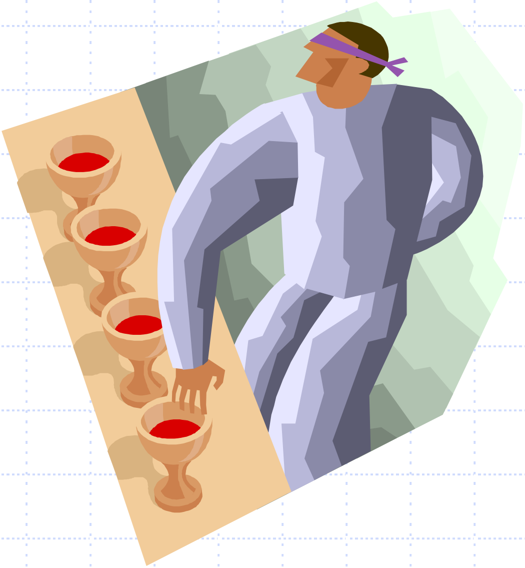
Selection



The Selection Problem

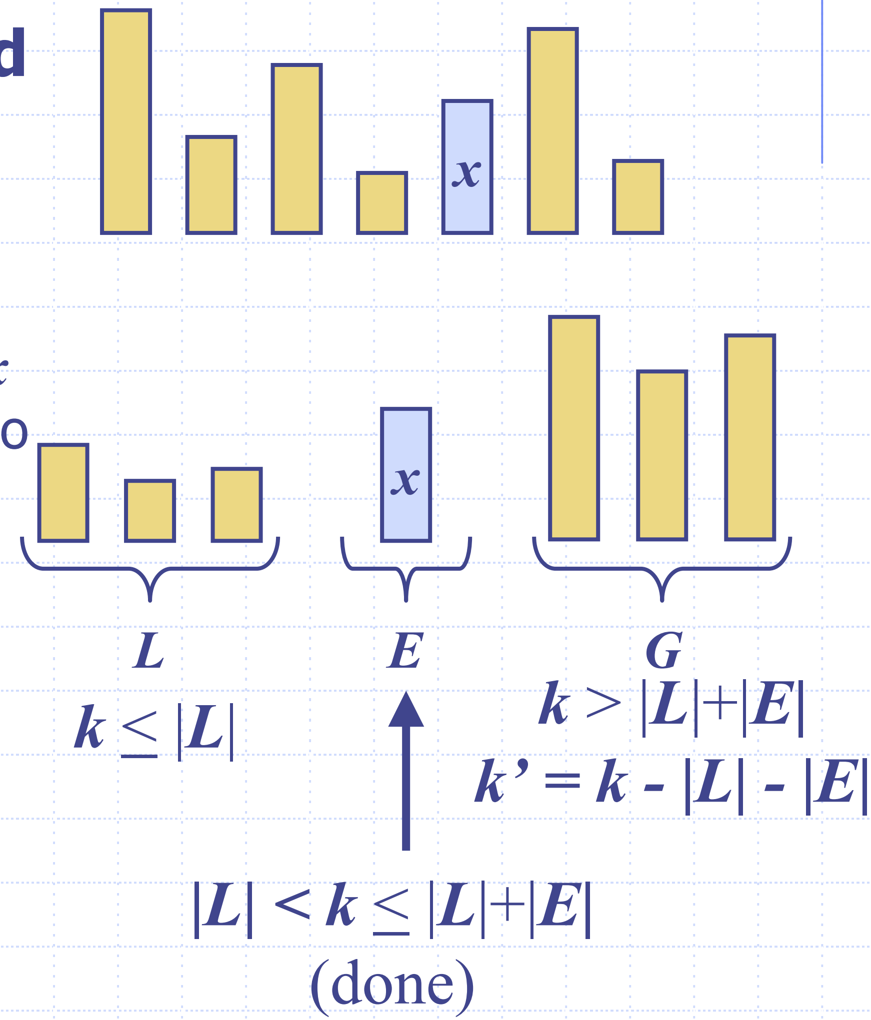
- Given an integer k and n elements $x_1, x_2, \ldots, x_n$, taken from a total order, find the k-th smallest element in this set.
- Of course, we can sort the set in O(n log n) time and then index the k-th element.

k=3 $\quad$ 7 4 9 <u>6</u> 2 $\rightarrow$ 2 4 <u>6</u> 7 9

- Can we solve the selection problem faster?

Quick-Select (§10.7)

- 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



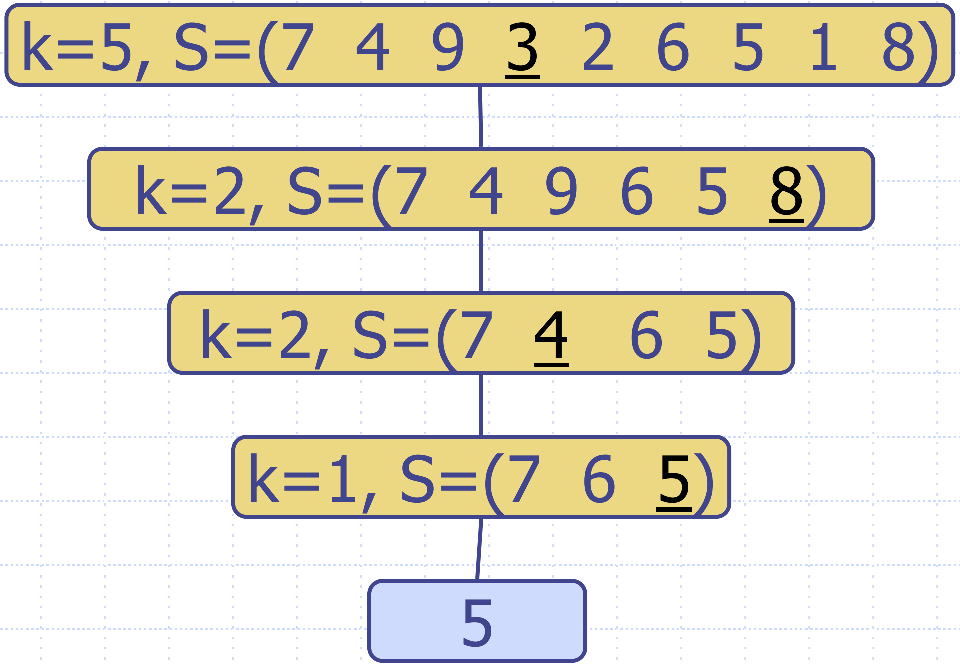
Partition

- 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

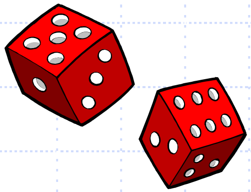
```
Algorithm partition(S, p)
    Input sequence S, position p of pivot
    Output subsequences L, E, G of the
        elements of S less than, equal to,
        or greater than the pivot, resp.
    L, E, G ← empty sequences
    x ← S.remove(p)
    while ¬S.isEmpty()
        y ← S.remove(S.first())
        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

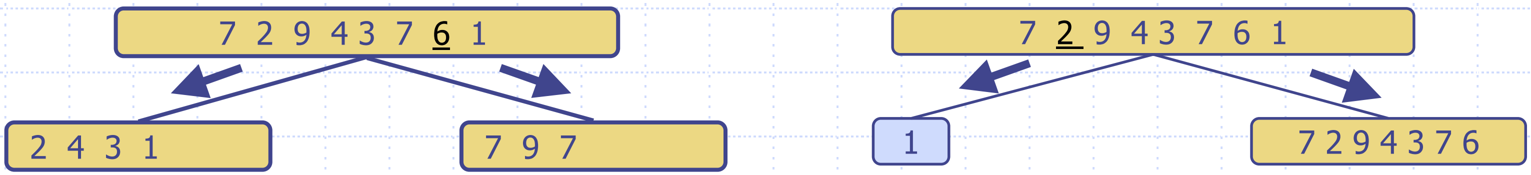
- An execution of quick-select can be visualized by a recursion path
  - Each node represents a recursive call of quick-select, and stores k and the remaining sequence



Expected Running Time

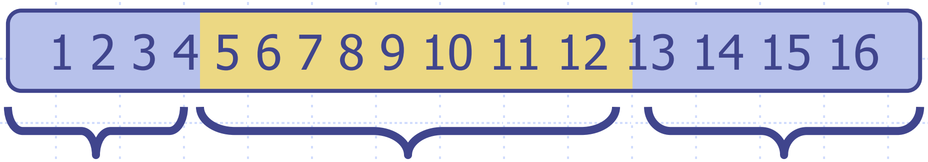


- Consider a recursive call of quick-select on a sequence of size s
  - **Good call:** the sizes of L and G are each less than $3s/4$
  - **Bad call:** one of L and G has size greater than $3s/4$



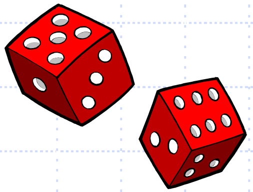
Good call **Bad call**

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



Bad pivots **Good pivots** **Bad pivots**

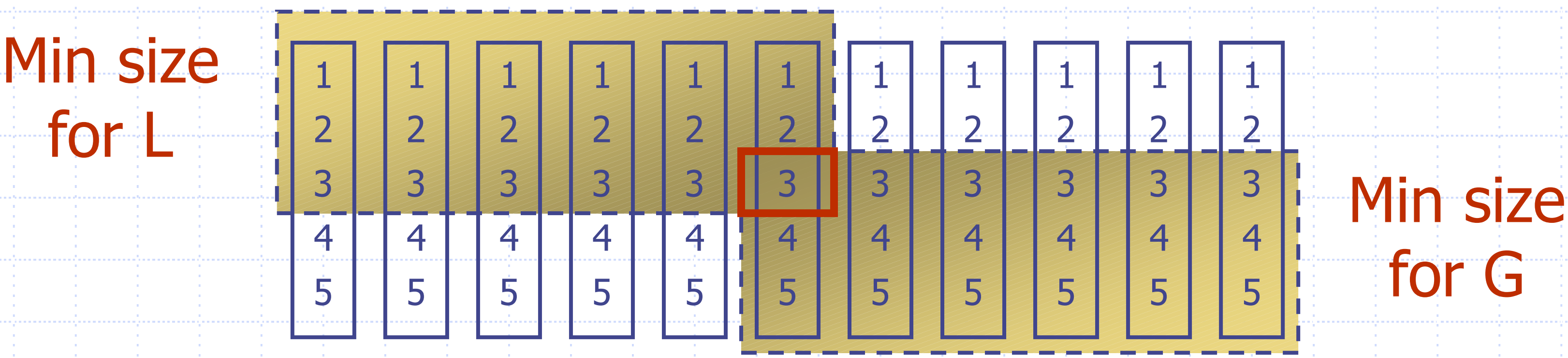
Expected Running Time, Part 2



- Probabilistic Fact #1: The expected number of coin tosses required in order to get one head is two
- Probabilistic Fact #2: Expectation is a linear function:
  - $E(X+Y) = E(X) + E(Y)$
  - $E(cX) = cE(X)$
- Let T(n) denote the expected running time of quick-select.
- By Fact #2,
  - $T(n) \le T(3n/4) + bn$*(expected # of calls before a good call)
- By Fact #1,
  - $T(n) \le T(3n/4) + 2bn$
- That is, T(n) is a geometric series:
  - $T(n) \le 2bn + 2b(3/4)n + 2b(3/4)^2n + 2b(3/4)^3n + \ldots$
- So T(n) is O(n).
- We can solve the selection problem in O(n) expected time.

Deterministic Selection

- We can do selection in O(n) worst-case time.
- Main idea: recursively use the selection algorithm itself to find a good pivot for quick-select:
  - Divide S into n/5 sets of 5 each
  - Find a median in each set
  - Recursively find the median of the "baby" medians.



- See Exercise C-4.24 for details of analysis.