Quicksort

Chapter 7

Sorting

- What's the running time for:
  - Insertion Sort
  - Merge Sort
  - Heapsort
- Which of these algorithms sort in place?

Quicksort

- The Basic version of quicksort was invented by C. A. R. Hoare in 1960
- Divide and Conquer algorithm
- In practice, it is the fastest in-place sorting algorithm

Divide and Conquer

- Divide: Partition the array into two subarrays around a pivot x such that elements to the left are <= x and elements to the right are >= x

$\leq x$	X	$\geq x$

- Conquer: Recursively sort the two subarrays
- Combine: Trivial!

Key? Good Partitioning Subroutine!

Quicksort Pseudocode

QUICKSORT(A, p, r)

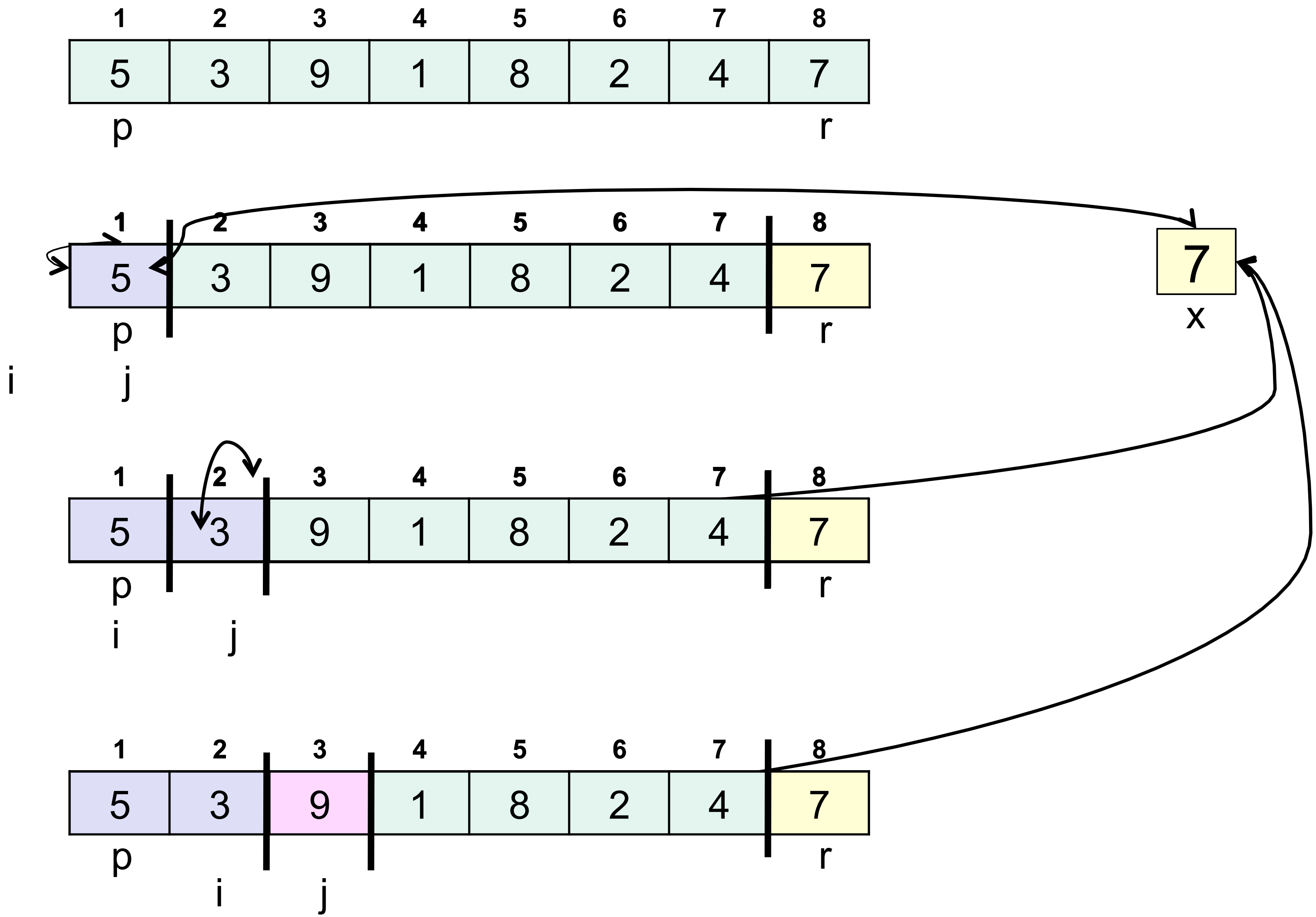
```
Quicksort(A, p, r) // A:Array; p,r: integer indexes
if p < r
  q = Partition(A, p, r);
  Quicksort(A, p, q-1);
  Quicksort(A, q+1, r);
```

- What's the call to sort the entire array?

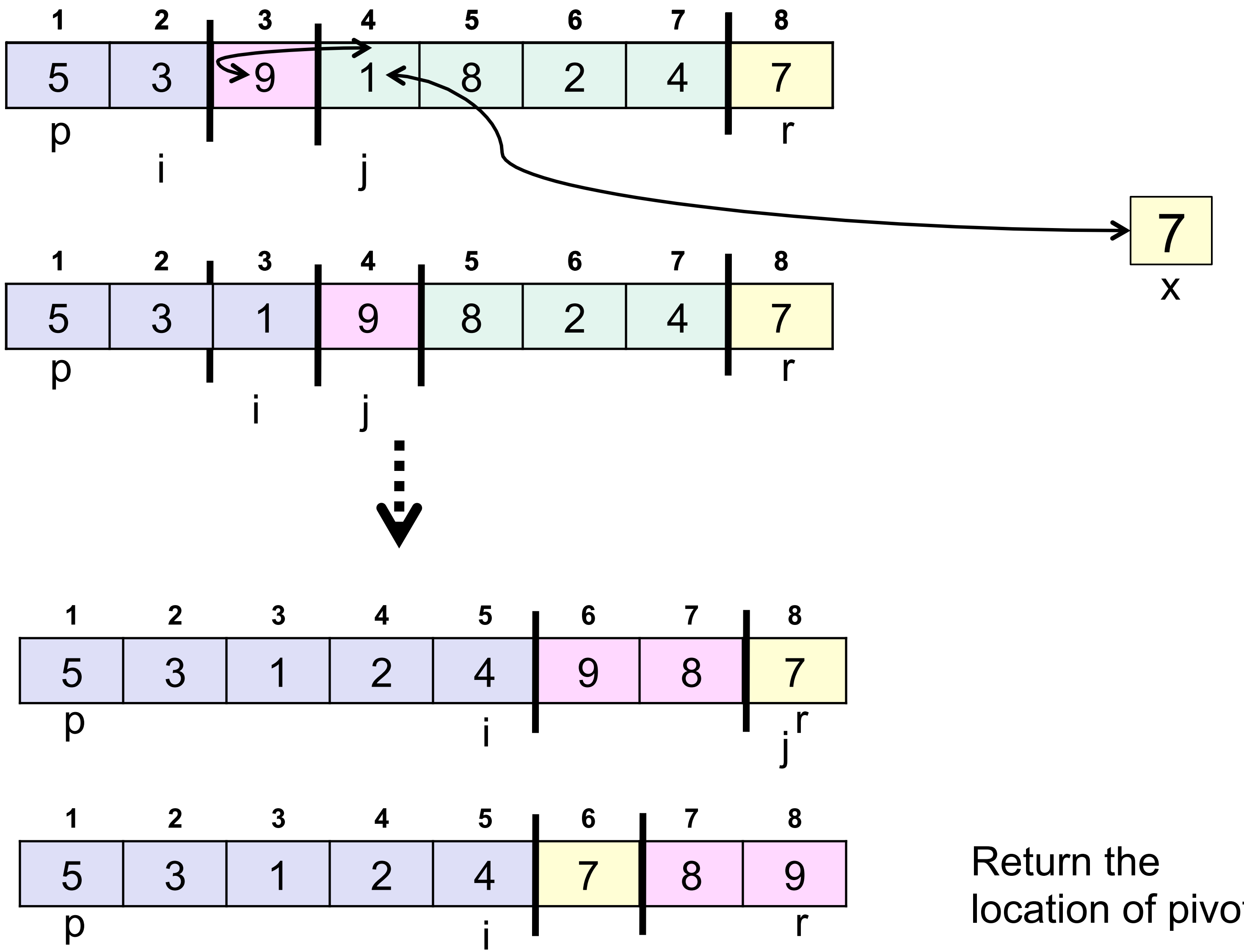
Partitioning the Array

```
Partition(A,p,r) // A:Array; p,r: integer indexes
x = A[r]
i = p - 1
for j = p to r-1
  if A[j] <= x
    i = i + 1
    swap(A[i], A[j])
swap (A[i+1], A[r])
return i+1
```

Example

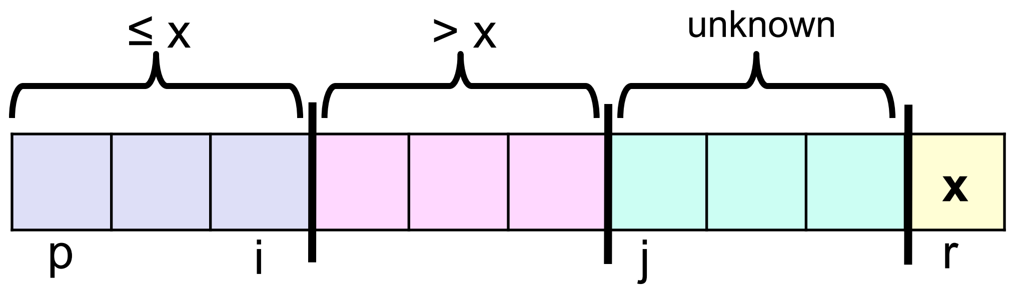

1 2 3 4 5 6 7 8
5 3 9 1 8 2 4 7
p r
1 2 3 4 5 6 7 8
5 3 9 1 8 2 4 7
p r
i j
7
x
1 2 3 4 5 6 7 8
5 3 9 1 8 2 4 7
p r
i j
1 2 3 4 5 6 7 8
5 3 9 1 8 2 4 7
p r
i j

Example


1 2 3 4 5 6 7 8
5 3 9 1 8 2 4 7
p i j r
7
x
1 2 3 4 5 6 7 8
5 3 1 9 8 2 4 7
p i j r
1 2 3 4 5 6 7 8
5 3 1 2 4 9 8 7
p i j r
1 2 3 4 5 6 7 8
5 3 1 2 4 7 8 9
p i r
Return the location of pivot

Correctness of Partition

- During the execution of PARTITION there are four distinct sections of the array:



Exercise - Partition the Following

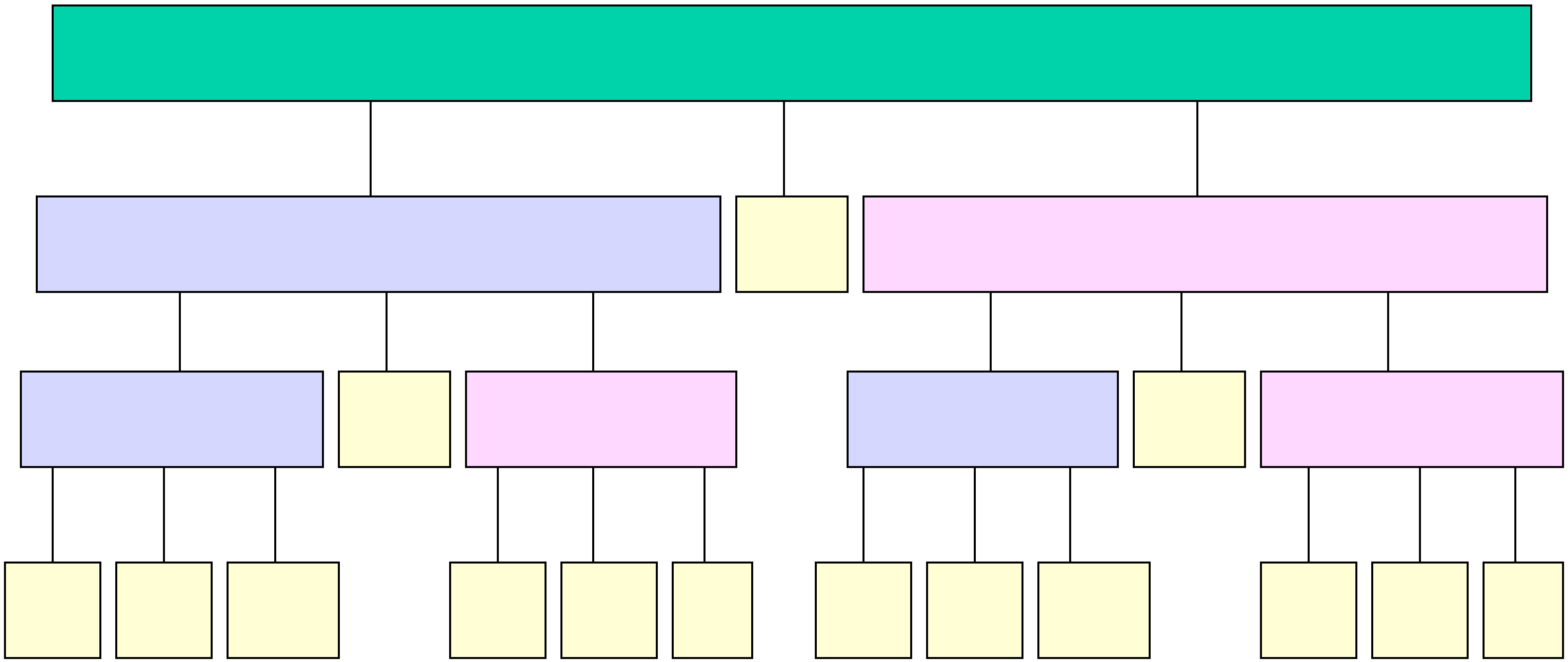
44	**75**	**23**	**43**	**55**	**12**	**64**	**77**	**33**	**41**

Analysis of Partition

- What is the running time of PARTITION?

```
Partition(A,p,r) // A:Array; p,r: integer indexes
x = A[r]
i = p - 1
for j = p to r-1
  if A[j] <= x
    i = i + 1
    swap(A[i], A[j])
swap (A[i+1], A[r])
return i+1
```

Quicksort in Action



Exercise

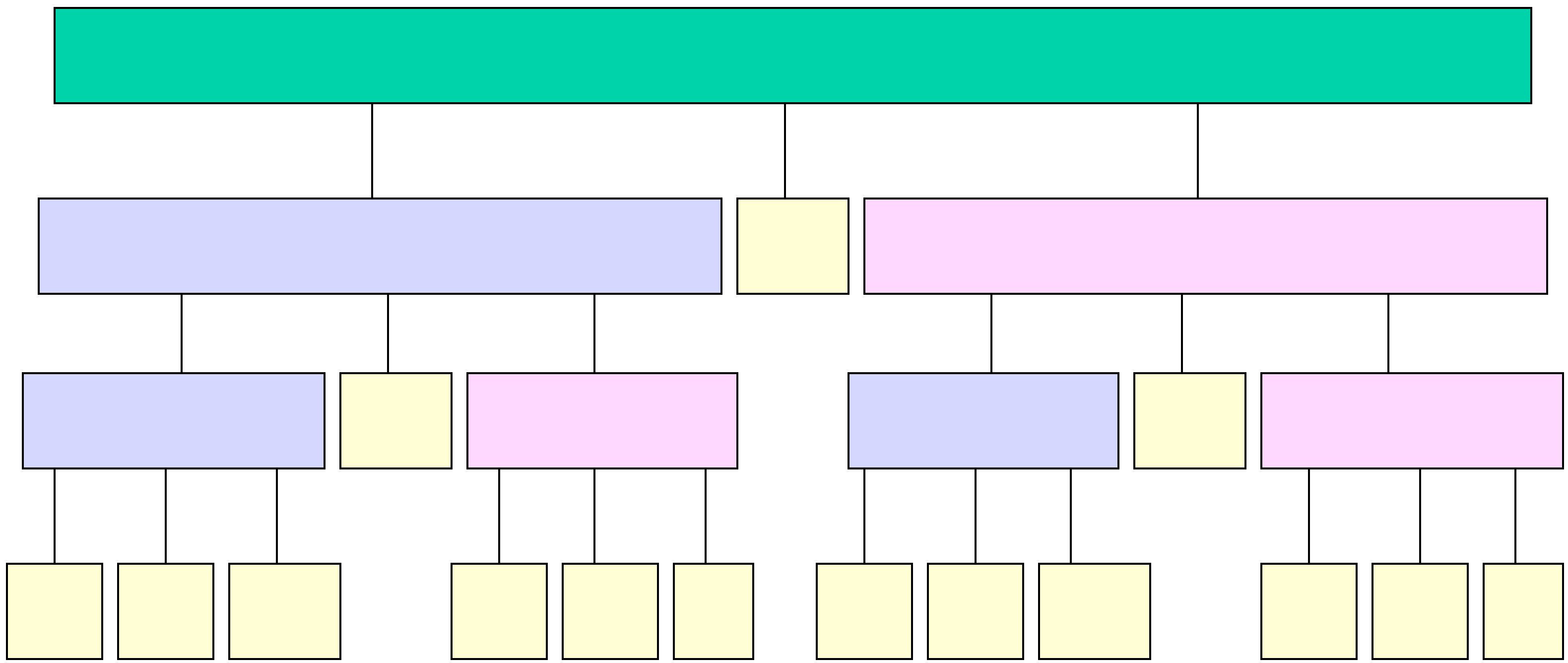
- Sort the following array using quicksort

3	4	2	5	1

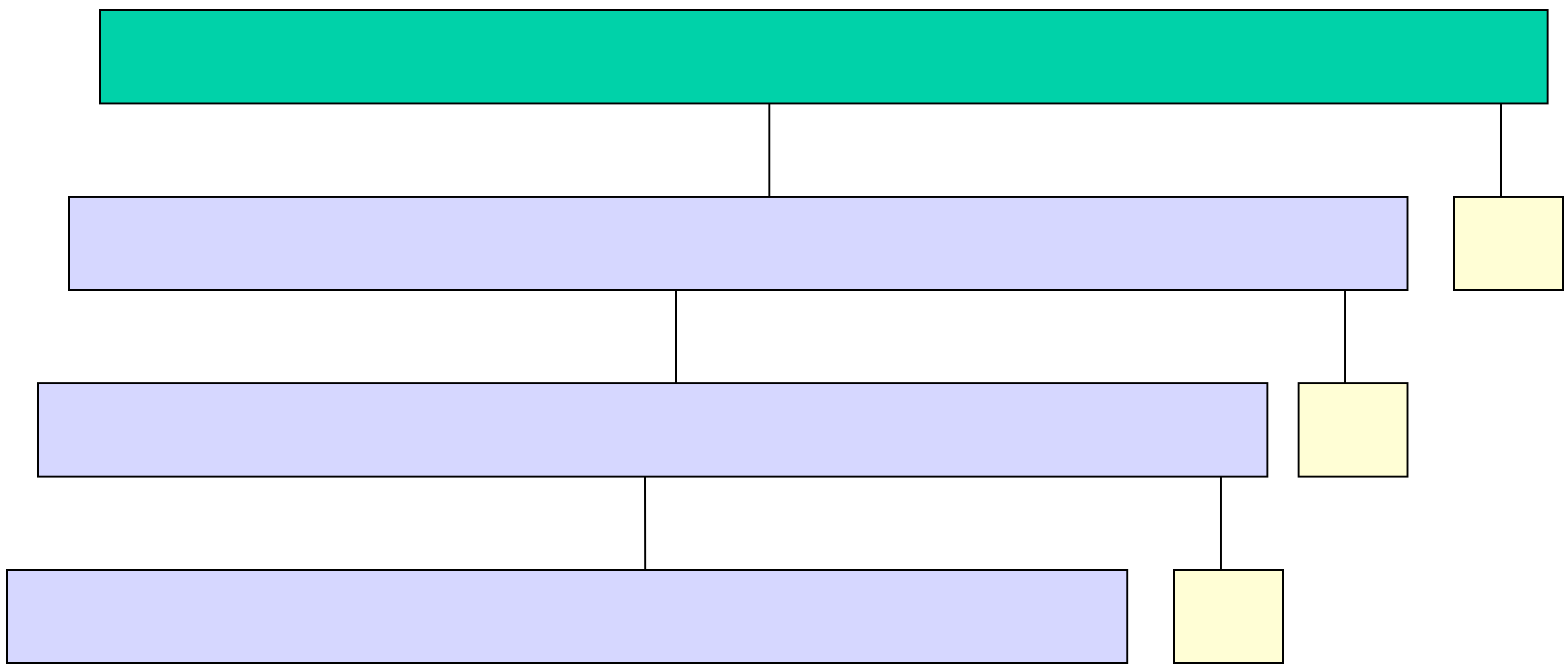
Performance of Quicksort

- What does the performance of quicksort depend on?
- What would give us the best case?

Best Case of Quicksort



Worst Case of Quick Sort



Quicksort Analysis

- To justify its name, Quicksort had better be good in the average case.
- Showing this requires some intricate analysis.

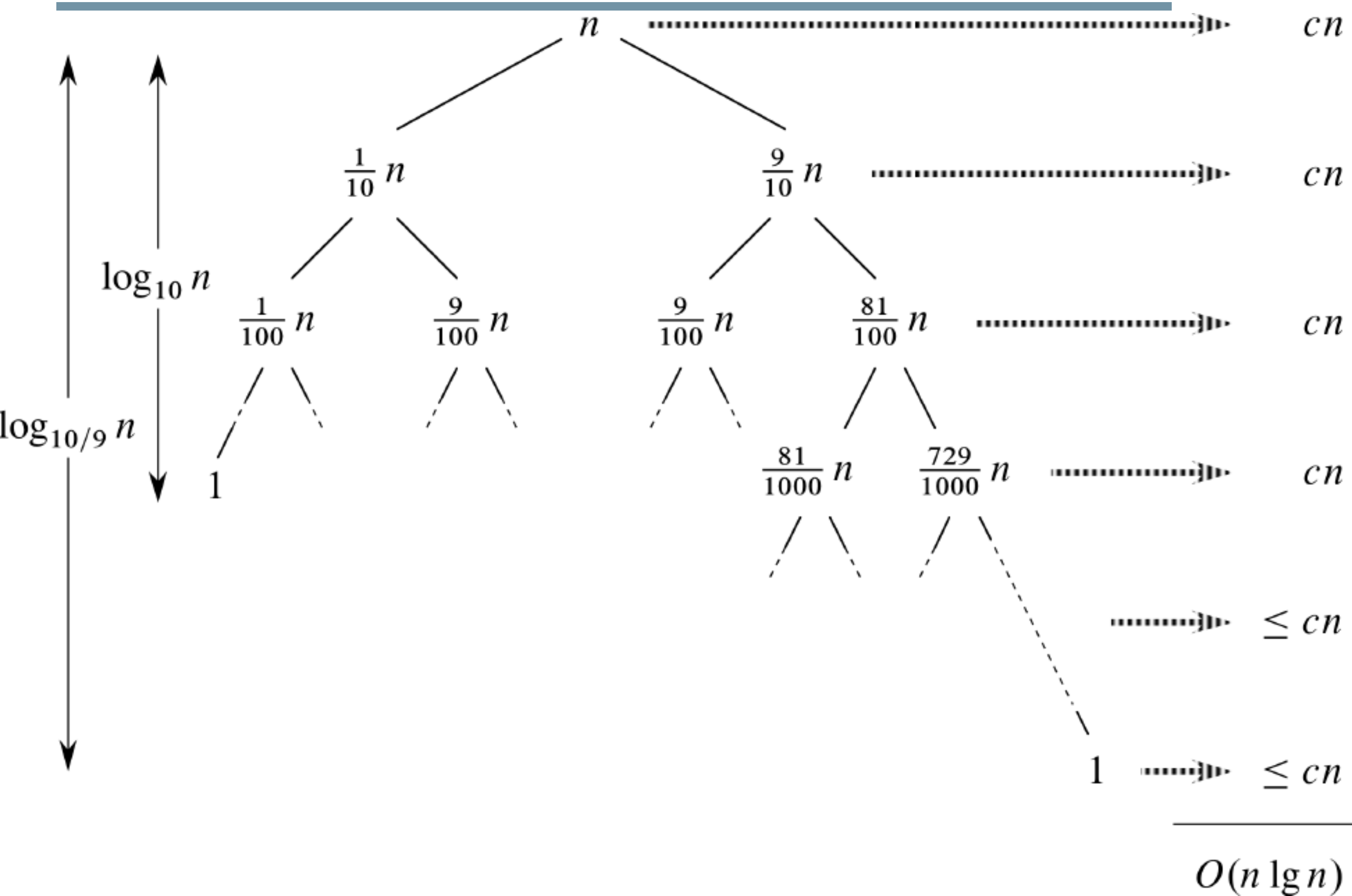
Average Case Analysis

- Let’s look at this by intuition
- Running quicksort on a random array is likely to produce a mix of balanced and unbalanced partitions
- It has been shown that 80% of the time partition produces good splits and 20% of the time it produces bad splits

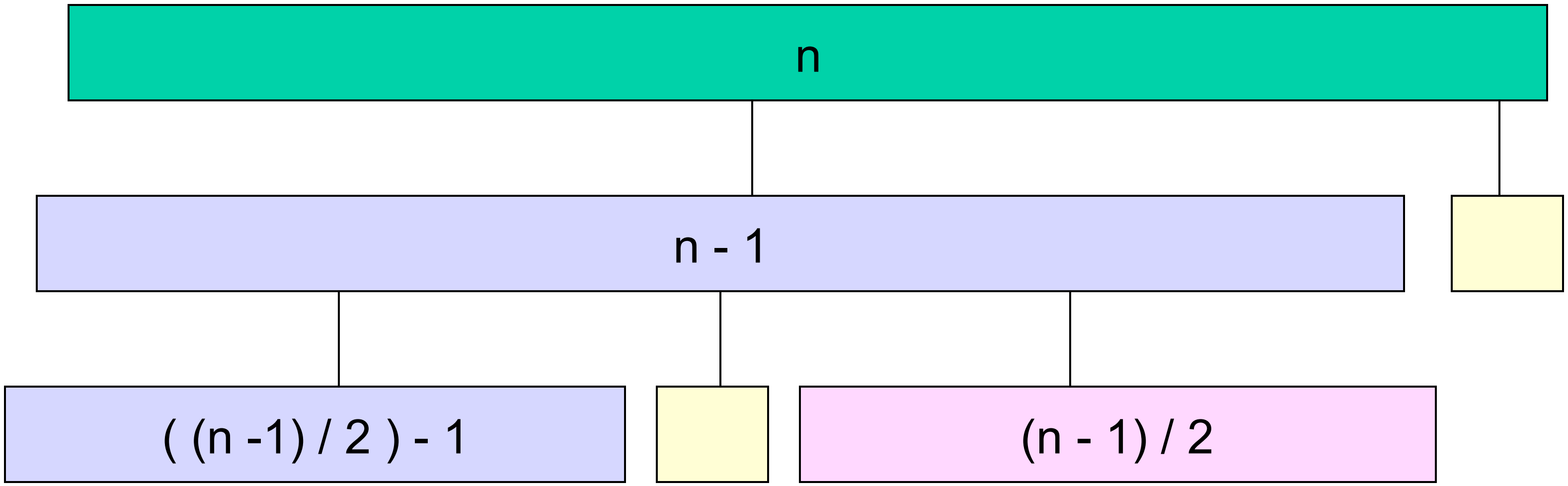
Assume 9-1 split, p 176

- Assume each partition is a 9 to 1 split.
  - constant proportionality
- What is the recurrence?

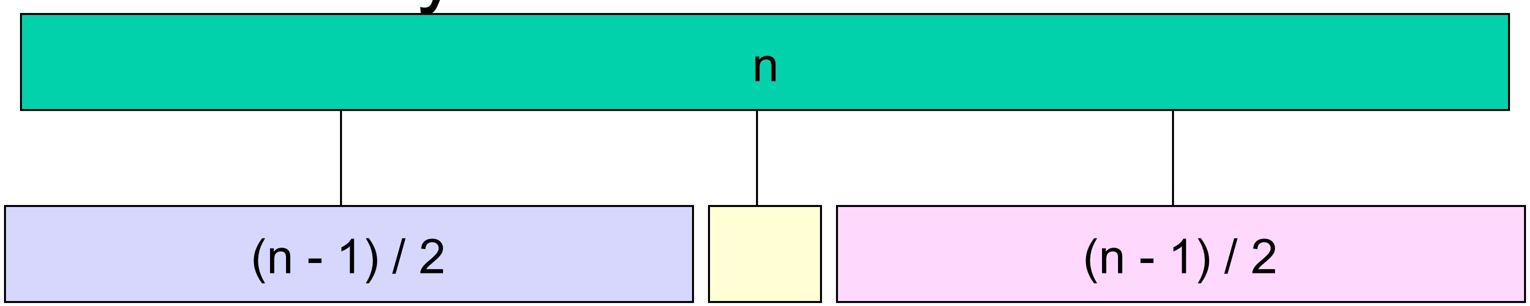
Fig 7.4 What does the recursion tree look like (9-1 split)?



Average Case Analysis



- This is really no different than:



- Thus, the O(n -1) of the bad split can be absorbed into the O(n) of the good split

Average Case Analysis

- The running time of quicksort when alternating good and bad splits is like the running time for good splits alone
- O(n lg n) but with a slightly larger constant hidden by the O-notation

Random Partition, p 179

```
Randomized-Partition(A, p, r)
i = RANDOM(p,r)
swap (A[r], A[i])
return PARTITION(A, p, r)
```

Hoare Partition, p 185

```
HoareParition(A,p,r)
x = A[p]
i = p -1
j = r + 1
while TRUE
   do
      j=j-1
   while(A[j] > x)
   do
     i = i+1
   while( A[i] < x)
   if ( i < j)
      swap (A[i], A[j])
   else return j
```