10.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:

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

all are $\leq A[s]$ all are $\geq A[s]$

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

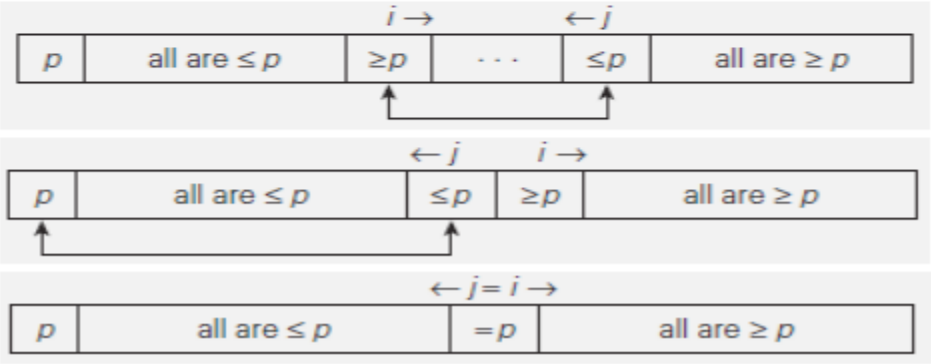
//Output: Subarray $A[l..r]$ sorted in non-decreasing order

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

$p \leftarrow A[l]$

$i \leftarrow l; j \leftarrow r+1$

repeat

  repeat $i \leftarrow i+1$ **until** $A[i] \geq p$

  repeat $j \leftarrow j-1$ **until** $A[j] \leq p$

  swap$(A[i], A[j])$

until $i \geq j$

swap$(A[i], A[j])$ //undo last swap when $i \geq j$

swap$(A[l], A[j])$

return j

0	1	2	3	4	5	6	7
	i						*j*
5	3	1	9	8	2	4	7
			i			*j*	
5	3	1	9	8	2	4	7
			i			*j*	
5	3	1	4	8	2	9	7
				i	*j*		
5	3	1	4	8	2	9	7
				i	*j*		
5	3	1	4	2	8	9	7
				j	*i*		
5	3	1	4	2	8	9	7
2	3	1	4	**5**	8	9	7
	i		*j*				
2	3	1	4				
	i	*j*					
2	3	1	4				
	i	*j*					
2	1	3	4				
	j	*i*					
2	1	3	4				
1	**2**	3	4				
1							
			i j				
		3	4				
		j	*i*				
		3	4				
			4				
						i	*j*
					8	9	7
						i	*j*
					8	7	9
						j	*i*
					8	7	9
					7	**8**	9
					7		
							9

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

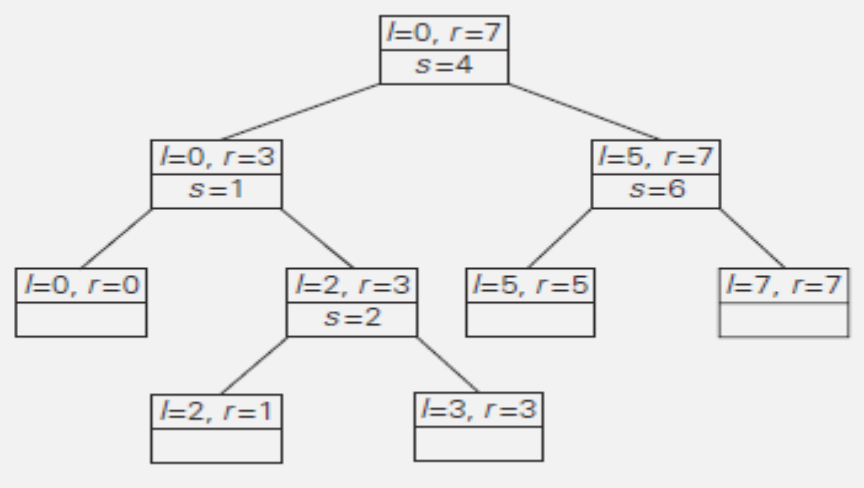


FIGURE 2.12 Tree of recursive calls to *Quicksort* with input values *l* and *r* of subarray bounds and split position *s* of a partition obtained.

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

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

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

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

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

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

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

ROHINI COLLEGE OF ENGINEERING AND TECHNOLOGY
PALKULAM, KANYAKUMARI
OBSERVE OPTIMIZE OUTSPREAD