Quick Sort

→ In Quick Sort, the division into 2 subarrays is made, so that the sorted subarrays do not need to be merged later.

→ Rearrange the elements in a[1:n] such that $a[i] \leq a[j]$.

→ For all i between 1 and m.

For all j between m+1 and n for some m,

$1 \leq m \leq n$

Eg:- The function is initially invoked (called) as partition(a, 1, 10)

a[1]	a[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	i	j
65	(70)	75	80	85	60	55	50	(45)	+∞	2	9
↑ pivot element											
65	45	(75)	80	85	60	55	(50)	70	+∞	3	8
65	45	50	(80)	85	60	(55)	75	70	+∞	4	7
65	45	50	55	(85)	(60)	80	75	70	+∞	5	6
(65)	45	50	55	(60)	85	80	75	70	+∞	6	5 (i>j)
				↓	i					swap	
60	45	50	55	65	85	80	75	70	+∞	a[i] ↔ a[j]	

↑ pivot
partition(a, 1, 4)

pivot ↑
Partition(a, 6, 9)

Function partition produces 2 sets S_1 and S_2

All elements in S_1 are $\leq$ All the elements in S_2

Hence S_1 and S_2 can be sorted independently.

Each set is sorted by reusing the function partition.

→ First element of the array is assumed to be pivot i.e., the partitioning element

→ Thus the elements in $a[1:m]$ and $a[m+1:n]$ can be independently sorted. No merge is needed.

→ The rearrangement of the elements is accomplished by picking some element of $a[\,]$, say $t = a[s]$ and then reordering the other elements so that

→ All elements appearing before t in $a[1:n]$ are $\leq t$ and

→ All elements appearing after t are $\geq t$

This rearranging is referred to as partitioning.

Algorithm Quicksort (a, P, q)

// sorts the elements a[P], a[P+1], ... a[q] which
// reside in the global array a[1:n] into
// ascending order ; a[n+1] is considered to be defined
// and must be ≥ all the elements in a[1:n].

{
    if (P < q) then // if there are more than one element
    {
        // divide P into subproblems (subarrays)
        j := partition(a, P, q+1);
        // j is the position of the partitioning element
        // solve the subproblem or sort the subarrays
        Quicksort (a, P, j-1);
        Quicksort (a, j+1, q);
        // there is no need to merge the subarrays
    }
}

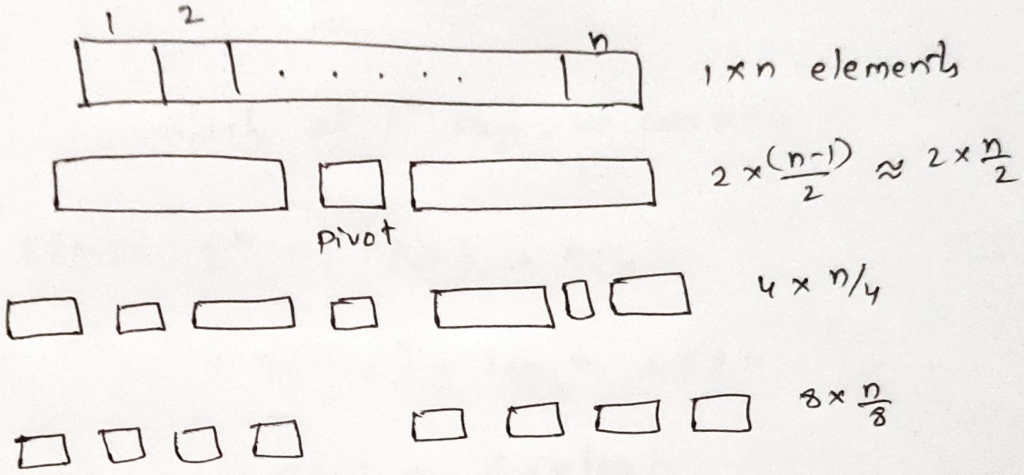
Algorithm Partition (a, m, p)

// within a[m], a[m+1], ... a[p-1] the elements
// are rearranged in such a manner that
// if initially t = a[m], then after completion
// a[q] = t for some q b/w m and p-1,

Time Complexity of Quick Sort :-

Best Case :-

In the best case, the pivot element is in the middle, which partitions the array into 2 equal sized subarrays.



1 × n elements

$2 \times \frac{(n-1)}{2} \approx 2 \times \frac{n}{2}$

$4 \times n/4$

$8 \times \frac{n}{8}$

∴ Time complexity recursive formula.

$$T(n) = \begin{cases} a & \text{if } n=1 \\ 2T(n/2) + cn & \text{if } n > 1 \end{cases}$$

cn — is the time taken for partitioning array

The running time = Time taken for two recursive calls (next) to sort subarrays + Linear time taken for partitioning the array (first)

$n = 2^k$

$k = \log_2 n$

$T(n) = 2T(n/2) + cn$ 1st step

$= 2\left[2T(n/4) + c \cdot n/2\right] + cn = 2^2 T(n/2^2) + 2cn$ 2nd step

$= 2^2\left[2T(n/8) + c \cdot n/4\right] + 2cn = 2^3 T(n/2^3) + 3cn$ 3rd step

⋮

Similarly at k^{th} step, we can write

$T(n) = 2^k T(n/2^k) + kcn$

$= n T(n/n) + \log_2 n + c \times n$

$= n T(1) + c \times n \log n$

$= na + c \times n \log n$

$T(n) = cn \log n + an$

$T(n) \propto n \log n$

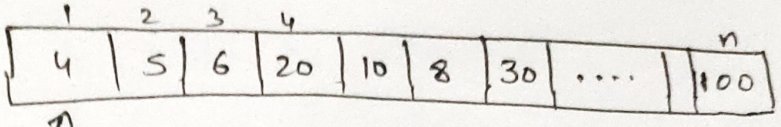
$\therefore T(n) = O(n \log n)$

Average case Time complexity:

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

Worst Case:

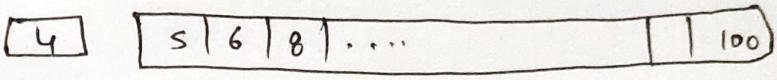
If we take pivot as smallest (1st element) the array would not be divided into 2 equal sized partitions, but one of length 0 and one of length (n-1)



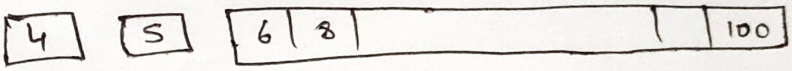
a

n elements

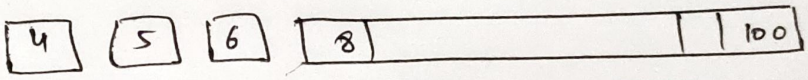
pivot element



(n-1) elements.



(n-2)



(n-3)

$T(n) = T(i) + T(n-i-1) + cn \qquad T(0) = 1$

$T(n) = T(0) + T(n-1) + cn \qquad$ if $i = 0$

$$T(n) = T(n-1) + cn$$
$$= T(n-2) + c(n-1) + cn$$
$$= T(n-3) + c(n-2) + c(n-1) + cn$$
$$\vdots$$

At n^{th} step, we can write

$$T(n) = T(n-n) + c\left[[n-(n-1)] + [n-(n-2)] + \dots + n\right]$$
$$= T(0) + c\left[1 + 2 + 3 + \dots + (n-1) + n\right]$$
$$= 1 + c \times \frac{n(n+1)}{2} \quad //$$

$$T(n) = 1 + c \times \frac{n^2 + n}{2}$$

$$T(n) \propto n^2$$

$$\therefore T(n) = O(n^2)$$