QUICK SORT

Quicksort is the other important sorting algorithm that is based on the divide-and-conquer approach. quicksort divides input elements according to their value. A partition is an arrangement of the array's elements so that all the elements to the left of some element $A[s]$ are less than or equal to $A[s]$, and all the elements to the right of $A[s]$ are greater than or equal to it:

$A[0] \ldots A[s-1]$ $\quad\quad$ $A[s]$ $\quad\quad$ $A[s+1] \ldots A[n-1]$

allareSA[s] $\quad\quad$ all areSA[s]

Sort the two subarrays to the left and to the right of $A[s]$ independently. No work required to combine the solutions to the sub problems.

Here is pseudocode of quicksort: call *Quicksort(A[0..n − 1])* where As a partition algorithm use the *Hoare Partition.*

ALGORITHM *Quicksort(A[l..r])*

//Sorts a subarray by quicksort

//Input: Subarray of array $A[0..n-1]$, defined by its left and right indices l and r

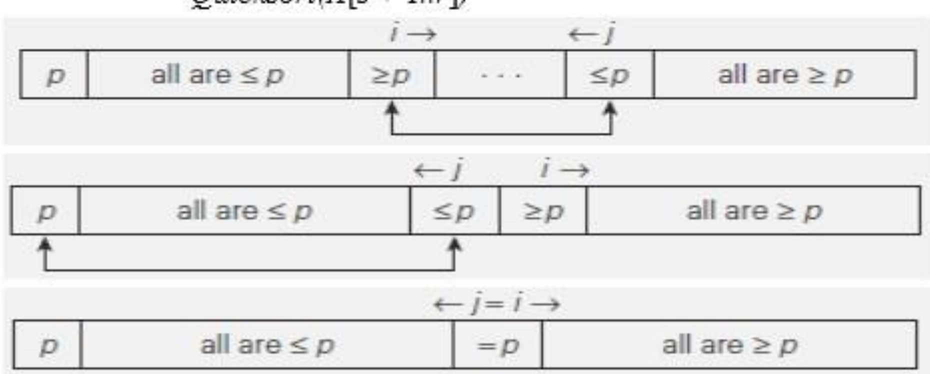
//Output: Subarray $A[l..r]$ sorted in non-decreasing order

if $l < r$

$s \leftarrow$ *Hoare Partition(A[l..r])* //s is a split position

Quicksort(A[l..s − 1])

Quicksort(A[s + 1..r])



ALGORITHM *Hoare Partition(A[l..r])*

```
//Partitions a subarray by Hoare's algorithm, using the first element as a pivot
//Input: Subarray of array A[0..n − 1], defined by its left and right indices l and r (l<r)
//Output: Partition of A[l..r], with the split position returned as this function's value
p←A[l]
i ←l; j ←r + 1
repeat
        repeat i ←i + 1 until A[i]≥ p
        repeat j ←j − 1 until A[j ]≤ p
        swap(A[i], A[j ])
until i ≥ j
swap(A[i], A[j ]) //undo last swap when i ≥ j
swap(A[l], A[j ])
return j
```

0	1	2	3	4	5	6	7
5	$\overset{i}{3}$	1	9	8	2	4	$\overset{j}{7}$
5	3	1	$\overset{i}{9}$	8	2	$\overset{j}{4}$	7
5	3	1	$\overset{i}{4}$	8	2	$\overset{j}{9}$	7
5	3	1	4	$\overset{i}{8}$	$\overset{j}{2}$	9	7
5	3	1	4	$\overset{i}{2}$	$\overset{j}{8}$	9	7
5	3	1	4	$\overset{j}{2}$	$\overset{i}{8}$	9	7
2	3	1	4	**5**	8	9	7
2	$\overset{i}{3}$	1	$\overset{j}{4}$				
2	$\overset{i}{3}$	$\overset{j}{1}$	4				
2	$\overset{i}{1}$	$\overset{j}{3}$	4				
2	$\overset{j}{1}$	$\overset{i}{3}$	4				
1	**2**	3	4				
1							
		3	$\overset{i\,j}{4}$				
		$\overset{j}{\mathbf{3}}$	$\overset{i}{4}$				
			4				
					8	$\overset{i}{9}$	$\overset{j}{7}$
					8	$\overset{i}{7}$	$\overset{j}{9}$
					8	$\overset{j}{7}$	$\overset{i}{9}$
					7	**8**	9
					7		
							9

FIGURE 2.11 Example of quicksort operation of Array with pivots shown in bold.

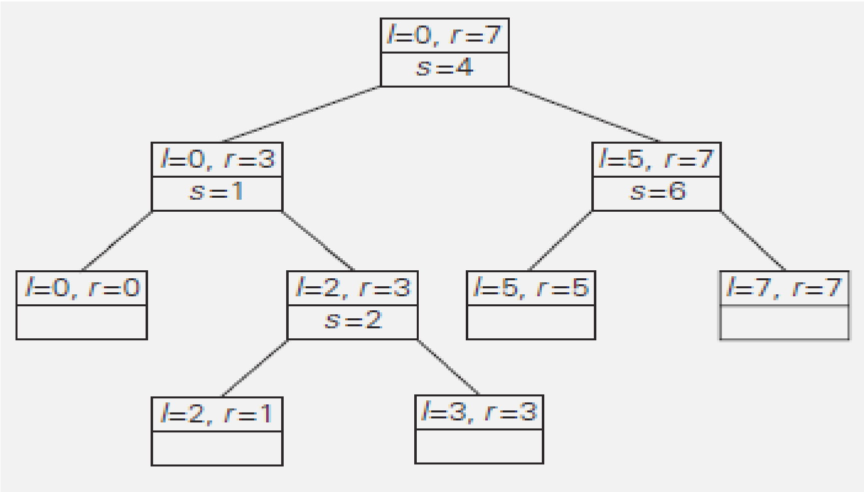


FIGURE 2.12 Tree of recursive calls to *Quicksort* with input values *l* and *r* of subarray bounds and split position *s* of a partition obtained.

The number of key comparisons in the best case satisfies the recurrence

$C_{best}(n) = 2C_{best}(n/2) + n$ for $n>1$, $\quad C_{best}(1) = 0$.

By Master Theorem, $C_{best}(n) \in \Theta(n \log_2 n)$; solving it exactly for $n = 2^k$ yields $C_{best}(n) = n \log_2 n$. The total number of key comparisons made will be equal to

$C_{worst}(n) = (n + 1) + n + \ldots + 3 = ((n + 1)(n + 2))/2 - 3 \in \Theta(n^2)$.

$$C_{avg}(n) = \frac{1}{n}\sum_{s=0}^{n-1}[(n+1) + C_{avg}(s) + C_{avg}(n-1-s)] \quad \text{for } n > 1,$$

$$C_{avg}(0) = 0, \quad C_{avg}(1) = 0.$$

$$C_{avg}(n) \approx 2n \ln n \approx 1.39 n \log_2 n.$$