k-Selection; Median

• Select the $k$-th smallest item in the list
• Naïve Solution
  - Sort;
  - pick the $k$-th smallest item in sorted list.
    \(O(n \log n)\) time complexity
• Randomized solution: Average case \(O(n)\)
• Improved Solution: worst case \(O(n)\)
QuickSort(A, p, r)
  if (p < r) then
    q = Partition(A, p, r)
    QuickSort(A, p, q)
    QuickSort(A, 1+1, r)

Partition(A, p, r)
  x = A[p]
  i = p-1
  j = r+1
  while TRUE do
    repeat
      j--
    until (A[j] <= x)
    repeat
      i++
    until (A[i] >= x)
    if (i < j) SWAP(A[i], A[j])
    else return j
The Partition code can be rewritten so that it accepts another parameter, namely, the pivot value. Let’s call this new variation as PivotPartition. This change does not affect its time complexity. RandomizedPartition as used in RandomizedSelect picks the pivot uniformly at random from among the elements in the list to be partitioned.
Randomized Selection

**RandomizedSelect**\((A, p, r, i)\)

if \((p = r)\) then
   return \(A[p]\)

\(q = \text{RandomizedPartition}(A, p, r)\)

\(k = q - p + 1\)

if \((i <= k)\)
   return **RandomizedSelect**\((A, p, q, i)\)

else
   return **RandomizedSelect**\((A, q+1, r, i-k)\)
Randomized Selection: Rewritten

```plaintext
RandomizedSelect(A, p, r, i)
    if (p = r) then
        return A[p]
    Pivot = A[random(p,r)]
    q = PivotPartition(A, p, r, Pivot)
    k = q - p + 1
    if (i <= k)
        return RandomizedSelect(A, p, q, i)
    else
        return RandomizedSelect(A, q+1, r, i-k)
```
k-Selection & Median: Improved Algorithm

- Start with initial array
k-Selection & Median: Improved Algorithm (Cont’d)

- Use median of medians as pivot

\[ T(n) < O(n) + T(n/5) + T(3n/4) \]
Improved Selection

**ImprovedSelect**\(A, p, r, i\)

- if \((p = r)\) then
  - return \(A[p]\)
- else \(N = r - p + 1\)
- Partition \(A[p..r]\) into subsets of 5 elements and collect all the medians of the subsets in \(B[1..(N/5)]\).

Pivot = **ImprovedSelect** \((B, 1, \lceil N/5 \rceil, \lceil N/10 \rceil)\)

\(q = \text{PivotPartition}(A, p, r, \text{Pivot})\)

\(k = q - p + 1\)

- if \((i \leq k)\)
  - return **ImprovedSelect**\((A, p, q, i)\)
- else
  - return **ImprovedSelect**\((A, q+1, r, i-k)\)
Upper & Lower Bounds

- Algorithm A solves problem P if it terminates and gives the correct output on every possible input.
- Algorithm A solving problem P has time complexity $f(n)$ if it takes time at most $f(n)$ for every input of length $n$.
- $U(n)$ is an upper bound on the time complexity of P, if there exists an algorithm A that solves P and has time complexity $U(n)$.
- $L(n)$ is a lower bound on the time complexity of P, if there exists NO algorithm that solves P and has time complexity asymptotically less than $L(n)$.
Naïve Algorithm $A$ solves the Maximum problem, because it terminates in $n$ iterations for every possible input of length $n$ and outputs the correct maximum.

Naïve Algorithm $A$ has time complexity $O(n)$.

$O(n)$ is an upper bound on the time complexity of the maximum problem.

$(n-1)$ is a lower bound on the time complexity of the maximum problem, because there exists NO algorithm that solves it with less than $n-1$ comparisons.

WHY? In 1 comparison, at most 1 item is eliminated from being the maximum. How many to eliminate?

Therefore, no matter how smart you are you cannot design an algorithm that solves the Maximum problem in less than $n-1$ comparisons on all inputs of length $n$. 
Upper Bound on Sorting $n$ items

- $O(n \log n)$ is the upper bound for sorting.
- **WHY?**
  - HeapSort
  - MergeSort
- **What about QuickSort?**
  - $O(n^2)$ in the worst case!
Lower Bound for Sorting: Decision Tree Model

- The **decision tree model** models all comparison-based algorithms that solve the **sorting** problem. These algorithms perform no other "algebraic" operations on input values. They may perform data movements & other statements.
- Imagine a binary tree that models the algorithm, where
  - each node corresponds to a **comparison**
  - the edges to the children correspond to the two outcomes of the comparison: YES/NO
  - Leaves correspond to the output. **WHAT IS THE OUTPUT?**
- Decision tree for InsertionSort on 4 items?
- What can we say about such **decision** trees?
- Given an input, the algorithm follows a path from the root to a leaf.
Lower Bound for Sorting: Cont’d

- Leaves correspond to outputs.
- Paths correspond to a path followed on a specific input. Time complexity = height of decision tree.
- Different input orders must force different paths or else the output will end up being the same, giving rise to incorrect sorted orders.
- Therefore number of leaves is at least as large as the number of different input orders.
  - **HOW MANY?**
    - \( n! \)
- Height of the decision tree is at least \( \log(n!) \). Hence lower bound is \( O(\log(n!)) = O(n \log n) \)