Quick Sort

Lecture 8

Quick Sort

- Definitely a “greatest hit: algorithm
- Prevalent in practice
- Beautiful analysis
- O(n lg n) time “on average” works in place
  - i.e., minimal extra memory needed

The Sorting Problem

- Input : array of n numbers, unsorted
- Output : Same numbers, sorted in increasing order

  Assume : all array entries distinct.

3	8	2	5	1	4	7	6

Partitioning Around a Pivot

- Key Idea : partition array around a pivot element
  -- Pick element of array
  -- Rearrange array so that
  -- Left of pivot => less than pivot
  -- Right of pivot => greater than pivot
  Note : puts pivot in its “rightful position”

3	8	2	5	1	4	7	6

QuickSort: High-Level Description

[Hoare circa 1961]

- QuickSort (array A, length n)

```
-- If n=1     return
-- p = ChoosePivot(A,n)
-- Partition A around p
-- Recursively sort 1st part
-- Recursively sort 2nd part
```

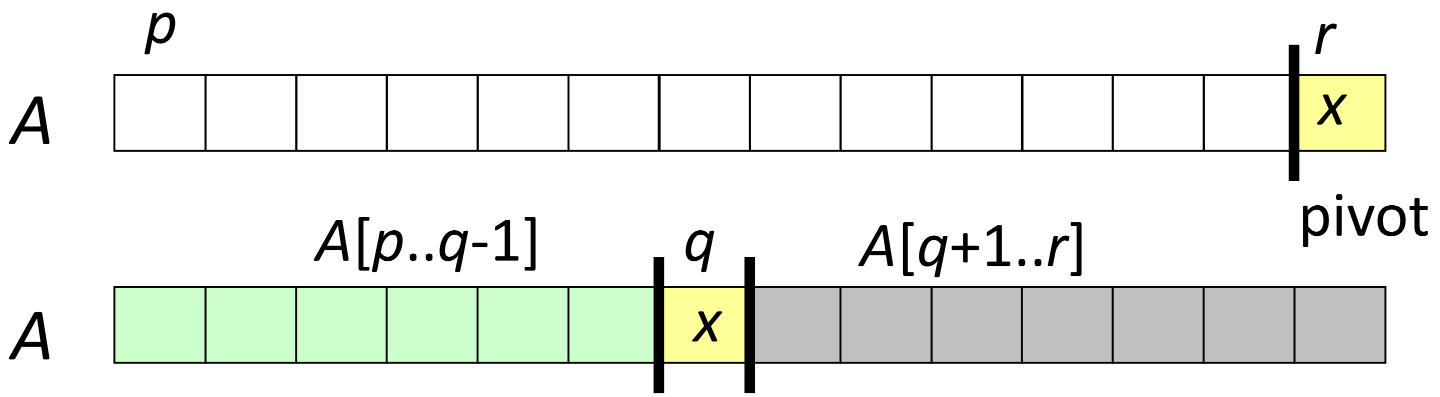
Two Cool Facts About Partition

- Linear O(n) time, no extra memory
- Reduces problem size

Quicksort

***Quicksort*(*A*[*p*..*r*])**

Divide



Conquer

Combine

```
Quicksort(A, p, r)
1  If $p < r$ then
2    $q \leftarrow$ Partition($A$, $p$, $r$)   /* divide */
3    Quicksort($A$, $p$, $q$-1)       /* conquer */
4    Quicksort($A$, $q$+1, $r$)       /* conquer */
```

Partition

- The Easy Way Out
- (using O(n) extra memory)

3	8	2	5	1	4	7	6

Partition High Level Idea

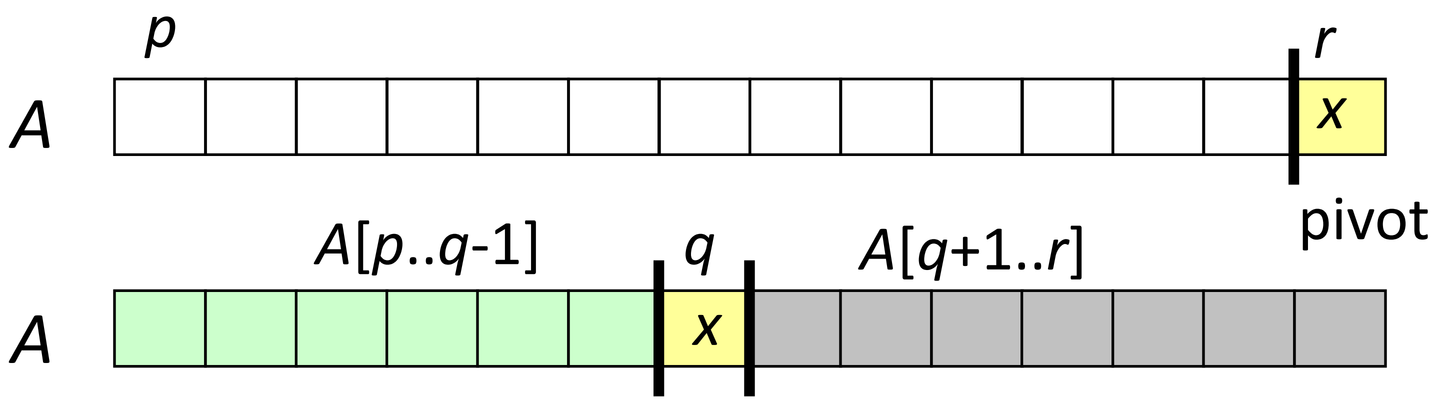
- In Place Algorithm

3	8	2	5	1	4	7	6

Quicksort

***Quicksort*(*A*[*p*..*r*])**

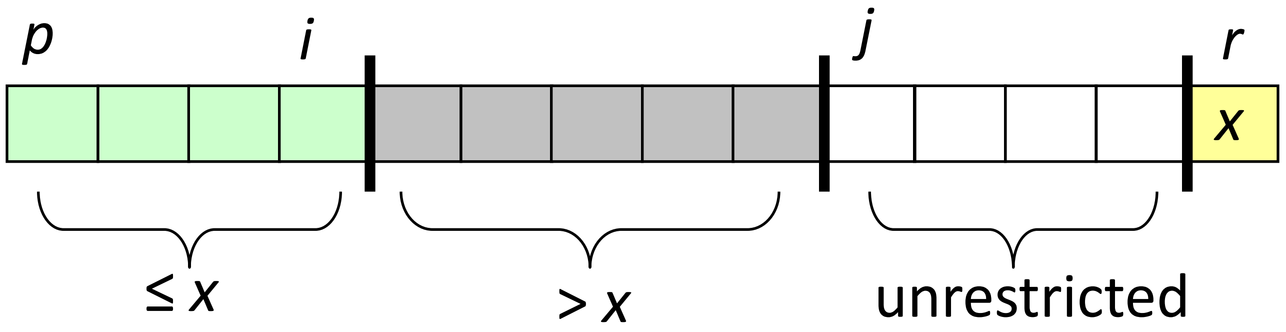
Divide

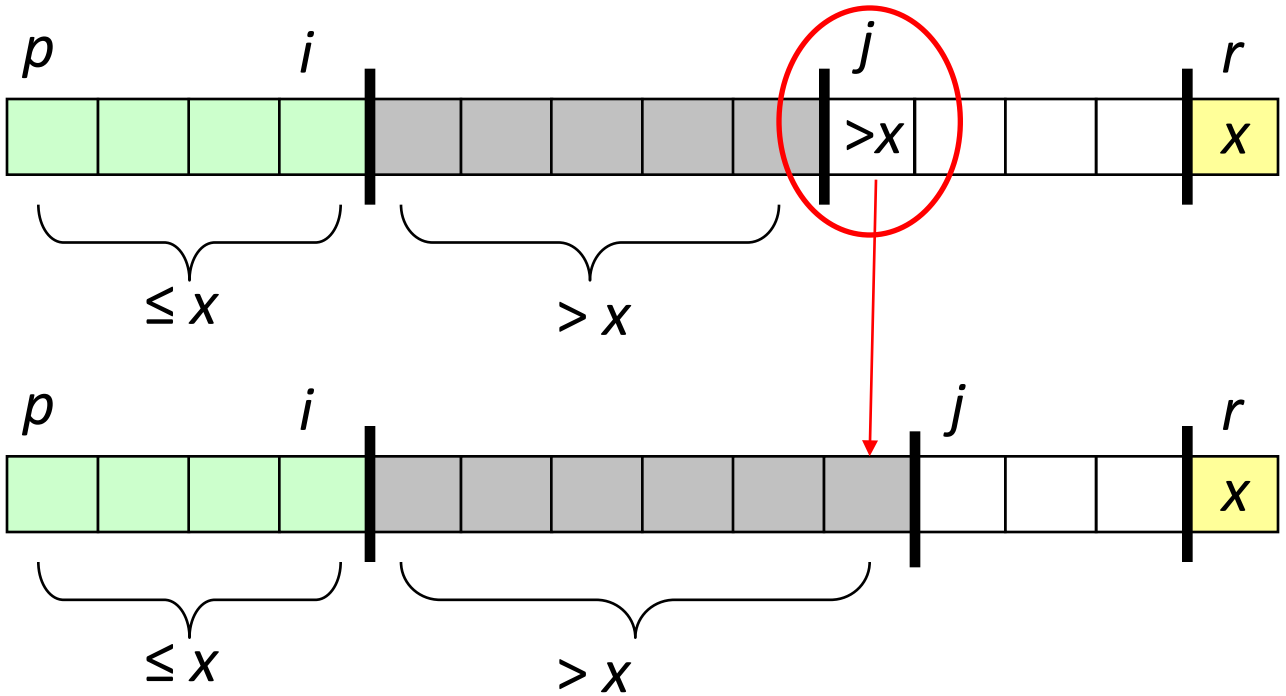


Conquer

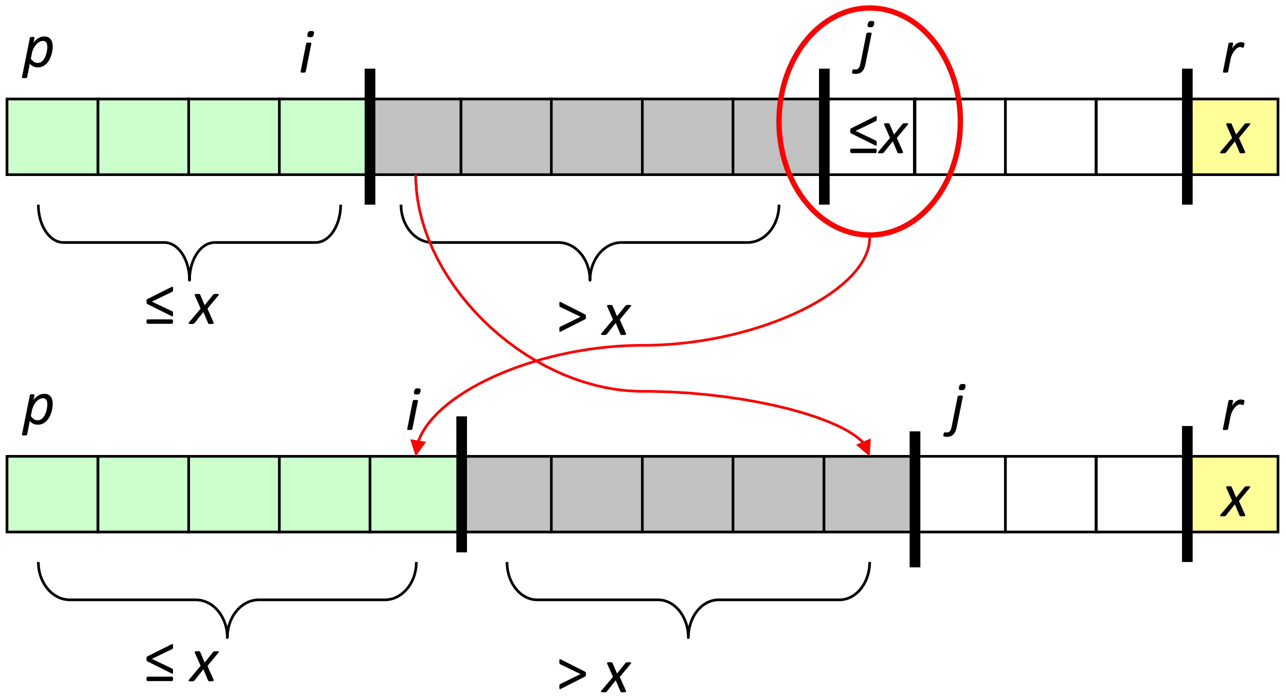
Combine

```
Quicksort(A, p, r)
1  If $p < r$ then
2    $q \leftarrow$ Partition(A, p, r)   /* divide */
3    Quicksort(A, p, q-1)         /* conquer */
4    Quicksort(A, q+1, r)         /* conquer */
```

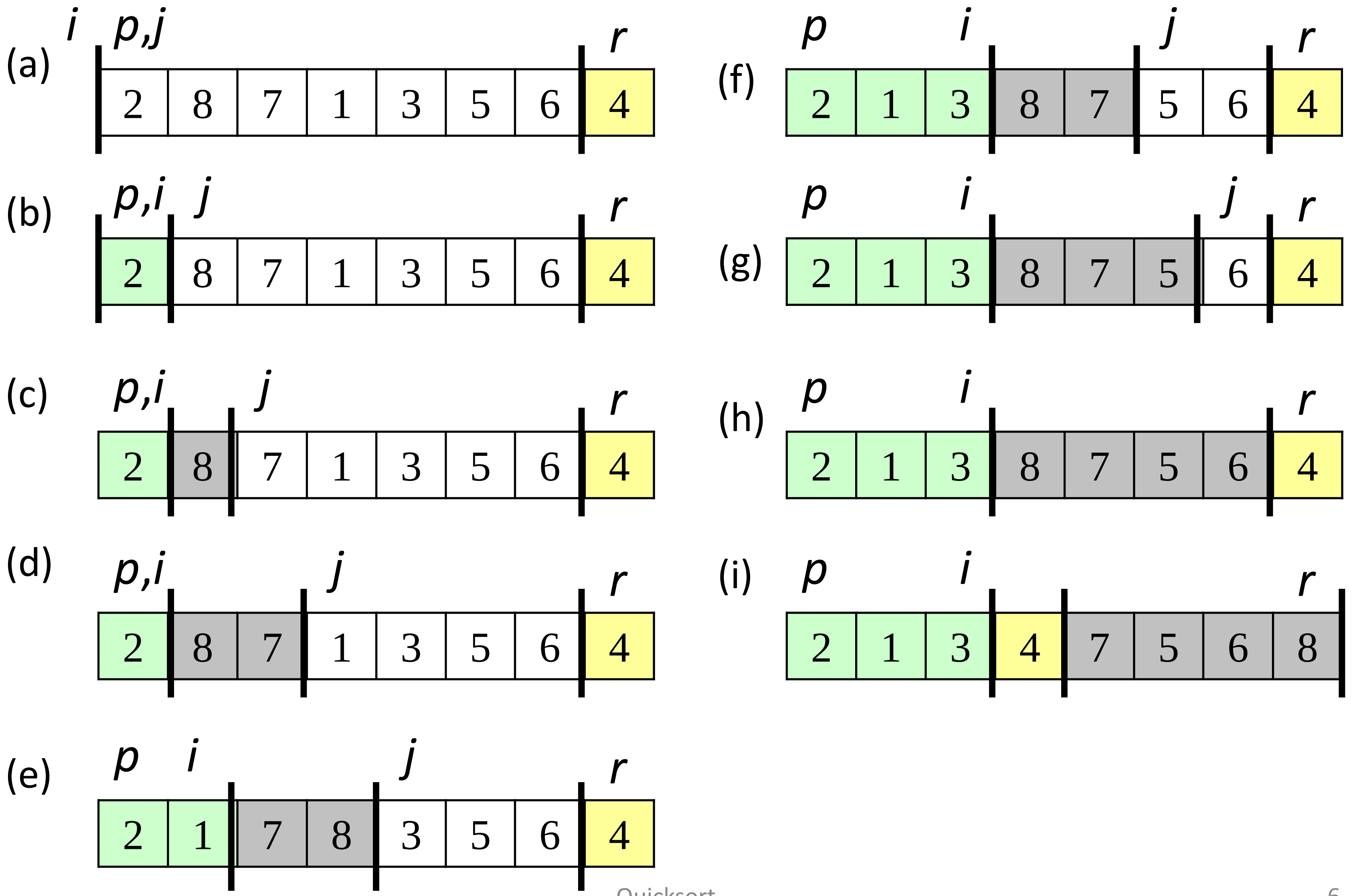

p
i
j
r
x
$\leq x$
$> x$
unrestricted



p
i
j
r
>x
x
≤ x
> x
p
i
j
r
x
≤ x
> x


p
i
j
r
≤x
x
≤ x
> x
p
i
j
r
x
≤ x
> x

(Partition, x=A[r]=4)



```
Partition(A, p, r)
1  $x \leftarrow A[r]$
2  $i \leftarrow p-1$
3  for $j \leftarrow p$ to $r-1$
4    do if $A[j] \leq x$
5    then $i \leftarrow i+1$
6         exchange $A[i] \leftrightarrow A[j]$
7  exchange $A[i+1] \leftrightarrow A[r]$
8  return $i+1$
```

Loop Invariant

```
Partition(A, p, r)
1   $x \leftarrow A[r]$
2   $i \leftarrow p-1$
3   for $j \leftarrow p$ to $r-1$
4     do if $A[j] \leq x$
5     then  $i \leftarrow i+1$
6           exchange $A[i] \leftrightarrow A[j]$
7   exchange $A[i+1] \leftrightarrow A[r]$
8   return $i+1$
```

Loop Invariant:

1. A[p] to A[i] are all less than or equal to pivot
2. A[i+1] to A[j-1] are all greater than pivot

Question 1

Suppose we implement Quicksort so that ChoosePivot always selects the first element of the array. What is the running time of this algorithm on an input array that is already sorted?

a) Not enough information to answer question
b) $\Theta(n)$
c) $\Theta(n \lg n)$
d) $\Theta(n^2)$

Question 1

Suppose we implement Quicksort so that ChoosePivot always selects the first element of the array. What is the running time of this algorithm on an input array that is already sorted?

a) Not enough information to answer question
b) $\Theta(n)$
c) $\Theta(n \lg n)$
d) $\Theta(n^2)$

Answer: d) $\Theta(n^2)$

Question 2

Suppose we run Quicksort on some input, and, magically, every recursive call chooses the median its subarray as pivot. What’s the running time in this case?

a) Not enough information to answer question
b) $\Theta(n)$
c) $\Theta(n \lg n)$
d) $\Theta(n^2)$

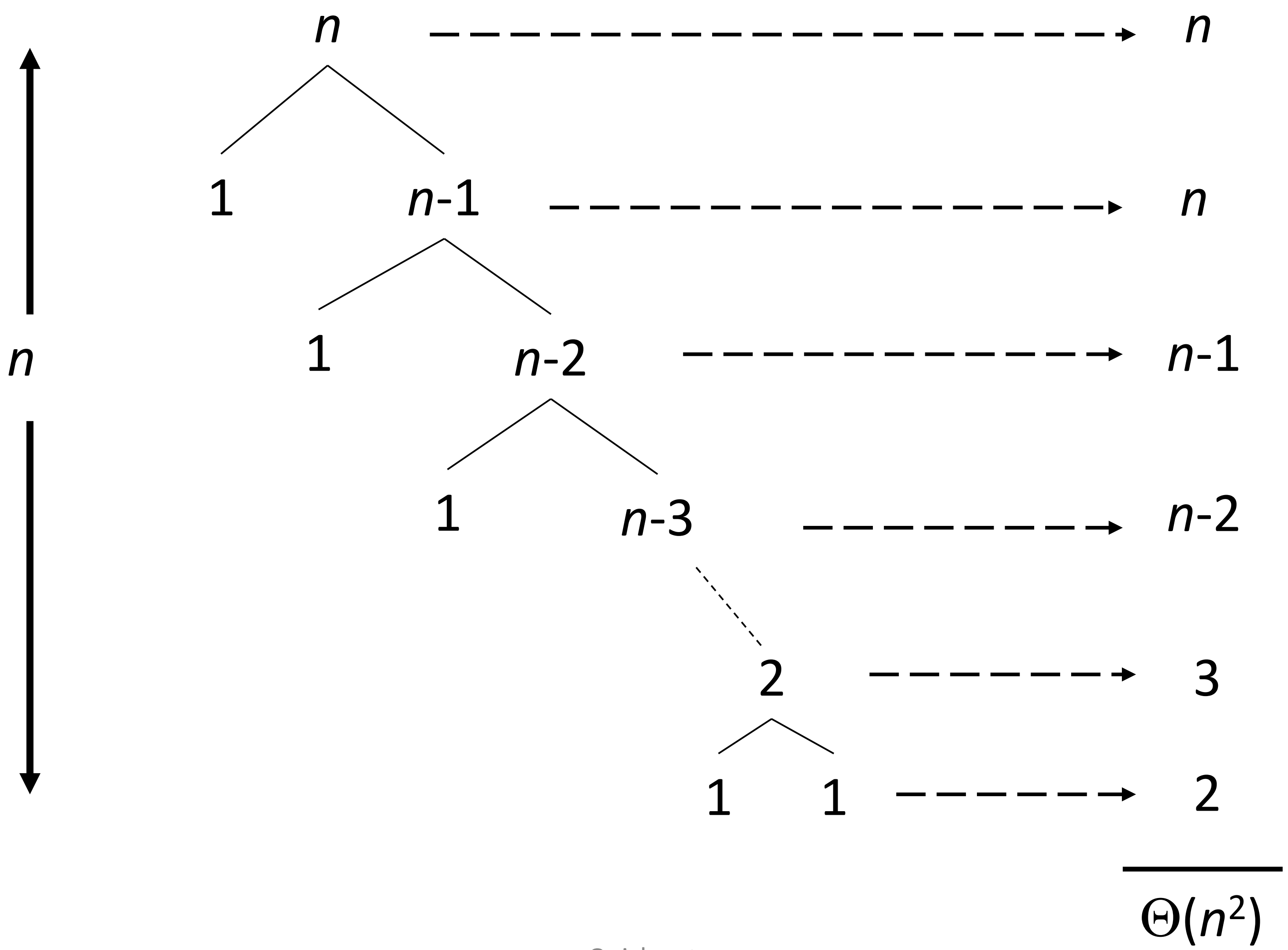
Question 2

Suppose we run Quicksort on some input, and, magically, every recursive call chooses the median its subarray as pivot. What’s the running time in this case?

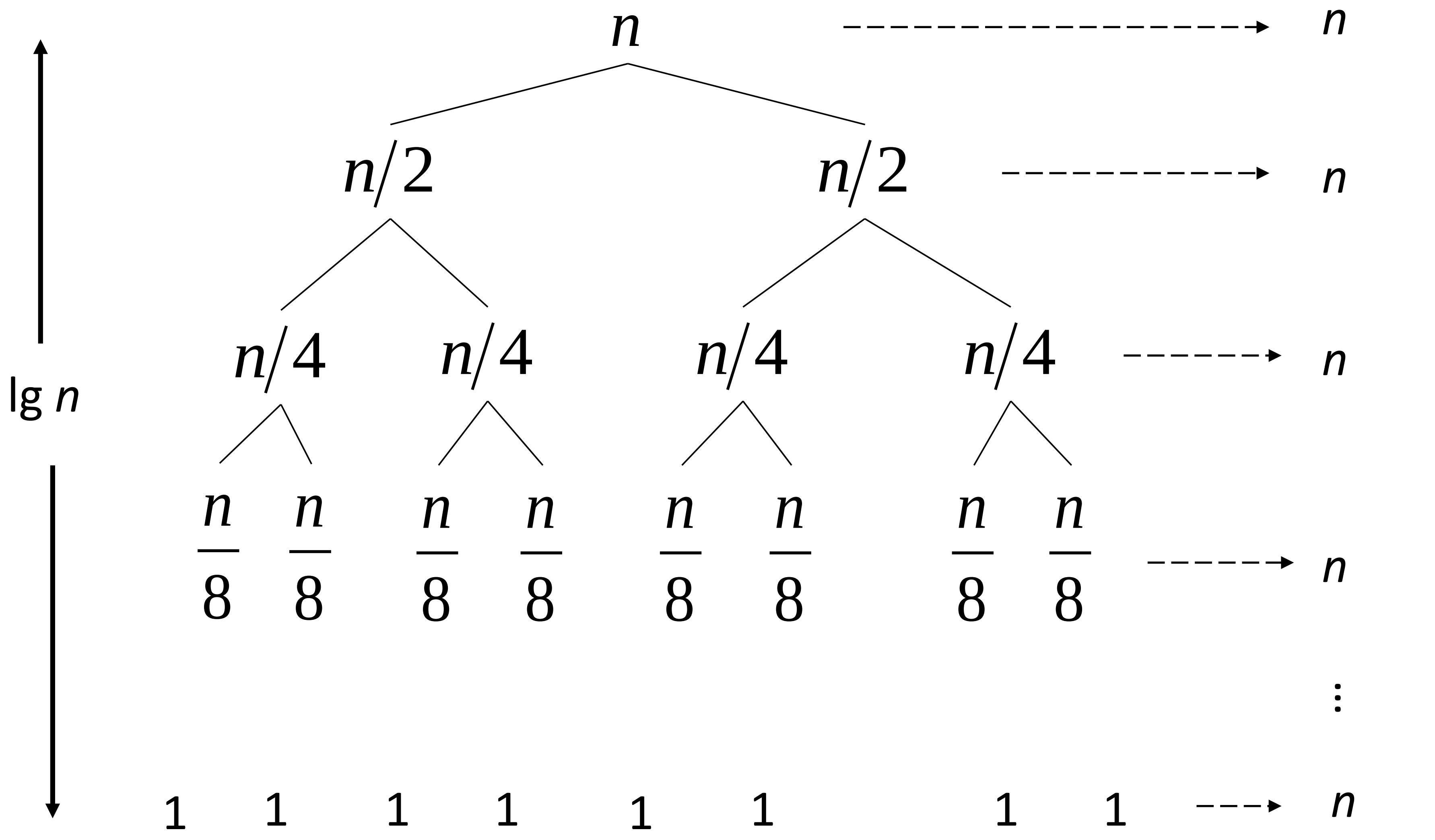
a) Not enough information to answer question
b) $\Theta(n)$
c) $\Theta(n \lg n)$
d) $\Theta(n^2)$

Answer: c) $\Theta(n \lg n)$

Question 1: Worst Case Analysis



Question 2: Best Case Analysis



Total:$\Theta(n \lg n)$

Key Question

How to Choose Pivot?

Quicksort running time depends on it.

It can be anywhere from $\Theta(n \lg n)$

to $\Theta(n^2)$ depending upon selection of pivot

BIG IDEA

Randomization

Allow algorithms to flip coins in code to get some benefit

Choose Pivot randomly

Randomized version of quicksort

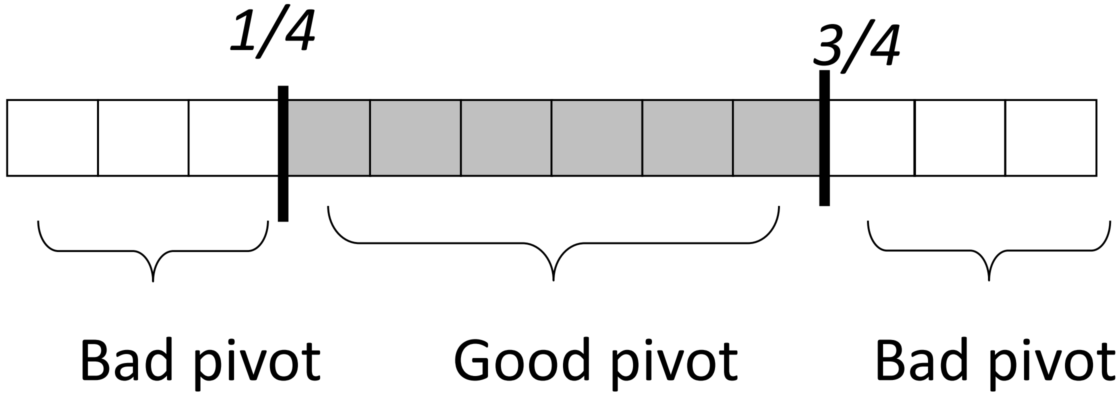
```
RANDOMIZED-PARTITION$(A, p, r)$
1  $i$ = RANDOM$(p, r)$
2  exchange $A[r]$ with $A[i]$
3  return PARTITION$(A, p, r)$
```

The new quicksort calls RANDOMIZED-PARTITION in place of PARTITION:

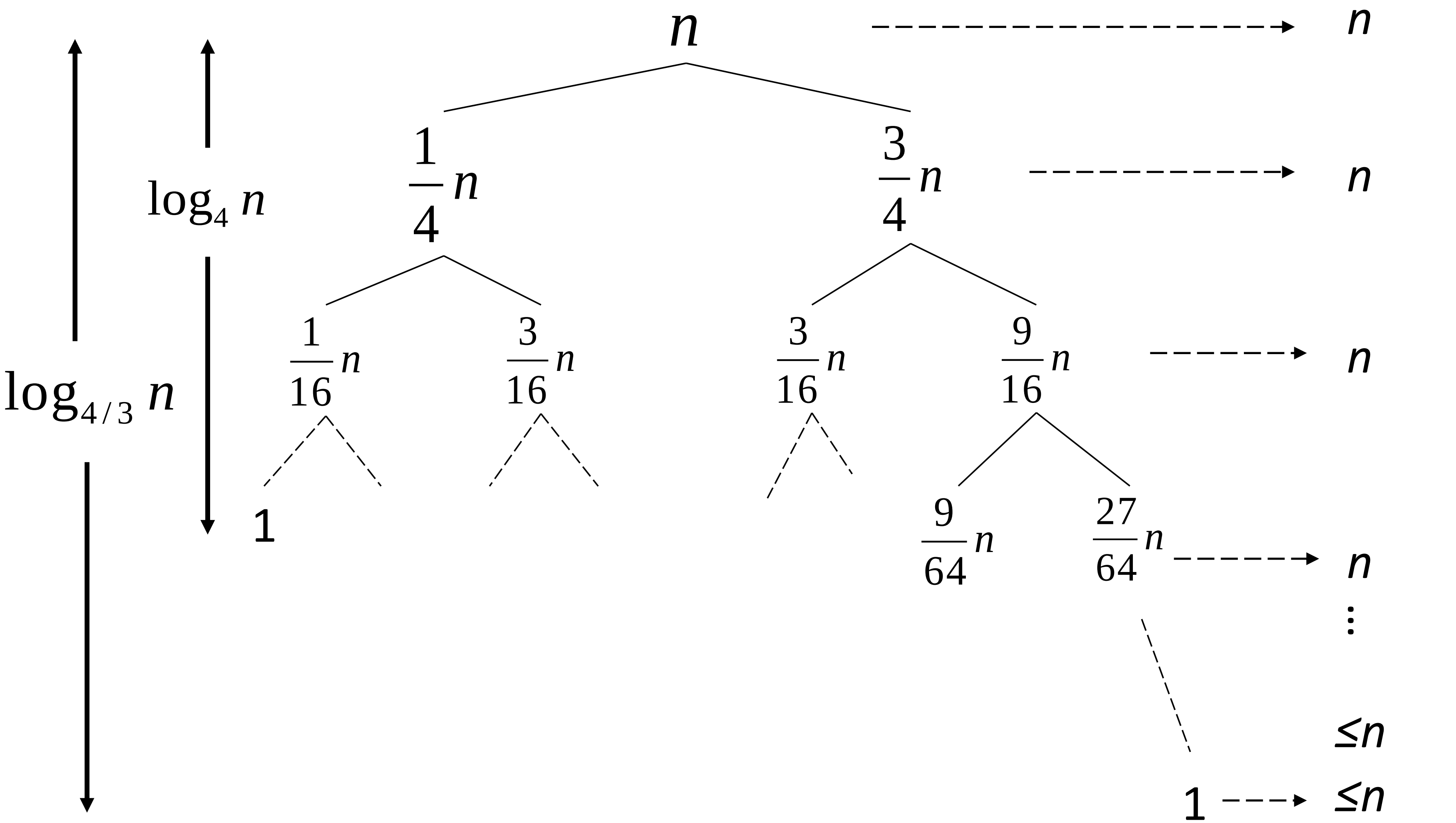
```
RANDOMIZED-QUICKSORT$(A, p, r)$
1  if $p < r$
2      $q$ = RANDOMIZED-PARTITION$(A, p, r)$
3      RANDOMIZED-QUICKSORT$(A, p, q - 1)$
4      RANDOMIZED-QUICKSORT$(A, q + 1, r)$
```

Random Pivot Selection

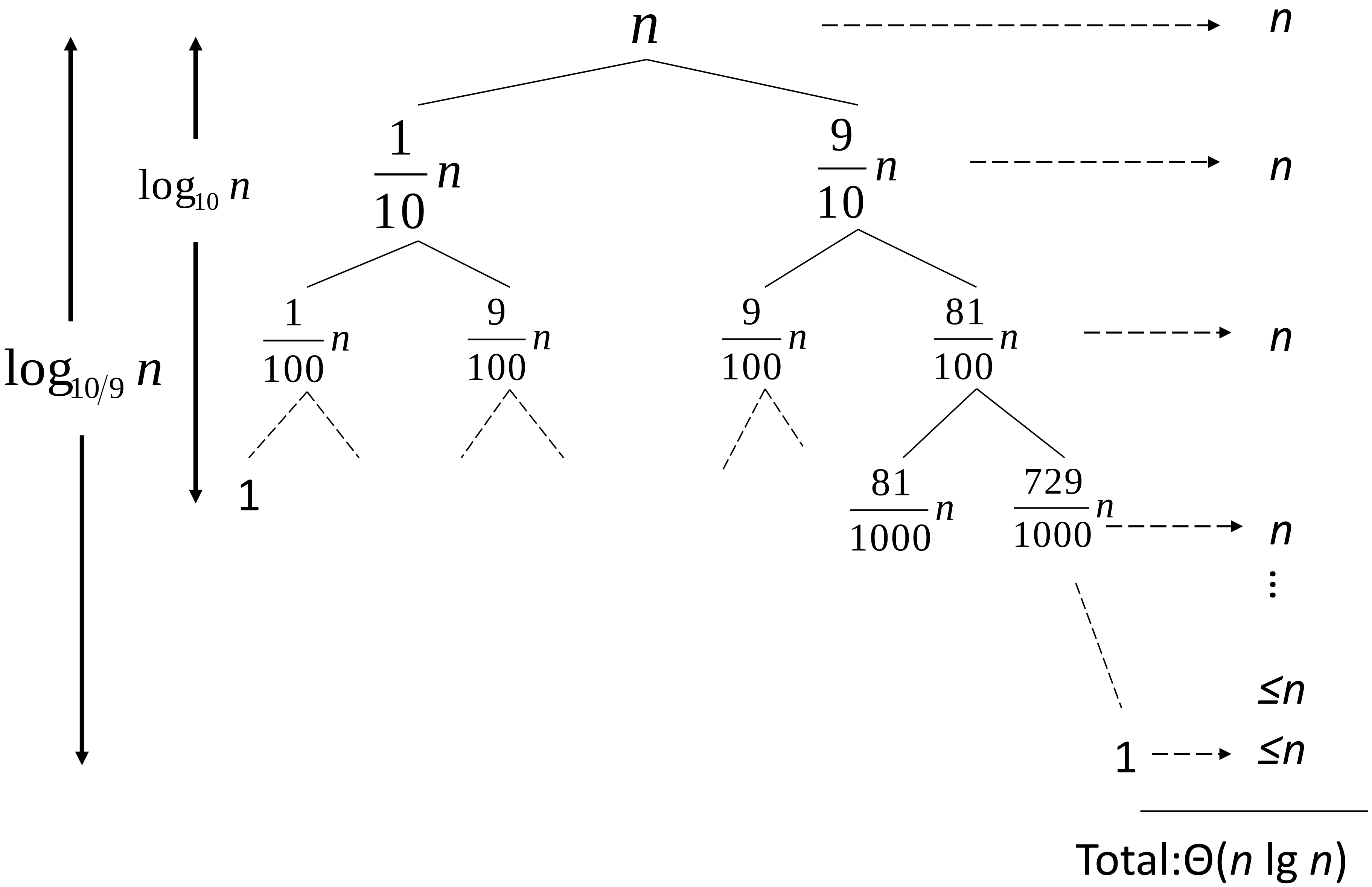
- **Good split**: The split is GOOD if the pivot is an element of rank n/4 to 3n/4.



- **Bad Split:** The split is BAD if the pivot is an element of rank less than n/4 or greater than 3n/4.



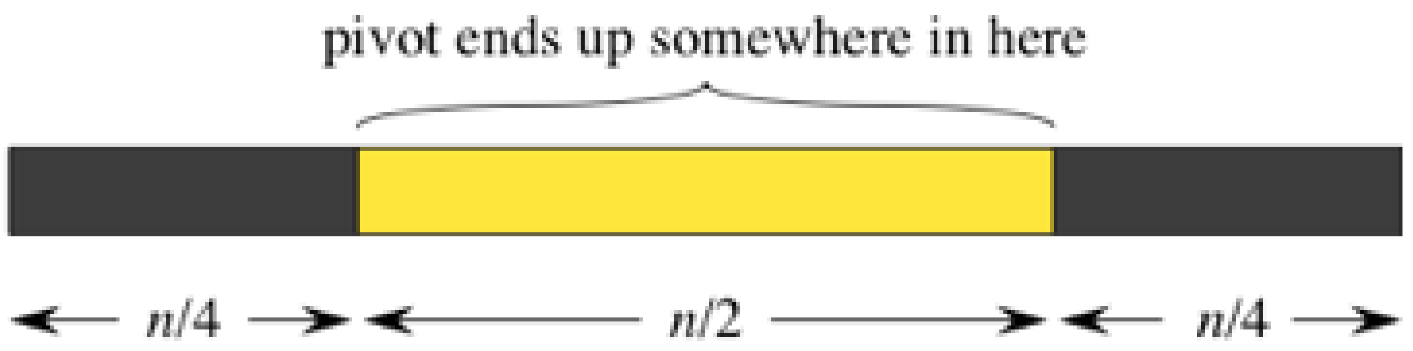
Total:$\Theta(n \lg n)$



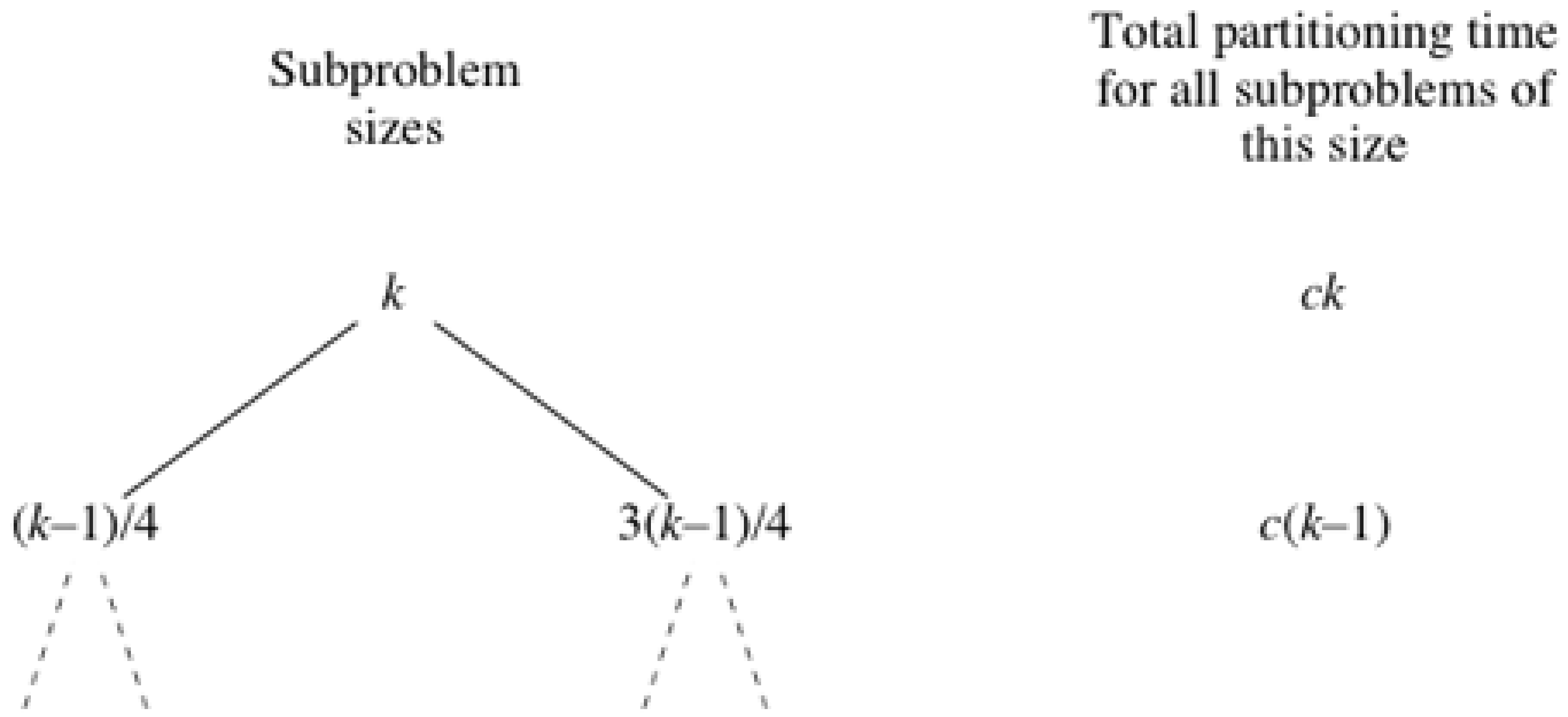
n
$\frac{1}{10}n$
$\frac{9}{10}n$
$\frac{1}{100}n$
$\frac{9}{100}n$
$\frac{9}{100}n$
$\frac{81}{100}n$
$\frac{81}{1000}n$
$\frac{729}{1000}n$
1
1
$\log_{10} n$
$\log_{10/9} n$
n
n
n
n
⋮
$\leq n$
$\leq n$
Total: $\Theta(n \lg n)$

Probability of Good Split

- If pivot is chosen randomly then probability of good split is 50%

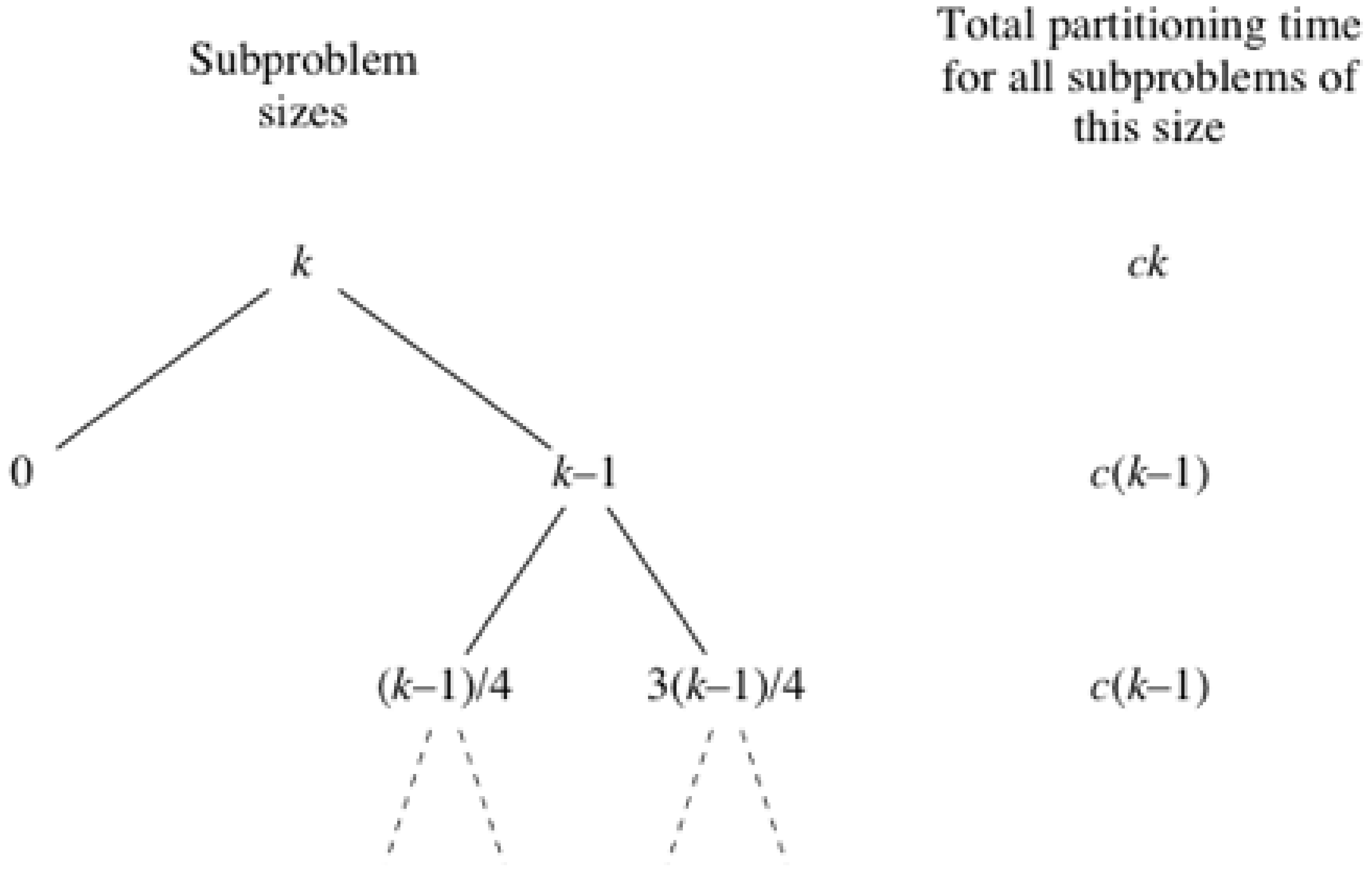


Good Split



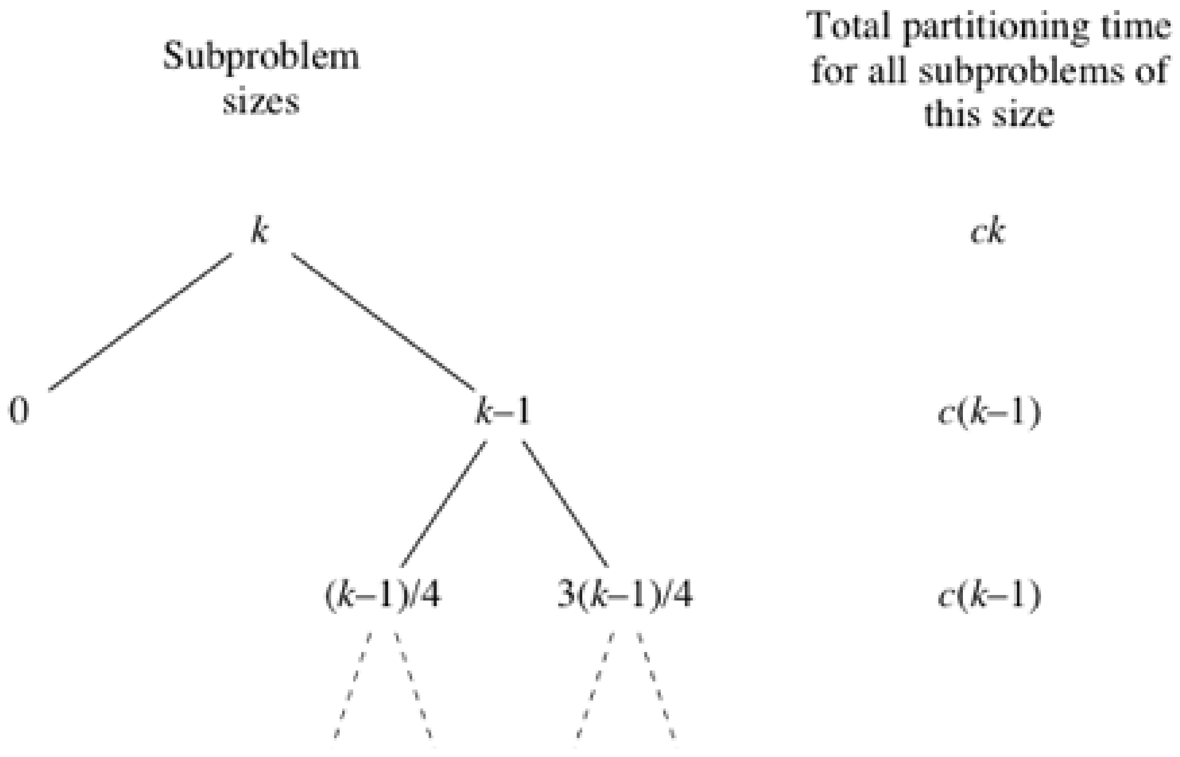
Subproblem sizes
Total partitioning time for all subproblems of this size
k
ck
$(k-1)/4$
$3(k-1)/4$
$c(k-1)$

Alternate between Good and Bad Split



Alternate between Good and Bad Split

- The tree height will be double the size of tree for good split so running time will be n $2\log_{4/3} n = O(n \lg n)$



Improve RANDOMIZED-QUICKSORT

- Choose the pivot as the median of a set of 3 elements randomly selected from the array.

Probability of Bad Split for Median of 3

- The split is bad if and only if
  1. The median of three randomly chosen elements has rank $< n/4$
  2. it has rank $> 3n / 4$

Probability of Bad Split Case 1

- Case(i) The median of three randomly chosen elements has rank < n/4
- It happens if and only if
  a) exactly 2 out of 3 random elements have rank < n / 4
  b) all 3 randomly chosen elements have a rank < n/4.

Probability of Bad Split Case 1 (a)

- Each random element will have rank < n / 4 with probability 1 / 4
- The probability that two fixed elements will have this rank and remaining element will have rank >= n / 4 is (1/4)* (1/4)*(3/4) = 3/64
- There are 3 ways to choose 2 elements out of 3 and 3 choose 2 = 3
- Case1 (a) happens with probability 3 * 3/64 = 9/64.

Probability of Bad Split Case 1 (b)

- all 3 randomly chosen elements have a rank $< n/4$
- $(1/4)* (1/4)* (1/4) = 1/64$

Probability of Bad Split Case 1

- Case 1 (a) happens with probability = 9/64
- Case 1 (b) happens with probability = 1/64
- Case 1 happens with probability = 9/64 +1/64

  = 10/64 = 5/32

Probability of Bad Split
Case 2

- Case 2: it has rank > 3n / 4
- Its probability is same as case 1 = 5/32

Probability of Bad Split for Median of 3

i. Probability that the median of three randomly chosen elements has rank < n/4 = 5/32

ii. Probability that it has rank > 3n / 4 = 5/32

Total probability = 5/32 + 5/32 = 10/32 = **5/16**

- Probability of Good split = 1 – 5/16 = 11/16

Probability of Good Split

Pivot	Probability	Percentage
One Random Pivot	1/2	50%
Median of 3	11/16	68.75%
Median of 5	203/256	79.3%
Median of 7	1759/2048	85.9%
Median of 9	59123/65536	90.2%
Median of 11	488293/524288	93.1%

Slide Credits

- http://people.cs.nctu.edu.tw/~sctsai/algo/notes/
- https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort