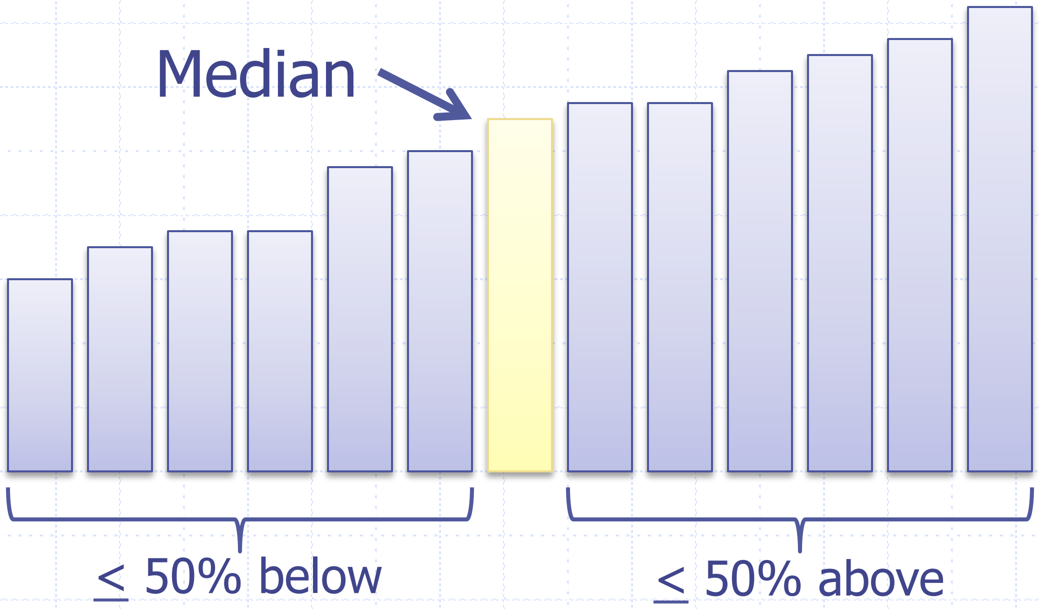
Presentation for use with the textbook, Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia, Wiley, 2015

Selection

Antarctica Gentoo Penguins. U.S. government image, 2010.
Credit: Lt. Elizabeth Crapo/NOAA.

Application: Finding Medians

- A common data analysis tool is to compute a **median**, that is, a value taken from among n values such that there are at most n/2 values larger than this one and at most n/2 elements smaller.
- Of course, such a number can be found easily if we were to sort the scores, but it would be ideal if we could find medians in O(n) time without having to perform a sorting operation.



The Selection Problem

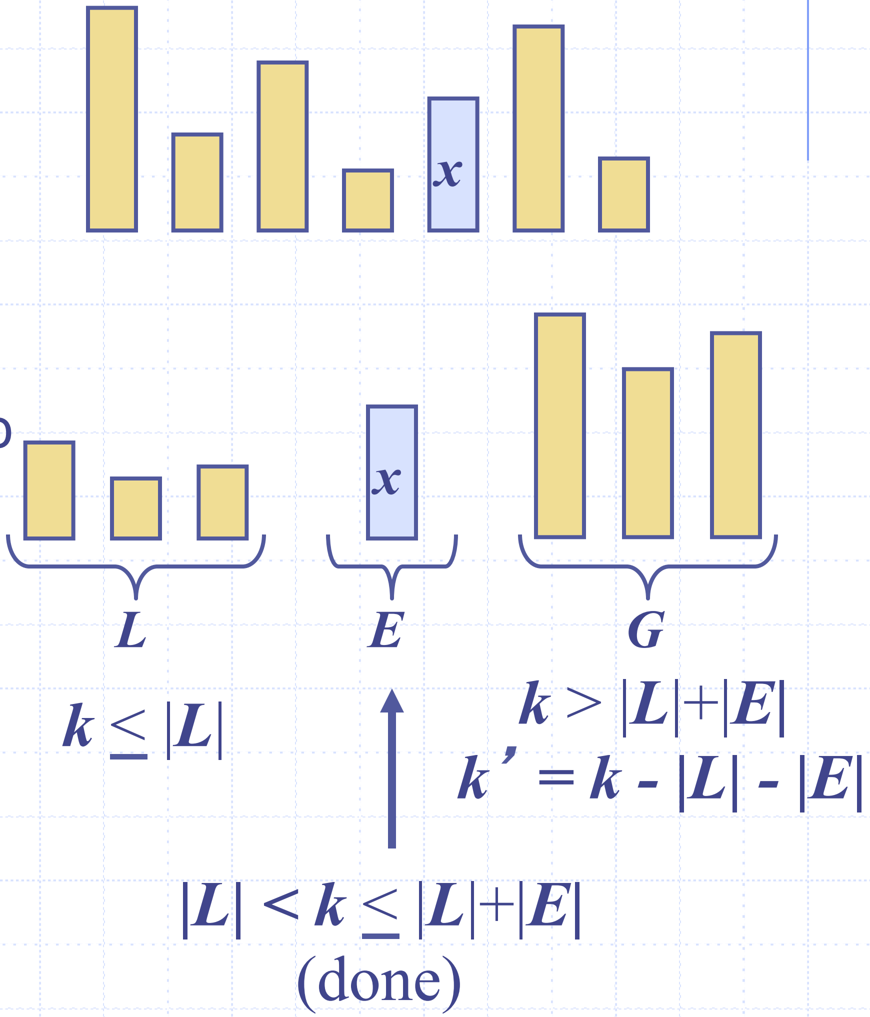
- Given an integer k and n elements $x_1, x_2, \ldots, x_n$, taken from a total order, find the k-th smallest element in this set.
- Of course, we can sort the set in O(n log n) time and then index the k-th element.

k=3 $\boxed{7 \;\; 4 \;\; 9 \;\; \underline{6} \;\; 2 \;\; \rightarrow \;\; 2 \;\; 4 \;\; \underline{6} \;\; 7 \;\; 9}$

- We want to solve the selection problem faster.

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 in either L or G



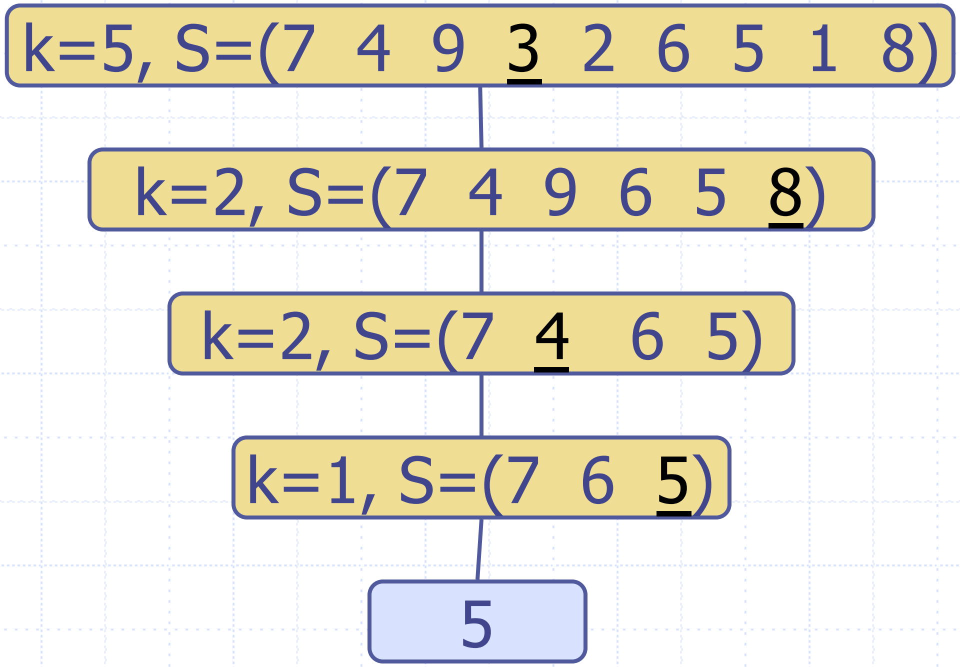
Pseudo-code

```
Algorithm quickSelect($S, k$):
    Input: Sequence $S$ of $n$ comparable elements, and an integer $k \in [1, n]$
    Output: The $k$th smallest element of $S$
    if $n = 1$ then
        return the (first) element of $S$
    pick a random element $x$ of $S$
    remove all the elements from $S$ and put them into three sequences:
        • $L$, storing the elements in $S$ less than $x$
        • $E$, storing the elements in $S$ equal to $x$
        • $G$, storing the elements in $S$ greater than $x$.
    if $k \leq |L|$ then
        quickSelect($L, k$)
    else if $k \leq |L| + |E|$ then
        return $x$        // each element in $E$ is equal to $x$
    else
        quickSelect($G, k - |L| - |E|$)
```

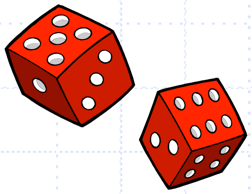
- Note that each call to quickSelect takes $O(n)$ time, not counting the recursive calls.

Quick-Select Visualization

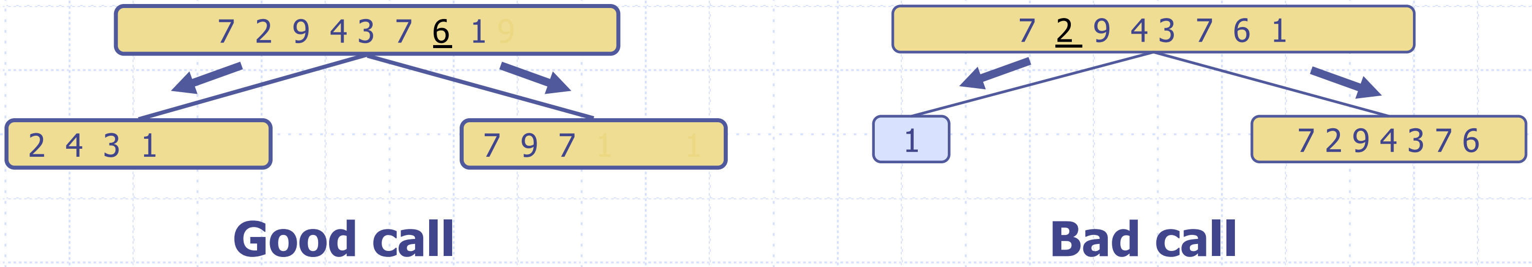
- An execution of quick-select can be visualized by a recursion path
  - Each node represents a recursive call of quick-select, and stores k and the remaining sequence



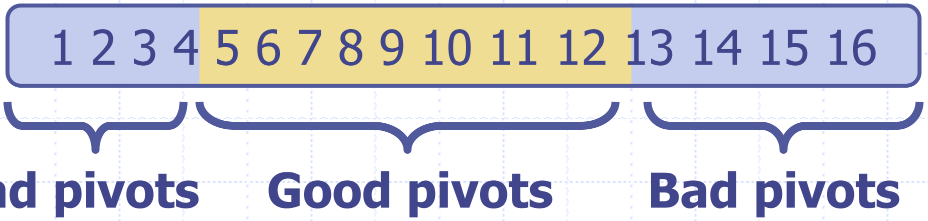
Expected Running Time



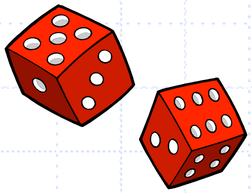
- Consider a recursive call of quick-select on a sequence of size s
  - Good call: the sizes of L and G are each less than $3s/4$
  - Bad call: one of L and G has size greater than $3s/4$



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

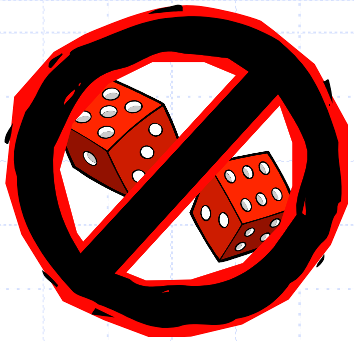


Expected Running Time, Part 2

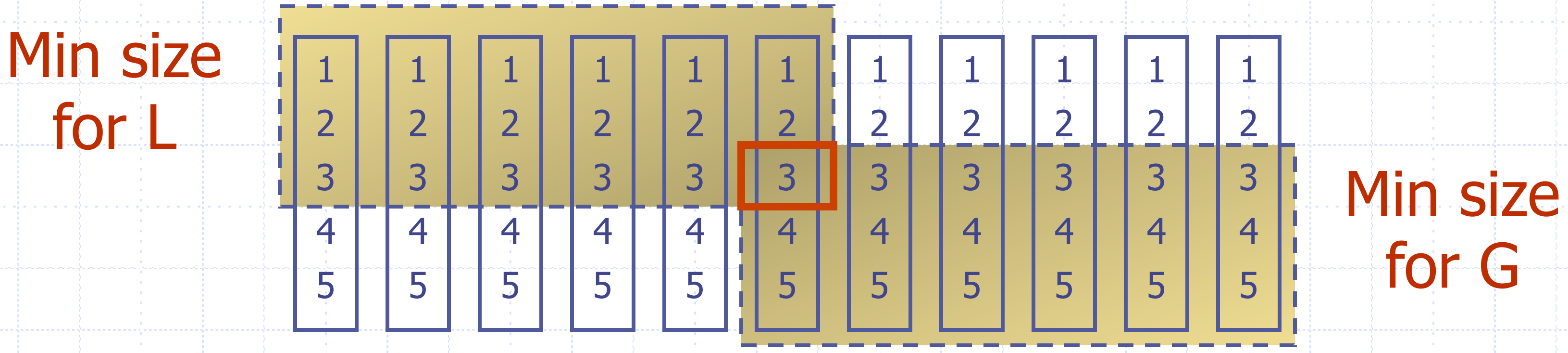


- Probabilistic Fact #1: The expected number of coin tosses required in order to get one head is two
- Probabilistic Fact #2: Expectation is a linear function:
  - $E(X+Y) = E(X) + E(Y)$
  - $E(cX) = cE(X)$
- Let T(n) denote the expected running time of quick-select.
- By Fact #2,
  - $T(n) \le T(3n/4) + bn*(\text{expected \# of calls before a good call})$
- 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.



Pseudo-code

```
Algorithm DeterministicSelect($S, k$):
    Input: Sequence $S$ of $n$ comparable elements, and an integer $k \in [1, n]$
    Output: The $k$th smallest element of $S$
    if $n = 1$ then
        return the (first) element of $S$
    Divide $S$ into $g = \lceil n/5 \rceil$ groups, $S_1, \ldots, S_g$, such that each of groups
    $S_1, \ldots, S_{g-1}$ has 5 elements and group $S_g$ has at most 5 elements.
    for $i \leftarrow 1$ to $g$ do
        Find the baby median, $x_i$, in $S_i$ (using any method)
    $x \leftarrow$ DeterministicSelect($\{x_1, \ldots, x_g\}$, $\lceil g/2 \rceil$)
    remove all the elements from $S$ and put them into three sequences:

        • $L$, storing the elements in $S$ less than $x$
        • $E$, storing the elements in $S$ equal to $x$
        • $G$, storing the elements in $S$ greater than $x$.

    if $k \leq |L|$ then
        DeterministicSelect($L, k$)
    else if $k \leq |L| + |E|$ then
        return $x$        // each element in $E$ is equal to $x$
    else
        DeterministicSelect($G, k - |L| - |E|$)
```

- Note that each call to DeterministicSelect takes $O(n)$ time, not counting the recursive calls.

Analysis

We now show that the above deterministic selection algorithm runs in linear time.

The algorithm has two recursive calls. The first one is performed on the set of baby medians, which has size

$$g = \lceil n/5 \rceil.$$

The second recursive call is made on either set L (elements smaller than the pivot, x) or set G (elements larger than the pivot, x). Recall that each group but one contains 5 elements and our pivot, x, is the median of the baby medians from all of these groups. Thus, we have that for $\lceil g/2 \rceil$ groups, at least half of the group elements are less than or equal to x. Since group S_g could be part of this half, we have that number of elements in S that are less than or equal to x is at least

$$3\left(\left\lceil \frac{g}{2} \right\rceil - 1\right) + 1 = 3\left\lceil \frac{1}{2} \cdot \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \geq \frac{3n}{10} - 2.$$

With a similar argument, we obtain that the above value is also a lower bound on the number of elements of S less than or equal to x.

We conclude that the second recursive call is performed on a set of size at most

$$n - \left(\frac{3n}{10} - 2\right) = \frac{7n}{10} + 2.$$

Overall, for a sufficiently large value of n, the running time for the deterministic selection algorithm, $T(n)$, can be characterized by the following recurrence relation:

$$T(n) \leq T(n/5 + 1) + T(7n/10 + 2) + bn.$$

where $b > 0$ is a constant.

Analysis, Part 2

$$T(n) \le T(n/5 + 1) + T(7n/10 + 2) + bn.$$

where $b > 0$ is a constant.

To solve the recurrence, we guess that $T(n) \le cn$, for some constant $c > 0$. Expanding the recurrence, we have the following:

$$\begin{aligned} T(n) &\le T(n/5 + 1) + T(7n/10 + 2) + bn \\ &\le cn/5 + c + 7cn/10 + 2c + bn \\ &= 9cn/10 + bn + 3c. \end{aligned}$$

Pick $c = 11b$. We obtain

$$T(n) \le 9cn/10 + bn + 3c \le 9cn/10 + cn/11 + 3c.$$

Thus, we have $T(n) \le cn$ for n large enough such that

$$cn/11 + 3c \le cn/10,$$

that is, for $n \ge 330$. Therefore, the running time of the deterministic selection algorithm is $O(n)$.

We summarize the above analysis with the following theorem.

Theorem 9.4: *Given an input sequence with n elements, the deterministic selection algorithm runs in $O(n)$ time.*