COT 5407: Introduction to Algorithms

Giri Narasimhan

ECS 254A; Phone: x3748

giri@cis.fiu.edu

http://www.cis.fiu.edu/~giri/teach/5407S17.html

https://moodle.cis.fiu.edu/v3.1/course/view.php?id=1494

Sorting Animations

- https://www.toptal.com/developers/sorting-algorithms
- https://visualgo.net/sorting
- https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html
- YouTube: https://www.youtube.com/watch?v=kPRA0W1kECg
- http://www.cs.ubc.ca/~harrison/Java/sorting-demo.html
- http://cg.scs.carleton.ca/~morin/misc/sortalg/
- http://home.westman.wave.ca/~rhenry/sort/
- http://cs.smith.edu/~dthiebaut/java/sort/demo.html

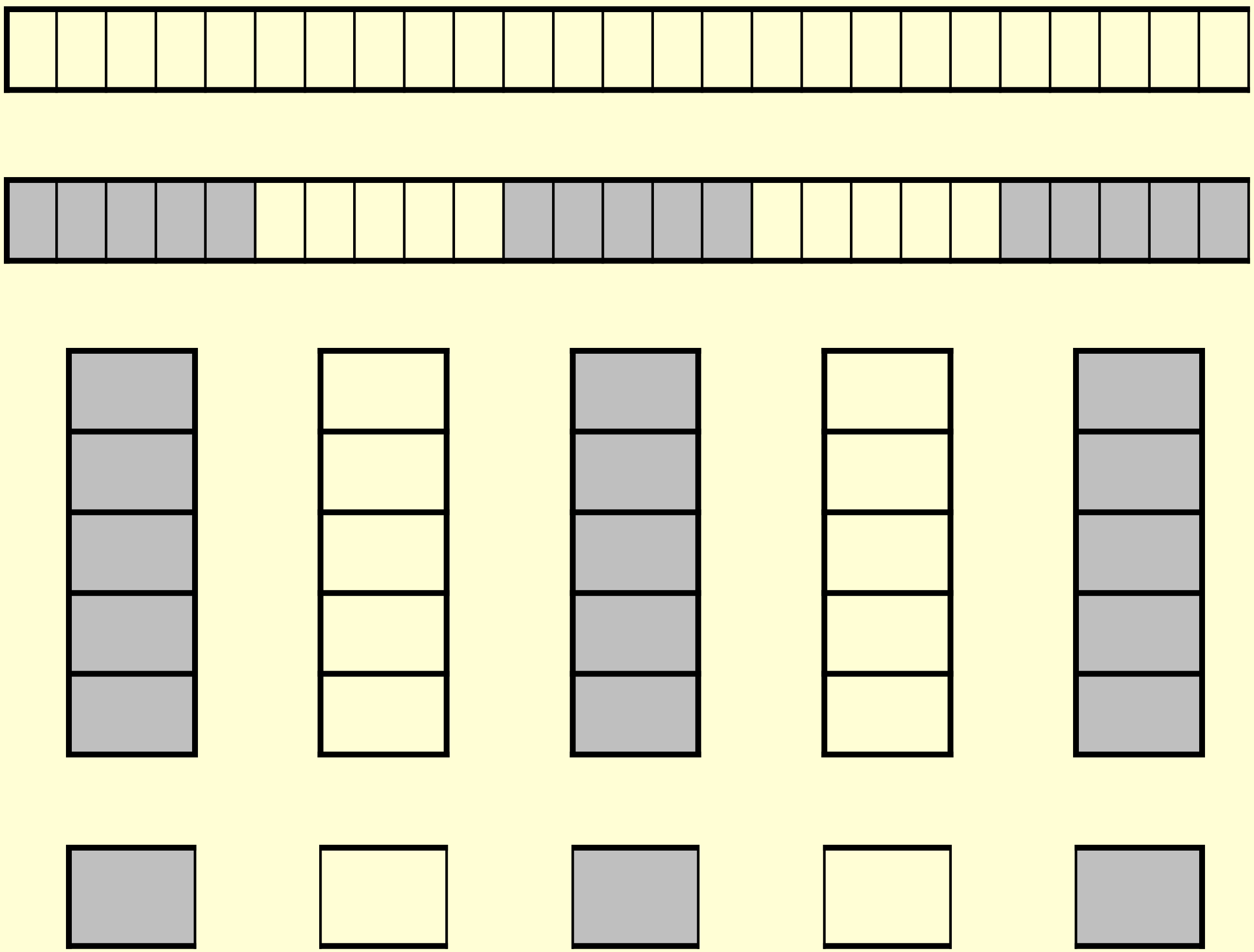
- **Which one did you like the most?**

Sorting Real Numbers

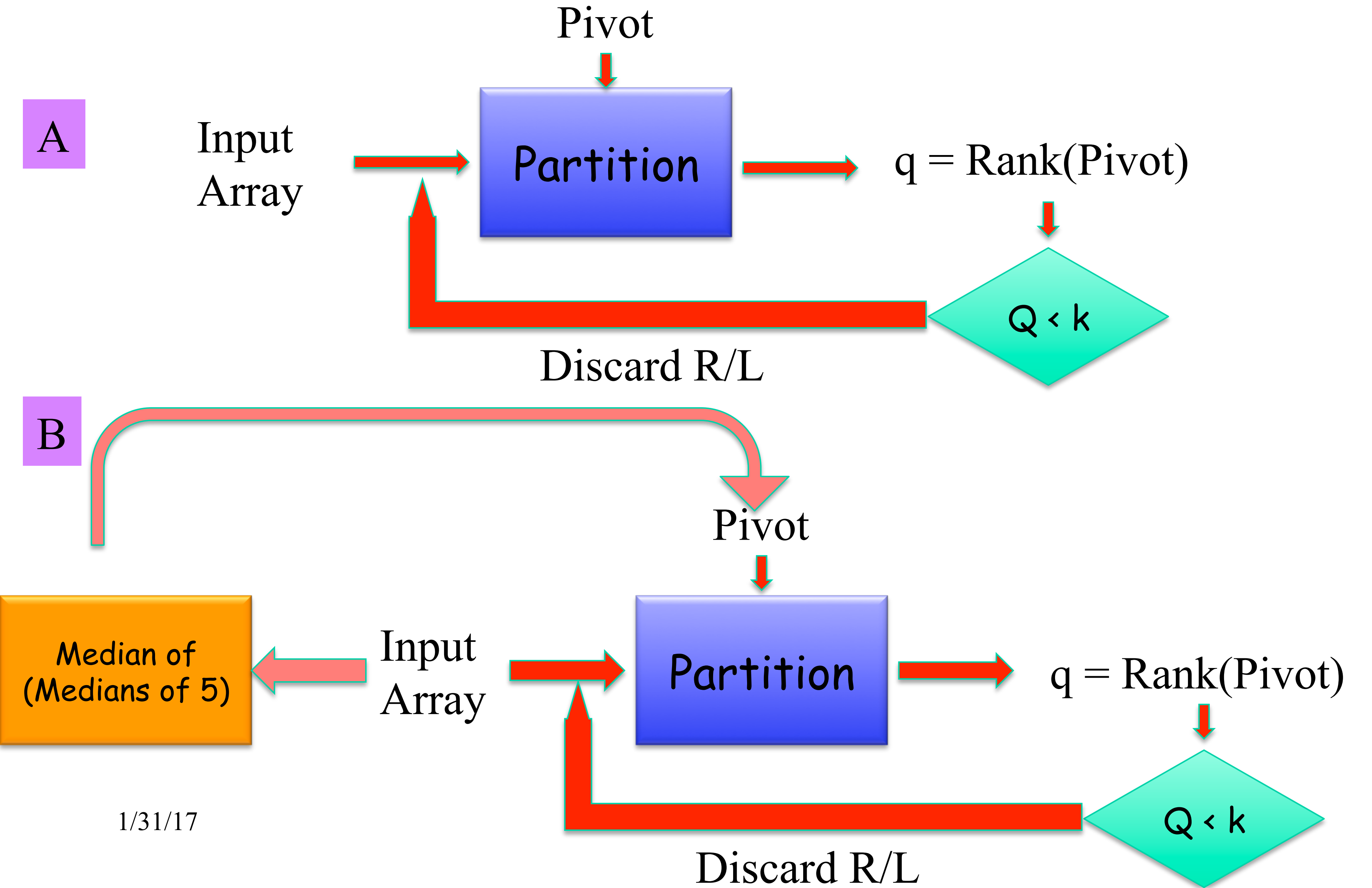
- **Real numbers** are infinite precision numbers and in some cases cannot be written down in their entirety.
- **Theorem**: There are an uncountable number of real numbers between any two real numbers.
- In particular, real numbers cannot be sorted using Bucket sort or radix sort or counting sort even if they are within a range.
- Real numbers stored on a real computer are not really "real numbers" because they are finite precision numbers. We can only approximate real numbers using a computer. Integers can be stored precisely on a computer. The integer n can be stored using roughly $\log_2 n$ bits.

k-Selection & Median: Improved Algorithm

- Start with initial array

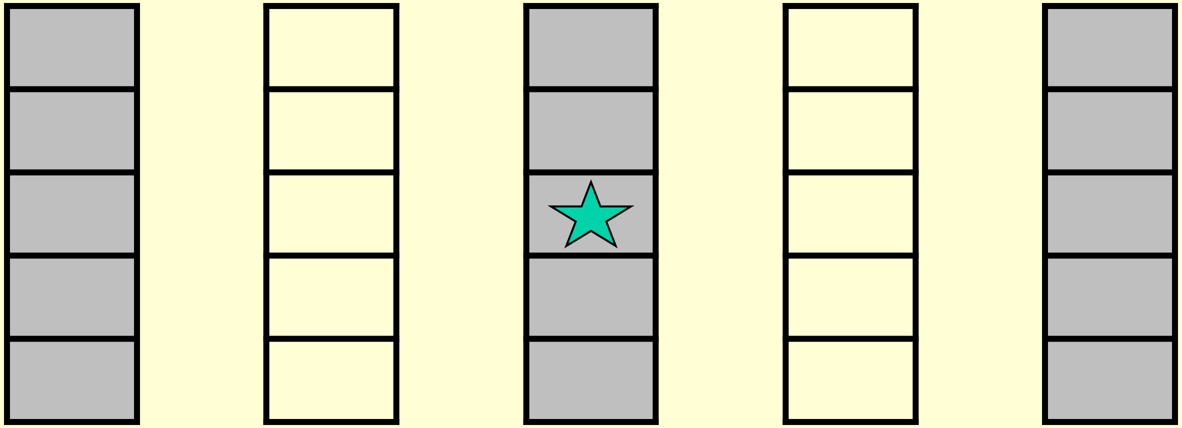


QuickSelect (A) & Improved Median (B)



k-Selection & Median: Improved Algorithm(Cont'd)

- Use median of medians as pivot



- $T(n) < O(n) + T(n/5) + T(3n/4)$
- Only way to solve is using the "Substitution Method"
- Solution: $T(n) = O(n)$
- Constants are large

ImprovedSelect

ImprovedSelect(*array* A, *int* k, *int* p, *int* r)

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

if $(p = r)$

  then return $A[p]$

  else $N \leftarrow r - p + 1$

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

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

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

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

if $(i = k)$

  then return $A[q]$

if $(i > k)$

  then return ImprovedSelect$(A, k, p, q - 1)$

  else return ImprovedSelect$(A, k - i, q + 1, r)$

PivotPartition

PIVOTPARTITION(*array* A, *int* p, *int* r, *item Pivot*)

$\triangleright$ Partition using provided *Pivot*

$i \leftarrow p - 1$

for $j \leftarrow p$ **to** 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

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

```
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]
           cop y's satellite data into z
   return y
```

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.