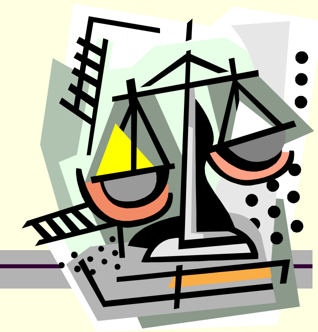
Quick Sort & Quick Select

Computer Science Department
University of Central Florida

COP 3502 Recitation Session

The Selection Problem



- Given an integer k and n elements $x_1, x_2, \ldots, x_n$, taken from a total order, find the k-th smallest element in this set.
- Naïve solution - SORT!
- we can sort the set in O(n log n) time and then index the k-th element.

7 4 9 <u>6</u> 2 → 2 4 <u>6</u> 7 9   k=3

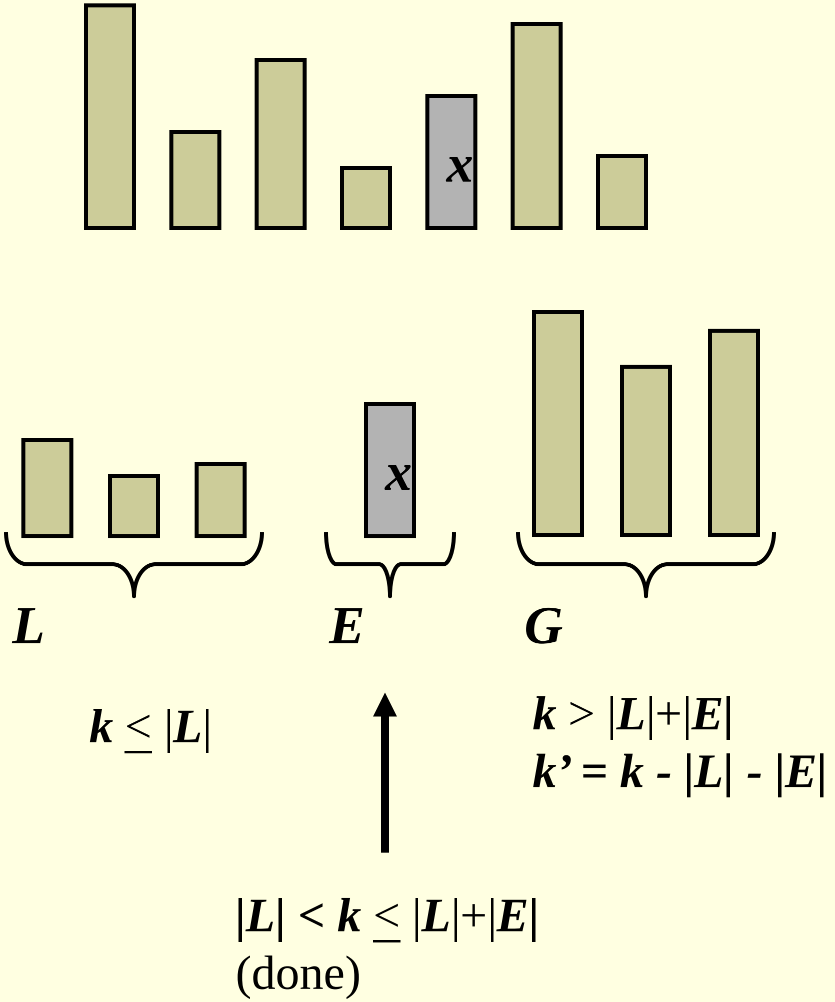
- Can we solve the selection problem faster?

The Selection Problem

- Can we solve the selection problem faster?
  - Of course we can!
  - We use Quick Select
- What is Quick Select?
  - Concept is very similar to Quick Sort
  - But in this case, we are not sorting
  - We don’t care about sorting the numbers
  - BUT, we do care about finding the specific element

Quick-Select

- Quick-select is a **randomized** selection algorithm based on the prune-and-search paradigm:
  - Prune: pick a random element x (called pivot) and partition S into
    - L elements less than x
    - E elements equal x
    - G elements greater than x
  - Search: depending on k, either answer is in E, or we need to recur on either L or G



Partition

- We partition an input sequence as in the quick-sort algorithm:
  - We remove, in turn, each element y from S and
  - We insert y into L, E or G, depending on the result of the comparison with the pivot x
- Each insertion and removal is at the beginning or at the end of a sequence, and hence takes $O(1)$ time
- Thus, the partition step of quick-select takes $O(n)$ time

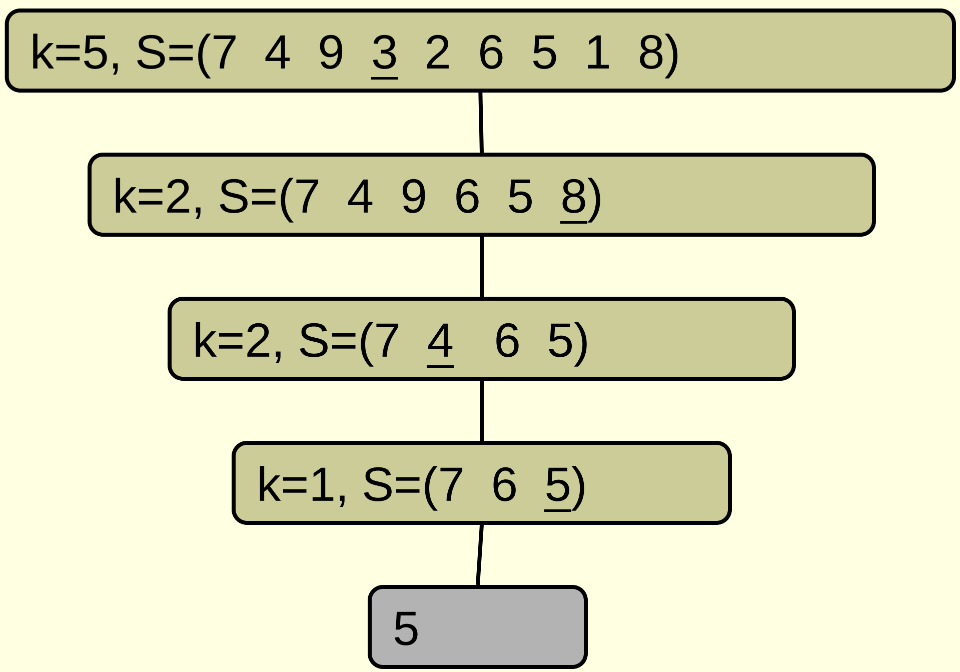
Algorithm ***partition***(S, p)

Input sequence S, position p of pivot
Output subsequences L, E, G of the elements of S less than, equal to, or greater than the pivot, resp.

```
L, E, G ← empty sequences
x ← S.remove(p)
while ¬S.isEmpty()
    y ← S.remove(S.first())
    if y < x
        L.insertLast(y)
    else if y = x
        E.insertLast(y)
    else { y > x }
        G.insertLast(y)
return L, E, G
```

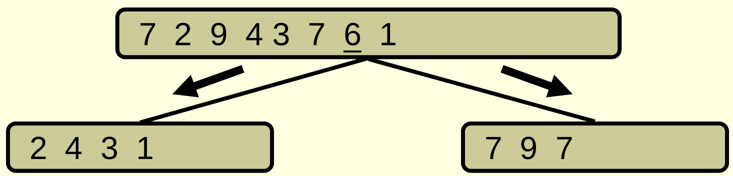
Quick-Select Visualization

- An execution of quick-select can be visualized by a recursion path
  - Each node represents a recursive call of quick-select, and stores k and the remaining sequence

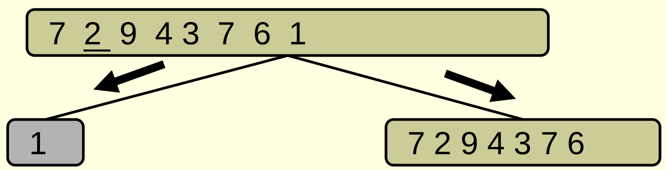


Running Time

- Best Case - even splits (n/2 and n/2)
- Worst Case - bad splits (1 and n-1)

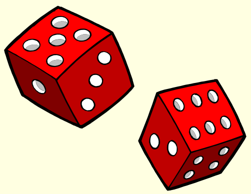


Good call

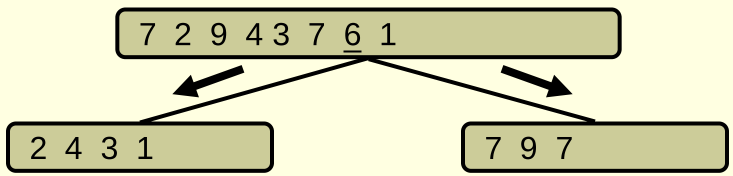


Bad call

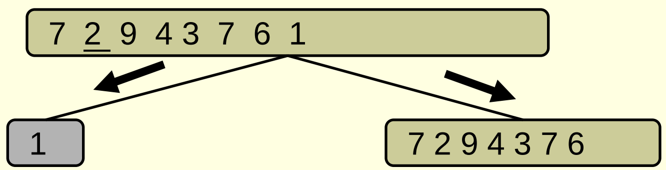
Expected Running Time



- Consider a recursive call of quick-select on a sequence of size s
  - **Good call:** the sizes of L and G are each less than $3s/4$
  - **Bad call:** one of L and G has size greater than $3s/4$

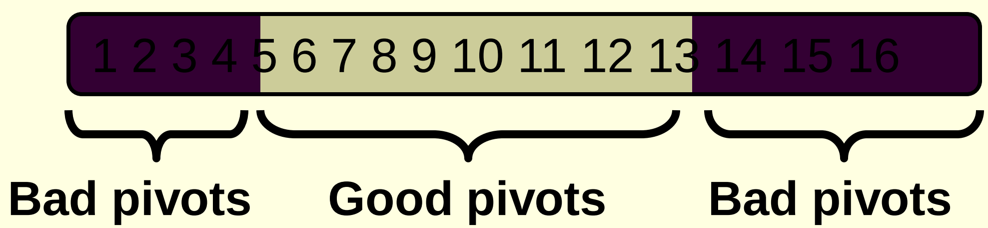


Good call



Bad call

- A call is good with probability $1/2$
  - 1/2 of the possible pivots cause good calls:



quickSelect Summary

- Recall: the Selection problem
  - Find the *k*th smallest element in an array *a*
- quickSelect(*a*, *k*):

1. If *a*.length = 1, then *k*=1 and return the element.
2. Pick a pivot $v \in a$.
3. Partition $a - \{v\}$ into a_1 (left side) and a_2 (right side).
   - if $k \leq a_1$.length, then the *k*th smallest element must be in a_1. So return quickSelect(a_1, *k*).
   - else if $k = 1 + a_1$.length, return the pivot *v*.
   - Otherwise, the *k*th smallest element is in a_2. Return quickSelect(a_2, $k - a_1$.length - 1).

Quick Sort & Quick Select

Computer Science Department
University of Central Florida

COP 3502 Recitation Session