1 Quick-Sort Review

- The last two lectures we have considered Quick-Sort:
  - Divide $A[1...n]$ (using PARTITION) into subarrays $A' = A[1..q-1]$ and $A'' = A[q+1...n]$ such that all elements in $A''$ are larger than $A[q]$ and all elements in $A'$ are smaller than $A[q]$.
  - Recursively sort $A'$ and $A''$.

- We discussed how split point $q$ produced by PARTITION only depends on last element in $A$.

- We discussed how randomization can be used to get good expected partition point.

- Analysis:
  - Best case ($q = n/2$): $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$.
  - Worst case ($q = 1$): $T(n) = T(1) + T(n-1) + \Theta(n) \Rightarrow T(n) = \Theta(n^2)$.
  - Expected case for randomized algorithm: $\Theta(n \log n)$.

2 Selection

- If we could find element $e$ such that $\text{rank}(e) = n/2$ (the median) in $O(n)$ time we could make quick-sort run in $\Theta(n \log n)$ time worst case.
  - We could just exchange $e$ with last element in $A$ in beginning of PARTITION and thus make sure that $A$ is always partition in the middle.

- We will consider a more general problem than finding the $i$’th element:
  - Selection problem
    
    SELECT($i$) is the $i$’th element in the sorted order of elements

- Note: We do not require that we sort to find SELECT($i$)
- Note: SELECT(1)=minimum, SELECT($n$)=maximum, SELECT($n/2$)=median
Special cases of \( \text{SELECT}(i) \)

- Minimum or maximum can easily be found in \( n - 1 \) comparisons
  - Scan through elements maintaining minimum/maximum
- Second largest/smallest element can be found in \( (n - 1) + (n - 2) = 2n - 3 \) comparisons
  - Find and remove minimum/maximum
  - Find minimum/maximum

- Median:
  - Using the above idea repeatedly we can find the median in time \( \sum_{i=1}^{n/2} (n-i) = n^2/2 - \sum_{i=1}^{n/2} i = n^2/2 - (n/2 \cdot (n/2 + 1))/2 = \Theta(n^2) \)
  - We can easily design \( \Theta(n \log n) \) algorithm using sorting

- Can we design \( O(n) \) time algorithm for general \( i \)?

If we could partition nicely (which is what we are really trying to do) we could solve the problem

- by partitioning and then recursively looking for the element in one of the partitions:

```
SELECT(A, p, r, i)

IF p = r THEN RETURN A[p]
q = PARTITION(A, p, r)

\( k = q - p + 1 \)
IF \( i \leq k \) THEN
    RETURN SELECT(A, p, q, i)
ELSE
    RETURN SELECT(A, q + 1, r, i - k)
FI
```

Select \( i \)'th elements using \( \text{SELECT}(A, 1, n, i) \)

- If the partition was perfect \( (q = n/2) \) we have

\[
T(n) = T(n/2) + n \\
= n + n/2 + n/4 + n/8 + \cdots + 1 \\
= \sum_{i=0}^{\log n} \frac{n}{2^i} \\
= n \cdot \sum_{i=0}^{\log n} \left(\frac{1}{2}\right)^i \\
\leq n \cdot \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i \\
= \Theta(n)
\]
Note:

* The trick is that we only recurse on one side.
* In the worst case the algorithm runs in \( T(n) = T(n-1) + n = \Theta(n^2) \) time.
* We could use randomization to get good expected partition.
* Even if we just always partition such that a constant fraction (\( \alpha < 1 \)) of the elements are eliminated we get running time \( T(n) = T(\alpha n) + n = n \sum_{i=0}^{\log n} \alpha^i = \Theta(n) \).

- It turns out that we can modify the algorithm and get \( T(n) = \Theta(n) \) in the worst case
  - The idea is to find a split element \( q \) such that we always eliminate a fraction of the elements:

    ```
    \text{SELECT}(i)
    * Divide \( n \) elements into groups of 5
    * Select median of each group (\( \Rightarrow \lceil \frac{n}{5} \rceil \) selected elements)
    * Use SELECT recursively to find median \( q \) of selected elements
    * Partition all elements based on \( q \)
    
    ![Diagram](https://example.com/diagram.png)
    
    * Use SELECT recursively to find \( i \)’th element
      - If \( i \leq k \) then use SELECT\((i)\) on \( k \) elements
      - If \( i > k \) then use SELECT\((i-k)\) on \( n-k \) elements
    
    - If \( n' \) is the maximal number of elements we recurse on in the last step of the algorithm the running time is given by \( T(n) = \Theta(n) + T(\lceil \frac{n}{5} \rceil) + \Theta(n) + T(n') \)

- Estimation of \( n' \):
  - Consider the following figure of the groups of 5 elements
    * An arrow between element \( e_1 \) and \( e_2 \) indicates that \( e_1 > e_2 \)
    * The \( \lceil \frac{n}{5} \rceil \) selected elements are drawn solid (\( q \) is median of these)
    * Elements \( > q \) are indicated with box

![Diagram](https://example.com/diagram.png)
- Number of elements > \( q \) is larger than \( 3\left(\frac{1}{2}\left\lceil \frac{n}{5} \right\rceil - 2\right) \geq \frac{3n}{10} - 6 \)
  * We get 3 elements from each of \( \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \) columns except possibly the one containing \( q \) and the last one.
- Similarly the number of elements < \( q \) is larger than \( \frac{3n}{10} - 6 \)

\[ \downarrow \]

We recurse on at most \( n' = n - \left(\frac{3n}{10} - 6\right) = \frac{7}{10}n + 6 \) elements.

- So Selection(i) runs in time \( T(n) = \Theta(n) + T(\left\lceil \frac{n}{5} \right\rceil) + T(\frac{7}{10}n + 6) \)

- Solution to \( T(n) = n + T(\left\lceil \frac{n}{5} \right\rceil) + T(\frac{7}{10}n + 6) \):
  - Guess \( T(n) \leq cn \)
  - Induction:

\[
T(n) = n + T(\left\lceil \frac{n}{5} \right\rceil) + T(\frac{7}{10}n + 6)
\leq n + c \cdot \left\lceil \frac{n}{5} \right\rceil + c \cdot \left(\frac{7}{10}n + 6\right)
\leq n + c \cdot \frac{n}{5} + c + \frac{7}{10}cn + 6c
= \frac{9}{10}cn + n + 7c
\leq cn
\]

If \( 7c + n \leq \frac{1}{10}cn \) which can be satisfied (e.g. true for \( c = 20 \) if \( n > 140 \))

- Note: It is important that we chose every 5’th element, not all other choices will work (homework).