Design and Analysis of Algorithms

Lecture-12

Dharmendra Kumar (Associate Professor)
Department of Computer Science and Engineering
United College of Engineering and Research,
Prayagraj

Quicksort

- Quicksort, like merge sort, applies the divide-and-conquer paradigm.

- Quicksort uses three steps to sort the elements.

- **Divide**
- **Conquer**
- **Combine**

Quicksort

Algorithm

It divides the large array into smaller sub-arrays. And then quicksort recursively sort the sub-arrays.

Pivot

1. Picks an element called the "pivot".

Partition

2. Rearrange the array elements in such a way that the all values lesser than the pivot should come before the pivot and all the values greater than the pivot should come after it.

This method is called partitioning the array. At the end of the partition function, the pivot element will be placed at its sorted position.

Recursive

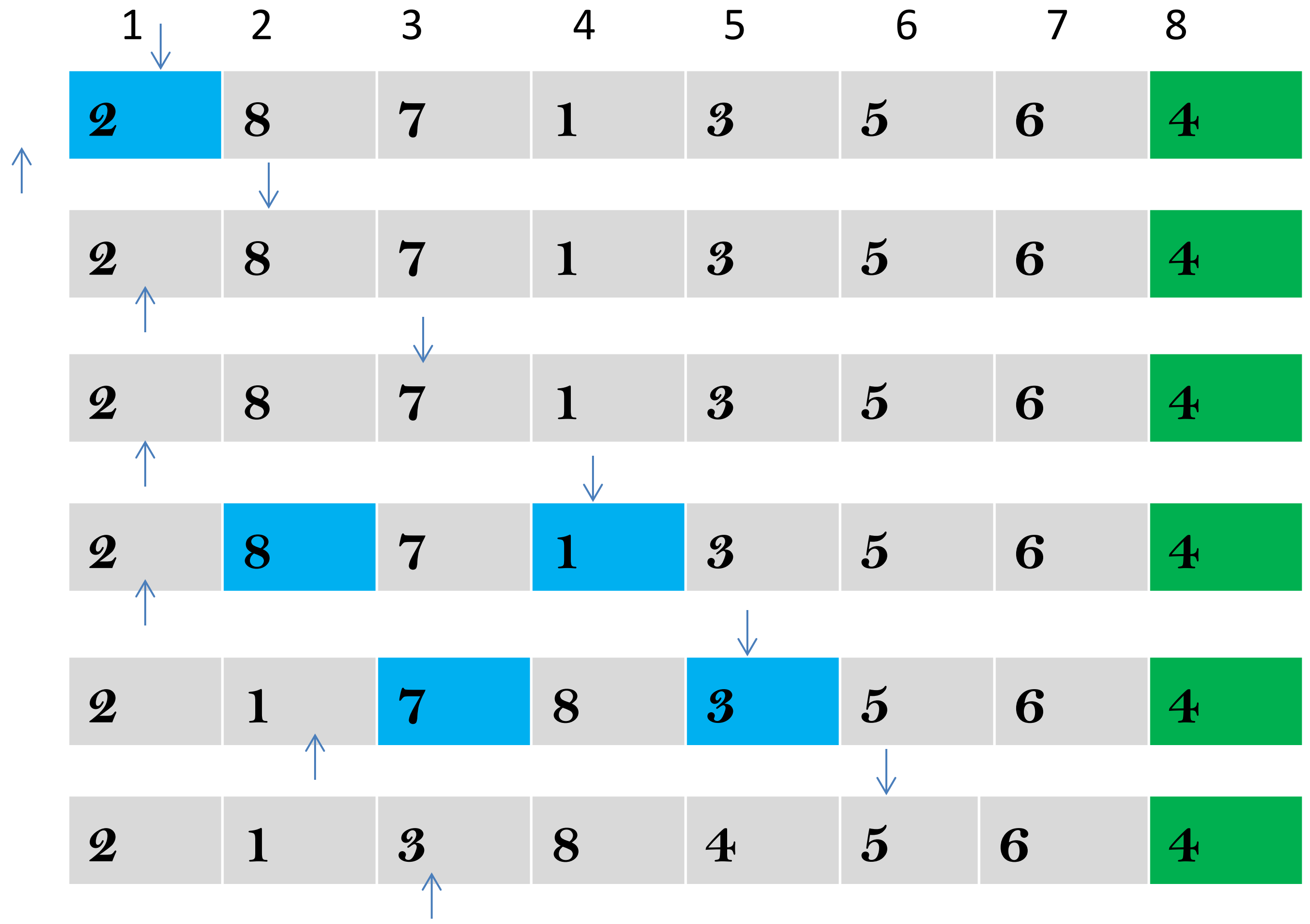
3. Do the above process recursively to all the sub-arrays and sort the elements.

Quicksort

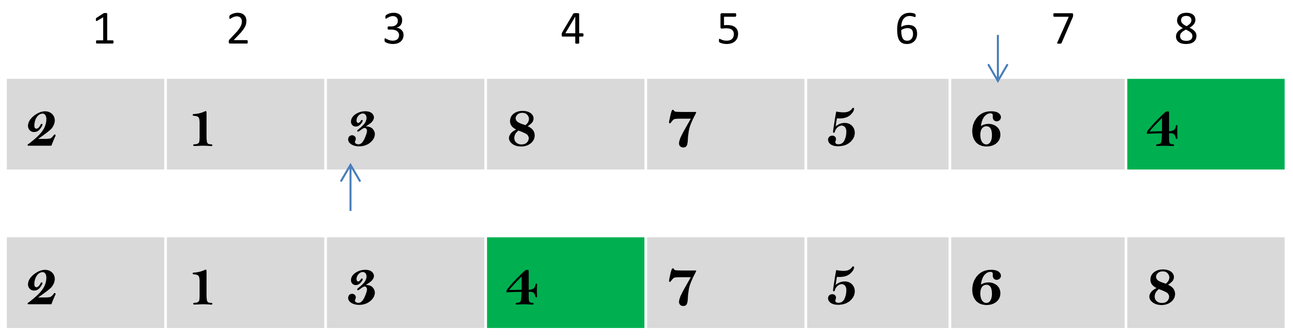
Example: Sort the following elements using quicksort

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

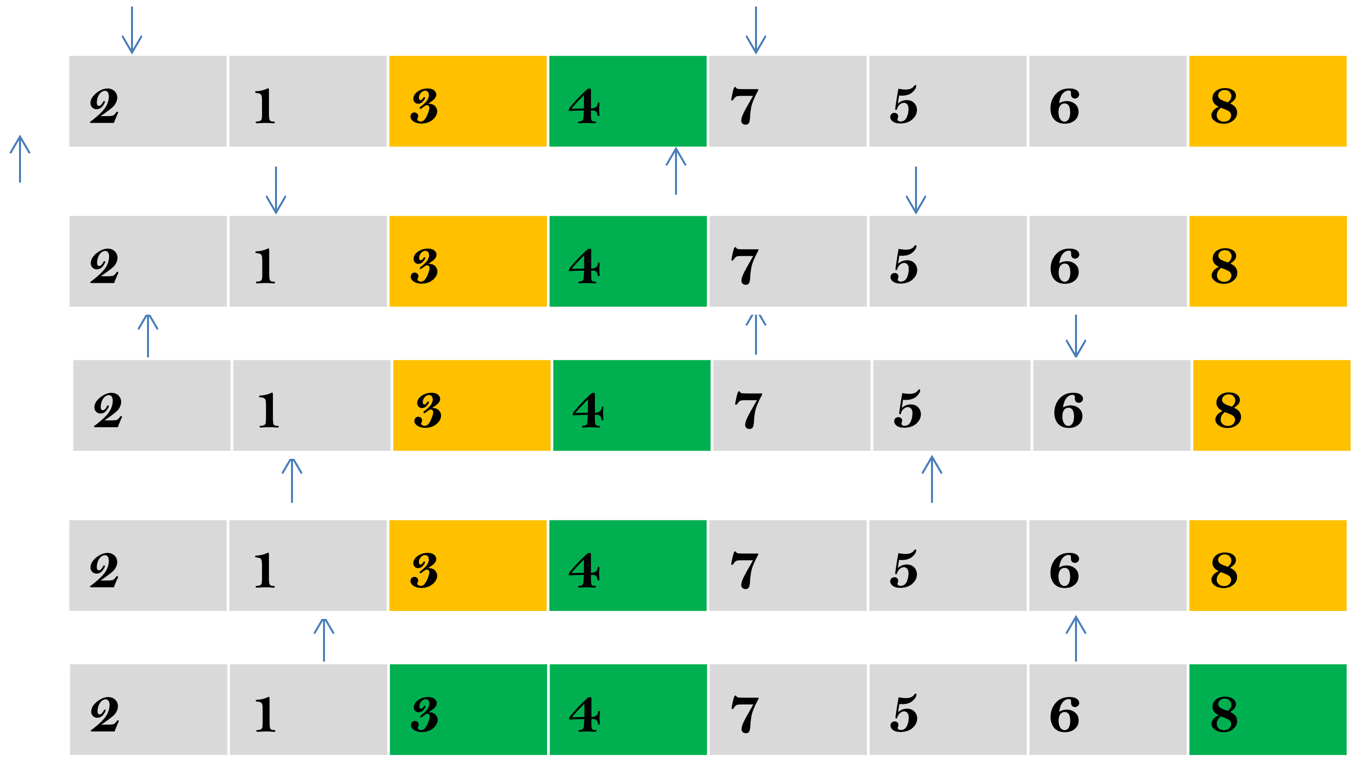
Solution: Here, we pick the last element in the list as a pivot



Quicksort

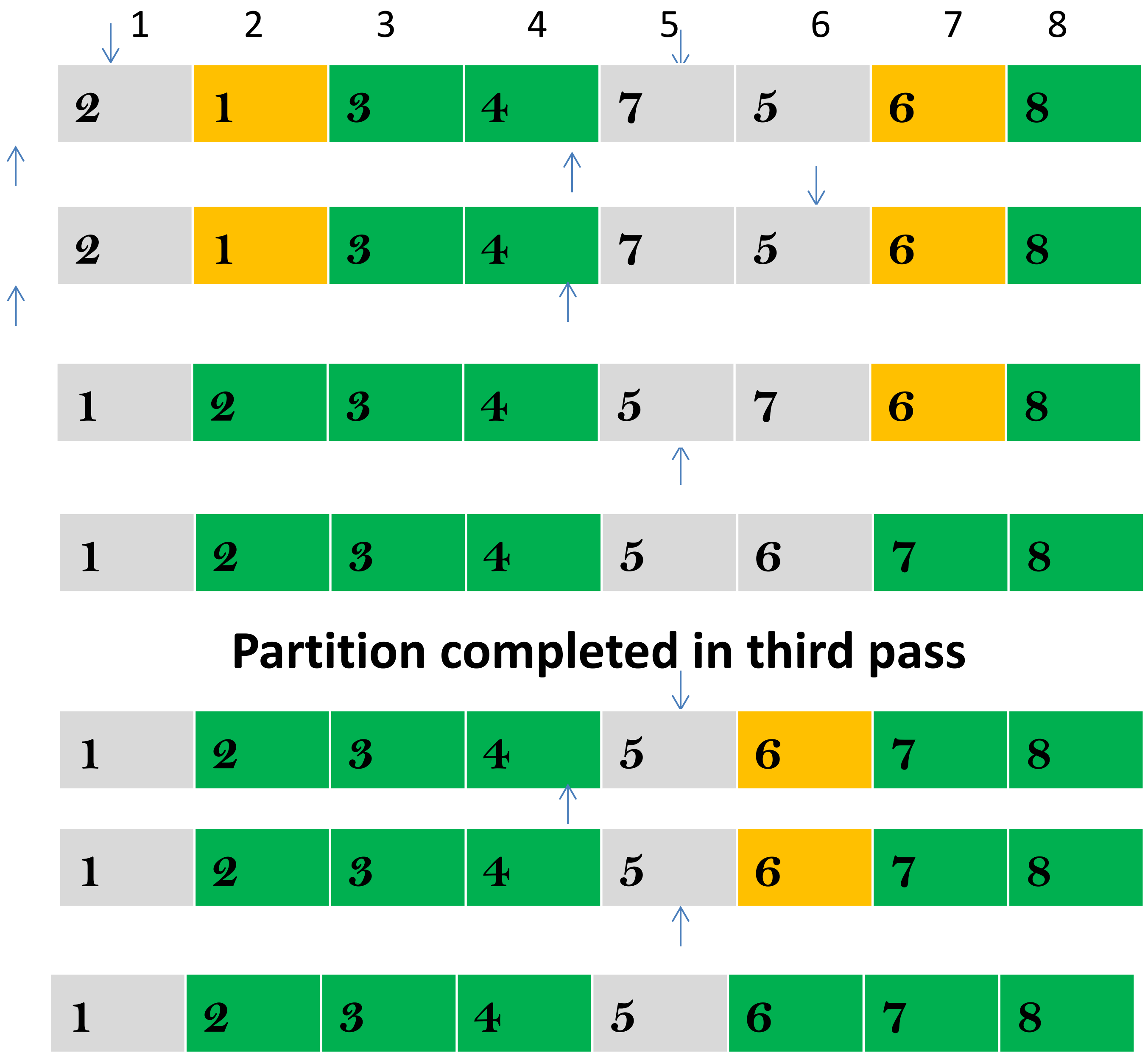


Partition completed in first pass



Partition completed in second pass

Quicksort



Partition completed in third pass

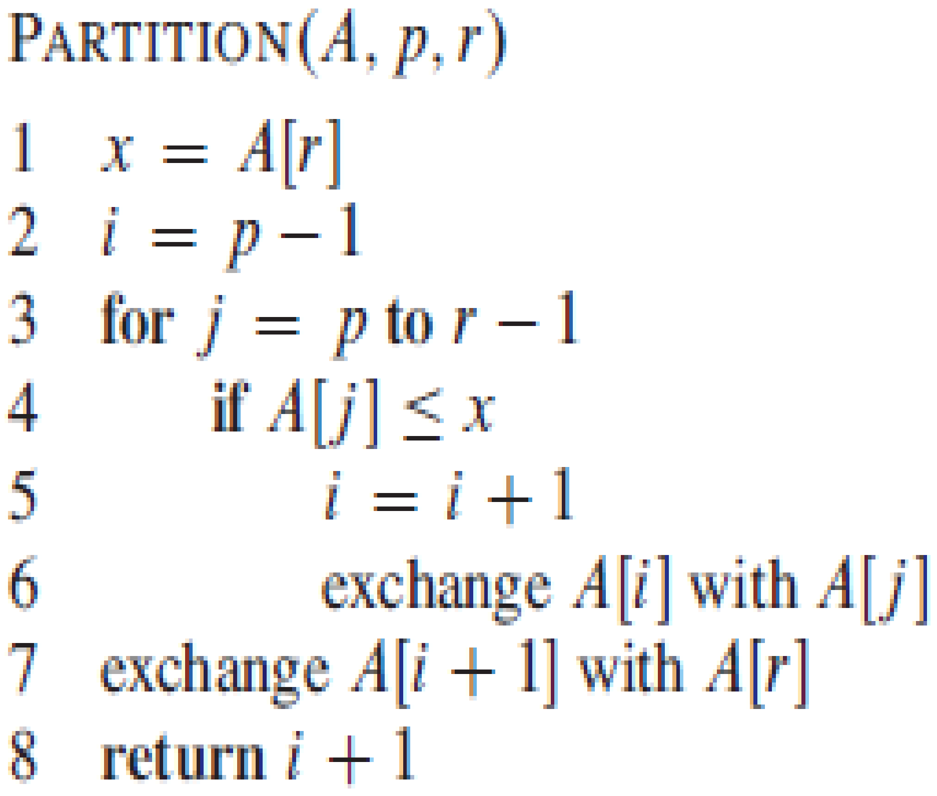
Process completed

Quicksort Algorithm

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

To sort an entire array A, the initial call is QUICKSORT(A, 1, *length[A])*.

Quicksort Algorithm



```
PARTITION($A, p, r$)
$x = A[r]$
$i = p - 1$
for $j = p$ to $r - 1$
    if $A[j] \le x$
        $i = i + 1$
        exchange $A[i]$ with $A[j]$
exchange $A[i + 1]$ with $A[r]$
return $i + 1$
```

Quicksort Algorithm Analysis

Time Complexity:

The running time of quicksort algorithm is

$$T(n) = T(n_1) + T(n_2) + \theta(n) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

Here n_1 and n_2 are the size of sub-problems.

- The running time of quicksort depends on whether the partitioning is balanced or unbalanced, which in turn depends on which elements are used for partitioning.
- If the partitioning is balanced, the algorithm runs asymptotically as fast as merge sort.
- If the partitioning is unbalanced, however, it can run asymptotically as slowly as insertion sort.

Quicksort Algorithm Analysis

Worst-case partitioning

➢The worst-case behavior for quicksort occurs when the partitioning routine produces one sub-problem with n-1 elements and one with 0 elements.

➢Let us assume that this unbalanced partitioning arises in each recursive call. Therefore,

$$T(n) = T(0) + T(n-1) + \theta(n)$$
$$T(n) = T(n-1) + \theta(n) \quad (\text{since } T(0) = \theta(1))$$

Quicksort Algorithm Analysis

Worst-case partitioning

After solving this recurrence relation, we get

$$T(n) = \theta(n^2)$$

Therefore the worst-case running time of quicksort is no better than that of insertion sort.

Moreover, the $\theta(n^2)$ running time occurs when the input array is already completely sorted—a common situation in which insertion sort runs in $O(n)$ time.

Quicksort Algorithm Analysis

Best-case partitioning

If PARTITION produces two sub-problems, each of size no more than n/2, since one is of size $\lfloor n/2 \rfloor$ and one of size $\lceil n/2 \rceil - 1$. In this
case, quicksort runs much faster. The recurrence for the running time is then

$$T(n) = 2T(n/2) + \theta(n)$$

After solving this recurrence relation, we get

$$\mathbf{T(n) = \theta(n\ lgn)}$$

Note: By equally balancing the two sides of the partition at every level of the recursion, we get an asymptotically faster algorithm.

Quicksort Algorithm Analysis

Balanced partitioning

The average-case running time of quicksort is much closer to the best case than to the worst case.

Suppose that the partitioning algorithm always produces a 9-to-1 proportional split. In this case, running time will be

$$T(n) = T(9n/10) + T(n/10) + \theta(n)$$

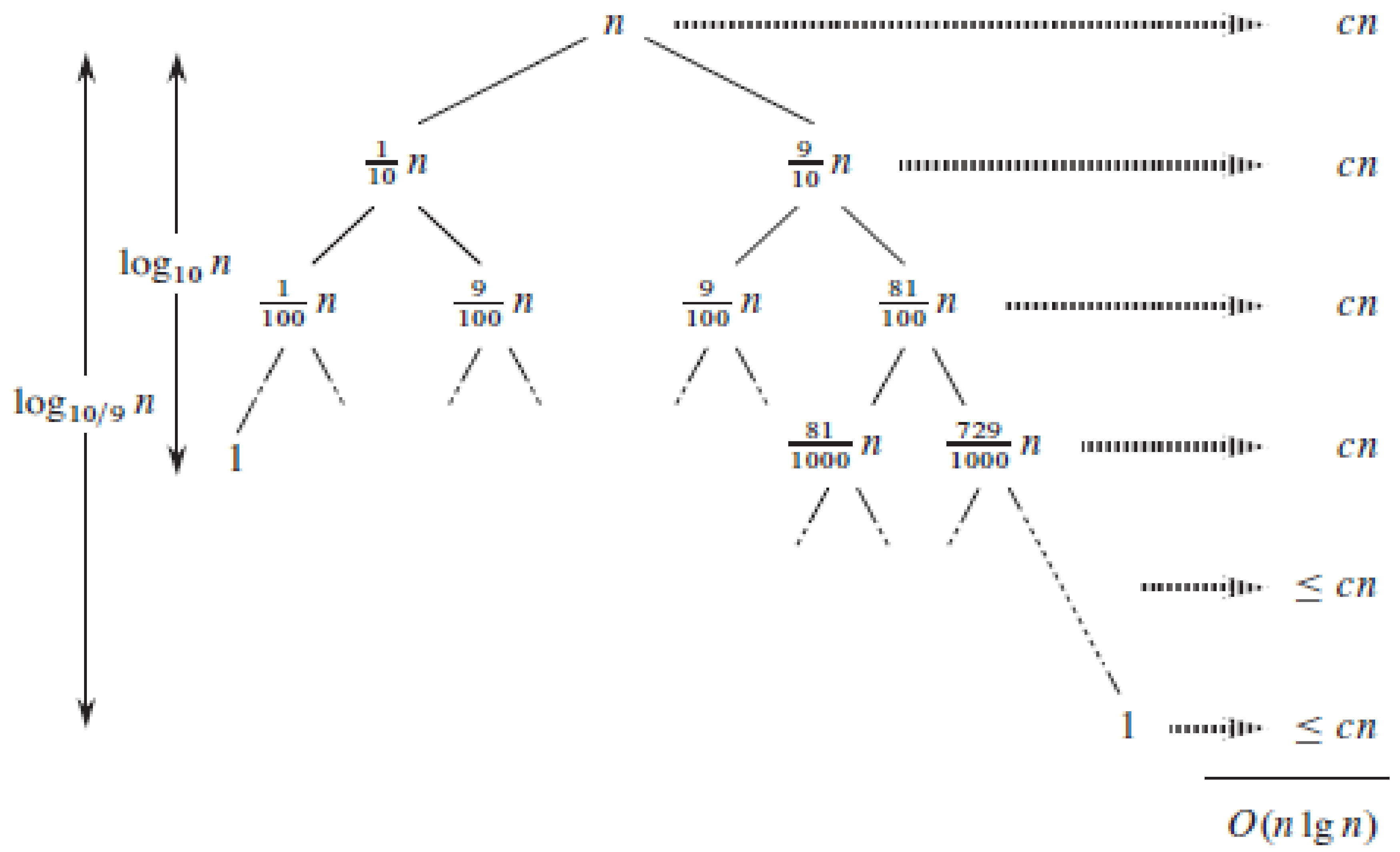
This relation is equivalent to the following

$$T(n) = T(9n/10) + T(n/10) + cn$$

We can solve this recurrence using recurrence tree method.

Quicksort Algorithm Analysis

Recurrence tree will be



Quicksort Algorithm Analysis

Therefore, the solution of recurrence relation will be

$$T(n) \le cn + cn + cn+\ldots\ldots\ldots\ldots\ldots\ldots..+cn$$
$$= cn(1+\lg_{10/9}n)$$
$$= cn + cn\lg_{10/9}n$$

Therefore, **$T(n) = O(n\lg n)$**

Quicksort Algorithm Analysis

Exercise

1. Sort the following elements using quicksort

   13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11.

2. What is the running time of QUICKSORT when all elements of array A have the same value?

3. Show that the running time of QUICKSORT is, $\theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.

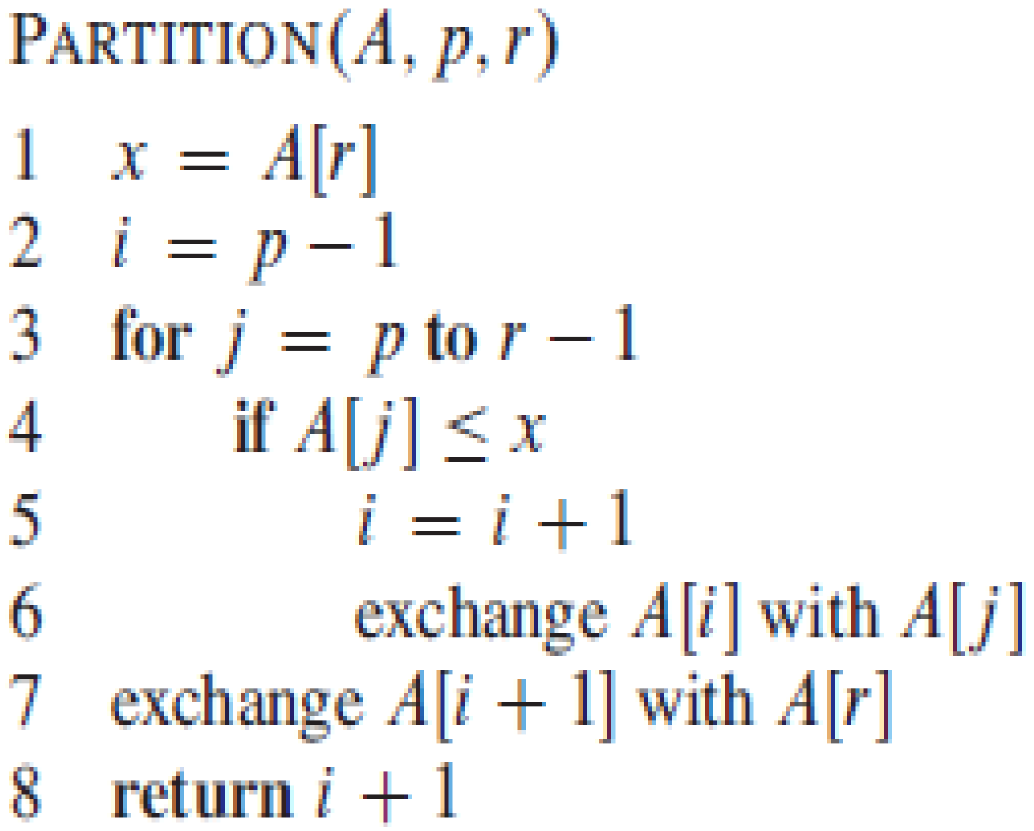
4. Suppose that the splits at every level of quicksort are in the proportion $1-\alpha$ to α, where $0 < \alpha \leq 1/2$ is a constant. Show that the minimum depth of a leaf in the recursion tree is approximately $\lg n/ \lg \alpha$ and the maximum depth is approximately $\lg n/ \lg(1-\alpha)$. (Don't worry about integer round-off.)

A randomized version of quicksort

```
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($A, q + 1, r$)
```

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

A randomized version of quicksort



```
PARTITION($A, p, r$)
$x = A[r]$
$i = p - 1$
for $j = p$ to $r - 1$
    if $A[j] \le x$
        $i = i + 1$
        exchange $A[i]$ with $A[j]$
exchange $A[i + 1]$ with $A[r]$
return $i + 1$
```

The running time of randomized quick sort will be

$T(n) = O(n\lg n)$