2 SELECTION

Finding largest: \( n - 1 \) comparisons

Finding second largest: .

- Two scans: \( (n - 1) + (n - 2) = 2n - 3 \) comparisons

- Divide & Conquer:
  
  \[
  W(n) = 2W(n/2) + 2
  \]
  
  \[
  = 2 + 2(2 + 2W(n/2^2))
  \]
  
  \[
  = 2 + 2^2 + 2^2W(n/2^2)
  \]
  
  \[
  = 2 + 2^2 + 2^2(2 + 2W(n/2^3))
  \]
= 2 + 2^2 + 2^3 + 2^3W(n/2^3)

Let \( n = 2^k \),

We know, \( W(2) = 1 \) \( \Rightarrow \) \( W(n/2^{k-1}) = 1 \)

\[
W(n) = 2 + 2^2 + 2^3 + \cdots + 2^{k-1} + 2^{k-1}W(n/2^{k-1})
\]

\[
= 2^1 + 2^2 + \cdots + 2^{k-1} + 2^{k-1}
\]

\[
= (2^k - 2) + 2^{k-1}
\]

\[
= n - 2 + n/2
\]

\[
= 3n/2 - 2
\]

- Using Tournament Tree

To build a tournament tree requires \( n - 1 \) comparisons.

\[
W(n) = n - 1 \text{ for max}
\]

\[
W(n) = n - 1 + (\log_2 n - 1) \text{ for 2max}
\]
2.1 Selection of $k$th smallest/largest

1. Sorting based

\[ W(n) = O(n \log n) \]

2. Tournament tree based

\[ W(n) = O(n + k \log n) = O(n) \text{ for } k \leq \frac{n}{\log n} \]

or \( k \geq \frac{n}{\log n} \)
2.2 A Good Average Case Algorithm

Divide & conquer

\textbf{Selection}(A[p, r], k):
\[
\begin{align*}
    j &= \text{Partition2}(A, p, r) \\
    \textbf{if} \ k < j \\
    \quad &\text{return Selection}(A[p..(j - 1)], k) \\
    \textbf{else if} \ k = j &\textbf{ then return} \ L[j] \\
    \textbf{else} \ \{k > j\} \\
    \quad &\text{return Selection}(A[j + 1..r], k - j)
\end{align*}
\]
\[ W(n) = n - 1 + W(n - 1) = O(n^2) \]
\[ A(n) \approx n - 1 + A(n/2) \]
\[ = (n - 1) + (n/2 - 1) + (n/4 - 1) + \cdots + 1 \]
\[ < 2n \in O(n) \]
(gross simplification)
2.3 Worst-case $O(n)$ algorithm

To improve $W(n)$, we must ensure a good split point.

**Selection’**($A[p, r], k$)

1. Divide $A$ in $\frac{n}{r}$ sublist ($r = 5, 7, \text{ etc.}$), $n = r - p + 1$.
2. Find median of each of the $\frac{n}{r}$ sublists.
3. Recursively find median of these $\frac{n}{r}$ medians.
4. Use median of medians (MM) as the pivot in the previous algorithm for selection:

   Let MM be at index $i$. Swap($A[p], A[i]$)
\( j = \text{Partition2}(A, p, r) \)

5. Choose the appropriate partition for further search:

\[
\text{if } k < j \\
\text{return } \text{Selection'}(A[p..(j - 1)], k) \\
\text{else if } k=j \text{ then return } L[j] \\
\text{else } \{k > j\} \text{ return } \text{Selection'}(A[j + 1..r], k - j)
\]
2.4 Time Complexity

$T(n) \leq cn$ (Steps 1, 2, 4: for finding $n/r$ medians and for partitioning the array based on pivot chosen as median of medians)

$+ T(n/5)$ (for step 3, recursively finding median of $n/5$ medians)

$+ T(3n/4)$ (for recursive call to larger partition)

**To Prove:** Let

$$T(n) \leq cn + T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right)$$
for $n \geq 5$. Then

$$T(n) \leq 20cn.$$ 

Choose $c$ large enough such that $T(n) \leq cn$ for $n \leq 24$ (for list of size less than 5, there is no recursive call).

**Basis:** For $n \leq 24$, by choice of $c$, $T(n) \leq cn \leq 20cn$.

**Hypothesis:** Assume for $k \geq 24$, $T(k) \leq 20ck$.

**Induction:**

To show that $T(k + 1) \leq 20c(k + 1)$.

$$T(k + 1) \leq c(k + 1) + T\left(\frac{k + 1}{5}\right)$$
$$T\left(\frac{3(k + 1)}{4}\right)$$

$$\leq c(k + 1) + 20\left(\frac{k + 1}{5}\right)c$$

$$20\left(\frac{3(k + 1)}{4}\right)c$$

$$= (k + 1)(c + 4c + 15c)$$

$$= 20(k + 1)c$$
How to make quicksort $O(n \log n)$? Use selection algo to find the median.
Use median as partitioning element in Quicksort.
Complexity $T(n) = cn + 2T(n/2) = O(n \log n)$