UNIT- II

DIVIDE AND CONQUER

Topic no: (1)

Divide and Conquer Methodology :-

In Divide and Conquer method, a given Problem is

(i) Divided into Smaller Sub Problems.

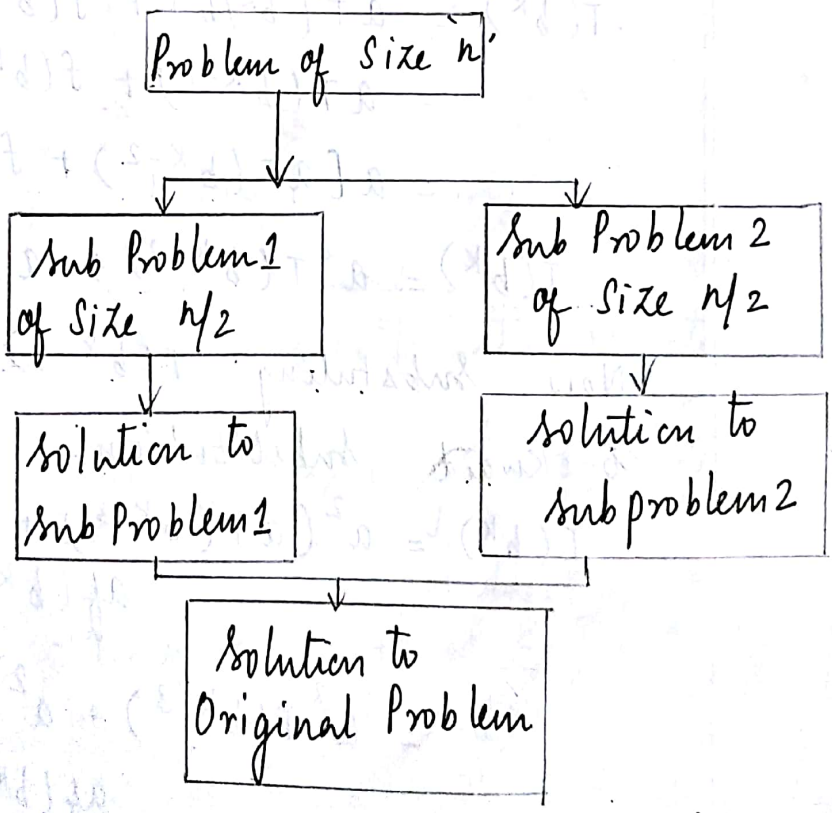
(ii) These Sub Problems are solved independently.

(iii) Combining all the solutions of Sub Problems into a solution of the whole.

✓ If the Sub Problems are large enough, then the divide and Conquer is reapplied.

✓ The generated Sub Problems are usually of Same type as the Original Problem.

∴ Hence the recursive algorithms are used in divide and Conquer methodology.



If we want to divide a Problem of size 'n' into a size of 'n/b'.

taking $f(n)$ time to divide and Conquer, then we set up a recurrence relation for obtaining time for size 'n' is

$$T(n) = a\,T(n/b) + f(n)$$

- $T(n)$ → Time for size 'n'
- a → no. of sub instances
- $T(n/b)$ → Time for size n/b.
- $f(n)$ → Time required for dividing the Problem.

∴ The above equation is called divide and Conquer recurrence relation.

Efficiency Analysis of Divide and Conquer:

Let the recurrence relation be

$$T(n) = a\,T(n/b) + f(n)$$

- Consider $a \geq 1$ and $b \geq 2$
- Assume $n = b^k$ where $k = 1, 2 \ldots$

$$T(b^k) = a\,T(b^k/b) + f(b^k)$$
$$= a\,T(b^{k-1}) + f(b^k)$$
$$= a[a\,T(b^{k-2}) + f(b^{k-1})] + f(b^k)$$
$$T(b^k) = a^2\,T(b^{k-2}) + a\,f(b^{k-1}) + f(b^k)$$

Now substituting $T(b^{k-2})$ by using backward substitution.

$$T(b^k) = a^2(a\,T(b^{k-3}) + a^2 f(b^{k-2}) \cdot$$
$$a f(b^{k-1}) + f(b^k).$$
$$\vdots$$
$$= a^3\,T(b^{k-3}) + a^2 f(b^{k-2}).$$
$$a f(b^{k-1}) + f(b^k).$$
$$\vdots$$

$$T(b^k) = a^k T(b^{k-k}) + a^{k-1} f(b^1) + a^{k-2} f(b^2) + \ldots + a^0 f(b(k))$$

$$T(b^k) = [a^k T(1) + a^{k-1} f(b) + a^{k-2} f(b^2) + \ldots + a^0 f(b^k)]$$

∴ This can be written as

$$T(b^k) = a^k T(1) + \frac{a^k}{a} f(b) + \frac{a^k}{a^2} f(b^2) + \ldots \frac{a^k}{a^k} f(b^k).$$

Taking a^k as common factor,

$$T(b^k) = a^k \left[T(1) + \frac{f(b)}{a} + \frac{f(b^2)}{a^2} + \ldots \frac{f(b^k)}{a^k} \right]$$

$$= a^k \left[T(1) + \sum_{j=1}^{k} \frac{f(b^j)}{a^j} \right]$$

By Property,

$$a \log_b x = x \log_b a.$$

∴ We can write a^k as

$$a^k = a^{\log_b n} = n^{\log_b a}.$$

∴ We can rewrite the equation.

$$T(n) = a^k \left[T(1) + \sum_{j=1}^{k} f(b^j)/a^j \right]$$

$$\boxed{T(n) = n^{\log_b a} \left[T(1) + \sum_{j=1}^{\log_b n} f(b^j)/a^j \right]}$$

Topic no: (2) **Divide and Conquer – Examples:**

Merge-Sort

The Merge sort is an sorting algorithm that uses Divide and Conquer strategy. In this method, division is dynamically Carried out.

– Merge sort on an input array with 'n' elements Consists of two steps.

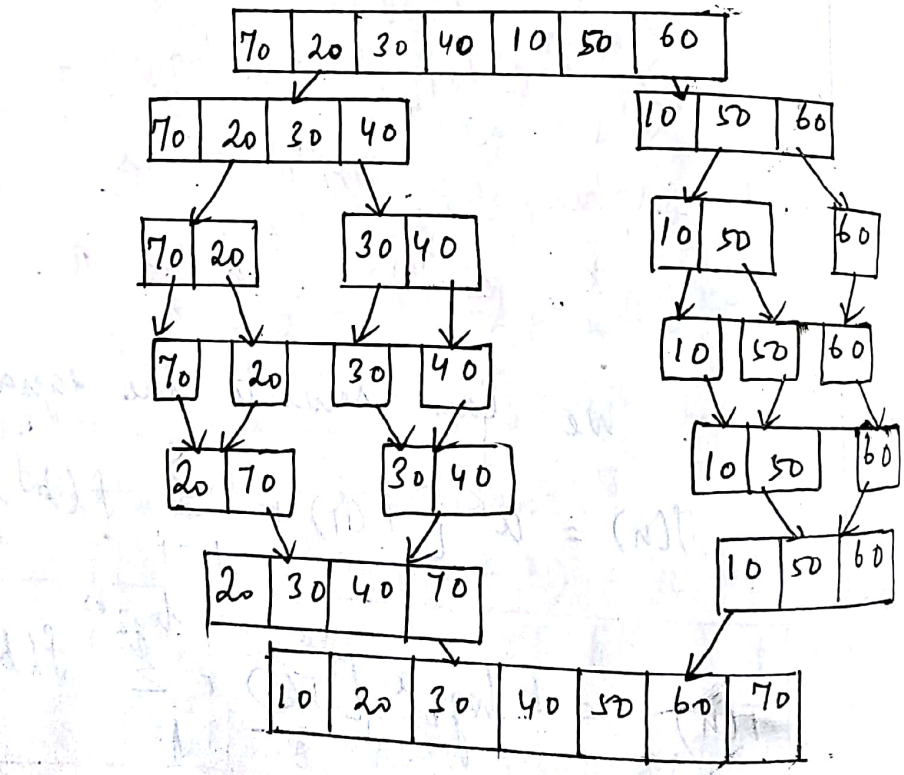
- ✓ Divide
- ✓ Conquer.

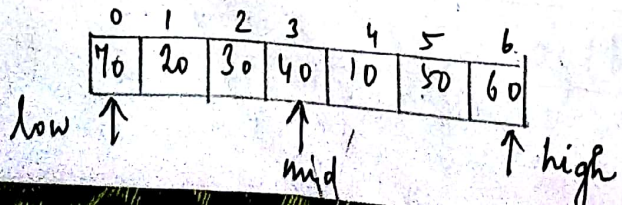
Example :– Merge Sort:

Consider the element,

70 20 30 40 10 50 60

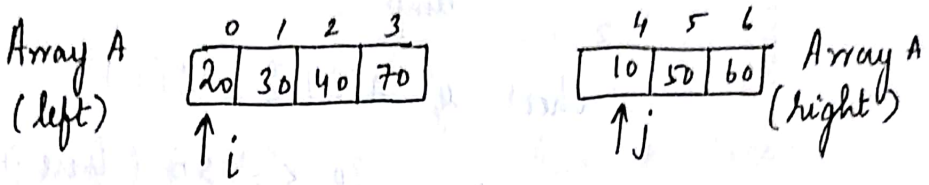
✓ Now we split the above lists into two sub-lists.

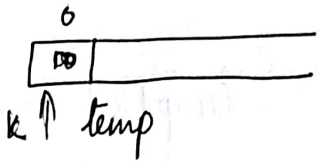




- After dividing the above list of elements, the last step before sorting has two sub lists to be merged.

eg: The two sub lists are.





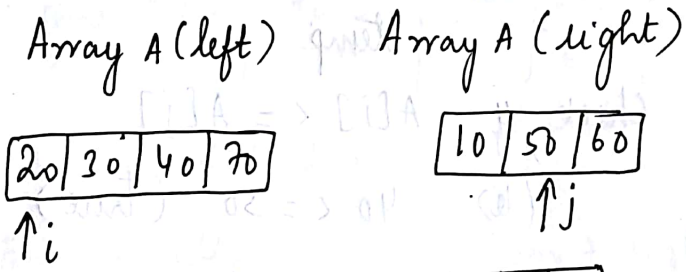
Initially K = 0 :-

Check if A[i] <= A[j]

(ie) 20 <= 10 (NO)

goes to else Part:

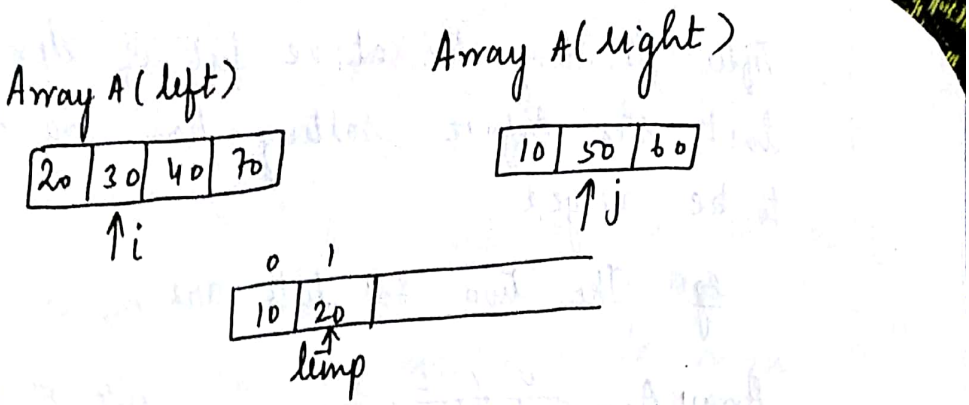
temp[K] ← A[j]	temp[0] ← 10
j ← j+1	j ← 1
K ← K+1	K ← 1



Check if A[i] <= A[j]

20 <= 50 (true)

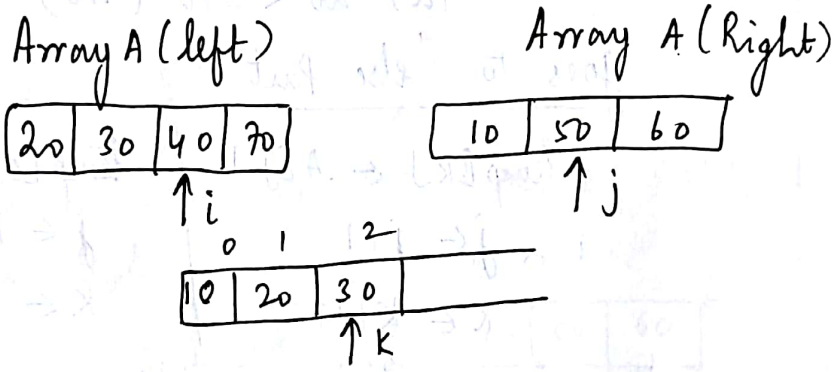
temp[K] ← A[i]	temp[1] ← 20
i ← i+1	i ← 1
K ← K+1	K ← 2



check if A[i] <= A[j]

30 <= 50 (true)

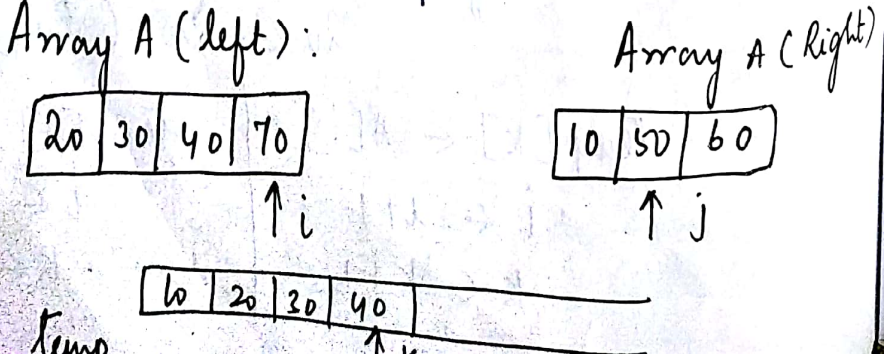
temp[k] ← A[i]	temp[2] ← 30
i ← i+1	i ← 2
k ← k+1	k ← 3



Check if A[i] <= A[j]

(ie) 40 <= 50 (true)

temp[k] ← A[i]	temp[3] ← 40
i ← i+1	i ← 3
k ← k+1	k ← 4



Check if A[i] <= A[j]

70 <= 50 [NO]

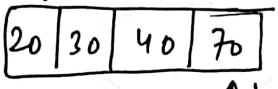
goes to else Part:

temp[k] ← A[j] | temp[4] ← 50

j ← j+1 | j ← 2

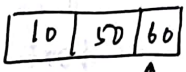
k ← k+1 | k ← 5

Array A (left)

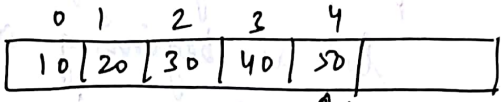


↑ i

Array A (right)



↑ j



↑ k

temp

Check if A[i] <= A[j]

(ie) 70 <= 60 (NO)

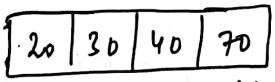
goes to else Part.

temp[k] ← A[j] | temp[5] ← 60

j ← j+1 | j ← 3

k ← k+1 | k ← 6

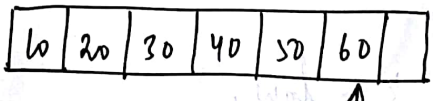
Array A (left)



↑ i

Array A (right)

↑ j



↑ k

temp

Check if A[i] <= A[j]

70 <= (?) no element in A[j]

∴ Finally we copy the A[i] to temp

temp:

10	20	30	40	50	60	70
						↑ k

Algorithm:-

```
Algorithm Mergesort (int A[0...n-1], low, high)
// Problem description: This algorithm is
// for sorting the elements using Merge-
// sort.
// Input: Array 'A' of unsorted elements
//    low as begining and high as end.
// output: Sorted Array A[0..n-1]
    if (low < high) then
    {
mid ← (low + high)/2 // Split the list
    Mergesort ( A, low, mid); // Sp First
                              left sub Array
    Mergesort ( A, mid+1, high); // Second
                                 left sub Array
    Combine ( A, low, mid, high);
                              // merging of 2
                                 arrays.
    }
```

```
Algorithm Combine ( A[0...n-1], low, mid, high)
{
    k ← low;
    i ← low;
    j ← mid + 1.
While ( i <= mid and j <= high) do
  {
```

```
if (A[i] <= A[j]) then
 {
    temp[k] ← A[i]
     i ← i+1
     k ← k+1
 }
else
 {
    temp[k] ← A[j]
     j ← j+1
     k ← k+1
 }
}
// Copy remaining elements of left sublist to
 temp array.
    While (i<=mid) do
    {
      temp[k] ← A[i]
       i ← i+1
       k ← k+1
    }
// Copy remaining elements of right sublist to
 temp array.
    While (j<=high) do
    {
      temp[k] ← A[j]
       j ← j+1
       k ← k+1
    }
```

Mathematical Analysis:-

In Merge sort, the two

recursive calls are made. Each recursion call focus on n/2 elements of the list. After 2 recursive calls, one call is made to Combine two sublists.

- Here we can write recurrence relation as

$$T(n) = T(n/2) + T(n/2) + C_n$$

- $T(n/2)$: Time taken by left sub list to get sorted
- $T(n/2)$: Time taken by right sublist to get sorted
- C_n: Time taken to Combine two sub lists.

and $T(1) = 0$

Substitution method :-

Let the recurrence relation for Merge Sort be

$$T(n) = T(n/2) + T(n/2) + C_n$$

(ie)

$$T(n) = 2T(n/2) + C_n$$

and $T(1) = 0$

Assume $n = 2^k$:

$$T(n) = 2T(n/2) + C_n$$

$$T(2^k) = 2T(2^k/2) + C \cdot 2^k \qquad (n = 2^k)$$

$$T(2^k) = 2T(2^{k-1}) + C \cdot 2^k$$

If we Put $k = k-1$ then,

$$T(2^K) = 2T(2^{K-1}) + C.2^K.$$

$$= 2[2T(2^{K-2}) + C.2^{K-1}] + C.2^K$$

$$= 2^2 T(2^{K-2}) + 2.C.2^{K-1} + C.2^K$$

$$= 2^2 T(2^{K-2}) + 2.C.\frac{2^K}{2} + C.2^K$$

$$= 2^2 T(2^{K-2}) + C.2^K + C.2^K$$

$$= 2^2 T(2^{K-2}) + 2C.2^K$$

Similarly, we can write,

$$T(2^K) = 2^3 T(2^{K-3}) + 3C.2^K$$

$$= 2^4 T(2^{K-4}) + 4C.2^K$$

$$\vdots$$

$$T(2^K) = 2^K T(2^{K-K}) + K.C.2^K$$

$$= 2^K T(2^0) + K.C.2^K$$

$$= 2^K T(1) + KC.2^K$$

$$= 2^K (0) + K.C.2^K$$

$$T(2^K) = K.C.2^K$$

But we assumed $n = 2^K$, taking log on both sides.

(ie) $\log_2 n = K$

$$T(n) = \log_2 n . Cn$$

$$\therefore \boxed{T(n) = O(n . \log_2 n)}$$

∴ Time Complexity:

Best Case: $O(n \log_2 n)$ Worst Case: $O(n \log_2 n)$

Average Case: $O(n \log_2 n)$

Topic no: (3)

Quick Sort:

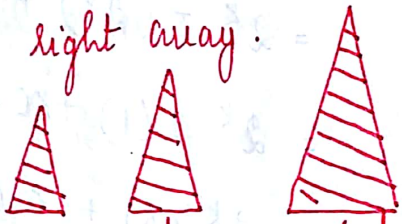
Quick Sort is a sorting algorithm that uses divide and Conquer strategy.

Steps:-

(I). ✓ Split the array into 2 sub arrays that each element in the left sub array is less than or equal to the middle element and each element in the right sub array is greater than the middle element.

✓ The splitting of the array into 2 sub array is based on Pivot element.

✓ All the elements that are less than Pivot should be in left sub array and all the elements that are more than Pivot should be in right array.



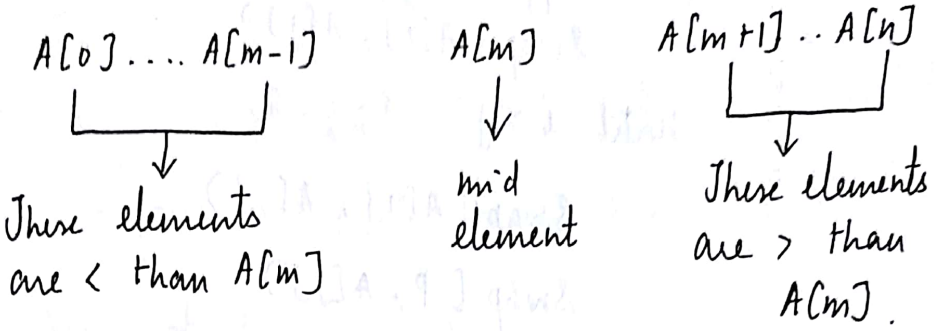
(elements that are < than Pivot) element (Pivot) (element that are > than Pivot)

(II). Recursively sort the two sub arrays.

(III). Combine all the sorted elements.

In Merge sort, the divisions is based on the Positions of array element, but in quicksort, the division is based on Actual Value of the element.

- Consider an Array A[i], where 'i' is ranging from 0 to n-1. We can formulate the division as.



Algorithm:-

```
Algorithm QuickSort (A[0...n-1], low, high)
// Problem Description: This algorithm Performs
   sorting of elements in an array.
// Input: The Array A[0...n-1]. The low
   indicates the leftmost elements in list
   and high indicates the rightmost elements
// Output: Sorted Array.
    if (low < high) then
    {
      m ← Partition (A[low ... high])
      Quicksort (A[low ... m-1])
      Quicksort (A[mid+1 .... high])
    }
```

In the above algorithm, the Partition Performs arrangement of the elements in ascending order.

```
Algorithm Partition (A[low..... high])
{
    P ← A[low]
    i ← low
    j ← low+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])
    swap [p, A[j])
return j.
```

Mathematical Analysis :-

The recurrence relation for Quick sort is

$$C(n) = C(n/2) + C(n/2) + n$$

- $C(n/2)$: Time required to sort left sub Array
- $C(n/2)$: Time required to sort right sub array
- n : Time required for Partition sub array

$$\therefore C(n) = 2C(n/2) + n$$

$$C(1) = 0$$

Substitution method :-

$$C(n) = 2C(n/2) + n \quad \text{——} (1)$$

$$C(1) = 0 \quad \text{——} (2)$$

$2^k, 2^{-1}, 2^{k-1}$

Assume $n = 2^k$ in (1)

$$C(2^k) = 2C(2^k/2) + 2^k$$

$$= 2C(2^{k-1}) + 2^k \quad \text{——} (3)$$

$$C(2^{k-1}) = 2C\left(\frac{2^{k-1}}{2}\right) + 2^k$$

$C(2^{k-1}) = 2C(2^{k-2}) + 2^k$ — (4)

Sub (4) in (3)

$C(2^k) = 2[2C(2^{k-2}) + 2^k] + 2^k$

$= 2^2 C(2^{k-2}) + 2^k + 2^k$ $(2^k[1+1])$

$C(2^k) = 2^2 C(2^{k-2}) + 2 \cdot 2^k$ — (5) ✓

Now Sub $n = 2^{k-2}$ in (1)

$C(2^{k-2}) = 2C\left(\frac{2^{k-2}}{2}\right) + 2^{k-2}$

$= 2C(2^{k-3}) + 2^{k-2}$ — (6)

Sub (6) in (5)

$\therefore C(2^k) = 2^2[2C(2^{k-3}) + 2^{k-2}] + 2 \cdot 2^k$

$= 2^3 C(2^{k-3}) + 2^2 2^{k-2} + 2 \cdot 2^k$

$= 2^3 C(2^{k-3}) + 2^k + 2 \cdot 2^k$

$C(2^k) = 2^3 C(2^{k-3}) + 3 \cdot 2^k$ — (7)

Similarly we can write,

$C(2^k) = 2^4 C(2^{k-4}) + 4 \cdot 2^k$

$= 2^k C(2^{k-k}) + k \cdot 2^k$

$= 2^k C(2^0) + k \cdot 2^k$

$= 2^k \cdot C(1) + k \cdot 2^k$

$= 2^k(0) + k \cdot 2^k$ $[C(1) = 0]$

$C(2^k) = k \cdot 2^k$ — (8)

∴ We assumed $n = 2^k$, taking log on both sides.

$k = \log_2 n$

Sub k in (8)

$C(n) = k \cdot n$

$C(n) = \log_2 n \cdot n$

∴ The best case Complexity of Quick sort is $O(n \cdot \log_2 n)$

Worst Case: $O(n^2)$

Average Case: $O(n \log_2 n)$

Problem: - Quick Sort:

	0	1	2	3	4	5	6	7
Pivot	50	30	10	90	80	20	40	70
	↑ i							↑ j

Check if A[i] <= Pivot

A[0] <= 50

50 <= 50 then i++

i = 1

	0	1	2	3	4	5	6	7
	50	30	10	90	80	20	40	70
		↑ i						↑ j

Check if A[i] <= Pivot

A[i] <= 50

30 <= 50, then i++

i = 2

	0	1	2	3	4	5	6	7
	50	30	10	90	80	20	40	70
			↑ i					↑ j

Again A[i] <= Pivot, then i++

0	1	2	3	4	5	6	7
(50)	30	10	90	80	20	40	70
			↑ i				↑ j

(i) A[i] <= Pivot (Condition fails)

90 <= 50.

∴ (ii) A[j] >= Pivot (Condition true)

70 >= 50 then j--;

∴ **j = 6**

0	1	2	3	4	5	6	7
(50)	30	10	90	80	20	40	70
			↑ i			↑ j	

(ii) A[j] >= Pivot fails

(iii) A[i] >= Pivot and A[j] <= Pivot

(i) 90 >= 50 and 40 <= 50

then Swap A[i] and A[j]

0	1	2	3	4	5	6	7
(50)	30	10	40	80	20	90	70
			↑ i			↑ j	

Again Condition (i) is true

(i) A[i] <= Pivot

40 <= 50 then i++

i = 4

0	1	2	3	4	5	6	7
(50)	30	10	40	80	20	90	70
				↑ i		↑ j	

Condition (i) fails.

Condition (ii) true - A[j] >= Pivot

90 >= 50 then j--

j = 5

0	1	2	3	4	5	6	7
(50)	30	10	40	80	20	90	70
				↑ i	↑ j		

Condition (ii) fails. ∴ Condition (iii) is Satisfied.

A[i] >= Pivot and A[j] <= Pivot

(ii) 80 >= 50 and 20 <= 50

∴ Swap A[i] and A[j]

0	1	2	3	4	5	6	7
50	30	10	40	20	80	90	70
				↑ i	↑ j		

Condition (i) is Satisfied ∴ i++ i=5

0	1	2	3	4	5	6	7
50	30	10	40	20	80	90	70
					↑ i,j		

Condition (i) fails, Condition (ii) is Satisfied

A[i] >= Pivot

80 >= 50 (j--) ∴ j=4

0	1	2	3	4	5	6	7
50	30	10	40	20	80	90	70
				↑ j	↑ i		

As i>j, then Swap A[j] and Pivot.

0	1	2	3	4	5	6	7
20	30	10	40	50	80	90	70

Now 50 - Pivot is in Correct Position

✓ Divide the below Output of Quicksort into two sub arrays, (i) left and right sub array.

0	1	2	3
20	30	10	40

left sub array

0	1	2
80	90	70

right sub Array.

Sort the above left and right sub array by applying Quicksort Procedure recursively. Thus the final sorting after applying Quicksort Procedure is

0	1	2	3	4	5	6	7
10	20	30	40	50	70	80	90

Topic no: (4)

Multiplication of Large Integers.

Multiplication:

Multiply the multiplicand by each of multiplier and then add up all the Property of shifted results. This method is also called grade-school multiplication.

Eg:-

```
  42
  34 x
 ----
 168
1260      [Padding '0' at Unit Position]
----
1428
```

- But this method is not convenient for Performing multiplication of large integers. So we are in interesting algorithm of multiply large numbers.

Let us formulate this method,

Let $c = a * b$ — (1)

$c = c_2 10^2 + c_1 10^1 + c_0$ — (2)

Sub (1) in (2)

(ie) $a * b = c_2 b^2 + c_1 b^1 + c_0$ — (3)

Where,

$c_2 = a_1 * b_1$

$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$

$c_0 = a_0 * b_0$

Example :-

Consider $42 * 34$

$a = a_1 a_0$, $a_1 = 4$, $a_0 = 2$

$b = b_1 b_0$, $b_1 = 3$, $b_0 = 4$

To obtain the values c_0, c_1 and c_2.

$c_0 = a_0 * b_0$

$= 2 * 4$

$\boxed{c_0 = 8}$

$c_2 = a_1 * b_1$

$= 4 * 3$

$\boxed{c_2 = 12}$

$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$

$= (4 + 2) * (3 + 4) - (12 + 8)$

$= 6 * 7 - 20$

$\boxed{c_1 = 22}$

Now Sub the values of c_0, c_1, c_2 in (3)

$a * b = c_2 10^2 + c_1 10^1 + c_0$

$= 12 \times 10^2 + 22 \times 10^1 + 8$

$42 * 34 = 1428$

Mathematical Analysis :-

$c_0 = a_0 * b_0$ — Multiplication ①

$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$ — multiplication ②

$c_2 = a_1 * b_1$ — Multiplication ③

If there are 'n' digits to be multiplied, then multiplication of 'n' digit requires 3 multiplication of n/2 digit number.

✓ Heme the recurrence relation can be written as

$c(n) = 3c(n/2), \quad n > 1$ — ①

$c(n) = 1$, $n = 1$ — ②

Assume $n = 2^k$,

$c(2^k) = 3c(2^k/2) = 3c(2^{k-1})$ — ③

$c(2^{k-1}) = 3c(2^{k-1}/2) = 3c(2^{k-2})$ — ④

Sub ④ in ③

$\Rightarrow 3[3c(2^{k-2})]$ — ⑤

$c[2^{k-2}] = 3c[2^{k-2}/2]$

$= 3c[2^{k-3}]$ — ⑥

Sub ⑥ in ⑤

$\Rightarrow 3[3[3c[2^{k-3}]]]$

$\Rightarrow 3^3[c[2^{k-3}]]$

$\vdots$

$\Rightarrow 3^k[c[2^{k-k}]]$

$\Rightarrow 3^k[c[2^0]]$

$\Rightarrow 3^k c[1]$

$\Rightarrow 3^k(1)$

$$\boxed{c(2^k) = 3^k} \quad \text{---} \quad (7)$$

$n = 2^k$,

$\therefore$ Take log on both sides, we get

$$\log(n) = \log 2^k \qquad \log n = \log_2 n$$

$$\log_2 n = k \log_2 2 \qquad \log_2 2 = 1$$

$$\therefore \boxed{k = \log_2 n}$$

Sub 'n' and 'k' in eq (7)

$$c(n) = 3 \log_2 n$$

(ie) $a \log_b c = c \log_2 a$ (formula)

$$\therefore c(n) = n \log_2 3$$

$$\boxed{c(n) \approx n^{1.58}}$$

---

no: (05) **Strassen's Matrix Multiplication :-**

Suppose we want to multiply two matrices A and B each of size 'N'. (ie)

$C = A \times B$, then

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

This above multiplication gives,

$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$

$C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}$

$C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$

$C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22}$.

Addition:

$C_{11} = A_{11} + B_{11}$

$C_{21} = A_{21} + B_{21}$

$C_{22} = A_{22} + B_{22}$

$C_{12} = A_{12} + B_{12}$

→ Thus to accomplish 2x2 matrix multiplication, that there are total of 8 multiplications and 4 additions.

- The above matrix multiplication requires more (ie) 8 multiplications, this is more costly ∴ Strassen's multiplication of two matrices requires only 7 multiplications.

Example:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} * \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$S_1 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$

$S_2 = (A_{21} + A_{22}) \times B_{11}$

$S_3 = A_{11} \times (B_{12} - B_{22})$

$S_4 = A_{22} \times (B_{21} - B_{11})$

$S_5 = (A_{11} + A_{12}) \times B_{22}$

$S_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$

$S_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$

$C_{11} = S_1 + S_4 - S_5 + S_7$

$C_{12} = S_3 + S_5$

$C_{21} = S_2 + S_4$

$C_{22} = S_1 + S_3 - S_2 + S_6$

Apply Strassen's method to multiply the following matrices.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 2 & 7 & 1 \\ 2 & 7 & 0 & 5 \\ 4 & 3 & 2 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 5 & 6 & 7 & 8 \\ 1 & 0 & 3 & 4 \\ 6 & 2 & 7 & 0 \\ 8 & 1 & 6 & 5 \end{bmatrix}$$

A_{11} A_{12} A_{21} A_{22} B_{11} B_{12} B_{21} B_{22}

$$\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 2 & 7 & 1 \\ \hline 2 & 7 & 0 & 5 \\ 4 & 3 & 2 & 1 \end{array} \quad * \quad \begin{array}{cc|cc} 5 & 6 & 7 & 8 \\ 1 & 0 & 3 & 4 \\ \hline 6 & 2 & 7 & 0 \\ 8 & 1 & 6 & 5 \end{array}$$

$S_1 = (A_{11} + A_{22}) * (B_{11} + B_{22})$

$$= \left[\begin{bmatrix} 1 & 2 \\ 5 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 5 \\ 2 & 1 \end{bmatrix} \right] * \left[\begin{bmatrix} 5 & 6 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 7 & 0 \\ 6 & 5 \end{bmatrix} \right]$$

$$= \begin{bmatrix} 1 & 7 \\ 7 & 3 \end{bmatrix} * \begin{bmatrix} 12 & 6 \\ 7 & 5 \end{bmatrix}$$

$$= \begin{bmatrix} 12+49 & 6+35 \\ 84+21 & 42+15 \end{bmatrix}$$

$$S_1 = \begin{bmatrix} 61 & 41 \\ 105 & 57 \end{bmatrix}$$

$$S_2 = (A_{21} * A_{22}) * B_{11}$$

$$= \left[\begin{bmatrix} 2 & 7 \\ 4 & 3 \end{bmatrix} + \begin{bmatrix} 0 & 5 \\ 2 & 1 \end{bmatrix}\right] \times \begin{bmatrix} 5 & 6 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 12 \\ 6 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 10+12 & 2+0 \\ 30+4 & 36+0 \end{bmatrix}$$

$$S_2 = \begin{bmatrix} 22 & 12 \\ 34 & 36 \end{bmatrix}$$

$$S_3 = A_{11} \times (B_{12} - B_{22})$$

$$= \begin{bmatrix} 1 & 2 \\ 5 & 2 \end{bmatrix} \times \left[\begin{bmatrix} 7 & 8 \\ 3 & 4 \end{bmatrix} - \begin{bmatrix} 7 & 0 \\ 6 & 5 \end{bmatrix}\right]$$

$$= \begin{bmatrix} 1 & 2 \\ 5 & 2 \end{bmatrix} \times \begin{bmatrix} 0 & 8 \\ -3 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} 0-6 & 8-2 \\ 0-6 & 40-8 \end{bmatrix}$$

$$S_3 = \begin{bmatrix} -6 & 6 \\ -6 & 32 \end{bmatrix}$$

$$S_4 = A_{22} \times (B_{21} - B_{11})$$

$$= \begin{bmatrix} 0 & 5 \\ 2 & 1 \end{bmatrix} \times \left[\begin{bmatrix} 6 & 2 \\ 8 & 1 \end{bmatrix} - \begin{bmatrix} 5 & 6 \\ 1 & 0 \end{bmatrix}\right]$$

$$= \begin{bmatrix} 0 & 5 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & -4 \\ 7 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0+35 & 0+5 \\ 2+7 & -8+1 \end{bmatrix}$$

$$S_4 = \begin{bmatrix} 35 & 5 \\ 9 & -7 \end{bmatrix}$$

$$S_5 = (A_{11} + A_{12}) \times B_{22}$$

$$= \begin{bmatrix} 1 & 2 \\ 5 & 2 \end{bmatrix} + \begin{bmatrix} 3 & 4 \\ 7 & 1 \end{bmatrix} \times \begin{bmatrix} 7 & 0 \\ 6 & 5 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 6 \\ 12 & 3 \end{bmatrix} \times \begin{bmatrix} 7 & 0 \\ 6 & 5 \end{bmatrix}$$

$$= \begin{bmatrix} 28+36 & 0+30 \\ 84+18 & 0+15 \end{bmatrix}$$

$$S_5 = \begin{bmatrix} 64 & 30 \\ 102 & 15 \end{bmatrix}$$

$$S_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$$

$$= \left[\begin{bmatrix} 2 & 7 \\ 4 & 3 \end{bmatrix} - \begin{bmatrix} 1 & 2 \\ 5 & 2 \end{bmatrix} \right] \times \left[\begin{bmatrix} 5 & 6 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 7 & 8 \\ 3 & 4 \end{bmatrix} \right]$$

$$= \begin{bmatrix} 1 & 5 \\ -1 & 1 \end{bmatrix} \times \begin{bmatrix} 12 & 14 \\ 4 & 4 \end{bmatrix}$$

$$= \begin{bmatrix} 12+20 & 14+20 \\ -12+4 & -14+4 \end{bmatrix}$$

$$S_6 = \begin{bmatrix} 32 & 34 \\ -8 & -10 \end{bmatrix}$$

$S_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$

$= \left[\begin{bmatrix} 3 & 4 \\ 7 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 5 \\ 2 & 1 \end{bmatrix}\right] \times \left[\begin{bmatrix} 6 & 2 \\ 8 & 1 \end{bmatrix} + \begin{bmatrix} 7 & 0 \\ 6 & 5 \end{bmatrix}\right]$

$S_7 = \begin{bmatrix} 25 & 0 \\ 65 & 10 \end{bmatrix}$

Sub $S_1, S_2, S_3, S_4, S_5, S_6, S_7$ in $C_{11}, C_{12}, C_{22}, C_{21}$

$C_{11} = S_1 + S_4 - S_5 + S_7$

$= \begin{bmatrix} 61 & 41 \\ 105 & 57 \end{bmatrix} + \begin{bmatrix} 35 & 5 \\ 9 & -7 \end{bmatrix} - \begin{bmatrix} 64 & 30 \\ 102 & 15 \end{bmatrix} + \begin{bmatrix} 25 & 0 \\ 65 & 10 \end{bmatrix}$

$= \begin{bmatrix} 96 & 46 \\ 114 & 50 \end{bmatrix} - \begin{bmatrix} 64 & 30 \\ 102 & 15 \end{bmatrix} + \begin{bmatrix} 25 & 0 \\ 65 & 10 \end{bmatrix}$

$= \begin{bmatrix} 96-64+25 & 46-30+0 \\ 114-102+65 & 50-15+10 \end{bmatrix}$

$C_{11} = \begin{bmatrix} 57 & 16 \\ 77 & 45 \end{bmatrix}$

$C_{12} = S_3 + S_5$

$= \begin{bmatrix} -6 & 6 \\ -6 & 38 \end{bmatrix} + \begin{bmatrix} 64 & 30 \\ 102 & 15 \end{bmatrix}$

$C_{12} = \begin{bmatrix} 58 & 36 \\ 96 & 53 \end{bmatrix}$

$C_{21} = S_2 + S_4$

$= \begin{bmatrix} 22 & 12 \\ 34 & 36 \end{bmatrix} + \begin{bmatrix} 35 & 5 \\ 9 & -7 \end{bmatrix}$

$$C_{21} = \begin{bmatrix} 57 & 17 \\ 43 & 29 \end{bmatrix}$$

$$C_{22} = S_1 + S_3 - S_2 + S_6$$

$$= \begin{bmatrix} 61 & 47 \\ 105 & 57 \end{bmatrix} + \begin{bmatrix} -6 & 6 \\ -6 & 38 \end{bmatrix} - \begin{bmatrix} 22 & 12 \\ 34 & 36 \end{bmatrix} + \begin{bmatrix} 32 & 34 \\ -8 & -10 \end{bmatrix}$$

$$C_{22} = \begin{bmatrix} 65 & 69 \\ 57 & 49 \end{bmatrix}$$

∴ The required solution is

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$= \begin{bmatrix} 57 & 16 & 58 & 36 \\ 7 & 45 & 96 & 53 \\ 77 & 17 & 65 & 69 \\ 43 & 29 & 57 & 49 \end{bmatrix}$$

Mathematical Analysis :-

Strassen's showed that 2x2 matrix multiplication can be accomplished in 7 multiplications and 18 additions or subtractions.

∴ The recurrence relation can be written as

$$T(n) = 7T(n/2) \quad \text{——} (1)$$

$$T(1) = 1 \quad \text{——} (2)$$

Sub $n = 2^k$ in (1).

$T(2^k) = 7T\left(\frac{2^k}{2}\right)$

$T(2^k) = 7T(2^{k-1})$ ——— ③

Sub $n = 2^{k-1}$ in ①

$T(2^{k-1}) = 7T\left(\frac{2^{k-1}}{2}\right)$

$= 7T(2^{k-2})$ ——— ④

Sub ④ in ③

$T(2^k) = 7[7T(2^{k-2})]$ — ⑤

Now Sub $n = 2^{k-2}$ in ①

$T(2^{k-2}) = 7T\left(\frac{2^{k-2}}{2}\right)$

$= 7T(2^{k-3})$ ——— ⑥

Sub ⑥ in ⑤

$T(2^k) = 7[7[7T(2^{k-3})]]$

⋮

$T(2^k) = 7^3 T\left(\frac{2^k}{2^3}\right)$

Sub $2^k = n$, we get $k = \log_2 n$, equation

$T(n) = 7^k T(n/2^k)$ — ⑦

$= 7^k T(n/n)$

$= 7^k T(1)$ $(T(1) = 1)$

$= 7^k (1)$

$\boxed{T(n) = 7^k}$

$T(n) = 7 \log_2 n$

$= n \log_2 7$

$= n^{2.81}$

$\therefore \boxed{T(n) = n^{2.81}}$

$\log^n = \log_2 n$

$\log_2 7 = 2.81$

Topic no: ⑥ Binary search:- ⑦ ⑧ ⑨ ⑩ ⑪ ⑫

✓ Binary search is one of the efficient searching method. While searching the elements in the array, the array should be the sorted one.

✓ An element which is to be searched from the list of elements stored in array A[0..n-1] is called Key element.

A[0]... A[m-1] | A[m] | A[m+1] ... A[n-1]

Search if Key < A[m] | Key | Search if Key > A[m]

✓ Let A[m] be the mid element of array A.

✓ Three conditions while searching the array.

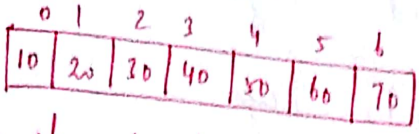
(i) If Key = A[m], the desired element is present in the list

(ii) If Key < A[m], then search the left sub array.

(iii) If Key > A[m], then search the right sub array.

Example:-

Consider a list of elements stored in array A as



– Array 'A'.

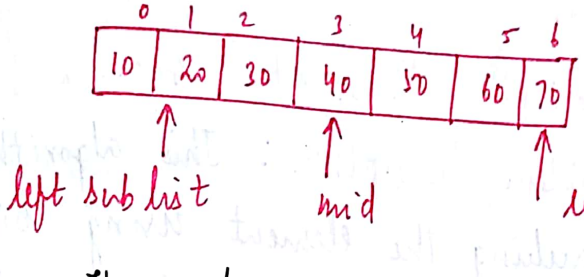
Key element : 60

Now obtain the middle element,

$mid = (low + high)/2$

$= (0+6)/2$

$mid = 3$



Then check A[m] == Key

(ie) A[3] == 60

40 == 60 (NO), (Condition 1+2 fails).

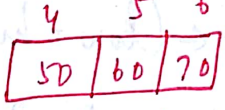
then 40 Key > A[m]

60 > A[3]

60 > 40. (Condition (iii) Satisfied)

∴ Search in right sub list

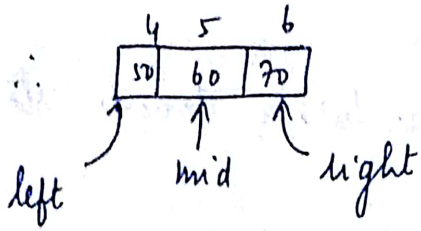
The right sub list is



Now again divide the list and find the middle element

$$m = \frac{low + high}{2} = \frac{4+6}{2} = \frac{10}{2}$$

$$\boxed{m = 5}$$



A[m] == Key

A[5] == 60

60 == 60 . ∴ Here Condition (i) is Satisfied with the Search element and it is Present in 5th Position in the list.

Algorithm:-

Algorithm Binsearch [A[0...n-1], Key]

// Problem description : This algorithm is for searching the element using binary search.

// Input: An array A from which the Key element is to be searched from Sorted order.

// Output: It returns the index of an array element.

```
low ← 0
high ← n-1
While (low < high) do
{
    m ← (low + high) / 2
if ( Key == A[m] ) then
    return m.
else if ( Key < A[m]) then
    high ← m-1    // Search the
                  left sub list
```

```
else
    low ← mid + 1    // Search the right sub list.
}
    Return -1        // If element is not found
                        in the list
```

Mathematical Analysis :-

✓ Basic Comparision is done for search key with array elements.

✓ The way to Analyze the effiency of binary search is to Count the number of times the search key is Compared with an element of the Array.

Worst-Case effiency :-

It is the algorithm Compares all the array elements for searching the desired element.

The recurrence relation for Worst-Case time Complexity is given by

$$C_{worst}(n) = C_{worst}(n/2) + 1 \quad \text{for } n > 1 \qquad (1)$$

$C_{worst}(n/2)$: Time required to Compare left or right sub list

$+1$: One Comparision made with middle element.

And

$$C_{worst}(n) = 1$$

Assume $n = 2^k$ then eq ① becomes,

$$C_{worst}(2^k) = C_{worst}\left(\frac{2^k}{2}\right) + 1$$

$$C_{worst}(2^k) = C_{worst}(2^{k-1}) + 1 \quad \text{— ③}$$

Sub $n = 2^{k-1}$ in eq ①

$$C_{worst}(2^{k-1}) = C_{worst}\left(\frac{2^{k-1}}{2}\right) + 1$$

$$C_{worst}(2^{k-1}) = C_{worst}(2^{k-2}) + 1 \quad \text{— ④}$$

Sub ④ in ③

$$C_{worst}(2^k) = [C_{worst}(2^{k-2}) + 1] + 1$$

$$C_{worst}(2^k) = C_{worst}(2^{k-2}) + 2 \quad \text{— ⑤}$$

now Sub $n = 2^{k-2}$

$$C_{worst}(2^{k-2}) = C_{worst}\left(\frac{2^{k-2}}{2}\right) + 1$$

$$= C_{worst}(2^{k-3}) + 1 \quad \text{— ⑥}$$

Sub ⑥ in ⑤

$$C_{worst}(2^k) = [C_{worst}(2^{k-3}) + 1] + 2$$

$$= C_{worst}(2^{k-3}) + 3 \quad \text{— ⑦}$$

⋮

$$C_{worst}(2^k) = C_{worst}(2^{k-k}) + k$$

$$= C_{worst}(2^0) + k$$

$$C_{worst}(2^k) = C_{worst} + k \quad \text{— ⑧}$$

eq ⑧ becomes $[C_{worst} = 1$

$\therefore C_{worst}(2^k) = 1+k$ — (9)

As we assumed $n = 2^k$, Take log on both sides.

$$\log n = \log_2 K$$
$$\log_2 n = k \log_2 2$$
$$\therefore k = \log_2 n$$

Sub K Values in eq (9), we get

$$C_{worst}(2^k) = 1 + \log_2 n \quad \text{for } n \geq 1$$

neglecting the Constants,

$\therefore$ The WorstCase Time Complexity is $O(\log_2 n)$

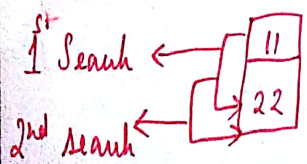
Average Case:-

✓ To obtain Average Case efficiency, of binary search, we will Consider some Sample of Input 'n'.

If $n=1$ and one element 11 is Present means, Only one search is required to Search some key.

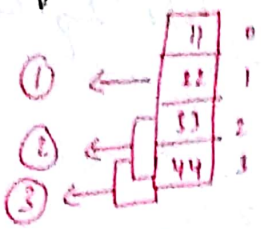
eg: [11] → 1 Search.

If $n=2$, and Search key we Consider as 22. (K = 22)



⇒ Two Comparisons are made to Search 22.

If $n=4$, and search key = 44



$\Rightarrow m = \frac{0+3}{2} = \frac{3}{2} = 1$

$\Rightarrow A[1] = 22$ and $22 < 44$, then search right sub list again

$\Rightarrow m = \frac{2+3}{2} = \frac{5}{2} = 2$

$\Rightarrow A[2] = 33$ and $33 < 44$ then search right sub list

$\Rightarrow A[4] = 44$. and the solution is obtained.

$\therefore$ So totally, (3) Comparisions are required.

If $n=8$, and search key = 88

$\therefore$ Thus total of (4) Comparisions are required to search 88.

From the above observations,

$$C = \log_2 n + 1$$

Assume if $n=2$

$C = \log_2 2 + 1$

$C = 1 + 1$

$C = 2$

If $n=8$

$C = \log_2 8 + 1$

$C = 3 + 1$

$C = 4$

$\therefore$ Then the Average Case for binary Search is : $\Theta(\log_2 n)$ $\therefore$

Best Case : O(1)

Average Case : O(log2 n)

Worst Case : O(log2 n)

Topic no:7

Brute - Force Technique :-

Definition :-

It is a straightforward approach of solving the Problem. It is directly based on the Problem statement and definitions of Concepts that are directly involved in the Problem.

Strength :-

- ✓ Simplicity
- ✓ applicable to Wide Variety of Problems.
- ✓ A good yardstick for better algorithms.
- ✓ It is an inefficient method, even though it is useful for solving small instances of a Problem.

Weakness :

- ✓ Rarely Produces effiient algorithm.
- ✓ Some brute force algorithms will be slow.

Topic no:8

Exhaustive Search :-

Definition : It is a method in which solution is obtained by searching each

element of given Problem.

✓ used to search for Combinatorial Problems.

✓ It make use of Brute force's approach.

✓ It implements a state space tree.

State Space Tree:- Given a Initial state and a goal state, a set of operations to attain the goal state.

- Evaluates Potential solutions One by one, disqualifying infeasible ones and for optimization Problem, Keeping track of the best one.

- The following are the problems that can be solved by Exhaustive search.

✓ Travelling Salesman Problem

✓ Knapsack Problem

✓ Assignment Problem.

Travelling salesman Problem:-

- Travelling Salesman Problem is one of the famous Problem in geaph theory.

- Consider that are 'n' cities and travelling salesman has to visit each city exactly once and have to return to the city from which he was

started.

- To model, this Problem, Weighted graphs can be used.
- The Vertices of such graph represents cities and the edge weight specifies the distance between the cities.
- This Problem can also be stated as finding "Hamiltonian Circuit" of a graph.

Hamiltonian Circuit: It is defined as a cycle, that Passes through all the Vertices.

	$\sum w_i$	$\sum v_i$	
Subset	Total Weight	Total Value.	
{ }	0	0	
{1}	2	$1	
{2}	3	$2	
{3}	4	$8	
{4}	5	$6	
{1,2}	2+3=5	$3	
{1,3}	2+4=6	$9	
{1,4}	2+5=7	$7	
{2,3}	3+4=7	$10	— optimal solution
{2,4}	3+5=8	$8	
{1,2,3}	2+3+4=9	$11	
{1,2,4}	2+3+5=10	$9	
{2,3,4}	3+4+5=12	$16	→ not feasible

since there are 'n' items, there are 2^n Possible Combination of items.

- Running time (or) Time Complexity is $O(2^n)$.

Assignment Problem:-

- Consider there are 'n' People who need to be assigned to execute 'n' jobs.
- only one person is assigned to execute one job at a time.
- Then Problem is to find such assignment that gives Smallest total cost.
- The cost can be computed as C[i,j] (ie) i^{th} Person is assigned to j^{th} Job.

Problem:-

Assign the jobs to Person A, B, C, D such that Cost is low.

Persons	Job 1	Job 2	Job 3	Job 4
A	9	5	4	5
B	4	3	5	6
C	3	1	3	2
D	2	4	2	6

Sol:

Cost Matrix 'C' =

	Job1	Job2	Job3	Job4
1	9	5	4	5
2	4	3	5	6
3	3	1	3	2
4	2	4	2	6

This Problem is to select one element in each row of the matrix, so that all selected elements are in different Columns and the total sum of the selected elements is the smallest Possible.

(1,2,3,4) = 9+3+3+6 = 21

(1,2,4,3) = 9+3+2+2 = 16

(1,3,2,4) = 9+5+1+6 = 19

✗ (1,3,4,2) = 9+5+2+4 = 20

(1,4,2,3) = 9+6+1+2 = 18 ✓

(1,4,3,2) = 9+6+3+4 = 22 ✓

(2,1,3,4) = 5+9+3+6 = 23

(2,3,1,4) = 5+5+2+2 = 24

(2,1,4,3) = 5+4+2+4 = 15

(2,3,4,1) = 5+5+2+2 = 14

(2,4,3,1) = 5+6+3+2 = 16

(2,4,1,3) = 5+6+3+2 = 16

(3,1,2,4) = 4+4+3+6 = 17

(3,1,4,2) = 4+4+2+4 = 14

(3,2,4,1) = 4+3+2+2 = (11) // optimal solution

(3,2,1,4) = 4+3+3+6 = 16

(3,4,1,2) = 4+6+3+4 = 17

(3,4,2,1) = 4+6+1+2 = 13

(4,3,2,1) = 5+5+1+2 = 13

(4,3,1,2) = 5+5+3+4 = 17

$(4,1,3,2) = 5+4+3+4 = 16$

$(4,1,2,3) = 5+4+1+2 = 12$

$(4,2,1,3) = 5+3+3+2 = 13$

$(4,2,3,1) = 5+3+3+2 = 13$

∴ Thus by trying 24 Permutations

[$n! = 4! = 4\times3\times2\times1 = 24$]

∴ The feasible solution is $(3,2,4,1)$

∴ **Cost = 4+3+2+2 = 11**

∴ Thus for Person A → Job '3' is Assigned
Person B → Job '2' is Assigned
Person C → Job '4' is Assigned
Person D → Job '1' is Assigned
With low Cost (11).

---

pic no (9)

Closest Pair Problem:

✓ The closest Pair Problem is used for finding the two closest Points in a set of 'n' points.

✓ For simplicity, the closest Pair Problem Can be Considered to be in two dimensional Case.

✓ Consider two Points specified by $P_i = (x_i, y_i)$ is a Point on a two dimensional Planes & $P_j = (x_j, y_j)$

✓ The distance between the 2 Points is denoted by Euclidean distance. It is denoted by

$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

✓ so brute force approach, is used to Compute the distance between each Pair of distinct Points and to find a Pair with the Smallest distance.

Algorithm:

```
Algorithm BruteforceclosestPair (P)
// Problem description: To find two closest
// Points in the Plane using brute force
// technique.
// Input: A list 'P' of n (n >, 2) Points
// P1 = (x1, y1) ... P(n) = (xn, yn)
// Output: return the indices of closest
// Pair of Points
{
   min-dist ← ∞
   for i ← 1 to n-1 do
      for j ← i+1 to n do
      dis ← Sqrt [(xi - xj)^2 + (yi - yj)^2]
      if dis < min-dist
            in1 ← i
            in2 ← j;
      return in1, in2.
}
```

Mathematical Analysis:-

- The basic operation of the algorithm is Computing the Euclidean distance b/w 2 Points.

- In most Case, Computing the Square root will give only approximate Value. To avoid this, if we replace $d \leftarrow sqrt((x_i - x_j)^2 + (y_i - y_j)^2)$.

- Now the basic operation of the algorithm will be the squaring of a number. The input size depends on the number of points.

$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2$$

$$= \sum_{i=1}^{n-1} (n-i)$$

$$= 2[(n-1)(n-2)\ldots+1]$$

$$= (n-1)(n)$$

$$C(n) = n^2 - n$$

Taking the highest Power, we get the time Complexity as $O(n^2)$

Example:-

List 'P' with Points P1(3, 9), P2(6, 4) P3(7, 3) [n = 3]

min_dist ← ∞,

i ← 1, // range of i = {1, 2} x_i, y_i

j ← 2 // range of j = {2, 3} x_j, y_j

i ← 1 :

dis ← Sqrt (B-9)^2 + (9-4)^2

dis ← 5.83

j ← 2 :

dis ← Sqrt (6-7)^2 + (4-3)^2

dis ← 1.4

i ← 2

Range of i = {2, 2}

j ← 3

Range of j = {3}

P1 and P2 :-

$(3-6)^2 + (9-4)^2 = (-3)^2 + (5)^2 = 9 + 25$

$= \sqrt{34} = 5.8$

P2 and P3 :

$(6-7)^2 + (4-3)^2 = (-1)^2 + (1)^2 = 1+1 = \sqrt{2} = 1.4$

P1 and P3 :

$(3-7)^2 + (9-3)^2 = (-4)^2 + (6)^2$

$= 16 + 36 = \sqrt{52} = 7.2$