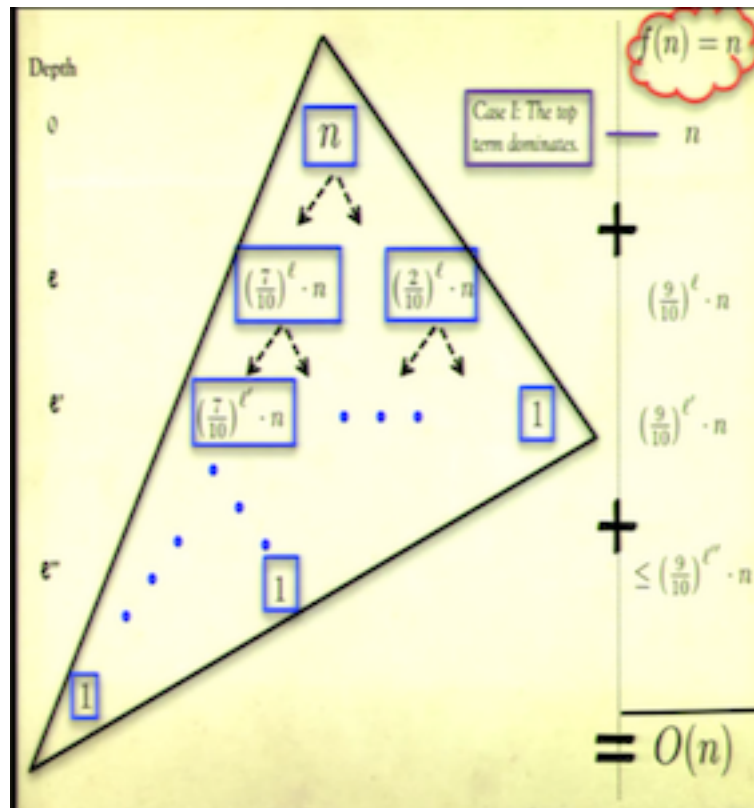
$$\Rightarrow T(n) \leq T\left(\frac{7n}{10}\right) + T\left(\frac{n}{5}\right) + O(n)$$

Note:

1. $T\left(\frac{n}{5}\right)$ term comes from finding the median of the medians.
 2. $T\left(\frac{7n}{10}\right)$ comes from that pivoting on the median of the medians gives a significantly smaller sub-problem.
 3. $O(n)$ comes from breaking in groups of size 5, finding the median of each group, and pivoting on the median of medians.
- But this does **not** fit with the Master Theorem!
 - The problem is not broken into the same sized sub-problems. One is $\frac{7n}{10}$, the other is $\frac{n}{5}$.

- This does not matter as we understand the proof of the Master Theorem,
 \Rightarrow Apply the Recursion Tree Method!





- Runtime: $T(n) = O(n)$
- Thus, we have a deterministic **linear time** algorithm to solve the selection problem (and, specifically, to find the median).
 - The reason why finding the selection problem is useful is that we try to find the median, but when we break it into two sub-problems, we are not finding the medians in the sub-problems since things would be shifted, and we might be finding the n over 10th problem.
 - This works a lot in induction. When you do induction proof, here we are using a stronger algorithm as our sub-routines, using k th selection problem, rather than median problem, which assumes that you have a stronger induction hypothesis. Often we use induction, it is hard to prove an easy result, since we can prove a general result, using induction.