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, q+1, r)

Partition(A, p, r)
  x = A[r]
  i = p-1
  for j = p to r-1 do
      if (A[j] <= x) then
          i++
          SWAP(A[i], A[j])
  SWAP(A[i+1], A[r])
  return i+1
```

Partition Procedure Revisited

- 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

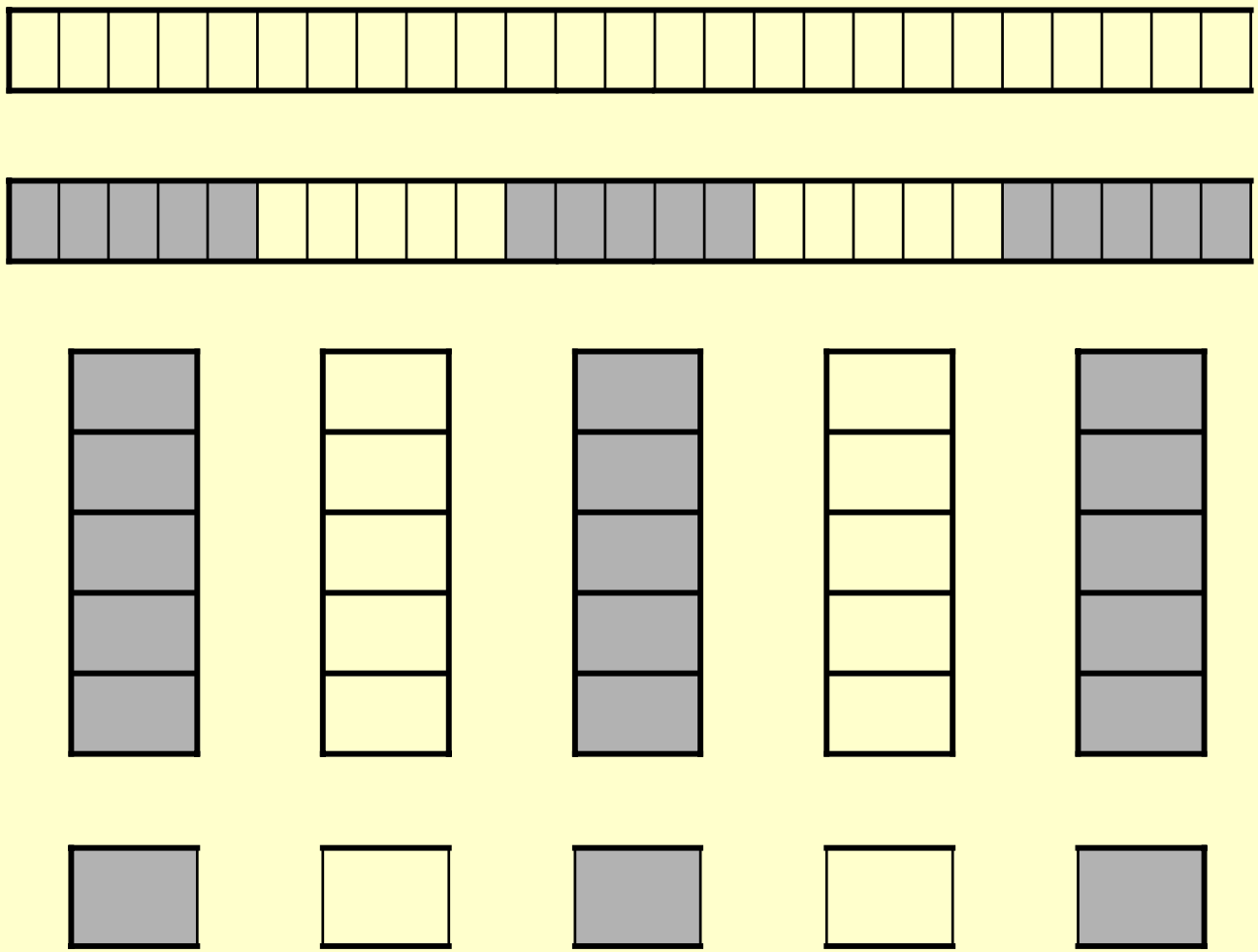
```
RandomizedSelect(A, p, r, i)
    if (p = r) then
        return A[p]
    q = RandomizedPartition(A, p, r)
    k = q – p + 1
    if (i = k)
        return A[i]
   else if (i < k)
        return RandomizedSelect(A, p, q-1, i)
    else
        return RandomizedSelect(A, q+1, r, i-k)
```

Randomized Selection: Rewritten

```
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 A[i]
    else if (i < k)
        return RandomizedSelect(A, p, q-1, i)
    else
        return RandomizedSelect(A, q+1, r, i-k)
```

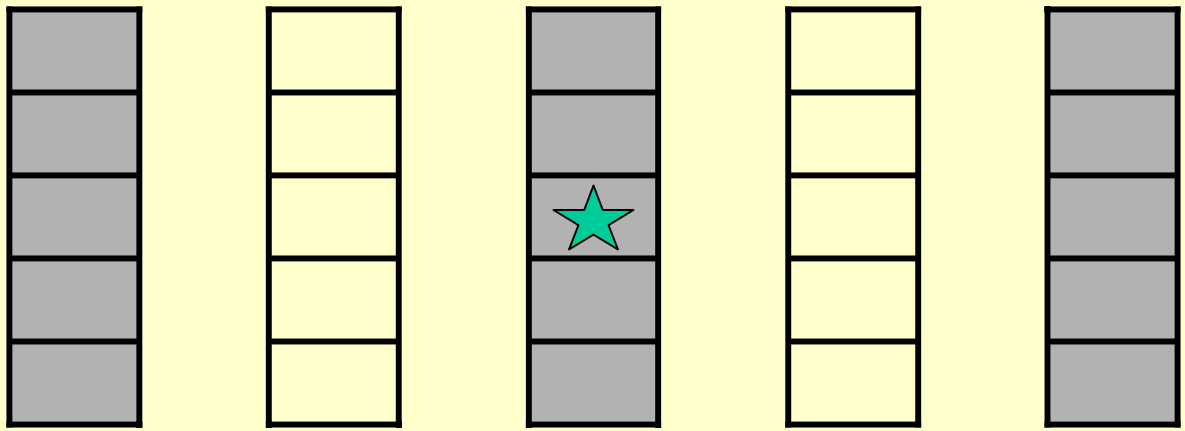
k-Selection & Median: Improved Algorithm

- 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, ⌈N/5⌉, ⌈N/10⌉ )
    q = PivotPartition (A, p, r, Pivot)
    k = q – p + 1
    if (i = k)
        return A[i]
    else if (i < k)
        return RandomizedSelect(A, p, q-1, i)
    else
        return RandomizedSelect(A, q+1, r, i-k)
```

Animations

- BST: http://babbage.clarku.edu/~achou/cs160/examples/bst_animation/BST-Example.html
- Rotations: http://babbage.clarku.edu/~achou/cs160/examples/bst_animation/index2.html
- RB-Trees: http://babbage.clarku.edu/~achou/cs160/examples/bst_animation/RedBlackTree-Example.html

Binary Search Trees

```
TreeSearch(x, k)                    // pg 257
// Search for key k in tree rooted at x
    if ((x = NIL) or (k = key[x]))
            return x
    if (k < key[x])
            return TreeSearch(left[x], k)
    else
            return TreeSearch(right[x], k)
```

Binary Search Trees

- TreeInsert (T,z) // pg 261

```
// Insert node z in tree T
        y = NIL
        x = root[T]                     // y follows x down the tree
                                        // when x is NIL, y points to a leaf
        while (x ≠ NIL) do
                y = x
                if (key[z] < key[x])
                        x = left[x]
                        x = right[x]
        p[z] = y
        if (y == NIL)
                root[T] = z
        else if (key[z] < key[y])
                left[y] = z
        else right[y] = z
```

Binary Search Trees

- TreeDelete(T,z)

```
// delete node z in tree T
        if (left[z] == NIL) or (right[z] == NIL) then
                y = z
        else    y = TreeSuccessor(z)        // y has at most 1 child
        if (left[y] ≠ NIL) then
                x = left[y]
        else    x = right[y]                // x points to a child of 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 data into z
        return y
```

Red-Black Trees

- RB-Insert (T,z) // pg 261

```
// Insert node z in tree T
y = NIL
x = root[T]
while (x ≠ NIL) do
        y = x
        if (key[z] < key[x])
                x = left[x]
                x =
right[x]
p[z] = y
if (y == NIL)
        root[T] = z
else if (key[z] < key[y])
        left[y] = z
else right[y] = z
// new stuff
left[z] = NIL[T]
right[z] = NIL[T]
color[z] = RED
RB-Insert-Fixup (T,z)
```

```
RB-Insert-Fixup (T,z)
while (color[p[z]] == RED) do
        if (p[z] = left[p[p[z]]]) then
            y = right[p[p[z]]]
            if (color[y] == RED) then        // C-1
                color[p[z]] = BLACK
                color[y] = BLACK
                z = p[p[z]]
            else  if (z == right[p[z]]) then // C-2
                    z = p[z]
                    LeftRotate(T,z)
                color[p[z]] = BLACK          // C-3
                color[p[p[z]]] = RED
                RightRotate(T,p[p[z]])
        else
            // Symmetric code: "right" ↔ "left"
            . . .
color[root[T]] = BLACK
```

Rotations

- LeftRotate(T,x) // pg 278
  // right child of x becomes x's parent.
  // Subtrees need to be readjusted.
  y = right[x]
  right[x] = left[y] // y's left subtree becomes x's right
  p[left[y]] = x
  p[y] = p[x]
  if (p[x] == NIL[T]) then
  root[T] = y
  else if (x == left[p[x]]) then
  left[p[x]] = y
  else right[p[x]] = y
  left[y] = x
  p[x] = y