The Selection Problem

- The $i^{th}$ order statistic of a set of $n$ elements is the $i^{th}$ smallest element.

- The selection problem is specified as follows:

  **Input:** A set $A$ of $n$ (distinct) numbers and a number $i$, with $1 \leq i \leq n$.

  **Output:** The element $x \in A$ that is larger than exactly $i-1$ other elements of $A$. 
Finding the Minimum Element

$$\text{Minimum}(A)$$

$$\text{min} = A[1]$$

$$\text{for } i = 2 \text{ to } A.\text{length}$$

$$\text{if } \text{min} > A[i]$$

$$\text{min} = A[i]$$

$$\text{return } \text{min}$$

Is That the Best We Can Do?

**Remark**

A decision tree for the MIN problem must have at least $$n$$ leaves since any one of the $$n$$ keys may be the output.
That Don’t Impress Me Much

- The decision tree model yields a $\Omega(lg n)$ lower bound for finding the minimum value.

- Surely any algorithm that returns a minimum value requires at least $\Omega(n)$ steps.

Lower Bound on Finding the Minimum

**Lemma**
Any algorithm to find the min of $n$ keys by comparisons of keys must do at least $n-1$ comparisons.

**Proof**
In any list with $n$ distinct entries, $n-1$ entries are not minimum. Any entry is not minimum if it is larger than at least one other entry on the list.

Hence, $n-1$ entries must be “winners” in comparisons done by algorithm $A$.

Each comparison has only one winner.
Simultaneous Minimum and Maximum

Remark
Finding the minimum and maximum elements independently requires $2n-2$ comparisons.

Challenge
Can we do better?

Strategy for Simultaneous Min/Max

1. Maintain min and max elements seen so far
2. Compare each pair of input elements to each other.
3. Compare smaller of two inputs to min & update.
4. Compare larger of two inputs to max & update.
Selection in Expected Linear Time

To find $i^{th}$ smallest element:
- $p = \text{Randomized-Partition}(A, lo, hi)$

```
if i = p-lo+1
    return A[p]
elseif i < p-lo+1
    search bottom half of array for i
else search top half of array for i-(p-lo+1)
```

Randomized-Select

```
Randomized-Select(A, lo, hi, i)
if lo = hi
    return A[lo]
p = Randomized-Partition(A, lo, hi)
if i = p-lo+1
    return A[p]
elseif i < p-lo+1
    return Randomized-Select(A, lo, p-1, i)
else return Randomized-Select(A, p+1, hi, i-(p-lo+1))
```
An Unhappy Choice in the Worst Case...

- $n-1$ elements

A Happy Choice on Average...

- $\lfloor n/2 \rfloor$
- $\lceil n/2 \rceil$
Selection in Worst-Case Linear Time

Select(A, k)

1. Divide the keys into sets of size five and find the median of each set via Insertion Sort.

![Diagram of selecting median from sets of size five]
Selection in Worst-Case Linear Time

Select(A, k)

1. Divide the keys into sets of size five and find the median of each set via Insertion Sort.

2. Apply Select recursively to find median \( m^* \) of \( \lceil n/5 \rceil \) medians.

Smaller to Left; Larger to Right

3. Compare each key in A and D to \( m^* \)
   - Left = C \( \cup \) (keys from A\( \cup \)D < \( m^* \))
   - Right = B \( \cup \) (keys from A\( \cup \)D > \( m^* \))
Smaller to Left; Larger to Right

3. Compare each key in A and D to \( m^* \)
   \[\text{Left} = C \cup \{\text{keys from } A \cup D < m^*\}\]
   \[\text{Right} = B \cup \{\text{keys from } A \cup D > m^*\}\]

4. if \(|\text{Left}|+1 = k\)
   return \( m^* \)
else if \( k \leq |\text{Left}|\)
   return Select(Left, \( k \))
else return Select(Right, \( k-|\text{Left}|+1 \))

Bounded the Size of Recursive Calls

\[\text{Left} = C \cup \{\text{keys from } A \cup D < m^*\}\]
\[\text{Right} = B \cup \{\text{keys from } A \cup D > m^*\}\]
Bounded the Size of Recursive Calls

\[ \text{Left} = C \cup \{\text{keys from } A \cup D < m^*\} \]
\[ \text{Right} = B \cup \{\text{keys from } A \cup D > m^*\} \]
Bounded the Size of Recursive Calls

Left = \{keys from A \cup D < m^*\}
Right = \{keys from A \cup D > m^*\}