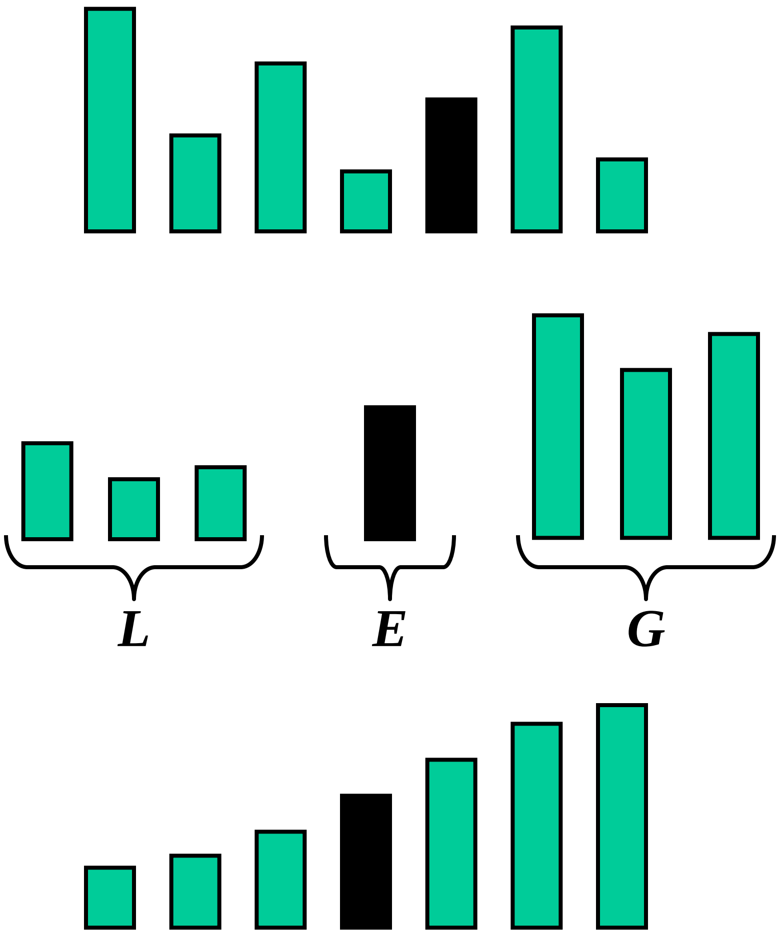
CSC 225

Algorithms and Data Structures I
Fall 2014
Rich Little

Quicksort as discussed in Textbook based on ADT Sequence

- Quick-sort is a randomized sorting algorithm based on the divide-and-conquer paradigm:
  - Divide: 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***
  - Recur: sort ***L*** and ***G***
  - Conquer: join ***L***, ***E*** and ***G***



Randomized Quicksort

- Even though the worst case running time of Quicksort is quadratic, in practice Quicksort is a very efficient sorting algorithm.

- Consider the expected running time of "Randomized Quicksort" where the index of the pivot is chosen randomly.

Randomized Quicksort

- ***Theorem*.** The **expected** running time of **randomized** Quicksort on a sequence of size n is $O(n \log n)$.
- Randomized means choosing a pivot randomly from the set of elements to be sorted.
- How can we prove this theorem?
  - To obtain $O(n \log n)$ **expected** time, we need to split up at least a fraction of n of all the elements.
  - Suppose we can show that we can split up a ¼ n elements not every time, but every other time we choose a pivot randomly, then we are done.

Random Pivot Selection

- Suppose our set of elements is sorted

- A "good" pivot is one that is in the red range
- How can we choose a pivot from the red range when the array is not sorted yet?
- Let us select a pivot randomly from the input set
- What are the chances that the pivot is in the red range?
  - 50 %
  - Probability ½
  - Basic coin toss
- Thus, every other time we choose a "good pivot" if we choose one randomly

Proof

- Now we have to estimate the height of the recursion tree, given that we we split up at least ¼ elements every other time.
- Suppose that we split up ¼ elements every time

$$\frac{1}{4}|S| \le |L| \le \frac{3}{4}|S| \qquad \frac{1}{4}|S| \le |G| \le \frac{3}{4}|S|$$

- Then the Quicksort recursion-tree is bounded in height by $\log_{4/3} n$
- Since we only choose a good pivot every other time the Quicksort recursion-tree is bounded in height by $2\log_{4/3} n$

Proof

- How many pivots do you have to pick to get $\log_{4/3} n$ good ones? $2\log_{4/3} n$

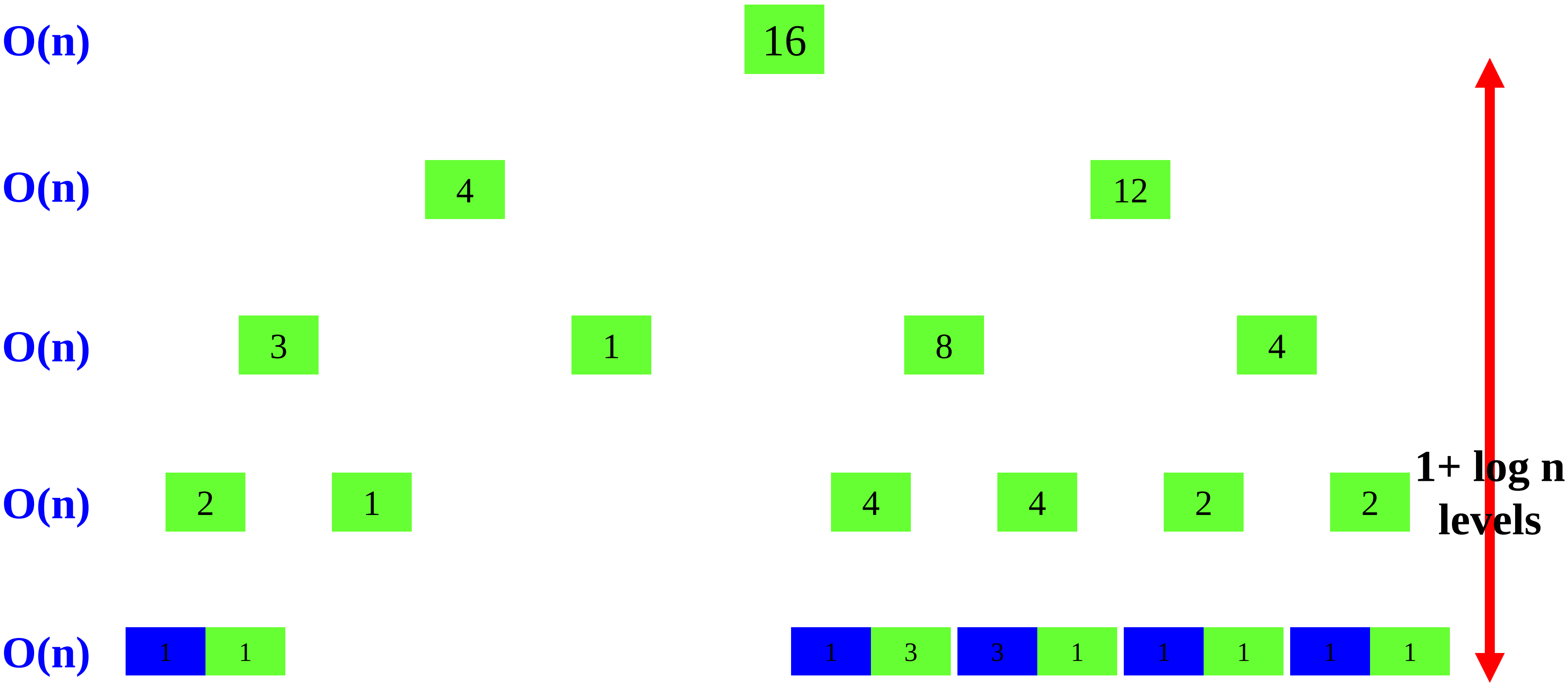
- What is the probability to pick a good pivot? $\frac{1}{2}$

- How many good pivots exist? $\frac{1}{2}n$

Proof

- Thus the tree has an expected bound in height of $2\log_{4/3} n$
- **Thus, the resulting expected running time for Randomized Quicksort is *O*(*n* log *n*)**

Height of Recursion Tree



$$T(n) = n \log_{4/3} n = n \frac{\log_2 n}{\log_2 4/3} = cn \log_2 n \in O(n \log n)$$

More on recurrence equations

$$T(n) = \begin{cases} b & \text{if } n \le 1 \\ T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1 & \text{if } n \ge 2 \end{cases}$$

T(n) is O(log n)

Using repeated substitutions

$$\text{For } n \geq 1$$

$$T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1$$

$$T\left(\left\lceil \frac{n}{2} \right\rceil\right) = T\left(\left\lceil \frac{n}{4} \right\rceil\right) + 1$$

T(n) is O(log n)

Using repeated substitutions

$$\text{For } n \geq 2$$

$$T(n) = T\left(\left\lceil \frac{n}{4} \right\rceil\right) + 2$$

$$T\left(\left\lceil \frac{n}{4} \right\rceil\right) = T\left(\left\lceil \frac{n}{8} \right\rceil\right) + 1$$

T(n) is O(log n)

Using repeated substitutions

$$\text{For } n \geq 3$$

$$T(n) = T\left(\left\lceil \frac{n}{8} \right\rceil\right) + 3$$

$T(n)$ is $O(\log n)$

Using repeated substitutions

$$\text{For } n \geq i$$

$$T(n) = T\left(\left\lceil \frac{n}{2^i} \right\rceil\right) + i$$

$T(n)$ is $O(\log n)$

$$T\left(\left\lceil \frac{n}{2^i} \right\rceil\right) = 1 \text{ for } i = \log n$$

$$\text{Then } T(n) = T(1) + \log n$$

$$\text{and } T(n) \text{ is } O(\log n).$$

Order Statistics or Selection

Problem

Given a sequence of n objects satisfying the total order property and an integer $k \leq n$, determine the k^{th} smallest object.

Selecting the k^{th} Smallest Element

- Sort the objects and return the k^{th} from the left (i.e., k^{th} smallest element) $O(n \log n)$
- How to improve on $O(n \log n)$?
  - ➢How much improvement is possible?
  - ➢Expected case and worst-case?
  - ➢Modify a known sorting algorithm
  - ➢Develop an algorithm from scratch

Modify Quicksort: QuickSelect

Input: Sequence S containing n elements, integer $k \le n$

Output: k^{th} smallest element in sorted sequence S

if S.length() = 1 **then return** S

Let L, E, G be empty sequences

$p \leftarrow$ pickPivot(S)

Partition(L, E, G, S, p)

QuickSort(L)

QuickSort(G)

Concatenate(L, E, G, S)

return S

Randomized QuickSelect

Input: Sequence S containing n elements, integer $k \leq n$
Output: k^{th} smallest element in sorted sequence S

if S.length() = 1 **then return** S
Let L, E, G be empty sequences
$p \leftarrow$ pickRandomPivot(S)
partition(L, E, G, S, p)
if $k \leq L$.length() **then return** QuickSelect(L, k)
else if $k \leq L$.length() + E.length() **then return** p
else return QuickSelect(G, $k - L$.length() $- E$.length())

Expected Time Analysis of Randomized QuickSelect

- Reuse the analysis for randomized Quicksort
- We split up ¼ n elements every time
- Thus, we have to continue partitioning at most ¾ n elements
- Thus, the height of the QuickSelect tree is at most $2\log_{4/3} n$
- How much work do we do at each level?

Expected Time Analysis of Randomized QuickSelect

$$T_{\mathrm{QS}}(n) = \begin{cases} b & \text{if } n = 1 \\ cn + T(\frac{3}{4}n) & \text{otherwise} \end{cases}$$

$$T_{\mathrm{QS}}(n) \in O(n)$$

- Show by repeated substitution

Expected Time Analysis of Randomized QuickSelect

$$T_{\mathrm{QS}}(n) = n + \frac{3}{4}n + \frac{3}{4}\frac{3}{4}n + \frac{3}{4}\frac{3}{4}\frac{3}{4}n + \frac{3}{4}\frac{3}{4}\frac{3}{4}\frac{3}{4}n + \ldots$$

$$T_{\mathrm{QS}}(n) = n\left[1 + \frac{3}{4} + \frac{9}{16} + \frac{27}{64} + \frac{81}{256} + \ldots\right]$$

$$T_{\mathrm{QS}}(n) = n\left[\left(\frac{3}{4}\right)^0 + \left(\frac{3}{4}\right)^1 + \left(\frac{3}{4}\right)^2 + \left(\frac{3}{4}\right)^3 + \left(\frac{3}{4}\right)^4 + \ldots + \left(\frac{3}{4}\right)^{2\log_{4/3} n - 1}\right]$$

$$T_{\mathrm{QS}}(n) = n \sum_{k=0}^{2\log_{4/3} n - 1} \left(\frac{3}{4}\right)^k$$

$$\left(\frac{1}{4}\right) T_{\mathrm{QS}}(n) = n\left[\left(\frac{3}{4}\right)^0 - \left(\frac{3}{4}\right)^{2\log_{4/3} n}\right]$$

$$T_{\mathrm{QS}}(n) = 4n\left[1 - \left(\frac{3}{4}\right)^{2\log_{4/3} n}\right] \approx 4n(1-0) \approx 4n \in O(n)$$

Expected Time Analysis of Randomized QuickSelect

- **Theorem.**
  Expected time of Randomized QuickSelect is $O(n)$.

Worst-case Analysis

- **Theorem.**
  The worst-case $T(n)$ of Quicksort is $O(n^2)$.
- **Theorem.**
  The expected-case $T(n)$ of Randomized Quicksort is $O(n \log n)$.
- **Theorem.**
  The expected-case $T(n)$ of Randomized QuickSelect is $O(n)$.
- **Theorem.**
  The worst-case $T(n)$ of QuickSelect is $O(n^2)$.

- Can we design a Selection algorithm with $O(n)$ time complexity?
- We have to guarantee that we split up a fraction of n elements every time with every partition.