The Selection Problem

- **Definition**
  - Given an array $L$ containing $n$ keys, find the $i$th smallest (or largest) key in $L$ ($1 \leq i \leq n$).

- **Different cases**
  - if $i = 1$, find the smallest key
  - if $i = 2$, find the second smallest key
  - by **median**, we mean:
    $$i = \begin{cases} 
      (n+1)/2 & \text{if } n \text{ is odd} \\
      \lfloor (n+1)/2 \rfloor & \text{if } n \text{ is even} 
    \end{cases}$$
    (tell the difference between median and average).
  - if $i = n$, find the largest key
First Try: Sorting

- The solution is trivial:
  1. Sort the sequence.
  2. Choose the \( i \)th element from the sorted sequence.

- What is the complexity?

Can we do better than this?
Problem 1: Finding the smallest key

MINNUM(A)

\[
\text{min} := A[1];
\]

\[
\text{for } i := 2 \text{ to } n \text{ do}
\]

\[
\quad \text{if } (\text{min} > A[i])
\]

\[
\quad \text{min} := A[i];
\]

\[
\text{return min;}
\]

Complexity: \( n - 1 \) comparisons (Note: this is the exact running time, not an asymptotic one)
Problem 2: Find the minimum and maximum simultaneously (straightforward way)

FIND-BOTH(A)

\[
\begin{align*}
\text{min} & := A[1]; \\
\text{max} & := A[1]; \\
\text{for } i := 2 \text{ to } n \text{ do} \\
& \quad \text{if } (\text{min} > A[i]) \\
& \quad \quad \text{min} := A[i]; \\
& \quad \text{if } (\text{max} < A[i]) \\
& \quad \quad \text{max} := A[i]; \\
\text{return min, max;}
\end{align*}
\]

Complexity: \(2(n - 1)\) comparisons (same as finding the largest and smallest keys independently)
Can we do better?

A smarter way:

- Pair the keys and find the minimum and maximum in each pair (about $n/2$ comparisons)

- Collect the smaller keys in a list and find the smallest (about $n/2$ comparisons)

- Collect the larger keys in a list and find the largest (about $n/2$ comparisons)

- Total number of comparisons:
Algorithm

**FIND-BOTH-SMARTER(A, n)**

if $n$ is odd
  $k := 2$;
else
  $k := 3$
  else

for $i := k$ to $n - 1$ by 2 do
  if $A[i] > A[i+1]$
    exchange $A[i]$ and $A[i+1]$;
  if $A[i] < min$
    $min := A[i]$;
  if $A[i+1] > max$
    $max := A[i+1]$;
What makes the difference here?

Using the ordinary way, each pair require 4 comparisons. With the “smarter” way, the number of comparisons is reduced to 3.
Problem 3: Find the $i$th smallest key

Idea: Divide and Conquer

Divide: split the input array recursively (using the routine “Partition” (in QuickSort))

Conquer: recursively solve ONE sub-problem (Process only the subarray which contains the $i$th smallest key (note that QuickSort processes both subarrays!))

Combine: no need to combine
Algorithm: first try

**SELECT**(A, p, r, i) /*Find the \( i \)th smallest element in \( A[p..r] \) */

if \( p == r \) return;

\( q \) := Partition(A, p, r);

\( k := q - p + 1; \)

if \( i == k \)
  return \( A[q] \);
else if \( i < k \)
  return Select(A, p, q-1, i);
else
  return Select(A, q + 1, r, i-k);
• If the partition is balanced \((q = n/2)\), we have \(T(n) = ?\)

• Worst Case, when Partition always results in 2 subarrays with 0 and \(n - 1\) elements: \(T_w(n) = ?\)

When will the worst-case happen?
Second Try: Selection in Worst-Case linear time

Basic Idea: to find a split element \( q \) such that we always eliminate a fraction \( \alpha \) of the elements:

\[
T(n) \leq T((1-\alpha)n) + \Theta(n) \quad \text{then} \quad T(n) = O(n)
\]

- For example, each time, if we can guarantee to eliminate at least 10% elements, then \( T(n) \leq T(0.9n) + cn \).

Since \( T'(n) = T'(0.9n) + cn \Rightarrow T'(n) = \Theta(n) \),

Then \( T(n) \leq T(0.9n) + cn \Rightarrow T(n) = O(n) \).
**Selection with Linear Time in Worst-Case**

**SELECT**(i)

1. Divide $n$ elements into groups of 5.
2. Select median of each group ($\Rightarrow \lceil \frac{n}{5} \rceil$ selected elements)
3. Use **SELECT** recursively to find median $q$ of the medians
4. Partition the array (all elements) based on $q$

```
\[ q \]
\[ \text{k} \quad \text{n-k} \]
```

5. Use **SELECT** recursively to find $i$th element
   - if $i == k$, we are done
   - if $i < k$, then **SELECT**(i) on $k - 1$ elements
   - if $i > k$, then **SELECT**(i - k) on $n - k$ elements
How the algorithm works

array of medians

median of medians

low

high

median of medians

<median of medians

>median of medians

CS404/504

Computer Science

Design and Analysis of Algorithms: Lecture 11
As our first step in the analysis, we are going to find a lower bound on the number of elements that are greater than the partitioning element $s$.

- at least $\frac{1}{2}$ of the medians found in step 2 are greater than or equal to $s$;

- at least $\frac{1}{2}$ of the $\lceil \frac{n}{5} \rceil$ groups contribute 3 elements that are greater than $s$, except for the one group that has fewer than 5 elements and the one group containing $s$ itself;

- Thus the number of elements $> s$ is at least $3\left(\lceil \frac{1}{2} \frac{n}{5} \rceil - 2\right) \geq \frac{3}{10}n - 6$; (Note: “3” is from “contribute 3 elements”; “$\lceil \rceil$” is from “at least”; “$\frac{n}{5}$” is the total number of groups, “-2” is from “except 2 groups”.)
• Similarly, the number of elements that are < $s$ is at least $\frac{3n}{10} - 6$.

• So no matter which sub-array is picked to continue the search, at least $\frac{3n}{10} - 6$ elements will be eliminated; Equivalently to say, the next call for SELECT will have an input size no bigger than $\frac{7n}{10} + 6$. 
Linear Time Selection: An Example

Select \((i=7, n=25)\)

\[
\begin{array}{cccccc}
24 & 12 & 9 & 21 & 2 \\
17 & 13 & 4 & 23 & 18 \\
1 & 6 & 19 & 16 & 10 \\
25 & 22 & 3 & 5 & 7 \\
8 & 11 & 14 & 15 & 20 \\
\end{array}
\]
Example, cont’d

Step 1:

Break the Array \( a \) into \( \lceil \frac{n}{5} \rceil = 5 \) groups of 5.

Step 2:

Sort each group of 5 elements using the insertion sort. This can be done using 8 comparisons.

\[
\begin{array}{cccccc}
2 & 4 & 1 & 3 & 8 \\
9 & 13 & 6 & 5 & 11 \\
12 & 17 & 10 & 7 & 14 \\
21 & 18 & 16 & 22 & 15 \\
24 & 23 & 19 & 25 & 20 \\
\end{array}
\]
Step 3:

Find the median of median of medians found in step 2. 12 is the median of medians in this case.

Step 4:

Partition the array about the median of medians.

**Lower side:**  2 9 12 1 6 10 3 5 7 11 4 8

**Upper side:**  21 24 17 18 23 14 15 20 16 19 22 25 13

So, $k = 12$
Step 5:

Call select recursively on

2 9 12 1 6 10 3 5 7 11 4 8

with \( i = 7 \)

As we saw last time, both the low side and high side of the partition have at most \( \frac{7n}{10} + 6 \) elements.
Complexity

Step 1: Divide elements into groups of 5; $\Theta(n)$

Step 2: To find the median of 5 elements requires constant time; total $\lceil \frac{n}{5} \rceil$ groups, so $\Theta(n)$.

Step 3: Total $\lceil \frac{n}{5} \rceil$ medians; To find the median of medians (a selection problem): $T(\lceil \frac{n}{5} \rceil)$

Step 4: Partition takes linear time: $\Theta(n)$.

Step 5: Recursively call SELECT with input size equal or smaller than $\frac{7n}{10} + 6$, complexity for this step: $\leq T(\frac{7n}{10} + 6)$.

Overall:

$$T(n) \leq T(\frac{7n}{10} + 6) + T(\lceil \frac{n}{5} \rceil) + \Theta(n)$$
Note:

\[ \frac{7n}{10} + 6 < n \text{ for all } n > 20 \text{ and let's take } n \leq 140 \text{ (nothing special about 140, you will see) as small size problems, and it takes constant time to solve them } O(1). \]

We will use the following recurrence relation for \( T(n) \):

\[
T(n) \leq \begin{cases} 
\Theta(1) & \text{if } n \leq 140 \\
T(\lceil \frac{n}{5} \rceil) + T(\frac{7n}{10} + 6) + \Theta(n) & \text{if } n > 140
\end{cases}
\]

We can show that \( T(n) = O(n) \) by substitution.
Proof using the Substitution Method:

Basis:
Assume that $T(n) \leq cn$ for some constant $c$ and all $n \leq 140$. This is true by assumption. (However, we have not specified $c$, yet).

Induction Step
Assume that $T(n) \leq cn$ holds for all $1 \leq n \leq k - 1$, or all numbers in $\{1, 2, \ldots, k - 1\}$,
Induction Step

We want to show that $T(n) \leq cn$ also holds for $n = k$, or $T(k) \leq ck$

$$T(k) \leq T(\lceil \frac{k}{5} \rceil) + T(\frac{7k}{10} + 6) + ak$$

$$\leq c\lceil \frac{k}{5} \rceil + c(\frac{7k}{10} + 6) + ak \quad \text{(by Induction Hypothesis, and because $\lceil \frac{k}{5} \rceil$ and $\frac{7k}{10} + 6$ are both in $\{1, 2, .. k-1 \}$)}$$

$$\leq c(\frac{k}{5} + 1) + c(\frac{7k}{10} + 6) + ak \quad \text{(by the definition of $\lceil \cdot \rceil$)}$$

$$= 9ck/10 + 7c + ak$$

$$= ck + (-ck/10 + 7c + ak)$$
• We want to prove that: \( \exists \ c, \text{ such that } T(k) \leq ck; \)

We can get this done by simply check if it is possible that 
\((-ck/10 + 7c + ak) \leq 0.\)

When \( n > 70, \ (-ck/10 + 7c + ak) \leq 0 \Leftrightarrow c \geq \frac{10ak}{k-70}, \)

so here (assume \( n > 140), \) we can choose a constant \( c \geq 20a, \)

then \( T(k) \leq ck. \) End of proof.

(Note: nothing special with 140; we could replace it by any integer strictly greater than 70 and then choose \( c \) accordingly)