CSC 225

Algorithms and Data Structures I

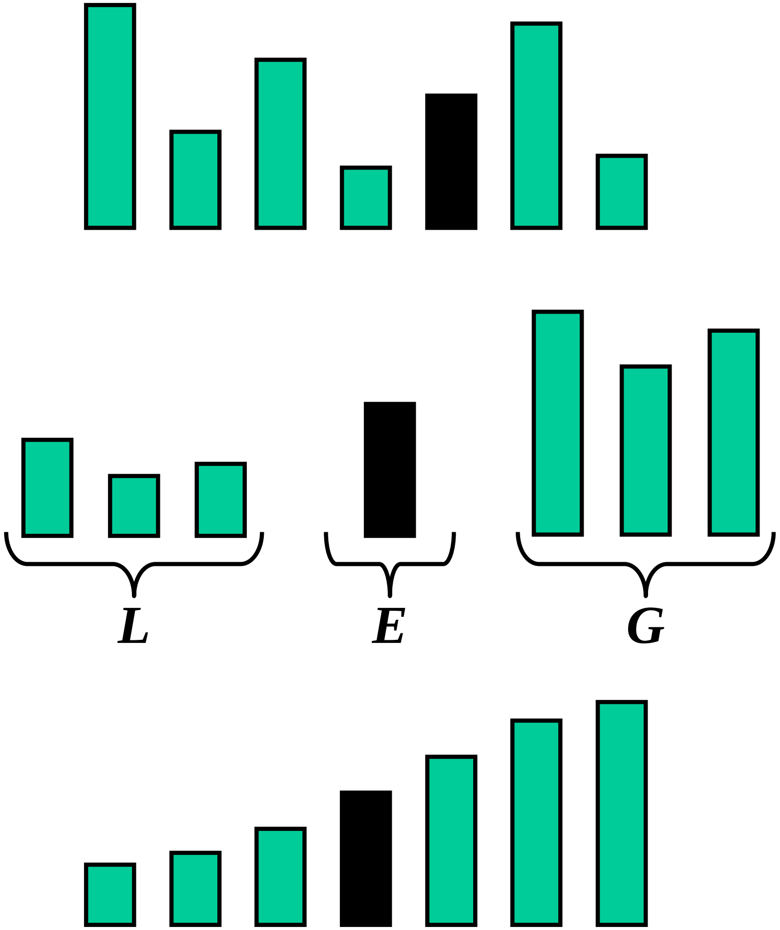
Rich Little
rlittle@uvic.ca
ECS 516

Algorithm Design Technique
Divide and Conquer: Quicksort

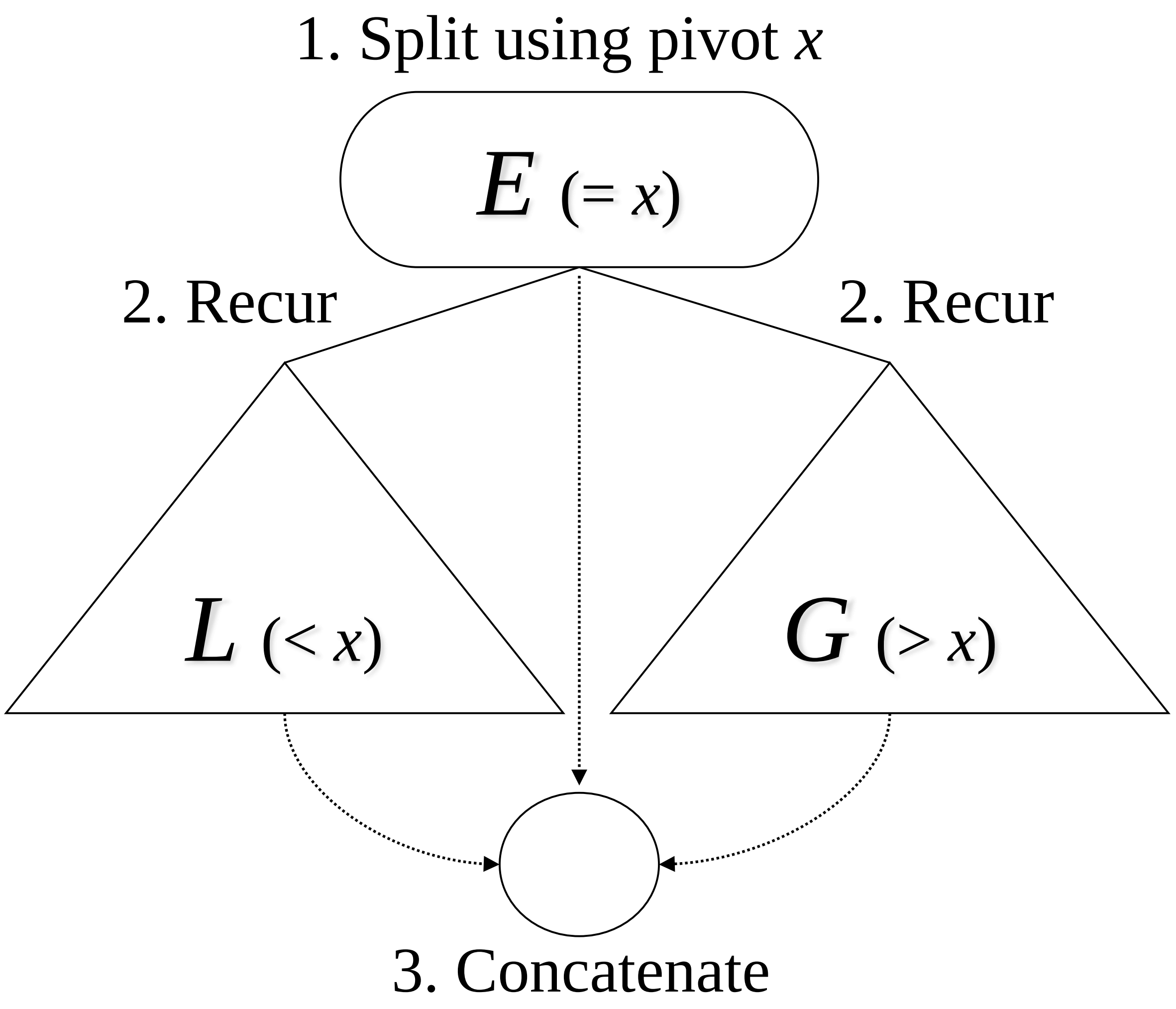
- Mergesort divides the input set according to the position of the elements (i.e., first and second part of sequence)
- Quicksort divides the input set according to the value of the elements

Quicksort
based on ADT Sequence

- Quick-sort is a sorting algorithm based on the divide-and-conquer paradigm:
  - Divide: pick an 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***



Quicksort Algorithm



Example

Let $S = [8,1,11,4,12,3,7,5]$ and sort using quick-sort.

Algorithm quickSort(S)

```
if S.size() < 2 then
    return S
x ← pickPivot(S)
split(L,E,G,S,x)
L ← quickSort(L)
G ← quickSort(G)
concatenate(L,E,G,S)
return S
```

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.

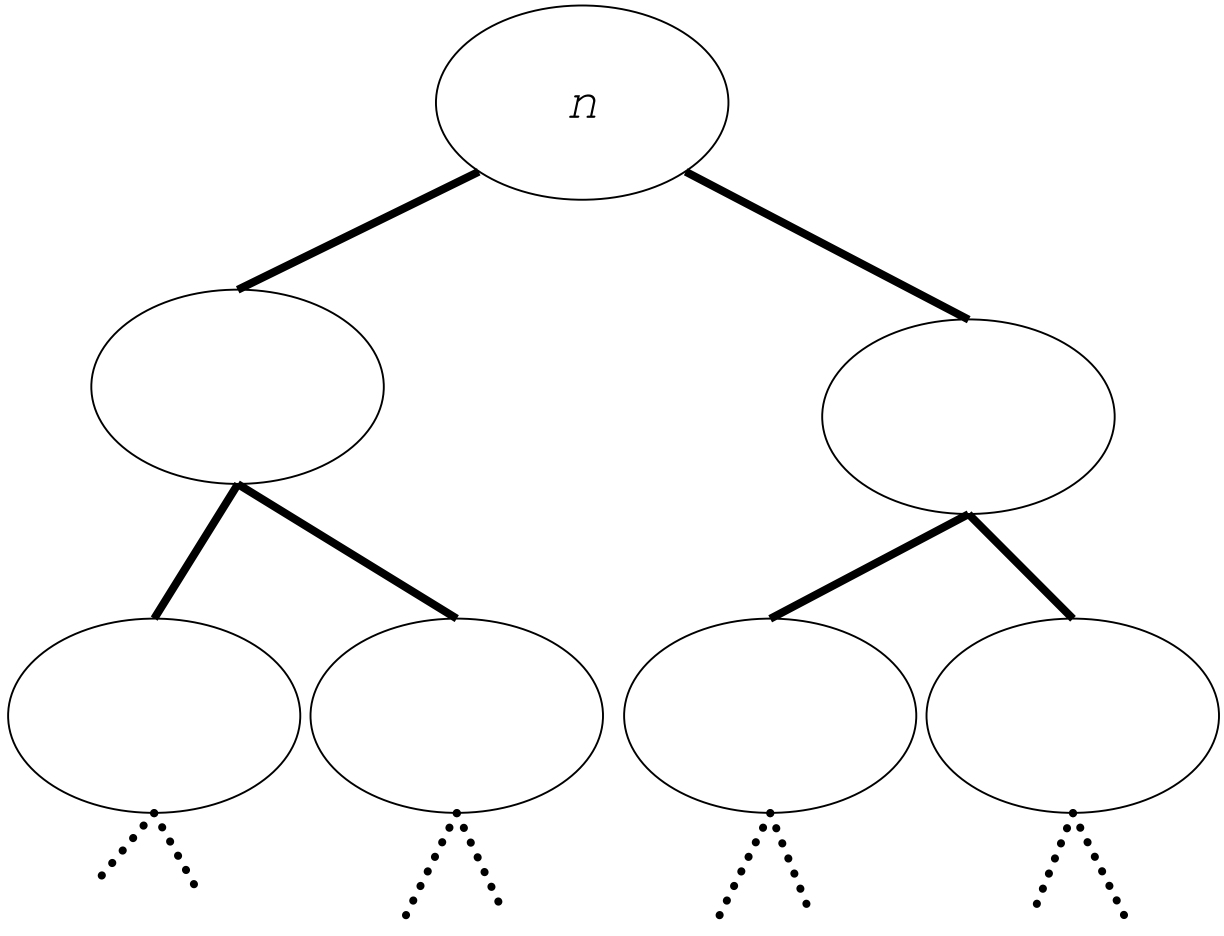
Algorithm concatenate(L, E, G, S)

- 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.

Quicksort: running time analysis

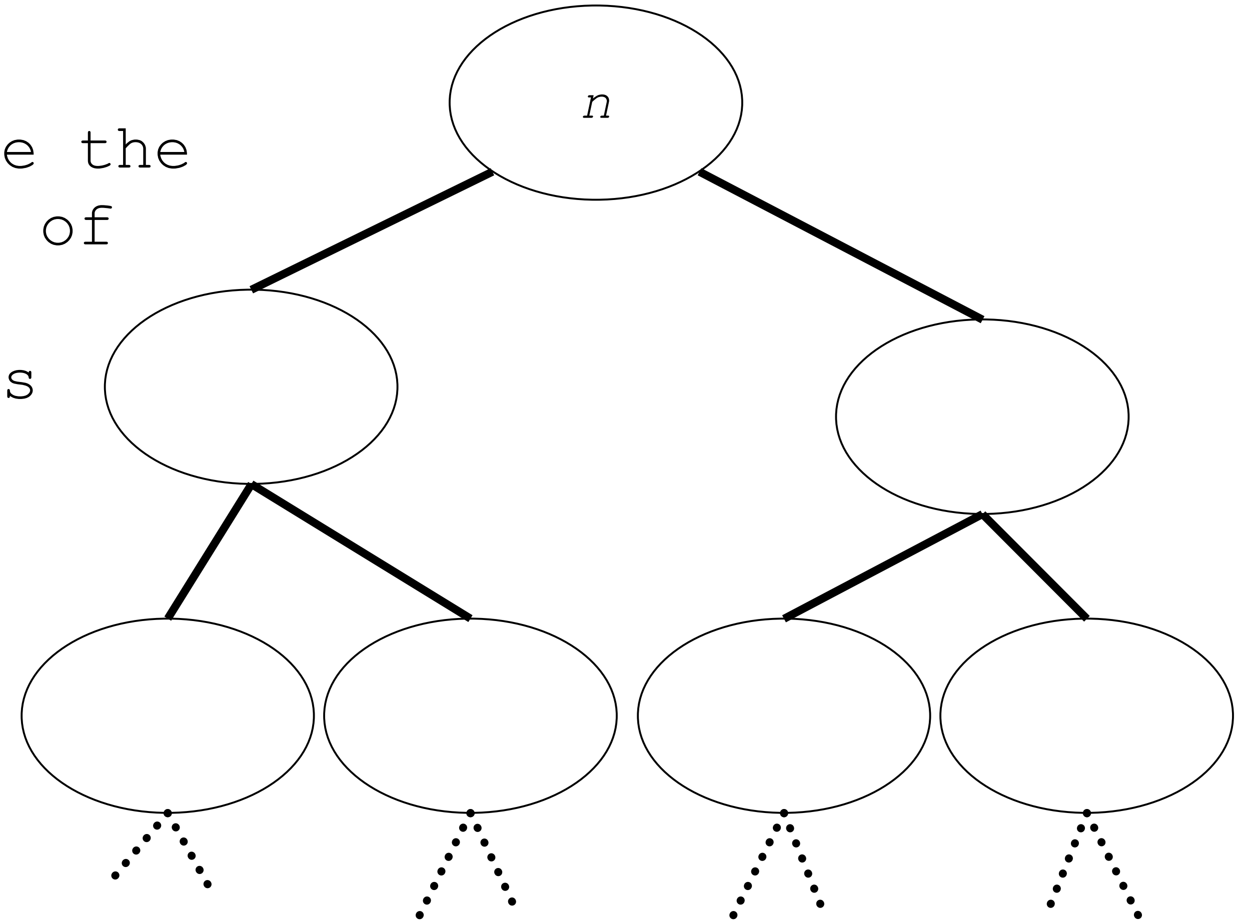
- How long can a branch in the Quicksort tree be?
- What is the worst-case running time of Quicksort?
- What sequences require the worst-case running time?
- What is the best-case running time?
- Why is Quicksort called *quick* sort?

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



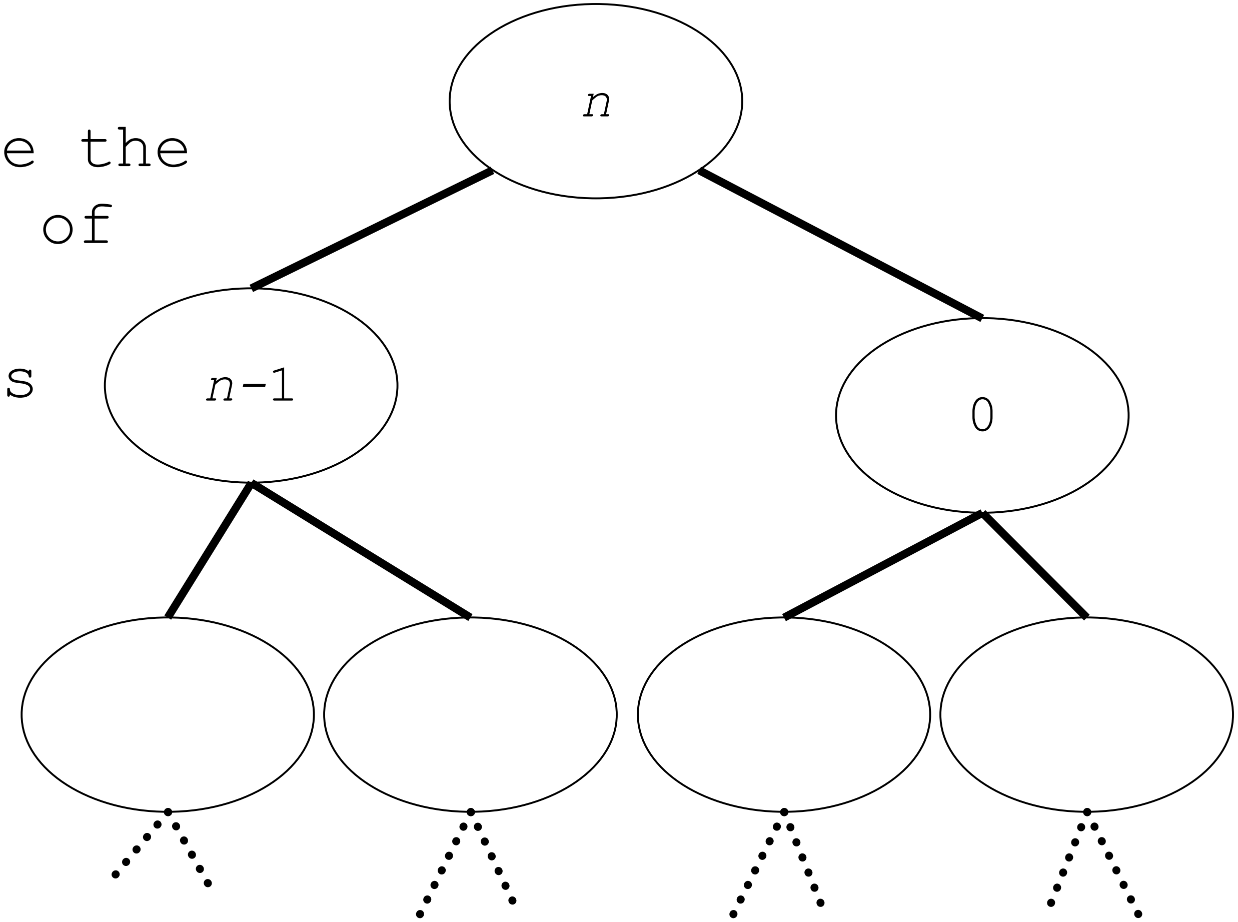
The pivot p and the length of sequences L and G

Let x be the largest of all elements



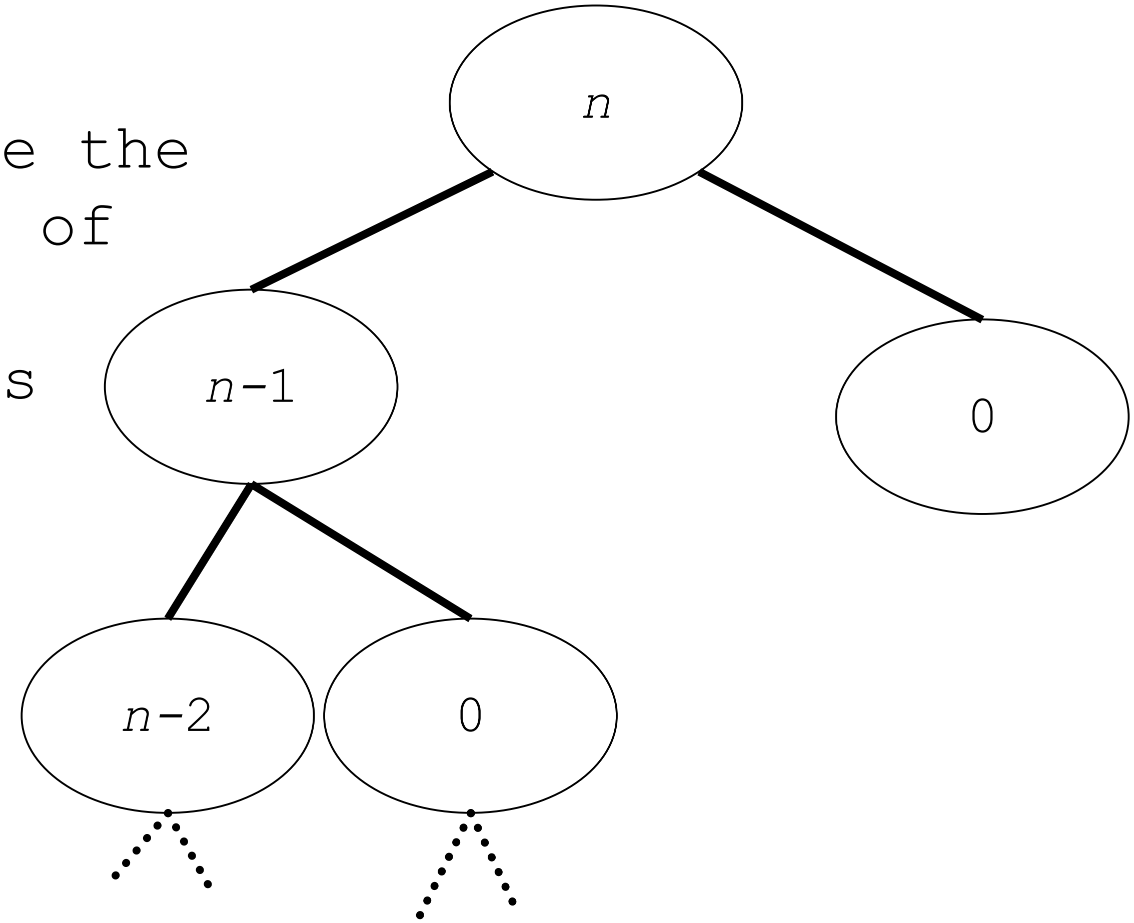
The pivot element and the length of sequences L and G

Let x be the largest of all elements



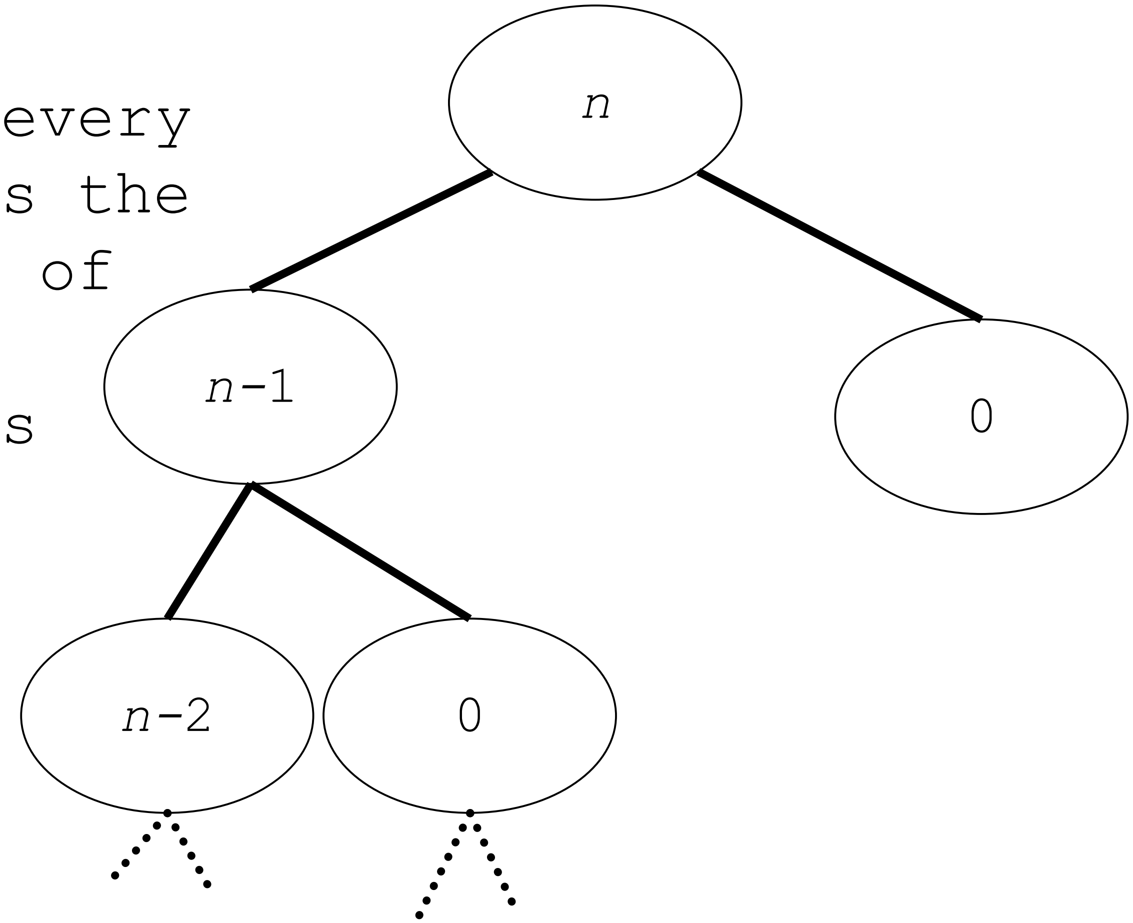
The pivot element and the length of sequences *L* and *G*

Let x be the largest of all elements



What sequences require the longest branch?

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



Worst-case Running Time of Quick-Sort

```
if S.size() < 2 then
    return S
x ← pickPivot(S)
split(L,E,G,S,x)
L ← quickSort(L)
G ← quickSort(G)
concatenate(L,E,G,S)
return S
```

Solve Recurrence Equation by Repeated Substitution

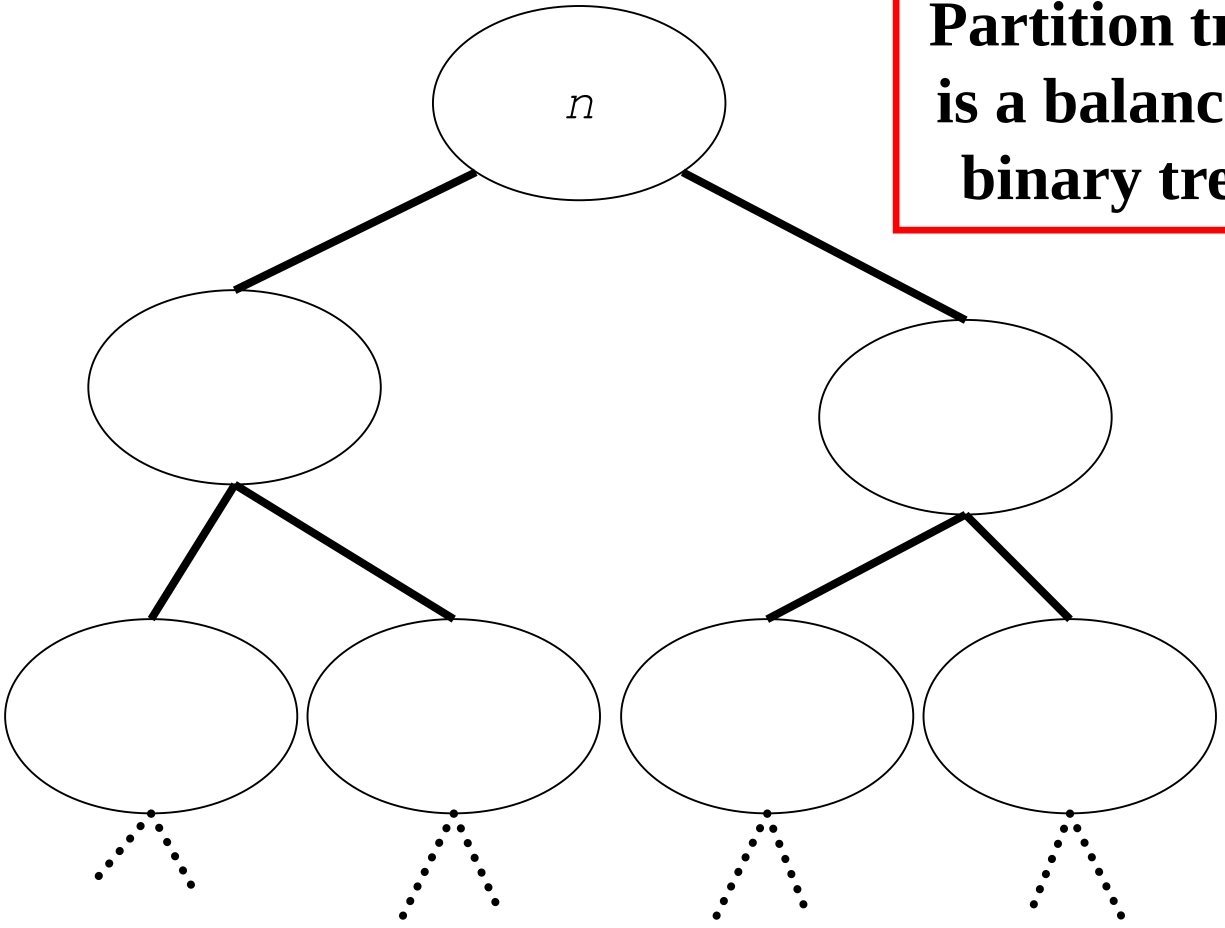
What sequences require the worst-case running time?

- Sorted sequences

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

- For simplicity sake we are pivoting on the last element here.

When is Quicksort fastest?



Partition tree is a balanced binary tree

A best case running time for Quicksort

$O(n \log n)$

Algorithm inPlaceQuickSort(S,a,b)

Input: Array S, ints a and b

Output: Subarray $S[a..b]$ sorted

```
if a ≥ b then return
  l ← inPlacePartition(S,a,b)
  inPlaceQuickSort(S,a,l-1)
  inPlaceQuickSort(S,l+1,b)
end
```

In-Place Partitioning

- Perform the partition using two indices to split S into L, E and G.

l *r*

3 2 5 1 0 7 3 5 9 2 7 9 8 9 7 9 **<u>6</u>** (pivot = 6)

- Repeat until *l* and *r* cross:
  - Scan *l* to the right until finding an element > *p*.
  - Scan *r* to the left until finding an element < *p*.
  - Swap elements

l *r*

3 2 5 1 0 $\boxed{7}$ 3 5 9 $\boxed{2}$ 7 9 8 9 7 9 **<u>6</u>**

Algorithm inPlacePartition(S,a,b)

```
Input: Array S, ints a ≤ b
Output: int l, pivot index
r ← randomInt(a,b)
swap(S[r],S[b])
p ← S[b]
l ← a
r ← b-1
while l ≤ r do
   while l ≤ r and S[l] ≤ p do
        l ← l + 1
   while l ≤ r and S[r] ≥ p do
        r ← r - 1
   if l < r then
        swap(S[l],S[r])
swap(S[l],S[b])
return l
```

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
  - Last element
  - Middle element
  - Average of three elements
  - Compute the average of 5 or 7 elements
  - Randomized selection of pivot

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 other $O(n \log n)$ worst-case sorting algorithms
- The actual running time makes the difference
  - $T_{Quick}(n) = 1.18\ n \log n$
  - $T_{Heap}(n) = 2.22\ n \log n$