Today

- Order statistics (e.g., finding median)
- Two $O(n)$ time algorithms:
  - Randomized: similar to Quicksort
  - Deterministic: quite tricky
- Both are examples of divide and conquer

Order statistics

Select the $i$th smallest of $n$ elements (the element with rank $i$).
- $i = 1$: minimum;
- $i = n$: maximum;
- $i = \lfloor (n+1)/2 \rfloor$ or $\lceil (n+1)/2 \rceil$: median.

How fast can we solve the problem?
- Min/max: $O(n)$
- General $i$: $O(n \log n)$ by sorting
- We will see how to do it in $O(n)$ time

Randomized Algorithm for Finding the $i$th element

- Divide and Conquer Approach
- Main idea: PARTITION

Example

Select the $i = 7$th smallest:

$\begin{array}{cccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11
\end{array}$

pivot

Partition:

$\begin{array}{cccccc}
2 & 8 & 6 & 8 & 13 & 10 & 11
\end{array}$

Select the $7 - 4 = 3$rd smallest recursively.
Analysis

What is the worst-case running time?

**Unlucky:**
\[ T(n) = T(n - 1) + \Theta(n) \]
\[ T(1) = \Theta(1) \]

Recall that a **lucky** partition splits into arrays with size ratio at most 9:1.

What if all partitions are lucky?

**Lucky:**
\[ T(n) = T(9n / 10) + \Theta(n) \]
\[ n \log_{10} 9 = n^0 = 1 \]

CASE 3

**Expected Running Time**

- The probability that a random pivot induces lucky partition is at least 9/10 (Lecture 4).
- Let \( t_i \) be the number of partitions performed between the \((i-1)\)-th and the \(i\)-th lucky partition.
- The total time is at most:

\[ T = t_0 n + t_1 (9/10)n + t_2 (9/10)^2 n + \ldots \]

- Geometric series

\[ p = (9/10) < 1 \]

\[ E[T] = E[t_1] * [n + (9/10)n + \ldots] \]

\[ = 10/8 * n * [1/(1-p)] \]

\[ = O(n) \]

**Digression: 9 to 1**

- Do we need to define the lucky partition as 9:1 balanced?
- No. Suffices to say that both sides have size \( \geq \alpha n \), for \( 0 < \alpha < \frac{1}{2} \).
- Need constant fraction of \( n \), regardless of what the fraction is. But not constant number.
- Probability of getting a lucky partition is \( 1 - 2\alpha \).

**Partitioning subroutine**

**Invariant:**

\[ x \leq i \leq j \leq r \]

**Summary of randomized order-statistic selection**

- Works fast: linear expected time.
- Excellent algorithm in practice.
- But, the worst case is very bad: \( \Theta(n^2) \).

**Q.** Is there an algorithm that runs in linear time in the worst case?

**A.** Yes, due to [Blum-Floyd-Pratt-Rivest-Tarjan’73].

**IDEA:** Generate a good pivot recursively.

**Worst-case linear-time order statistics**

**Select**(\( i, n \))

1. Divide the \( n \) elements into groups of 5. Find the median of each 5-element group by hand.
2. Recursively **Select** the median \( x \) of the \( \lceil n/5 \rceil \) group medians to be the pivot.
3. Partition around the pivot \( x \). Let \( k = \text{rank}(x) \).
4. If \( i = k \) then return \( x \)
   - else recursively **Select** the \( i\)th smallest element in the lower part
   - else recursively **Select** the \( (i-k)\)th smallest element in the upper part

Same as **Rand-Select**
Choosing the pivot

1. Divide the \( n \) elements into groups of 5. Find the median of each 5-element group by rote.

Analysis

At least half the group medians are \( \leq x \), which is at least \( \lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor \) group medians.

• Therefore, at least \( 3 \lfloor n/10 \rfloor \) elements are \( \leq x \).
**Introduction to Algorithms, Lecture 5**

**Introduction to Algorithms, Lecture 5**

**Order statistics, Median 4**

---

### Analysis

At least half the group medians are \( \leq x \), which is at least \( \lfloor n/5 \rfloor \) group medians.

- Therefore, at least \( 3 \lfloor n/10 \rfloor \) elements are \( \leq x \).
- Similarly, at least \( 3 \lfloor n/10 \rfloor \) elements are \( \geq x \).

### Developing the recurrence

1. Divide the \( n \) elements into groups of 5. Find the median of each 5-element group by rote.
2. Recursively SELECT the median \( x \) of the \( \lfloor n/5 \rfloor \) group medians to be the pivot.
3. Partition around the pivot \( x \). Let \( k = \text{rank}(x) \).
4. If \( i = k \), then return \( x \)
   - Else if \( i < k \), recursively SELECT the \( i \)th smallest element in the lower part
   - Else recursively SELECT the \( (i-k) \)th smallest element in the upper part

### Solving the recurrence

\[
T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + \Theta(n)
\]

**Substitution:**

\[
T(n) \leq \frac{1}{5}cn + \frac{7}{10}cn + \Theta(n)
\]

\[
= \frac{18}{20}cn + \Theta(n)
\]

\[
= cn - \left(\frac{2}{20}cn - \Theta(n)\right)
\]

\[
\leq cn
\]

if \( c \) is chosen large enough to handle the \( \Theta(n) \).

### Conclusions

- Since the work at each level of recursion is a constant fraction \( (9/10) \) smaller, the work per level is a geometric series dominated by the linear work at the root.
- In practice, this algorithm runs slowly, because the constant in front of \( n \) is large.
- The randomized algorithm is far more practical.

**Exercise:** Why not divide into groups of 3?