Algorithm Design Technique
Divide and Conquer: Quicksort

- Mergesort divides the input set according to the position of the elements (i.e., first and second part of sequence)
- Quicksort divides the input set according to the value of the elements

http://en.wikipedia.org/wiki/Quicksort

Quicksort

- The input data are stored in an array $A[L..R]$ where L and R are the leftmost and rightmost indices of the data in this array
- Approach: Partition the input set using a pivot into two (ideally) equal-sized subsets S_1 and S_2 using a pivot (i.e., typically a value from the input data)
- Apply the algorithm recursively for the two subsets S_1 and S_2 until size 1 is reached

Algorithm QuickSort($A[L..R]$)

```
if A.length > 1 then
  p ← pickPivot(A[L..R])
  M ← partition(A[L..R], A[p])
  QuickSort(A[L..M])
  QuickSort(A[M+1..R])
end
```

Pivot Computation

- Picking a pivot should be a $O(1)$ operation
- The median is the perfect pivot; computing the median takes $O(n)$ time
- Any value close to the median is still a good pivot
- The largest or smallest value would be a bad pivot, because it would split the array into subarrays of size 1 and n-1
- Constant time approaches for picking a pivot p
  - First element in subarray $A[L]$
  - Last element in subarray $A[L]$
  - Middle element of subarray $A[(L+R)/2]$
  - Average of three elements $(A[L] + A[R] + [(L+R)/2])/3$
  - Compute the average of 5 or 7 elements
  - Randomized selection of pivot—randomly select index in range L..R

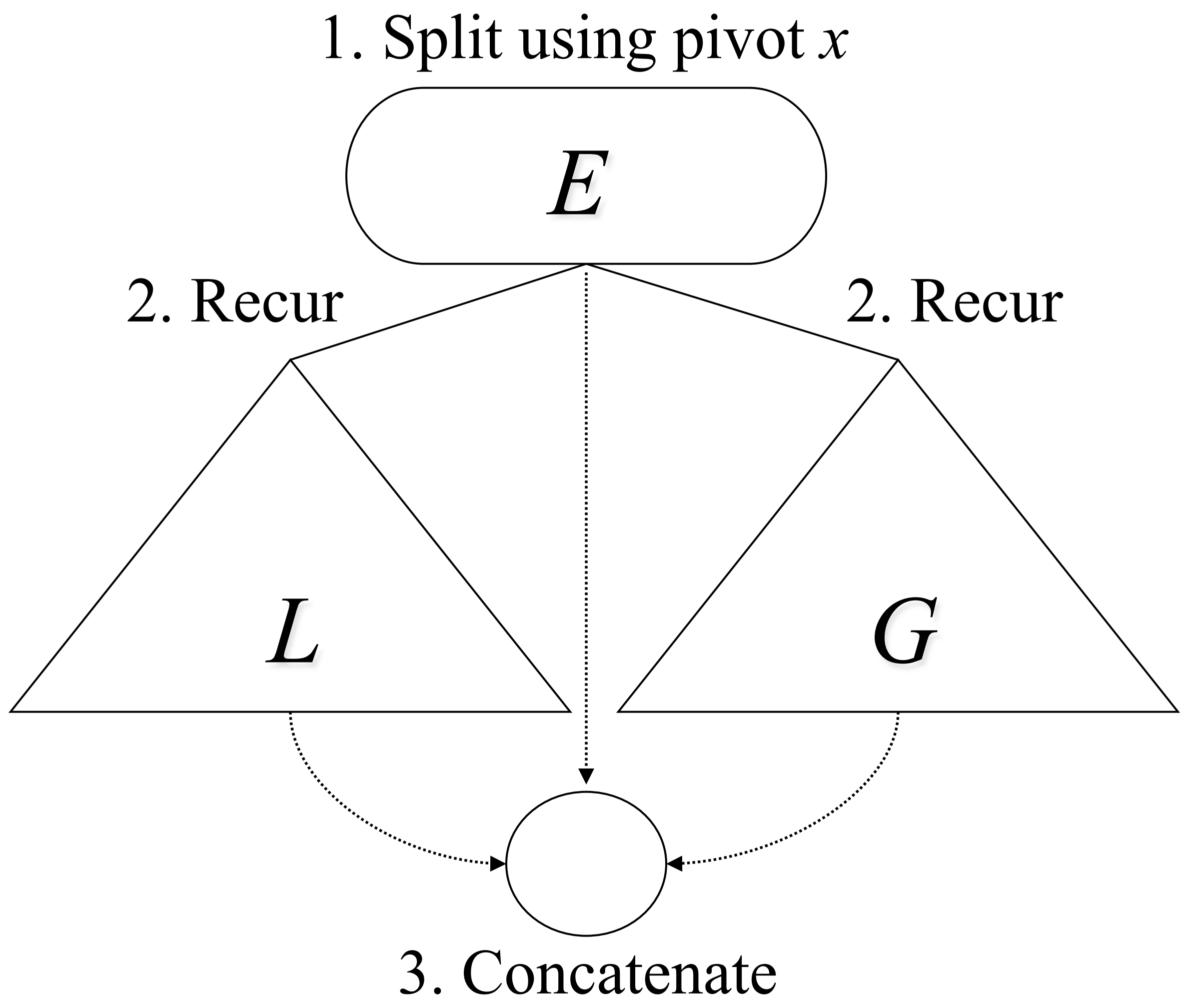
Why is Quicksort so fast?

- In practice Quicksort runs in $O(n \log n)$ and almost never exhibits its worst-case behaviour of $O(n^2)$
- Moreover, Quicksort performs better than $O(n \log n)$ worst-case sorting algorithms
- The actual running time makes the difference
  - $T_{Quick}(n) = 1.18 \text{ n log n}$
  - $T_{Heap}(n) = 2.22 \text{ n log n}$
- Sorting out sorting
  - http://www.youtube.com/watch?v=AUn7-36oluU

Quicksort as discussed in Textbook based on ADT Sequence

- Divide-and-conquer technique
- *Divide*: If sequence S has two or more elements:
  - Select *pivot* element x from S
  - Create sequence L storing the elements in S less than x
  - Create sequence E storing the elements in S equal to x
  - Create sequence G storing the elements in S greater than x
- *Recur*: Recursively sort L and G (Note that E is already sorted).
- *Conquer*: Put the elements back into S in order by first inserting the elements of L, then those of E, and finally those of G.

Quicksort Algorithm



Algorithm split(L, E, G, S, x)

- Let L, E, and G be empty sequences.
- Insert in L (and remove from S) all elements from S that are less than x.
- Insert in E (and remove from S) all elements from S that are equal to x.
- Insert in G (and remove from S) all elements from S that are greater than x.
- S is empty.

How fast can we implement algorithm split?

Algorithm concatenate(L, E, G, S)

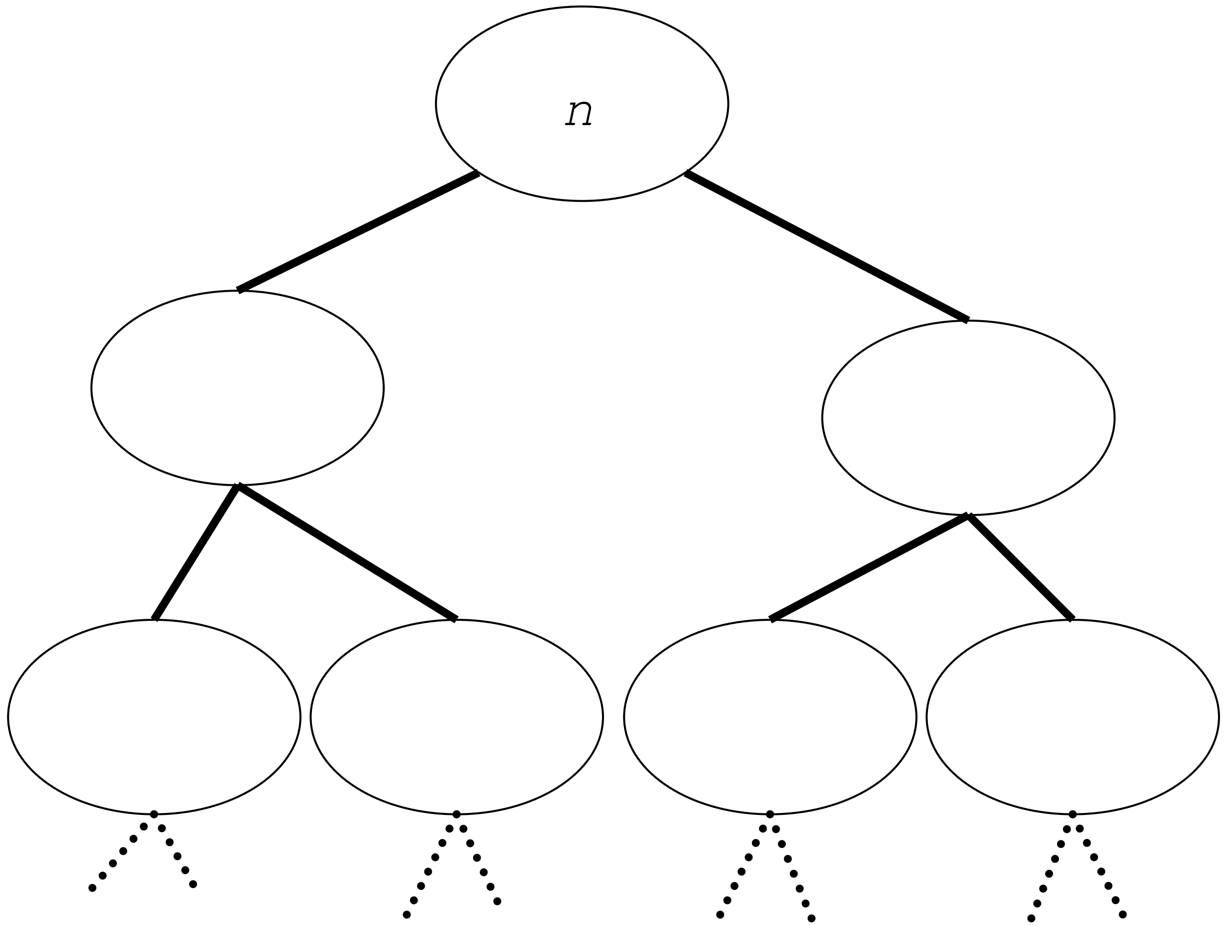
- Let S be an empty sequence.
- Put the elements back into S in order by first inserting the elements of L, then those of E, and finally those of G.

How fast can we implement concatenate?

Quicksort: running time analysis

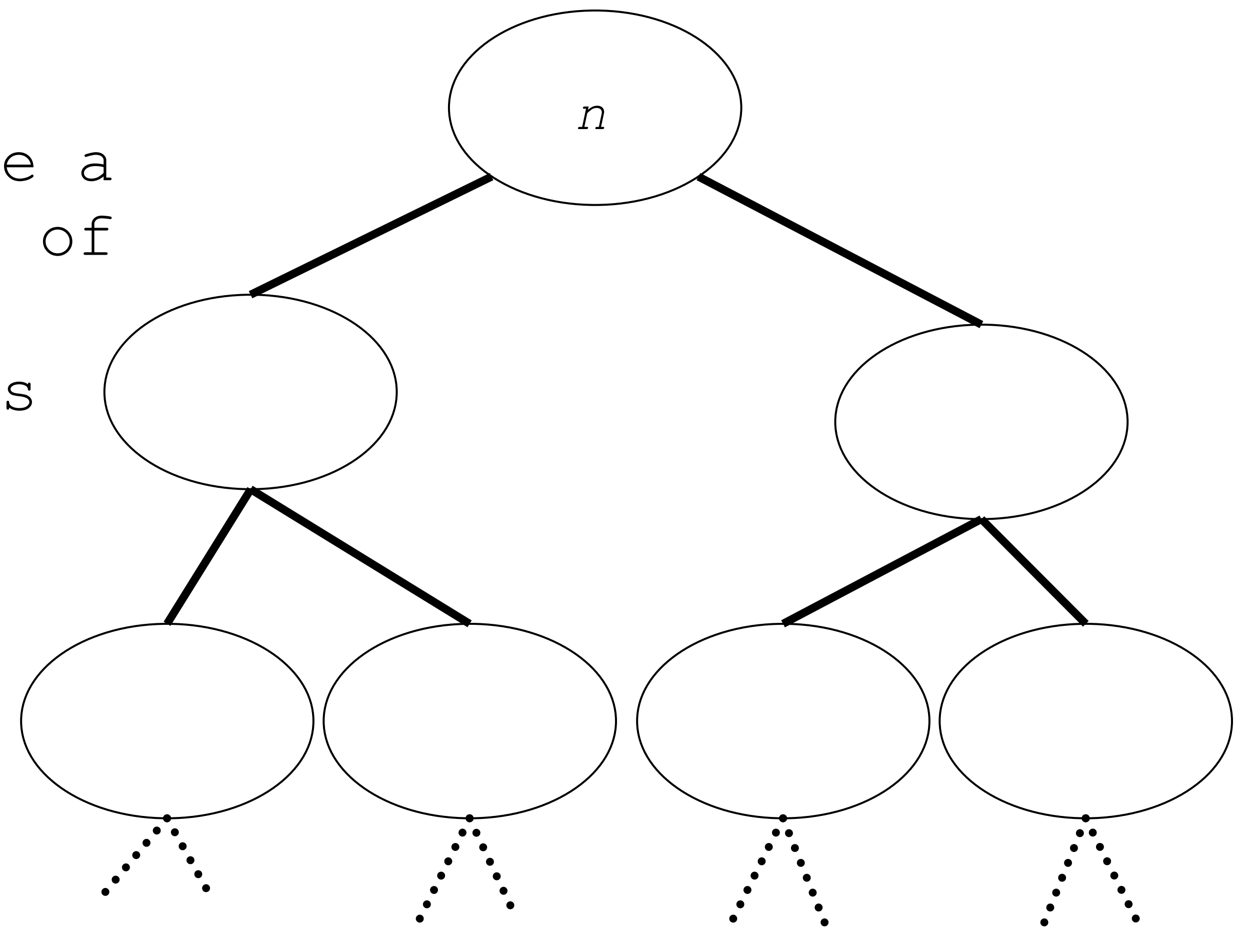
- How long can a branch in the Quicksort tree be?
- What is the worst-case running time of Quicksort?
- What sequences require the worst-case running time?
- What is the best-case running time?
- Why is Quicksort called *quick* sort?

How long can a branch in the Quicksort tree be in the worst case?



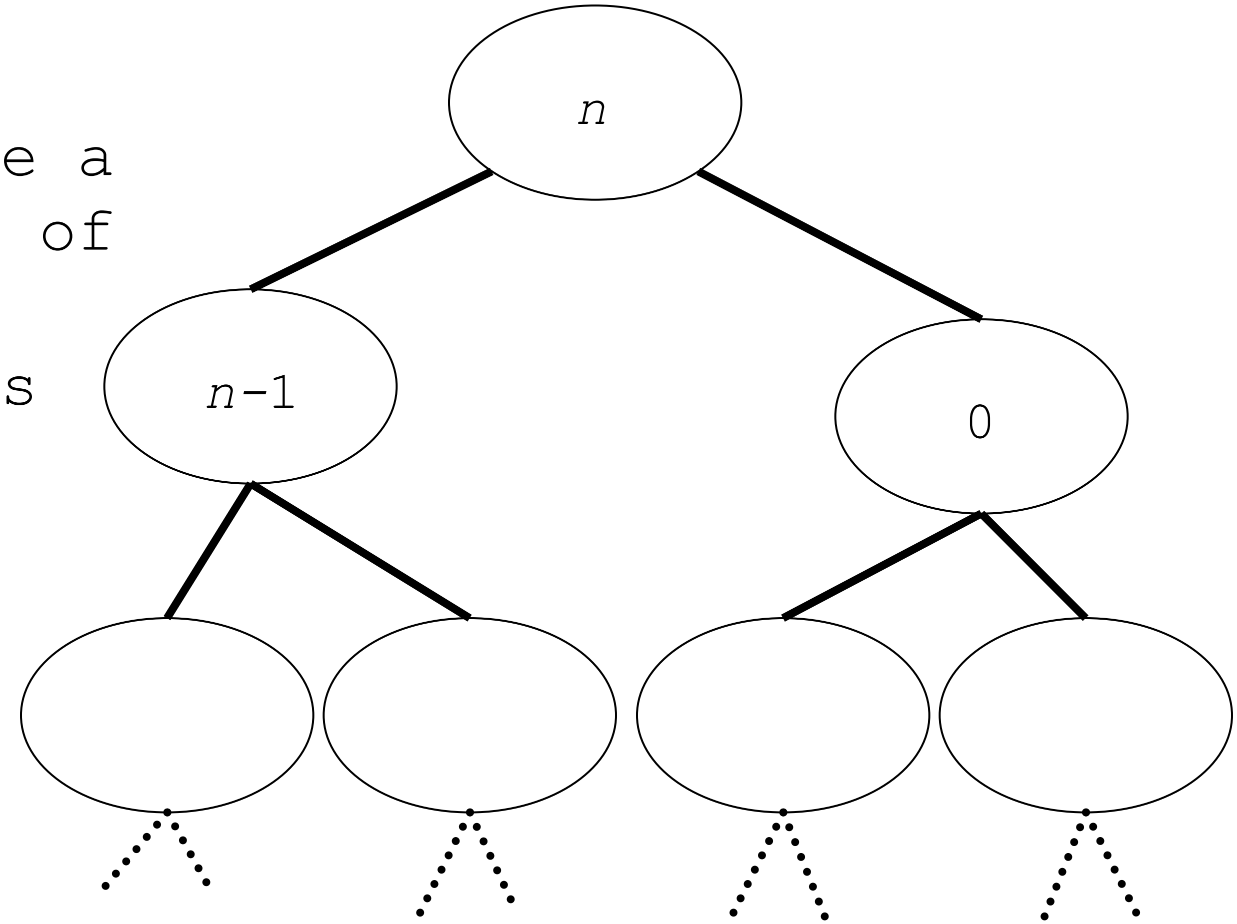
The pivot x and the length of sequences L and G

Let x be a largest of all elements



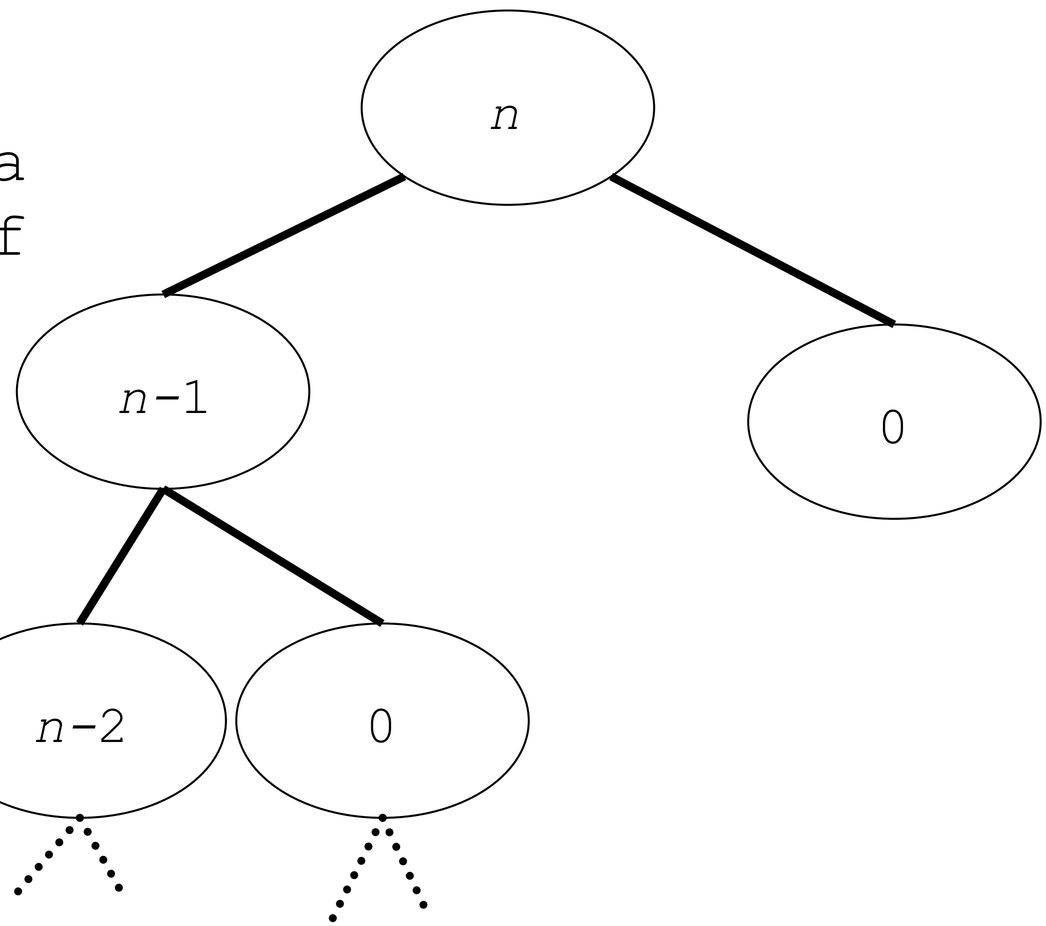
The pivot element and the length of sequences L and G

Let x be a largest of all elements



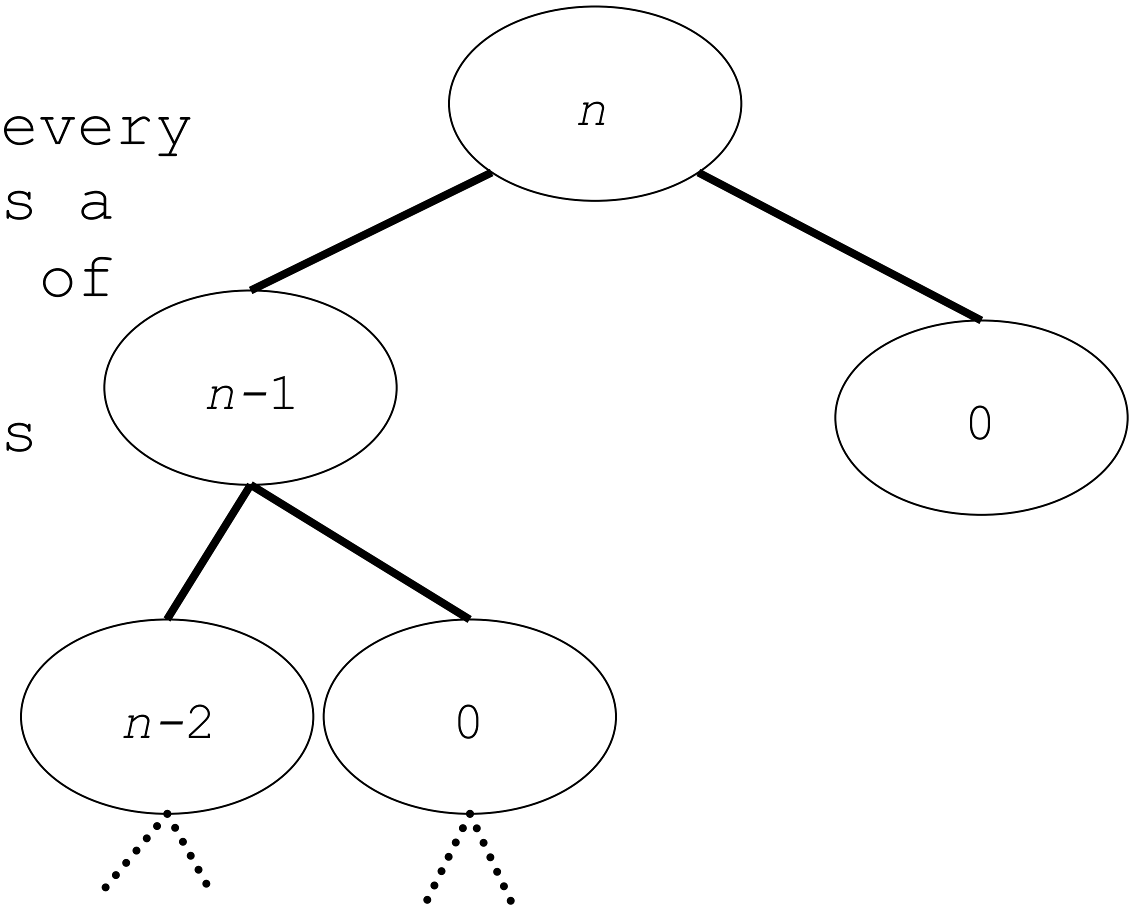
The pivot element and the length of sequences L and G

Let x be a largest of all elements

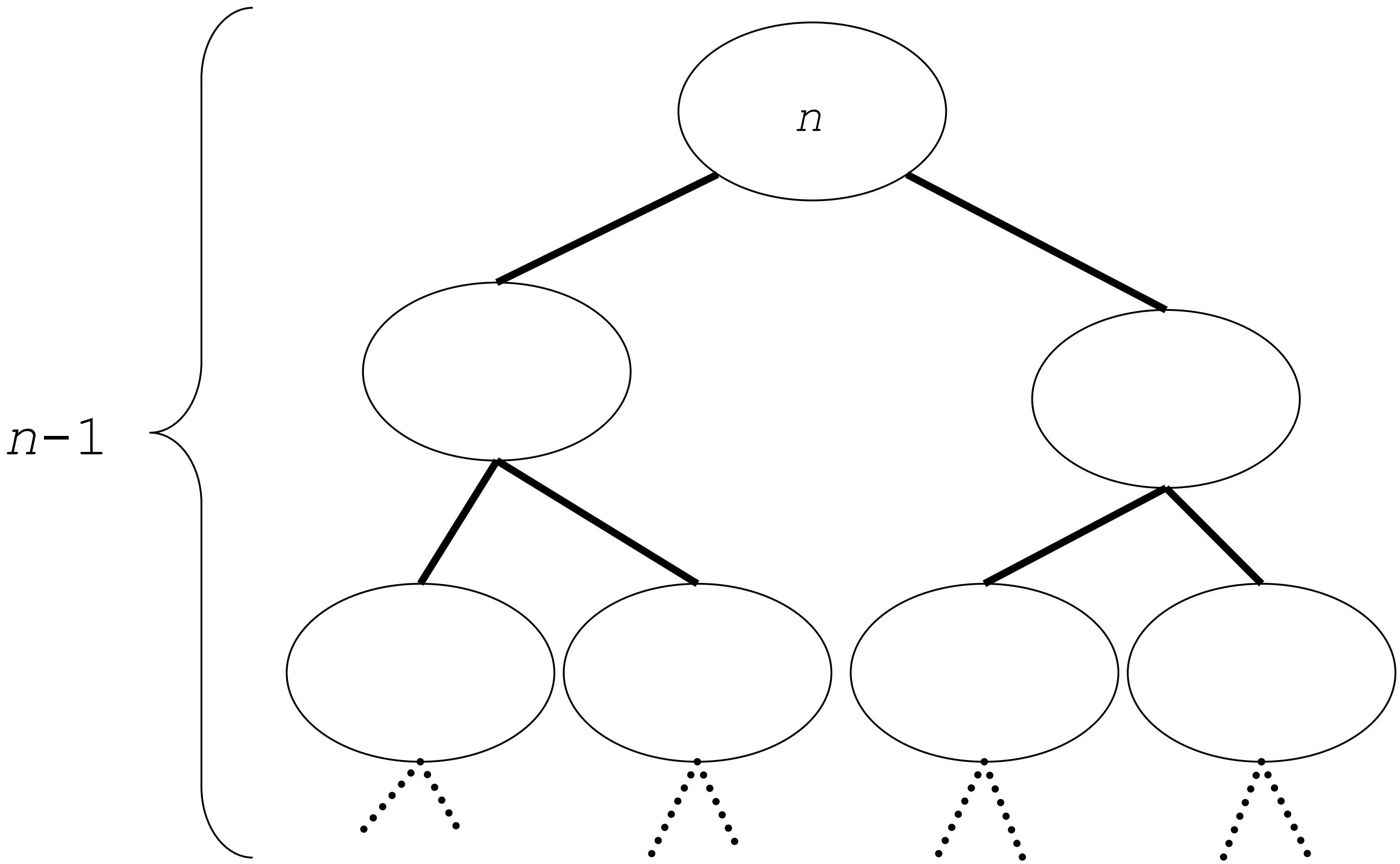


What sequences require the longest branch?

Assume every pivot is a largest of all the elements to sort



How long can be a branch in the quick-sort tree?



What sequences require the worst-case running time?

- Sorted sequences

1	2	3	4	5	6	7	8
8	7	6	5	4	3	2	1

What is the worst-case running time of Quicksort?

Create L, G and E in each level of the "tree".

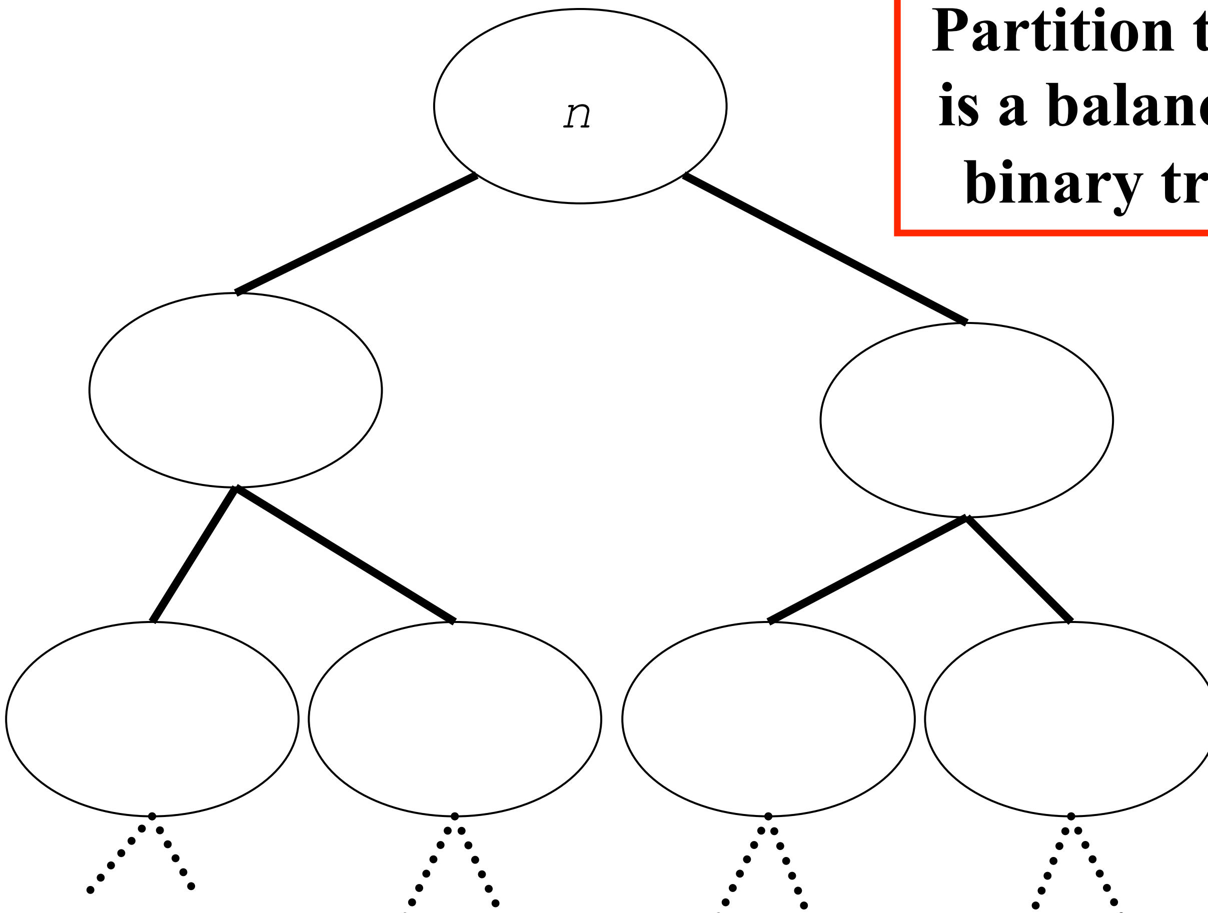
$$\sum_{i=1}^{n-1} i \text{ is } O(n^2)$$

Concatenate L, G and E in each level of the "tree".

$$\sum_{i=1}^{n} i \text{ is } O(n^2)$$

$\mathrm{O}(n^2)$

When is Quicksort fastest?



Partition tree is a balanced binary tree

A best case running time for Quicksort

$$O(n \log n)$$

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. Why that is the case we show a little later in the course.
- Suppose we can show that we can split up a $\frac{1}{4}n$ 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

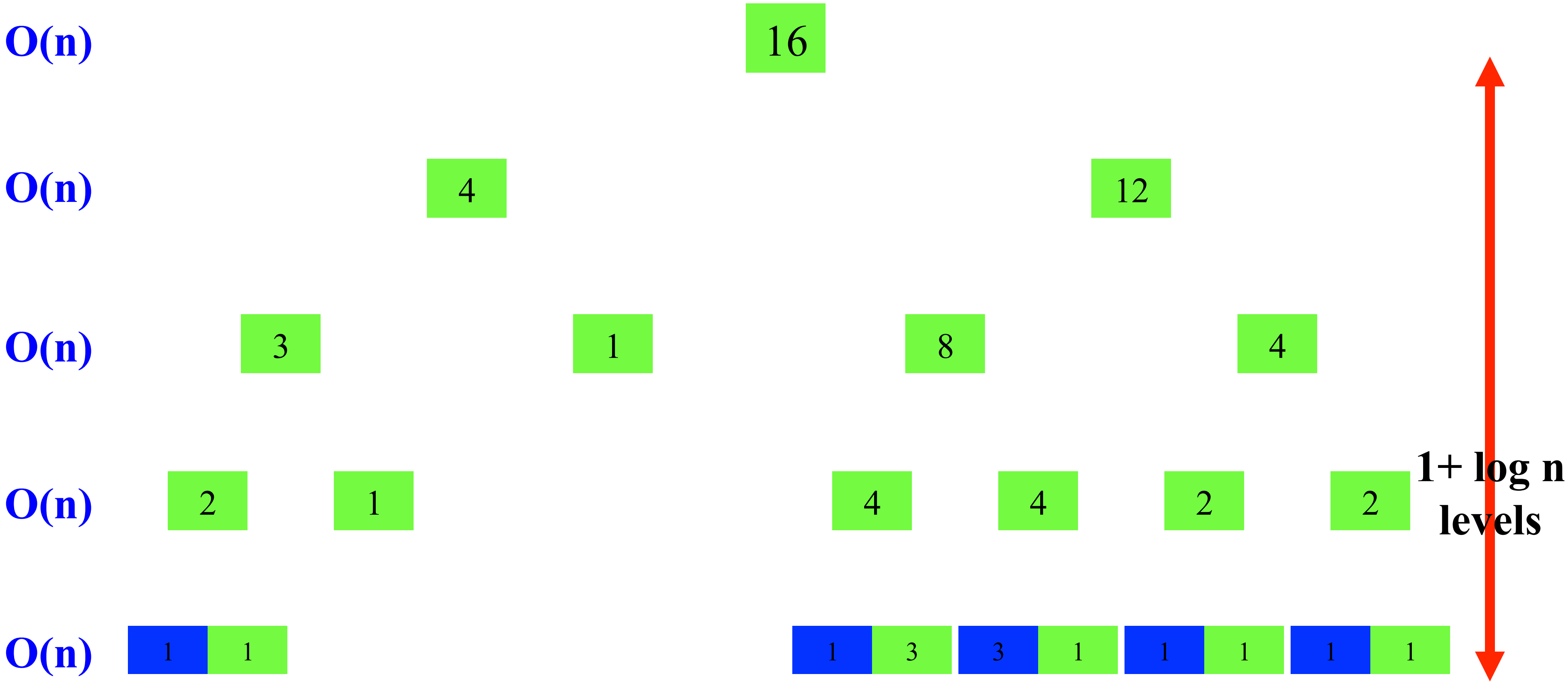
Proof

- 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 $2\log_{4/3} n$

Height of Recursion Tree



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

Proof

- How many pivot 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 $O(n \log n)$**

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 $k \leq n$
Output: k^{th} smallest element in sorted sequence S

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

Let L, E, G be empty sequences

$p \leftarrow$ pickPivot(S)

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

QuickSort(L)

QuickSort(G)

Concatenate(L, E, G, S)

return S

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
$p \leftarrow$ pickPivot(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())

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

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

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

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

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

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

$$\left(\frac{3}{4}\right)T_{QS}(n) = n\left[\left(\frac{3}{4}\right)^1 + \left(\frac{3}{4}\right)^2 + \left(\frac{3}{4}\right)^3 + \left(\frac{3}{4}\right)^4 + \left(\frac{3}{4}\right)^5 + \ldots + \left(\frac{3}{4}\right)^{2\log_{4/3} n} + \left(\frac{3}{4}\right)^{1+2\log_{4/3} n}\right]$$

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

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

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.