COT 5407: Introduction to Algorithms

Giri NARASIMHAN

www.cs.fiu.edu/~giri/teach/5407S19.html

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) = 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 complxty 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

Stable Sort

- 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)
- Rank(pivot) = 1 + # of items that are smaller than pivot
- If Rank(pivot) = k, we are done
- Else, recursively perform k-Selection in one of the two partitions

- On the average:
  - Rank(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)
  - $T(N) = O(N^2)$

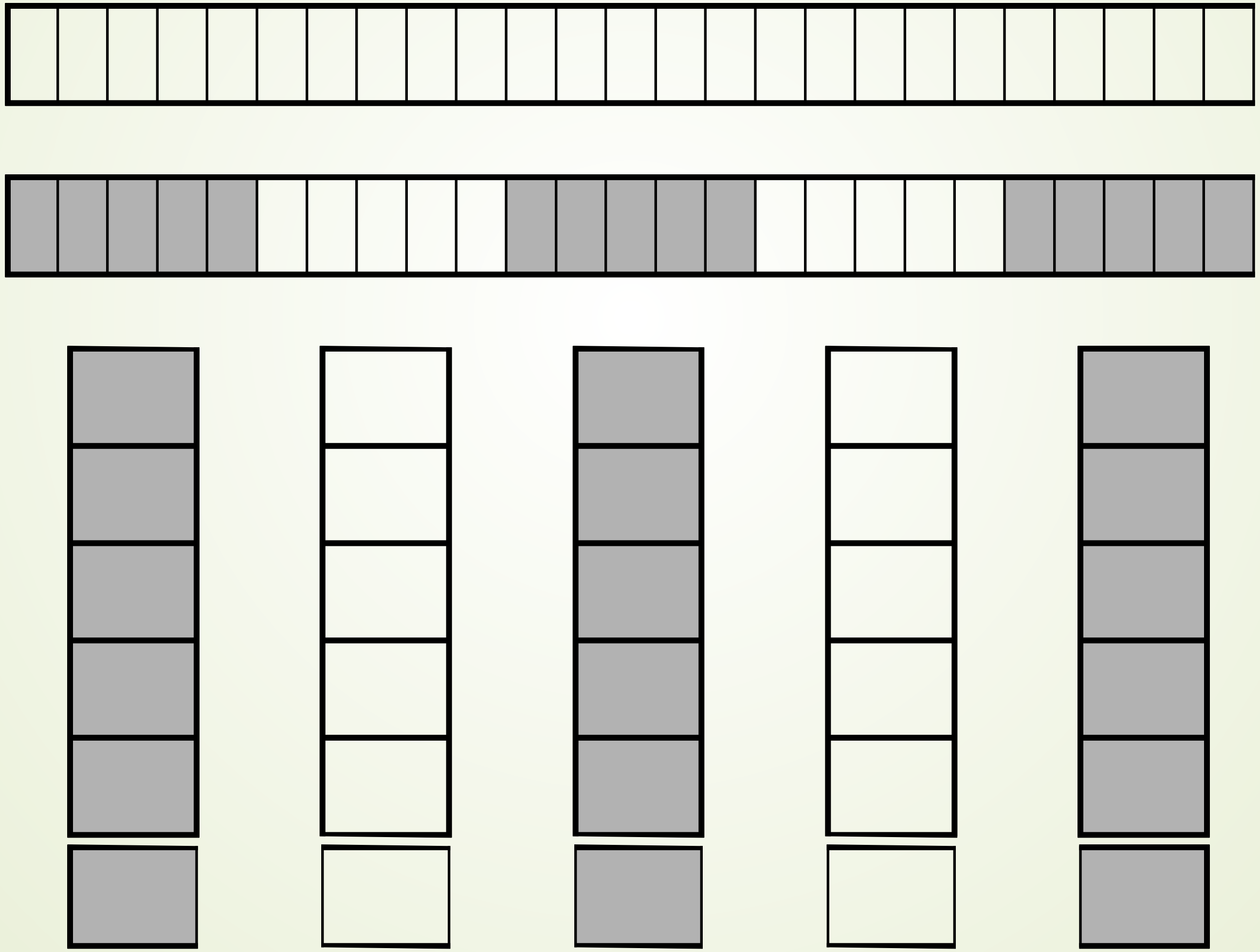
```
PARTITION(array A, int p, int r)
x ← A[r]                          ▷ Choose pivot
i ← p − 1
for j ← p to r − 1
        do if (A[j] ≤ x)
              then i ← i + 1
                   exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

Randomized Solution for k-Selection

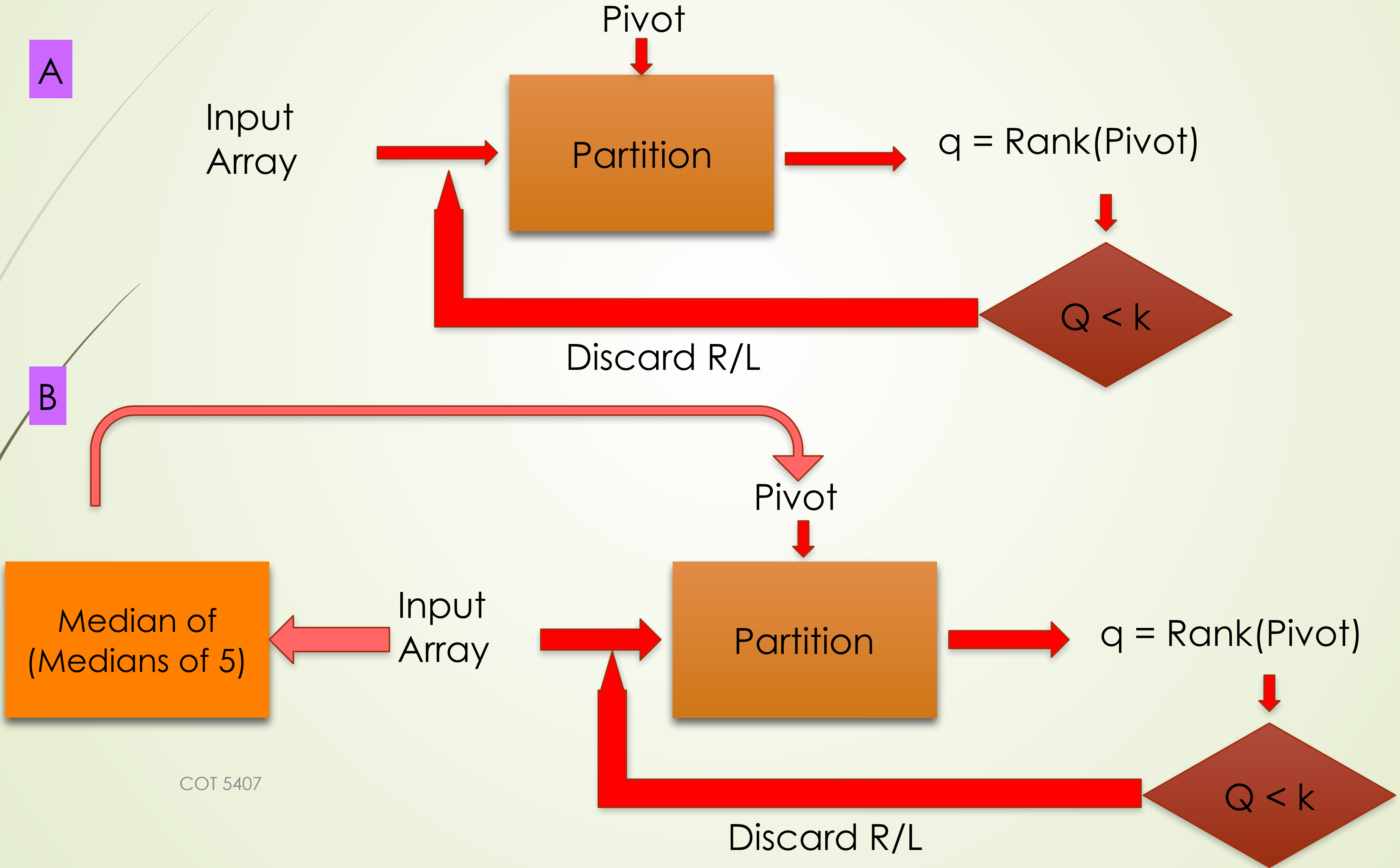
- Uses RandomizedPartition instead of Partition
  - RandomizedPartition picks the pivot uniformly at random from among the elements in the list to be partitioned.
- Randomized k-Selection runs in O(N) time on the average
- Worst-case behavior is very poor $O(N^2)$

k-Selection & Median: Improved Algorithm

- Start with initial array

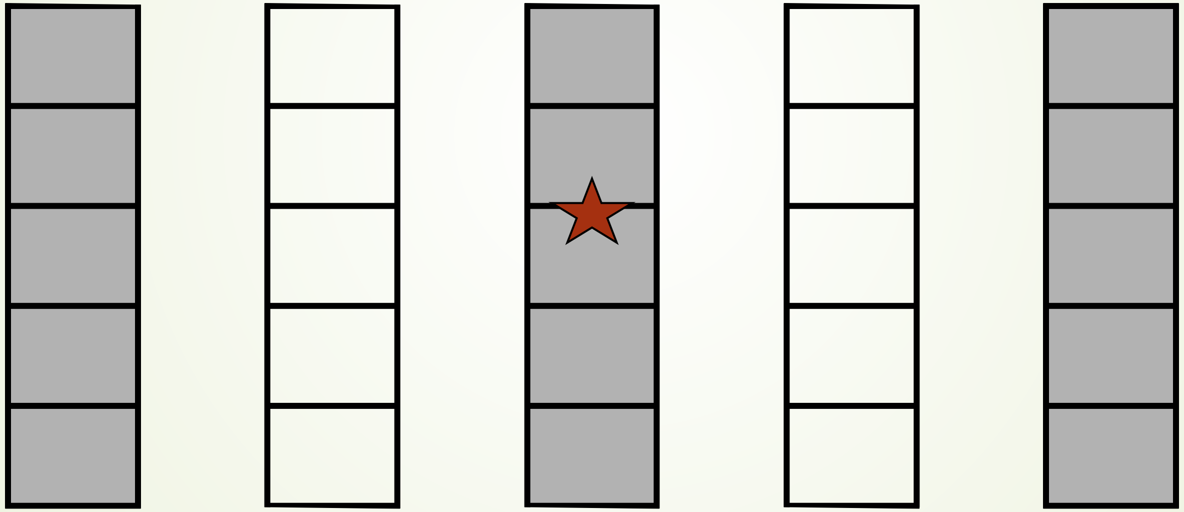


QuickSelect (A) & Improved Median (B)



k-Selection & Median: Improved Algorithm

- Use median of medians as pivot



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

ImprovedSelect

$\textsc{ImprovedSelect}(array\ A, int\ k, int\ p, int\ r)$

$\quad \triangleright$ Select k-th largest in subarray $A[p..r]$

if $(p = r)$

$\quad$ **then return** $A[p]$

$\quad$ **else** $N \leftarrow r - p + 1$

Partition $A[p..r]$ into subsets of 5 elements and collect all medians of subsets in $B[1..\lceil N/5 \rceil]$.

$Pivot \leftarrow \textsc{ImprovedSelect}(B, 1, \lceil N/5 \rceil, \lceil N/10 \rceil)$

$q \leftarrow \textsc{PivotPartition}(A, p, r, Pivot)$

$i \leftarrow q - p + 1 \qquad \triangleright$ Compute rank of pivot

if $(i = k)$

$\quad$ **then return** $A[q]$

if $(i > k)$

$\quad$ **then return** $\textsc{ImprovedSelect}(A, k, p, q - 1)$

$\quad$ **else return** $\textsc{ImprovedSelect}(A, k - i, q + 1, r)$

PivotPartition

$\textsc{PivotPartition}(array\ A, int\ p, int\ r, \boxed{item\ Pivot})$

$\triangleright$ Partition using provided $Pivot$

$i \leftarrow p - 1$

for $j \leftarrow p$ **to** $\boxed{r}$

  do if $(A[j] \leq Pivot)$

    then $i \leftarrow i + 1$

      exchange $A[i] \leftrightarrow A[j]$

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

```
TreeSearch(node x, key k)
  ▷ Search for key k in subtree rooted at node x
  if ((x = NIL) or (k = key[x]))
      then return x
  if (k < key[x])
      then return TreeSearch(left[x], k)
      else  return TreeSearch(right[x], k)
```

Time Complexity: O(h)
h = height of binary search tree

Not O(log n) — Why?

BST: Insert

```
TreeInsert(tree T, node z)
   ▷ Insert node z in tree T
   y ← NIL
   x ← root[T]
   while (x ≠ NIL)
         do y ← x
              if (key[z] < key[x])
                 then x ← left[x]
                 else  x ← right[x]
   p[z] ← y
   if (y = NIL)
      then root[T] ← z
      else  if (key[z] < key[y])
                 then left[y] ← z
                 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

```
TREEDELETE(tree T, node z)
  ▷ Delete node z from tree T
  if ((left[z] = NIL) or (right[z] = NIL))
     then y ← z
     else  y ← TREE-SUCCESSOR(z)
  if (left[y] ≠ NIL)
     then x ← left[y]
     else  x ← right[y]
  if (x ≠ NIL)
     then p[x] ← p[y]
  if (p[y] = NIL)
     then root[T] ← x
     else  if (y = left[p[y]])
              then left[p[y]] ← x
              else  right[p[y]] ← x
  if (y ≠ z)
     then key[z] ← key[y]
          copy y's satellite data into z
  return y
```

Time Complexity: O(h)
h = 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.