Computation Tree for A on n inputs

- Assume A is a comparison-based sorting alg
- Every node represents a comparison between two items in A
- Branching based on result of comparison
- Leaf corresponds to algorithm halting with output
- Every input follows a path in tree
- Different inputs follow different paths
- Time complexity on input x = depth of leaf where it ends on input x
Upper Bounds on Time Complexity

- Time complexity of algorithm A to solve problem P on specific input x
  - Simply count the steps $T(A, x) = \text{lower and upper bound}$
- Time complexity of algorithm A to solve problem P on any input of length N
  - $T_A(N) = \max_x T(A, x)$
- Time complexity of prob P on any input of length N
  - Time complexity of best algorithm to solve problem P on any input of length N
Lower Bounds on Time Complexity

- Lower Bound on Time complexity of algorithm A to solve problem P on any input of length N
  - Some function that lower bounds the time complexity of A on inputs of length N
- Time complexity of prob P on any input of length N
  - Lower bound on Time complexity of best algorithm to solve problem P on worst case input of length N
A sort is **stable** if equal elements appear in the same order in both the input and the output.

Which sorts are stable?
k-Selection; Median

- Select the $k$-th smallest item in list
- Naïve Solution
  - Sort;
  - pick the $k$-th smallest item in sorted list. 
    $O(n \log n)$ time complexity
- Idea: Modify Partition from QuickSort
  - How?
- Randomized solution: Average case $O(n)$
- Improved Solution: worst case $O(n)$
k-Selection Time Complexity

- Perform Partition from QuickSort (assume all unique items)
- $\text{Rank}(\text{pivot}) = 1 + \# \text{ of items that are smaller than pivot}$
- If $\text{Rank}(\text{pivot}) = k$, we are done
- Else, recursively perform k-Selection in one of the two partitions

- On the average:
  - $\text{Rank}(\text{pivot}) = n / 2$
- Average-case time
  - $T(N) = T(N/2) + O(N)$
  - $T(N) = O(N)$
- Worst-case time
  - $T(N) = T(N-1) + O(N)$
  - $T(N) = O(N^2)$

```
Partition(array A, int p, int r)
1 x ← A[r]            ▷ Choose pivot
2 i ← p - 1
3 for j ← p to r - 1
4     do if (A[j] ≤ x)
5         then i ← i + 1
7 exchange A[i + 1] ← A[r]
8 return i + 1
```
Randomized Solution for k-Selection

- Uses RandomizedPartition instead of Partition
  - RandomizedPartition picks the pivot uniformly at random from among the elements in the list to be partitioned.
- Randomized k-Selection runs in O(N) time on the average
- Worst-case behavior is very poor O(N^2)
k-Selection & Median: Improved Algorithm

- Start with initial array
QuickSelect (A) & Improved Median (B)

A

Input Array

 Partition

Pivot

q = Rank(Pivot)

Discard R/L

Q < k

B

Input Array

Median of (Medians of 5)

Partition

Pivot

q = Rank(Pivot)

Discard R/L

Q < k
k-Selection & Median: Improved Algorithm

- Use median of medians as pivot

- $T(n) < O(n) + T(n/5) + T(3n/4)$
ImprovedSelect

\[
\text{IMPROVEDSELECT}(\text{array } A, \text{int } k, \text{int } p, \text{int } r)
\]
\[
\quad \triangleright \text{ Select } k\text{-th largest in subarray } A[p..r]
\]
1. if \((p = r)\)
2. then return \(A[p]\)
3. else \(N \leftarrow r - p + 1\)
4. Partition \(A[p..r]\) into subsets of 5 elements and collect all medians of subsets in \(B[1..\lceil N/5 \rceil]\).
5. \(Pivot \leftarrow \text{IMPROVEDSELECT}(B, 1, \lceil N/5 \rceil, \lceil N/10 \rceil)\)
6. \(q \leftarrow \text{PIVOTPARTITION}(A, p, r, Pivot)\)
7. \(i \leftarrow q - p + 1\)  \(\triangleright\) Compute rank of pivot
8. if \((i = k)\)
9. then return \(A[q]\)
10. if \((i > k)\)
11. then return \(\text{IMPROVEDSELECT}(A, k, p, q - 1)\)
12. else return \(\text{IMPROVEDSELECT}(A, k - i, q + 1, r)\)
PivotPartition

\[
\text{PivotPartition}(\text{array } A, \text{ int } p, \text{ int } r, \text{ item } \text{Pivot})
\]

\[
\text{\hspace{1cm} Partition using provided Pivot}
\]

\[
1 \quad i \leftarrow p - 1
\]

\[
2 \quad \text{for } j \leftarrow p \text{ to } r
\]

\[
3 \quad \text{do if } (A[j] \leq \text{Pivot})
\]

\[
4 \quad \text{then } i \leftarrow i + 1
\]

\[
5 \quad \text{exchange } A[i] \leftarrow A[j]
\]

\[
6 \quad \text{return } i + 1
\]
Data Structure Evolution

- Standard operations on data structures
  - Search
  - Insert
  - Delete

- Linear Lists
  - Implementation: Arrays (Unsorted and Sorted)

- Dynamic Linear Lists
  - Implementation: Linked Lists

- Dynamic Trees
  - Implementation: Binary Search Trees
BST: Search

\[
\text{TreeSearch}(node \ x, \ key \ k) \\
\quad \triangleright \text{Search for key } k \text{ in subtree rooted at node } x \\
1 \quad \text{if } ((x = \text{NIL}) \text{ or } (k = \text{key}[x])) \text{ then return } x \\
2 \quad \text{if } (k < \text{key}[x]) \text{ then return TreeSearch(left}[x], k) \\
3 \quad \text{else return TreeSearch(right}[x], k) \\
\]

\text{Time Complexity: } \mathcal{O}(h) \\
\text{Not } \mathcal{O}(\log n) \quad \text{— Why?}

h = \text{height of binary search tree}
BST: Insert

```
TREE_INSERT(tree T, node z)
  ▷ Insert node z in tree T
1  y ← NIL
2  x ← root[T]
3  while (x ≠ NIL)
4    do y ← x
5    if (key[z] < key[x])
6      then x ← left[x]
7      else x ← right[x]
8  p[z] ← y
9  if (y = NIL)
10    then root[T] ← z
11  else if (key[z] < key[y])
12    then left[y] ← z
13    else right[y] ← z
```

Time Complexity: O(h)

h = height of binary search tree

Search for x in T

Insert x as leaf in T
BST: Delete

Time Complexity: $O(h)$

$h = \text{height of binary search tree}$

Set $y$ as the node to be deleted. It has at most one child, and let that child be node $x$. If $y$ has one child, then $y$ is deleted and the parent pointer of $x$ is fixed. The child pointers of the parent of $x$ is fixed. The contents of node $z$ are fixed.

\[
\text{TreeDelete}(tree\ T,\ node\ z)\\ 1. \text{if } ((\text{left}[z] = \text{NIL}) \text{ or } (\text{right}[z] = \text{NIL})) \\
2. \quad \text{then } y \leftarrow z \\
3. \quad \text{else } y \leftarrow \text{Tree-Successor}(z) \\
4. \text{if } (\text{left}[y] \neq \text{NIL}) \\
5. \quad \text{then } x \leftarrow \text{left}[y] \\
6. \quad \text{else } x \leftarrow \text{right}[y] \\
7. \text{if } (x \neq \text{NIL}) \\
8. \quad \text{then } p[x] \leftarrow p[y] \\
9. \text{if } (p[y] = \text{NIL}) \\
10. \quad \text{then } \text{root}[T] \leftarrow x \\
11. \quad \text{else if } (y = \text{left}[p[y]]) \\
12. \quad \quad \text{then } \text{left}[p[y]] \leftarrow x \\
13. \quad \quad \text{else } \text{right}[p[y]] \leftarrow x \\
14. \text{if } (y \neq z) \\
15. \quad \text{then } \text{key}[z] \leftarrow \text{key}[y] \\
16. \quad \cop y's\ satellite\ data\ into\ z \\
17. \text{return } y
\]