UNIT-I

⊛ Algorithm:-

"What is algorithm?" and "When it is required?"

Definition of algorithm: The algorithm is defined as a collection of unambiguous instructions occuring in some sequence and such an algorithm should produce output for given set of input in finite amount of time.

(or)

Algorithm is a set of steps to complete a task.

(or)

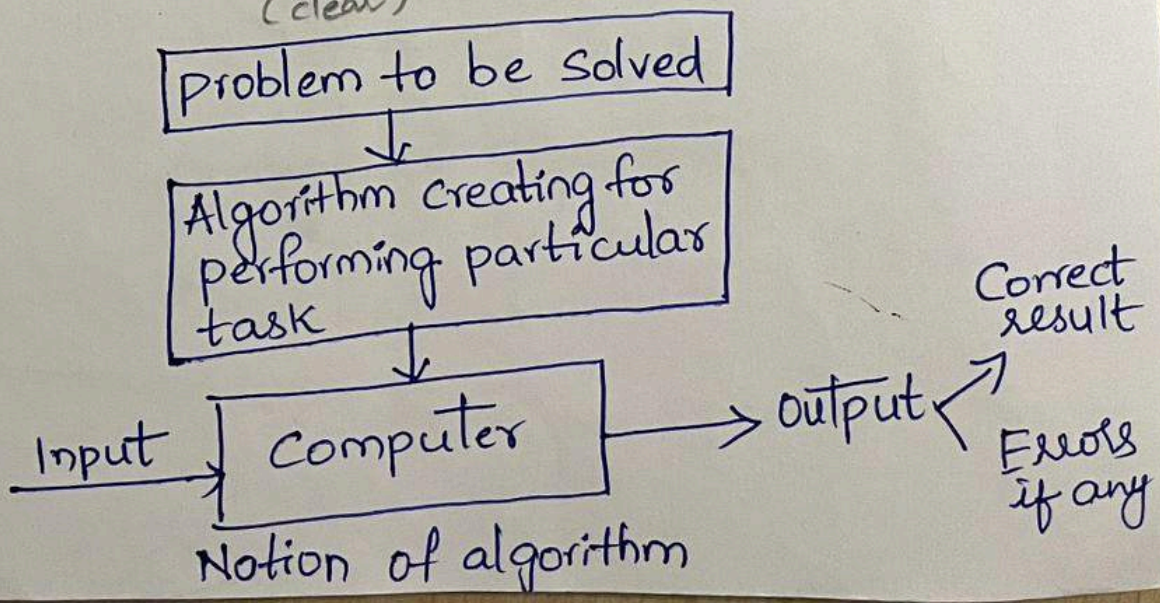
An Algorithm is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria

⊛ Input: Zero or more quantities are externally supplied.

⊛ Output: Atleast one quantity is produced.

⊛ Definiteness: Each instruction is clear and unambiguous.

(clear)



Notion of algorithm

properties of Algorithm :-

Algorithm should poses following properties -

1) Input: The input of zero or more number of quantities should be given to the algorithm
2) Output: The algorithm should produce at least one quantity, as an output.
3) Definiteness: Each instruction in algorithm should be specific and unambiguous.
4) Finiteness: The algorithm should be finite. That means after finite number of steps it should terminate.
5) Effectiveness: Every step of algorithm should be feasible. In other words, every step of algorithm should be such that it can be carried out by pen and pencil.

→ The implementation of such ~~pro~~ algorithms can be done programming language. Hence the term program can be defined as.

"program is the expression of algorithm using some progeamming language".

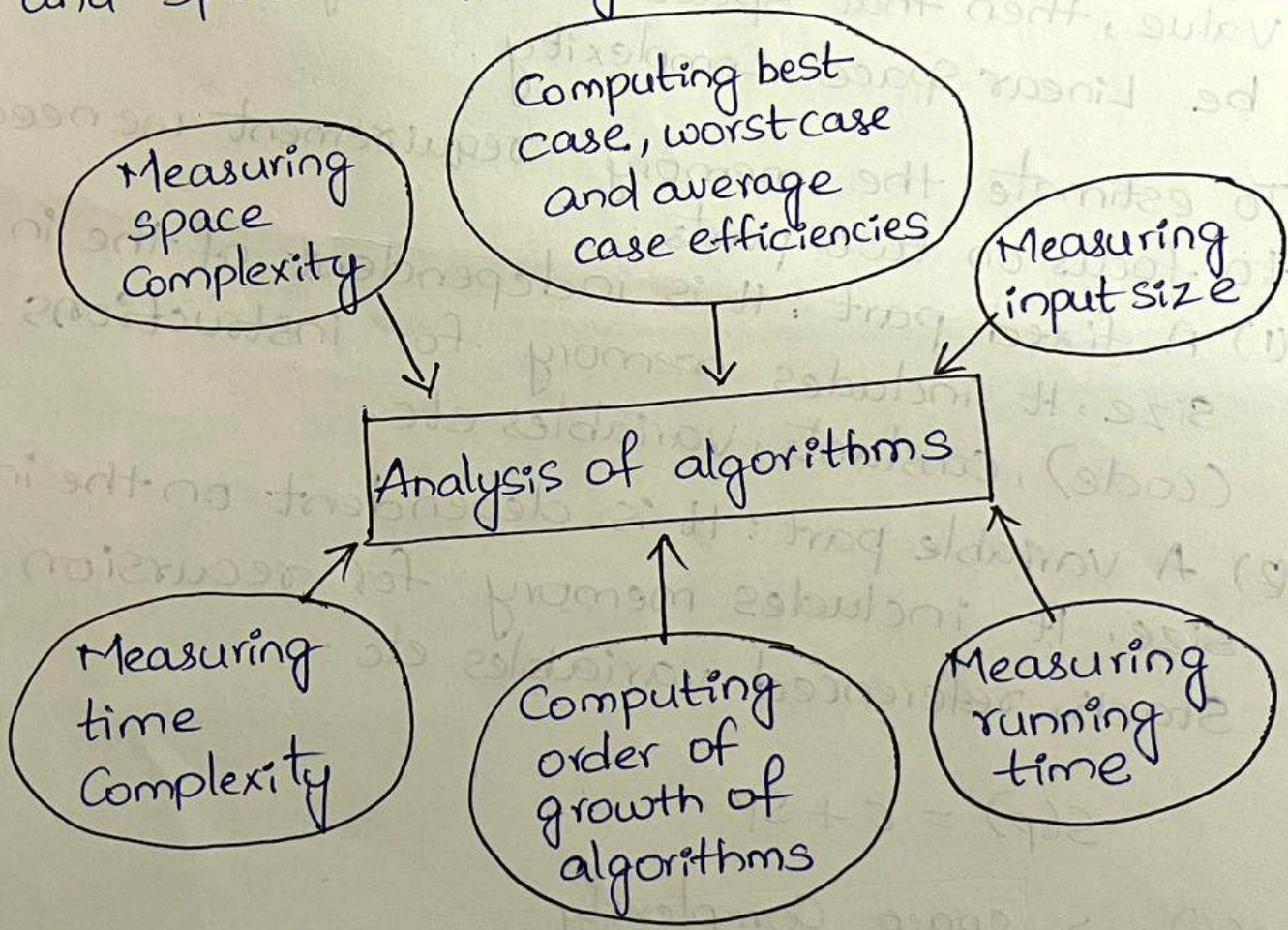
Performance Analysis:-

The efficiency of an algorithm can be decided by measuring the performance of an algorithm. We can measure the performance of an algorithm by computing two factors.

① Amount of time required by an algorithm to execute.

② Amount of storage required by an algorithm.

This is popularly known as time complexity and space complexity of an algorithm.



Analysis of algorithms.

Space Complexity:

The space complexity can be defined as amount of memory required by an algorithm to run.

(Or)

Total amount of computer memory required by an algorithm to complete its execution is called as space complexity of that algorithm.

→ If the amount of space required by an algorithm is increased with the increase of input value, then that space complexity is said to be Linear Space Complexity.

To estimate the memory requirement we need to focus on two parts.

(1) A fixed part: It is independent of the input size. It includes memory for instructions (code), constants, variables etc.

(2) A variable part: It is dependent on the input size. It includes memory for recursion stack, referenced variables etc.

$$S(p) = C + SP$$

$S(p)$ → space Complexity

C → Constant / independent part

SP → dependent part / variable part.

Ex:
```
algorithm Sum(p,q,r)
{
    p=1;
    q=2;
    r= p+q;
}
```

p → Constant → 1 unit
q → Constant → 1 unit
r → " → 1 unit

$S(p) = C + SP$
$= 3 + 0 = 3 \Rightarrow O(1)$
↳ Order of 1

Ex:- Sum of elements in an array.

```
algorithm Sum(S,n)
{
    total = 0;
    for i = 0 to n do
    total = total + S[i];
}
```

S → array name
n → size
n → 1 unit
total → 1 unit
i → 1 unit
S → n units.
↳ dependent variable

$S(p) = C + SP$

$= 1 + 1 + 1 + n$

$S(p) = 3 + n \rightarrow O(n) \rightarrow$ order of n.

↳ highest value

Time Complexity :-

Amount of time an algorithm requires to complete its execution is called time Complexity.

We have three types

① Best case : If an algorithm is taking minimum amount of time for its execution, it is called as best case (less time)

2 sec ↑ alg 1, 5 sec ↑ alg 2, 3 sec ↑ alg 3

② Worst case : (more time)
algorithm is taking maximum amount of time

③ Average case : If algorithm is taking average amount of time.

Ex : Linear Search

④

1	2	3	4	5
x	x	x	✓	

1st case : Search '1' → 1 Sec → Best
Search '5' → 5 Sec → Worst case
Search '3' → 3 Sec → Average

We have two approaches to calculate Tc

① Frequency Count
② Asymptotic Notations

Asymptotic Notations:-

Mathematical way of representing the time Complexity of an algorithm.

They are four Notations.

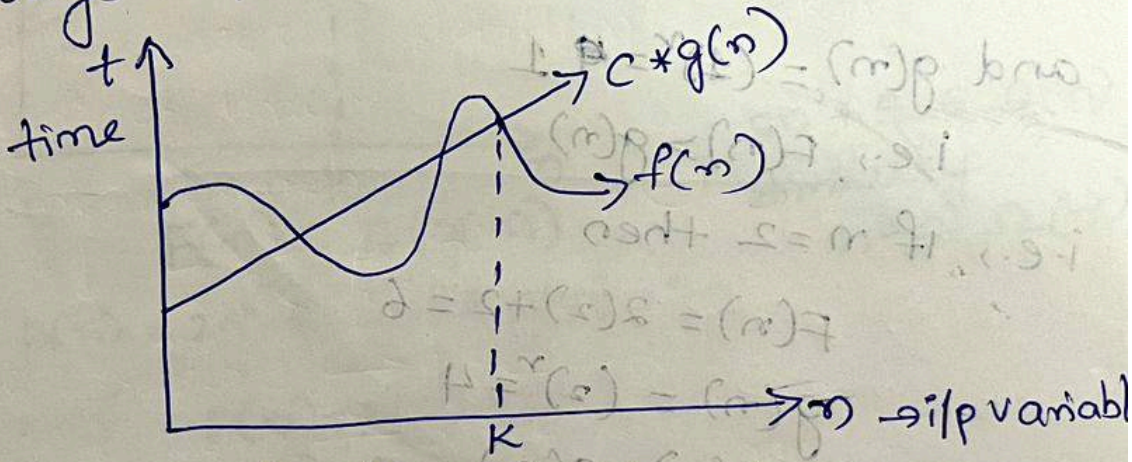
① Big-oh (O) Notation $F(n) \in O(g(n))$

→ It is denoted by 'O'.

→ It is a method of representing the upper bound of algorithm's running time. max time

→ Using this notation we can give longest amount of time taken by the algorithm to Complete.

⇒ Always represents the worst case TC only



$f(n)$ → is a function

$g(n)$ → another function.

We are representing $f(n)$ in terms of $g(n)$.

$$f(n) = O \cdot g(n)$$

Then according to Big-Oh

$$f(n) \leq c * g(n) \qquad c \rightarrow \text{Constant}$$

$c > 0$

$n \geq K, K \geq 0$

Ex:- $f(n) = 2n^2 + n$ $f(n) = O(?)$

$$2n^2 + n \leq c \cdot g(n^2)$$

$$2n^2 + n \leq c \cdot n^2$$

$2n^2+n \leq 1 \cdot n^2$ let $c=1$

$2n^2+n \leq 2n^2$ $c=2$

$2n^2+n \leq 3n^2$ $c=3$

Ex:- Consider Function $f(n)=2n+2$ and $g(n)=n^2$. Then we have to find some Constants So that $F(n) \leq c * g(n)$.

$F(n)=2n+2$ and $g(n)=n^2$

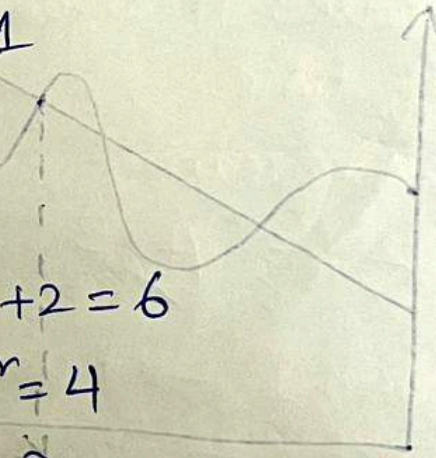
then we will find c for $n=1$

$F(n)=2(1)+2$

$F(n)=4$

and $g(n)=(1)^2=1$

i.e., $F(n) > g(n)$



i.e., if $n=2$ then

$F(n)=2(2)+2=6$

$g(n)=(2)^2=4$

$F(n) > g(n)$

If $n=3$ then

$F(n)=2(3)+2=8$

$g(n)=(3)^2=9$

$F(n) \leq g(n)$

Hence we conclude that for $n>2$, we obtain $F(n) \leq g(n)$

→ Least upper bound will be Considered

2) Omega Notation :

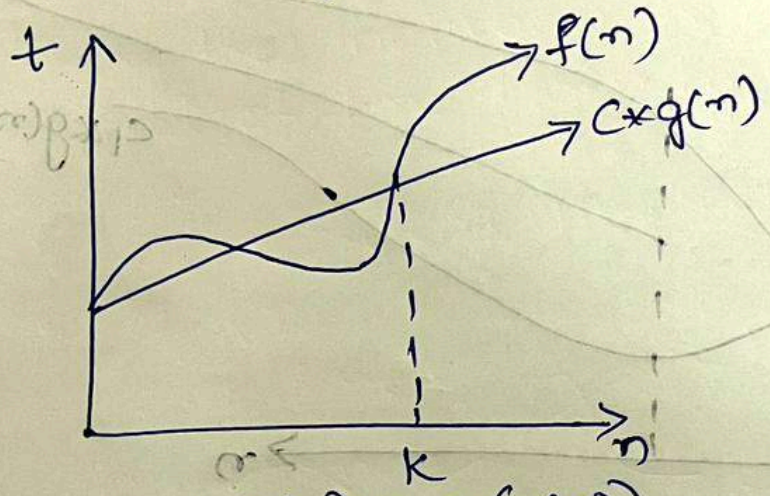
→ It is denoted by 'Ω'

→ It is used to represent the lower bound of algorithm's running time but greater lower bound.

→ We can denote shortest amount of time taken by algorithm.

→ best case time complexity is represented

$f(n) \geq c*g(n)$ $\quad F(n) \in \Omega(g(n))$



$F(n) \in \Omega(g(n))$

$2n^2+5 \in \Omega(n)$

Similarly

$n^3 \in \Omega(n^2)$

Ex:- $F(n) = 2n^2+5$ and $g(n) = 7n$

if $n=0$ $\quad F(n) = 2(0)+5 = 5$

$g(n) = 0$

$n=1$ $\quad F(n) = 2(1)^2+5 = 7$

$g(n) = 7$ $\quad$ i.e., $F(n) = g(n)$

$n=2$ $\quad F(n) = 2(2)^2+5 = 13$

$g(n) = 7 \times 2 = 14$

$n=3$ $\quad F(n) = 2(3)^2+5$

$= 18+5 = 23$

$g(n) = 7 \times 3 = 21$

$F(n) > g(n)$

Thus for $n=3$ we get $F(n) > c*g(n)$

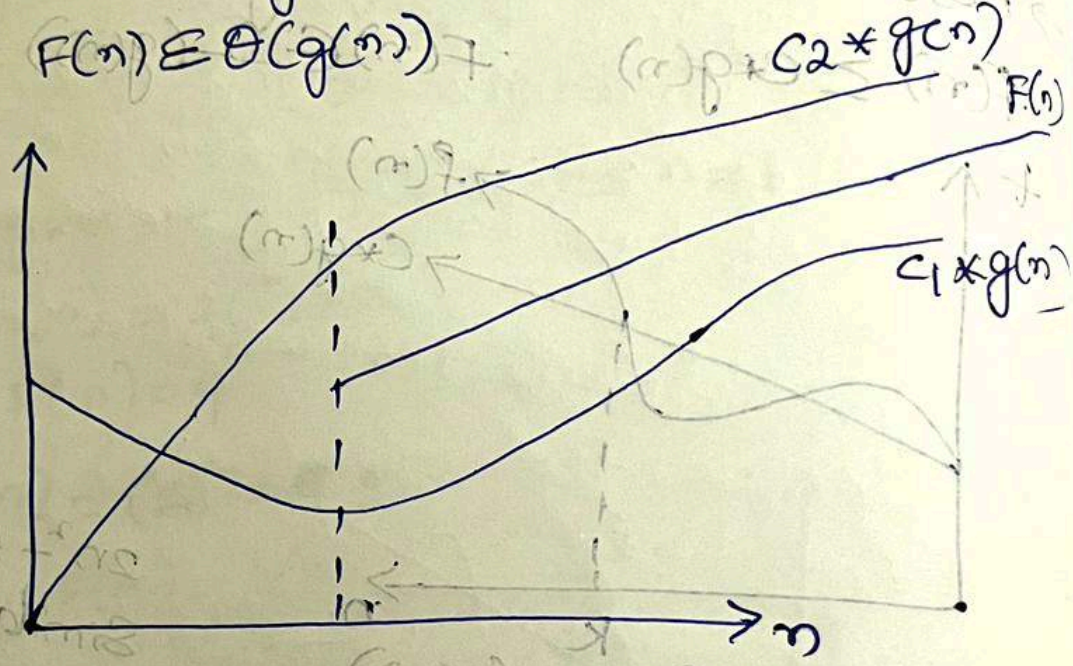
3) θ Notation (theta)

→ By this method the running time is between upper and lower bound.

Def:- Let $F(n)$ and $g(n)$ be two non negative functions. There are two positive Constants namely c_1 and c_2 such that

$$c_1 \leq g(n) \leq c_2 g(n)$$

$$F(n) \in \theta(g(n))$$



Example:

If $F(n) = 2n+8$ and $g(n) = 7n$

where $n \geq 2$

$F(n) = 2n+8 \quad g(n) = 7n$

~~$c_1 \leq 7n \leq c_2 7n$~~

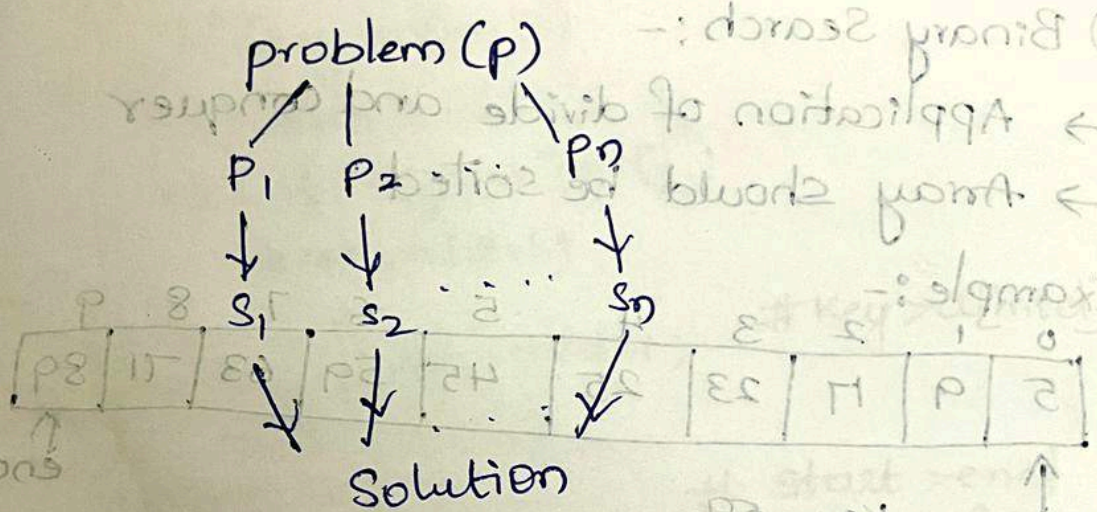
$5n \leq 2n+8 < 7n$

$c_1 = 5, \quad c_2 = 7 \quad n_0 = 2$

Divide and Conquer Method

One of the algorithm solving strategy

In this method,
- we divide a big problem into subproblems
- solve each subproblem
- Combine the solutions together.



Algorithm:-

$\{2+3\}+\{3-1\}\times\{4+3\}$

```
DAC(P)
{
  if (small(p))
  {
    S(p);
  }
  else{divide P in to P1, P2, ... Pn
       apply DAC(P1), DAC(P2), ... DAC(Pn)
       Combine (DAC(P1), ... DAC(Pn));
  }
```

Quick Sort :-

→ Arrange the array.

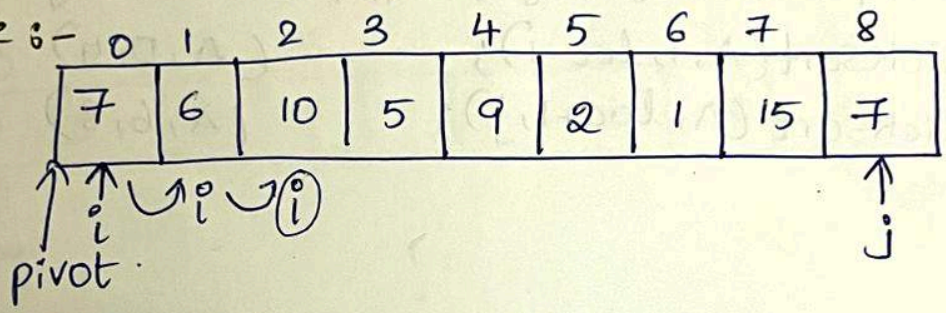
→ Application of divide and conquer → get the Solution.

First, we should apply partitioning algorithm

↓

finds proper position for pivot

Example :-



i ≤ j

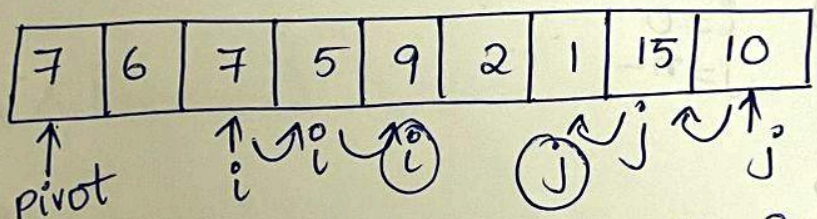
a[i] ≤ pivot

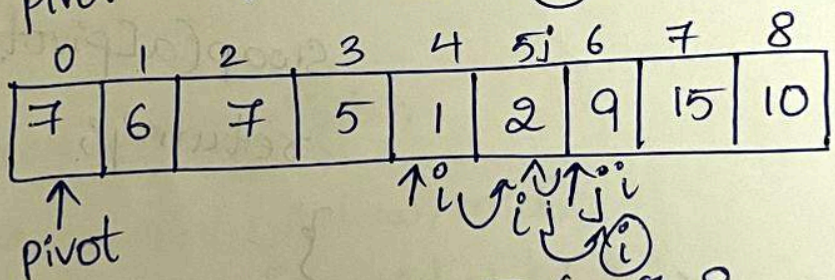
i++

a[j] > pivot

j--

Swap(i,j)





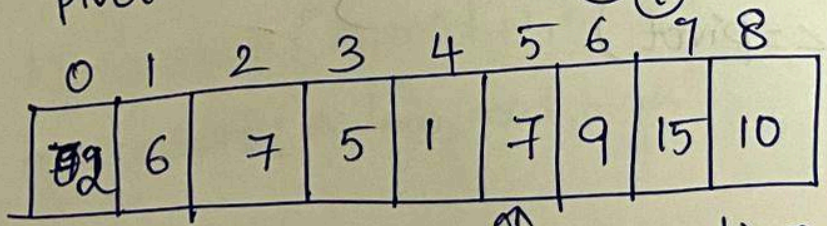
Swap.

i = 6

j = 5

when i > j

Swap(a[j], pivot)



pivot → location

partition

Algorithm Quick sort

```
Quicksort(A,i,j)
{
  if(i<j)
  {
    loc = partition(A,i,j);   # partition Alg.
    Quicksort(A,i,loc-1);
    Quicksort(A,loc+1,j);
  }
}
```

5

(A,0,4)

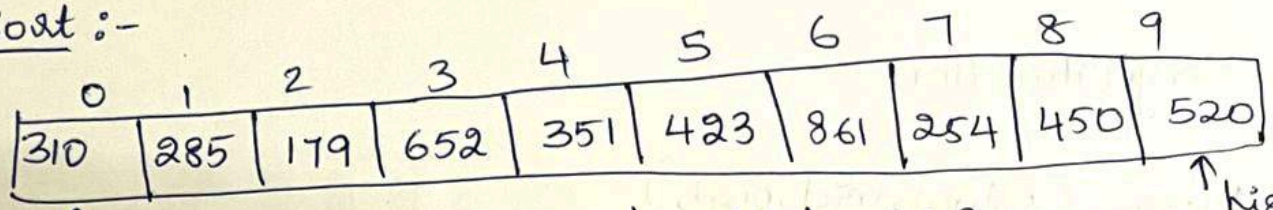
(A,6,8)

```
partition(A,i,j)
{
  pivot = a[i];
  i=0;
  j=n-1;
  while(i<j)
  {
    while(a[i]<=pivot)
    {
      i++;
    }
    while(a[j]>pivot)
    {
      j--;
    }
    if(i<j)
    { swap(a[i],a[j]);
    }
  } # end of while
```

i=0
j=n-1

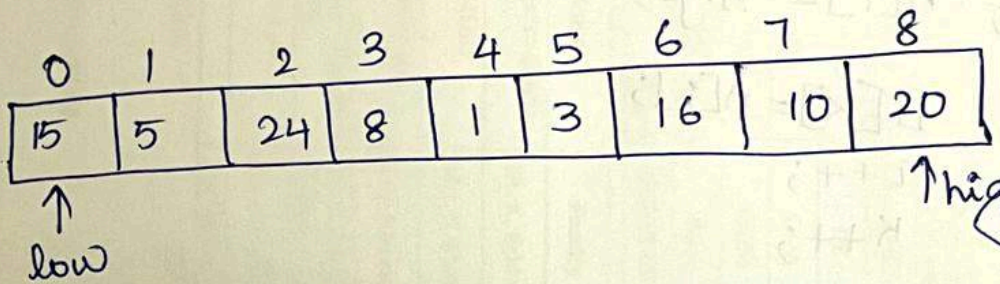
```
  swap(a[pivot],a[j]);
  return j;
}
```

Merge Sort :-



$mid = \frac{low + high}{2} = \frac{0+9}{2} = 4$

low, mid




```
Mergesort (A, low, high)
{
    if (low < high)
    {
        mid = (low + high) / 2;
        Mergesort (A, low, mid);
        Mergesort (A, mid+1, high);
        Mergesort (A, low, mid, high).
    }
}
```

Applications :-

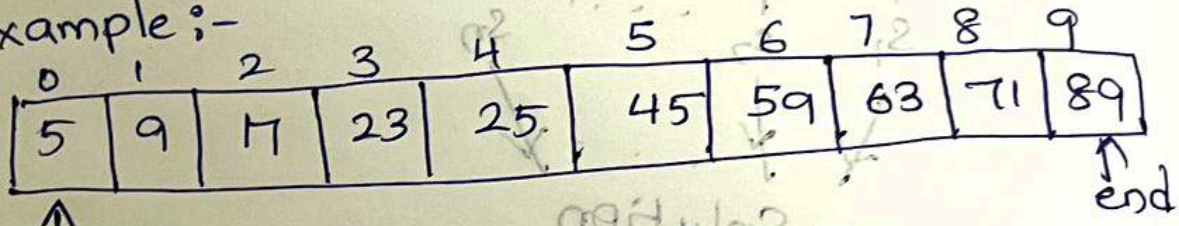
1. Binary Search
2. Quick Sort
3. Merge sort
4. Strassens matrix multiplication.

① Binary Search :-

→ Application of divide and conquer

→ Array should be soited

Example :-



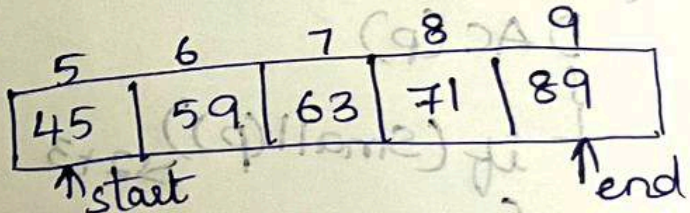
Key = 59

n = 10

$mid = \frac{start+end}{2} = \frac{0+9}{2} = 4.5$

$a[mid] = a[4] = 25$

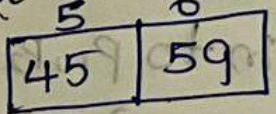
$start = 4+1 = 5$



$mid = \frac{5+9}{2} = 7$

$a[mid] = a[7] = 63$

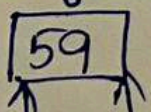
$end = mid - 1 = 7 - 1 = 6$



$mid = \frac{5+6}{2} = \frac{11}{2} = 5.5$

$mid = 5 \Rightarrow a[5] = 45$

$start = mid + 1 = 5 + 1 = 6$



$mid = \frac{6+6}{2} = 6$

$a[6] = 59$

Key = mid

return mid ;

Algorithm :-

binarysearch (a, n, key)
(a: array, n: no. of elements)

```
{
    start = 0, end = n-1;
    while (start ≤ end)
    {
        mid = (start+end)/2;
        if (key = a[mid])
            return mid;
        else if (key < a[mid])
            end = mid-1;
        else
            start = mid+1;
    }
    return -1;
}
```

key > a[mid]

start > end
unsucessful search

Example :- Key = 60 (which is not present in the array).

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89

start ↑ end ↑

mid = (0+9)/2 = 4.5 = 4

a[mid] = a[4] = 25

Key > a[mid]

start = mid+1 = 4+1 = 5

5	6	7	8	9
45	59	63	71	89

start ↑ end ↑

mid = (5+9)/2 = 14/2 = 7

a[7] = 63 Key < a[7]

end = mid-1 = 6

5	6
45	59

start ↑ end ↑

mid = (5+6)/2 = 5.5 = 5

a[5] = 45 Key > mid+1 = 5+1 = 6

6
59

start ↑ end ↑

mid = (6+6)/2 = 6

a[6] = 59

start = 6+1 = 7

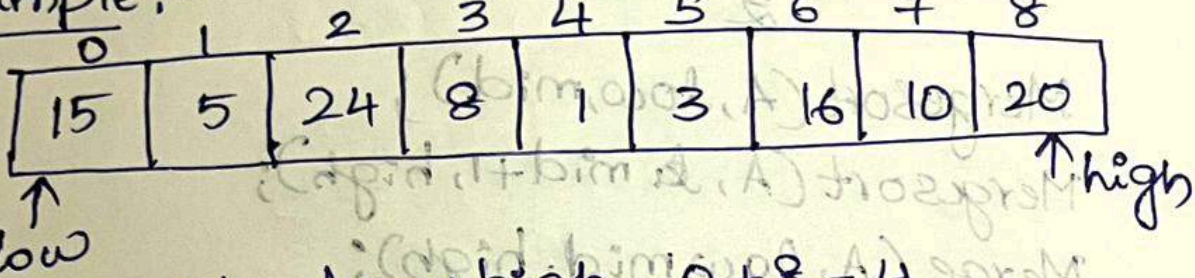
end = 6

③ Merge Sort

→ It is application of divide and conquer algorithm

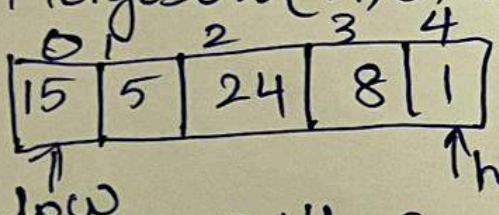
→ divide the given array into sub arrays and then merge the subarrays.
↳ Sorted manner.

Example :-



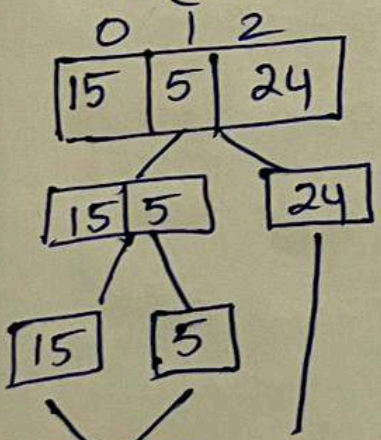
$$mid = \frac{low + high}{2} = \frac{0+8}{2} = 4$$

MergeSort(A, 0, 4)

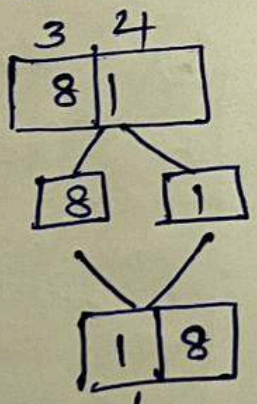


$$mid = \frac{0+4}{2} = 2$$

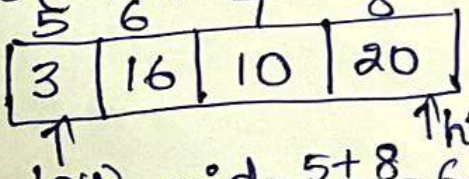
MS(0,2)



A(3,4)

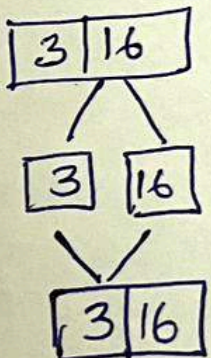


MS(A, 5, 8)

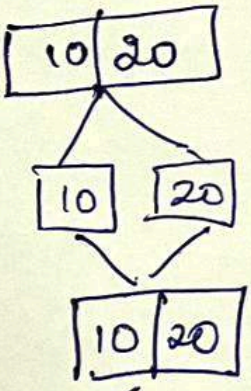


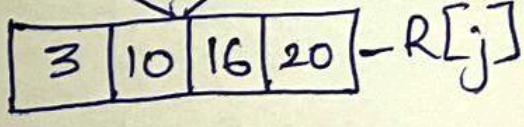
$$mid = \frac{5+8}{2} = 6$$

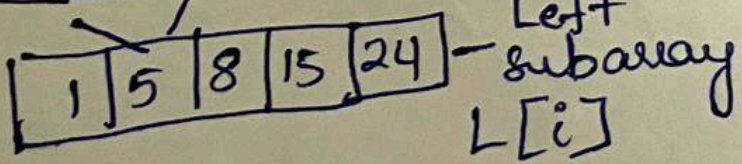
A(5,6)



A(7,8)

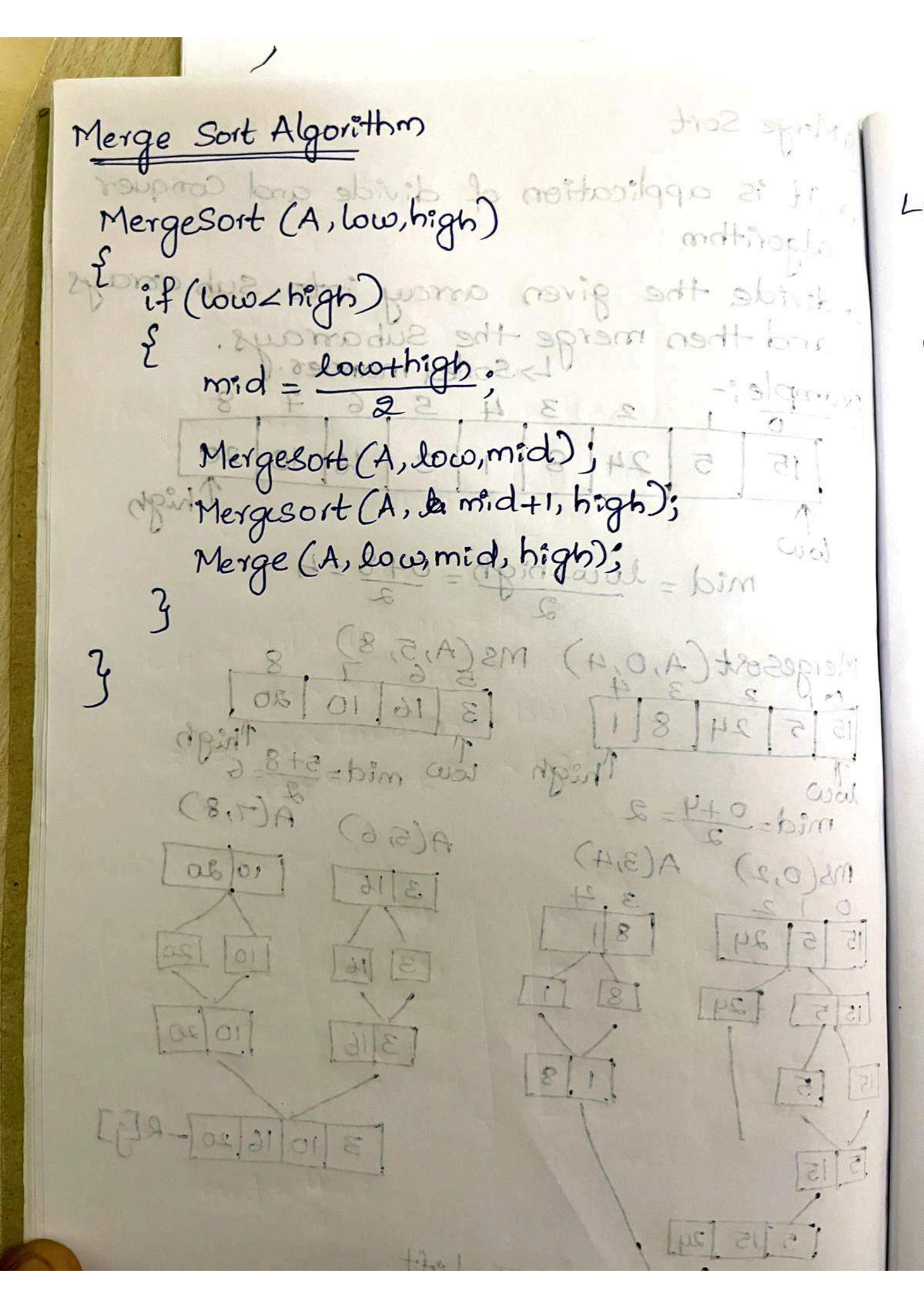






Merge Sort Algorithm

```
MergeSort (A, low, high)
{
    if (low < high)
    {
        mid = (low + high) / 2;
        Mergesort (A, low, mid);
        Mergesort (A, mid+1, high);
        Merge (A, low, mid, high);
    }
}
```



STRASSENS MATRIX MULTIPLICATION:

→ Application of divide and conquer

2×2 order — simple, (3,3)

4×4 order, 8×8 order — Multiplication

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$C = \text{Resultant} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21}$$
$$c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$$
$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$$
$$c_{22} = a_{21} * b_{12} + a_{22} * b_{22}$$

4×4 Matrix :-

$$\left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] \qquad \left[\begin{array}{cc|cc} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ \hline b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{array}\right]$$

A11 A12 A21 A22 B11 B12 B21 B22

$$mid = \frac{4}{2} = 2$$

$$mid = \frac{4}{2} = 2$$

Algorithm :-

```
Matrix_multiplication(A, B, n)
{
    if(n<=2)
    {
        c11 = a11 x b11 + a12 x b21;
        c12 = a11 x b12 + a12 x b22;
        c21 = a21 x b11 + a22 x b21;
        c22 = a21 x b12 + a22 x b22;
    }
    else    # n>2
    {
        mid = n/2;
        MM(A11, B11, n/2) + MM(A12, B21, n/2);
        MM(A11, B12, n/2) + MM(A12, B22, n/2);
        MM(A21, B11, n/2) + MM(A22, B21, n/2);
        MM(A21, B12, n/2) + MM(A22, B22, n/2);
    }
}
```

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

Strassens Formulae :- (7 formulas)

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$Q = (A_{21} + A_{22}) \cdot B_{11}$$

$$R = A_{11}(B_{12} - B_{22})$$

$$S = A_{22} \cdot (B_{21} - B_{11})$$

$$T = (A_{11} + A_{12}) \cdot B_{22}$$

$$U = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$$

$$V = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

$$C_{11} = P + S - T - V$$

$$C_{12} = R + T$$

$$C_{21} = Q + S$$

$$C_{22} = P + R - Q + U$$

Merge() function

```
Merge (A,low,mid,high)
{
    i= low, j=mid+1;
    while (i<=mid && j<=high)
    {
        if (A[i]<=A[j])
        {
            B[K]= A[i];
            i++;
            K++;
        }
        else
        {
            B[K] = A[j]
            j++;
            K++;
        }
    }
    if (i>mid)
    {
        while(j<=high)
        {
            B[K]= A[j];
            j++;
            K++;
        }
    }
    else(j>high)
    {
        while (i<=mid)
        {
            B[K]= A[i];
            i++; K++;
        }
    }
}
```

Space Complexity :-

Amount of space an algorithm requires to complete its execution is called space complexity.

$S(p) = C + SP$

$S(p) \rightarrow$ space complexity
$C \rightarrow$ Constant / independent part
$SP \rightarrow$ dependent part / variable part

Ex: algorithm for sum of two numbers

```
algorithm Sum(p,q,r)
{
    p=1;
    q=2;
    r=p+q;
}
```

p → Const - 1 unit
q → " - 1 unit
r → " → 1 unit

$S(p) = C + SP$
$= 1+1+1+0$
$S(p) = 3 \Rightarrow O(1)$
↳ Constant

② Sum of elements in an array

```
algorithm Sum(s,n)
{
    total=0;
    for i=0 to n do
        total = total+s[i];
}
```

s → array name
n → size

s → n units
n → 1 unit
total - 1 unit
i → 1 unit

$S(p) = C + SP$
$= 3 + n$
$= n \Rightarrow O(n)$

don't consider 3 only highest value with its coeff