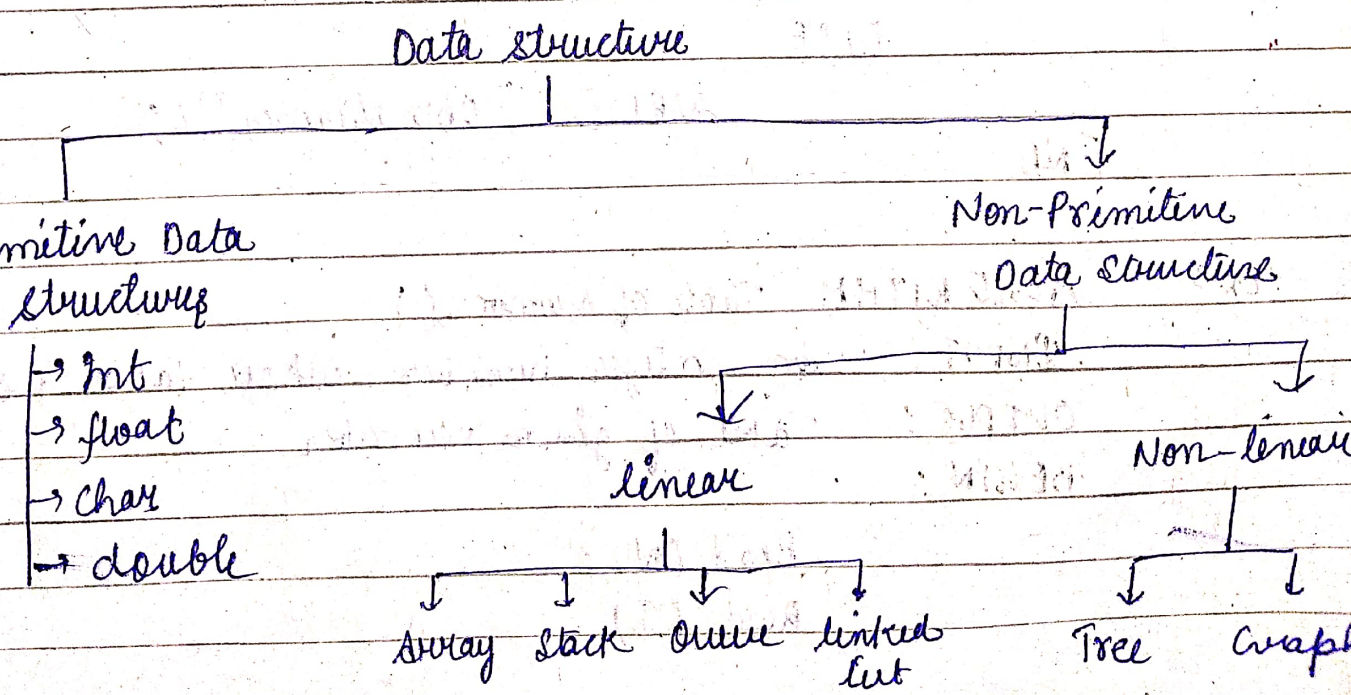
Data Structure :- It is a way of organizing or storing data into the memory so that operations on the data can be performed efficiently

Ex→ Array, stack, Queue, linked list etc.

Operation on Data Structure

1) Create
2) Insertion
3) Deletion
4) Traversing
5) Searching
6) Sorting
7) Merging

Classification of Data Structure



Pseudo code writing : Industry Pattern

ALGORITHM < Name of Algorithm > ( Argument list )
INPUT : < Description of input in words >
OUTPUT : < Description of output in words >
BEGIN

Instruction.

END

Ex→ To print Even or odd

ALGORITHM Even or odd ( )
INPUT : An Integer Number
OUTPUT: Nature of number, even or odd
BEGIN:

READ(N)
IF N%2 == 0 THEN
WRITE ("Even number");
ELSE
WRITE ("ODD Number");

END

Ex→ ALGORITHM Table of Number ( )
INPUT: An Integer number whose table we print.
OUTPUT: Table of given number
BEGIN:

Read (N)
Read (I)

```
        FOR I = 1 to 10 DO
            WRITE (N * i)
END;
```

Ex-1 Algorithm to find reverse of number.

```
ALGORITHM  Rev of number()
INPUT:     An Intiger Number
OUTPUT:    Reverse of number
BEGIN:
            Rev = 0
            Read (N)
            WHILE (N > 0) DO
                a = n % 10
                rev = rev * 10 + a
                N = N/10
            WRITE ("Reversed no. is")
            WRITE (Rev)
END;
```

a = Remaind

Ex- Algorithm to find Prime number

```
ALGORITHM  primenumber()
INPUT:     An Intiger number
OUTPUT:    To check Prime or not.
BEGIN:
            Read (N)
            For i = 1 to N DO
                (N % i == 0)
```

count ++

Complexity → Time, Space

Algorithm → Iterative, Recursive

Asymptotic Notation

Worst → Big-oh → Upper Bound → $O(n^2)$

Average → Big-theta → $\theta$ → middle Bound

Best → Big-omega → $\Omega$ → lower bound

```
for(i=1; i<=n; i++)
  {
     printf("Hello");
  }
```

⇒ 1, 2, 3, 4, 5 ⇒ $O(n)$

```
for(i=1 ; i<=n ; i=i+5)
  {
     printf("Hello");
  }
```

| iteration | 1 | 2 | 3 | 4 | ... | K |
|---|---|---|---|---|---|---|
| i | 1 | 1+5*1 | 1+5*2 | 1+5*3 | | 1+5*(K-1) |
| | | | | | | fail: K+1 |
| | | | | | | It: 5*K |

$1+5*K > n$

$K = \frac{n-1}{5}$ ← $f(n)$

complexity → $O(n)$

```
for(i=n ; i>=1 ; i=i-5)
  {
     pf("Hello");
  }
```

| iteration i | 1 | 2 | 3 | 4 | ... | K | K+1 |
|---|---|---|---|---|---|---|---|
| | n | n-5*1 | n-5*2 | n-5*3 | | n-5*(K-1) | n-5*K |

$n - 5*K < 1$

$5K = n$

$K = \frac{n}{5}$

complexity → $O(n)$

```
for(i=1 ; i<=n ; i=i*2)
  {
     pf("Hello");
  }
```

| iteration i | 1 | 2 | 3 | 4 | ... | K | fail K+1 |
|---|---|---|---|---|---|---|---|
| | 1 | 1*2 | 1*2² | 1*2³ | | 1*2^(K-1) | 1*2^K |

$1*2^K > n$

$K \log 2 > \log n$, complexity → $O(n)$

$K \log_2 2 = \log_2 n$

$K = \log_2 n$

→ `for (i=n ; i>=1 ; i=i/2)`
`{`
`pf("Hello");`
`}`

| Iteration | 1 | 2 | 3 | 4 | ... | K | K+1 |
|---|---|---|---|---|---|---|---|
| i | n | $n*2^{-1}$ | $n*2^{-2}$ | $n*2^{-3}$ | ... | $n*2^{-(K-1)}$ | $n*2^{-K}$ |

$n * 2^{-K} < 1$

$2^{-K} < n^{-1}$

$K \log 2 = \log n$

$\boxed{K = \log_2 n}$

→ `for(i=2 ; i<=n ; i=i^2`
`{`
`pf("Hello");`
`}`

| Iteration | 1 | 2 | 3 | 4 | ... | K | K+1 |
|---|---|---|---|---|---|---|---|
| i | 2 | ~~2*2~~ | ~~2*2~~ | ~~2*2~~ | ... | ~~2*2~~ | ~~2*2~~ |
| | $2^{2^0}$ | $2^{2^1}$ | $2^{2^2}$ | $2^{2^3}$ | ... | $2^{2^{(K-1)}}$ | $2^{2^K}$ |

~~$2*2^K$~~ $2^{2^K} > n$

~~$2^K = n$~~ $2^K \log_2 2 = \log n$

$2^K = \log n$

$K = (\log_2 \log_2 n)$

`for (i=n ; i>=2 ; i=√i)`
`{`
`pf("Hello");`
`}`

| iteration | 1 | 2 | 3 | 4 | ... | K+1 | K+1 |
|---|---|---|---|---|---|---|---|
| i | n | $n^{1/2}$ | $n^{1/2^2}$ | $n^{1/2^3}$ | ... | $n^{1/2^{K-1}}$ | $n^{1/2^K}$ |

$n^{1/2^K} < 2$

$\frac{1}{2^K} \log_n n = \log_2 2$

$\frac{1}{2^K} * \log n = 1$

$2^K = \log n$

$K \log_2 2 = \log_2 n$

$K = \log_2 n$

## Nested loops

→
```
for (i=1 ; i<=n ; i++)
{
  for (j=n ; j>=1 ; j=j/2)
  {
    pf("Hello");
  }
}
```

| Iteration | 1 | 2 | 3 |
|---|---|---|---|
| i | | | |
| j | | | |

→
```
for (i=1 ; i<=n ; i++)
{
  for (j=1 ; j<=n ; j*2)
  {
    pf("Hello");
  }
}
```

## Properties of Algorithm

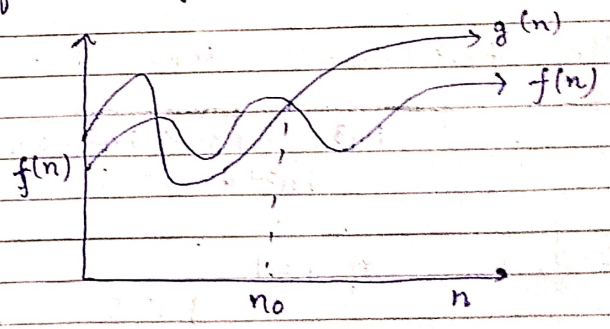
1) Input → may be zero or more
2) Output → must be 1 or more
3) Effectiveness
4) Finiteness
5) Definiteness

## * Asymptotic Notation

1) O Notation :-

Big-oh $f(n) = O \cdot g(n)$

This is true if their exist two constant C and $n_0$, such that $fn \leq C\,g(n)$ for all $n \geq n_0$ where small n may be no. of inputs or outputs. This mean that always Big O(n) will give maximum amount of time algorithm needs.



Q. $f(n) = 5n^3 + n^2 + 6n + 2$, show that $f(n) = O(n^3)$

$5n^3 + n^2 + 6n + 2 \leq 5n^3 + n^2 + 6n + 2$

$5n^3 + n^2 + 6n + 2 \leq 6n^3$

$c = 6$, if $n = 1$

$14 \leq 6$ (X)

$n = 2$

$58 \leq 48$ (X)

$n = 3$

$164 \leq 162$ (X)

$n = 4$

$362 \leq 384$ (✓)

$n = 5$

$682 \leq 750$ (✓)

$n_0 = 4$

Q. $f(n) = 4n^3 + 2n + 3$, show that $f(n) = O(n^3)$

$4n^3 + 2n + 3 \leq 4n^3 + 2n + 3$

$4n^3 + 2n + 3 \leq 5n^3$

$c = 5$, if $n = 1$

$9 \leq 5$ (X)

if $n = 2$

$39 \leq 40$

$n = 3$

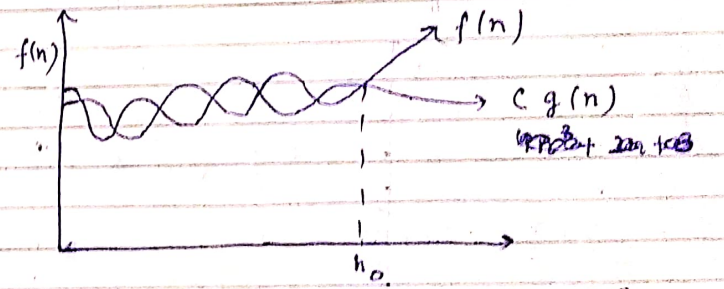
$117 \leq 135$

$c = 5$, $n_0 = 2$

* $\Omega$ Notation

$f(n) = \Omega(g(n))$ if there exist two positive constant $n$ and $n_0$ such that

$f(n) \geq c\,g(n)$ at $n_0$




which means $\Omega(n)$ will always give minimum amount of time the algorithm needs.

$f(n) = 2n + 1$

$\therefore f(n) = \Omega(n)$

$2n + 1 \geq 2n$, $n = 0$, $1 \geq 0$

$c = 2$, $n = 1$

$3 \geq 2$

$n = 2$

$5 \geq 4$

$n = 3$

$7 \geq 6$

$c = 2$, $n_0 = 0$ (Value satisfied)

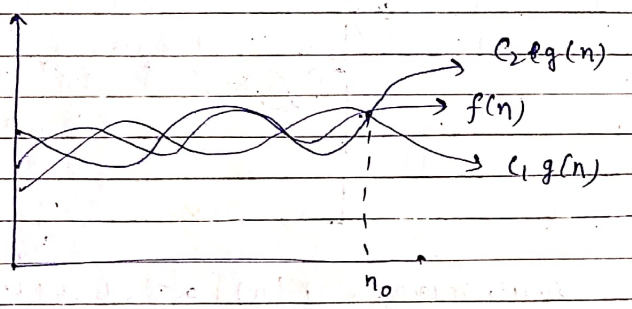
3. Θ Notation :-

Big - Theta

$f(n) = \Theta g(n)$ if and only if there exist +ve constant $c_1, c_2$ and $n_0$ for all $n \geq n_0$

$$f(n) = \Theta\ c g(n)$$

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




$f(n) = 3n + 5$

$4n \geq 3n+5 \geq 3n$

$c_2 = 4$, $c_1 = 3$

| for $n = 1$ | $n = 1$ |
|---|---|
| $4 \geq 8$ | $8 \geq 3$ |
| $n = 2$ | $n = 2$ |
| $8 \geq 11$ | $11 \geq 6$ |

$n = 0$

$5 \geq 0$

| $n = 3$ | $n = 3$ |
|---|---|
| $12 \geq 14$ | |
| $n = 4$ | |
| $16 \geq 17$ | |
| $n = 5$ | |
| $20 \geq 20$ | |
| $n = 6$ | |
| $24 \geq 23$ | |
| $n = 7$ | |
| $28 \geq 26$ | |

$n_0 = 5$

Array :-

→ Collection of similar type of data

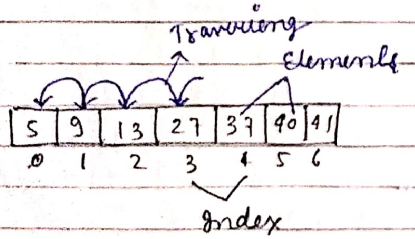
✓ Length of Array → Upper Bound - Lower bound + 1

Primitive operations in an array

a) Traversing → Visiting every value of array at least once.

```
Traverse (A, n)
    for i 1 to n DO
        Print A[i]
END;
```

```
Traverse (A, n)
1) Set I = 1
2) Repeat step 3 until
       while (I <= n)
3) write A[i]
4) Set I = I+1
5) Exit
```



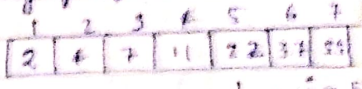

Time complexity → O(N)
Space complexity → O(1)

b) Insertion

→ To insert a data element in an array

→ A new element can be added at beginning, end, or at any given index based on the requirement

Ex:- Insert an element (key = 15) at specific index (i = 5) in array of size 6




↳ i = 5, key = 15

Algorithm

```
Insert_Array (A[ ], n, ele, pos)
1) Set I = n    // where n is the no. of elements
2) Repeat steps 3, 4 while (I >= pos)
3) Set A[i+1] = A[i]
4) Set I = I-1
   End of while
5) Set A[pos] = ele
6) Set n = n+1

→ for j = n to i step -1 DO } → shifting of element
      A[j+1] = A[j]

  A[i] = key  → placing at i index
  N = N+1
END
```

# Algorithm to delete an element from array.

Delete_An array (A[], pos, n)
1) Set I = Pos
2) Repeat steps 3 to 4 while (I < N)
3) Set A[I] = A[I+1]
4) Set I = I+1
   End of while
5) Set N = N-1
6) Exit

## Index Formula Derivation for Array

### Memory Representation of linear Array

| | 1000 | 1002 | 1004 | 1006 | 1008 | 1010 | 1012 | 1014 |
|---|---|---|---|---|---|---|---|---|
| marks[ ] = { | 99, | 88, | 27, | 28, | 88, | 92, | 42, | 58 } |
| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] |

Address (marks[4]) = ?

= Base address + w(K - lower Bound)

w = words per mem
K = Index

Derivation

Suppose there is one-dimensional A[L:U]


No. of elements / length of the array can be found by the formula → N = U - L + 1

U = Upper Bound (last index)
L = lower Bound (First index)

To find address of $i^{th}$ index element, we will take two assumptions

- Assumption 1 : 1 Byte storage for each element
- Assumption 2 : Index starts from 1.

Base Address of an array = α

A[1] = α // address of 1st element of array
A[2] = α + 1 // address of 2nd element [B.A + no. of bytes for 1st element]
A[3] = α + 2 // address of 3rd element [Address of 2nd element + no. of bytes for 1st]
A[4] = α + 3 // address of 4th element of array is address of 3rd element + no. of bytes for 3rd
⋮
A[i] = α + (i-1) → (1)

Step (2): Removal of first assumption i.e, 1 Byte storage for each element

→ An element can take 'n' no. of bytes depending upon datatype

→ A[i] = α + (i-1) → (eqn 1)
→ A[i] = α + (i-1) * n → (eqn 2)

General of 2nd assumption i.e. first element is at index

→ If index starts from L then, for some $i^{th}$ index element distance from first index will be $i - L + 1$.


Distance from i to L = $i - L + 1$

→ Address of $A[i] = \alpha + (i-1) * n$

Replacing i with $i - L + 1$

$A[i] = \alpha + [(i - L + 1) - 1] * n$

Address of $A[i] = \alpha + (i - L) * n$

Address of $A[i]$ = Base Address + $n * (i - \text{Lower Bound})$

Q Given $A[1:10]$, bytes per cell = 4
Base address = 2000
find address of $A[7]$

$n = 4$
Base Address = 2000 ($\alpha$)
Lower Bound = 1 - 1

Address of $A[7] = 2000 + 4 * (7 - (-1))$
$= 2000 + 4 * 8$
$= 2032$

Q Given $A[1:15]$, bytes per cell = 3
Base address = 1000
$A[9] = ?$

$\alpha = 1000$ $n = 3$
$L = 1$
$i = 9$

Address of $A[9] = 1000 + 3 * (9 - 1)$
$= 1024$

Consider the linear array $A[5:50]$, $B[-5:10]$, $C[10]$
a) Find the no. of elements in each array
b) Suppose Base $[A] = 300$, $w = 4$ for A
Find address of $A[15]$, $A[40]$ and $A[55]$

a) elements of Array $[A]$ = Upper Bound − Lower Bound + 1
$= 50 - 5 + 1$
$= 46$

Array $[B] = 10 - (-5) + 1$
$= 16$

Array $[C] = 10 - 0 + 1$
$= 11$

(b) $A[15] = 300 + 4 * (15 - 5)$
$= 340$

$A[40] = 300 + 4*(40-5)$
$= 300 + 4 \times 35$
$= 8440$

$A[55]$ = Array out of Bound
or
out of Index.

Two-Dimensional Array

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | A[1,1] | A[1,2] | A[1,3] | A[1,4] |
| 2 | A[2,1] | A[2,2] | A[2,3] | A[2,4] |
| 3 | A[3,1] | A[3,2] | A[3,3] | A[3,4] |

$A[1:10, 5:15]$
$A[10, 11]$
$M \times N$

* Row-Major order

| A[1,1] | A[1,2] | A[1,3] | . . . . . | A[3,3] | A[3,4] |
|---|---|---|---|---|---|

$A[J,K] = \text{Base address} + w(N(J - LB) + (K - LB))$
J: row, K: column

* Column-major order

| |
|---|
| A[1,1] |
| A[2,1] |
| A[3,1] |
| : |
| : |
| : |
| A[2,4] |
| A[3,4] |

$A[J,K] = B.A + w[(J - LB) + M*(K - LB)]$
J: row, K: column

Q Suppose a 2D Array A is declared as $A[-2:2, 2:6]$ words per cell = 4, Base address = 200
Find
a) no. of element, length of each dimension
b) Find address of $A[1,2]$ → for both row major and column major

$A[-2:2, 2:6]$

$A[5, 5]$
M N

a) no. of element = 25
length of row = 5
no. of column = 5

b) Row major

$A[1,2] = B.A + w[N(J - LB) + (K - LB)]$
$= 200 + 4[5(1 - (-2)) + (2 - 2)]$
$= 200 + 4[15]$
$= 260$

Column major
$A[1,2] = 200 + 4[(1 - (-2)) + 5*(2-2)]$
$= 212$

Q Calculate the address of $X[4,3]$ in a 2D array $X[1..5, 1..4]$, stored in a row major order. Assume $B.A = 1000$ and that each element required 4 words of (w)

X[1:5, 1:4]  , X[4, 3]

X[5, 4]
M N

→ Row major

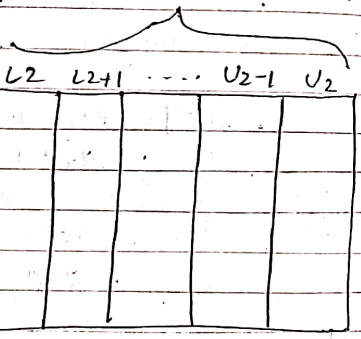
X[4,3] = 1000 + 4(4(4-1) + (3-1))
= 1000 + 4(12 + 2)
= 1000 + 56
= 1056

## * Row major order Arrangement

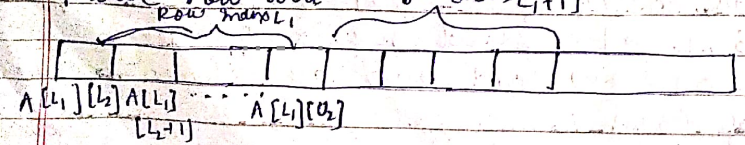
→ Suppose we have a 2D array A[L1:U1, L2:U2]

→ Row side indices are L1, L1+1, L1+2 .... U1-1, U1.

→ Column side indices are L2, L2+1, L2+2 .... U2-1, U2.




→ In a memory it will look like 1-D array as it will be stored row-wise




→ For simplicity we assume that first index is 1 and each element requires 1 byte for storage. The array becomes

[

Address of A[1,1] = α
" of A[1,2] = α + 1
" of A[1,3] = α + 2
" of A[1,U2] = α + (U2 - 1)
" of A[2,1] = α + (U2 - 1) + 1
= α + U2
" A[3,1] = α + 2U2
⋮
" A[i,1] = α + (i-1) * U2
" A[i,2] = α + (i-1) * U2 + 1

Similarly A[i,j] = α + [(i-1) * U2 + (j-1)]

Let us remove assumption that every element takes 1 byte storage with n bytes storage

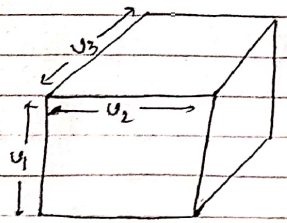
Address of A[i,j] = α + [(i-1) * U2 + (j-1)] * n

Replace 1 by L1 & L2

✓ Address of A[i,j] = α + [(i-L1) * (U2-L2+1) + (j-L2+1)] * n

A[i,j] = α + [(i-L1) * (U2-L2+1) + (j-L2)] * n

## 3-Dimensional Array (Row Major Order)




Array is : $A[L_1:U_1, L_2:U_2, L_3:U_3]$

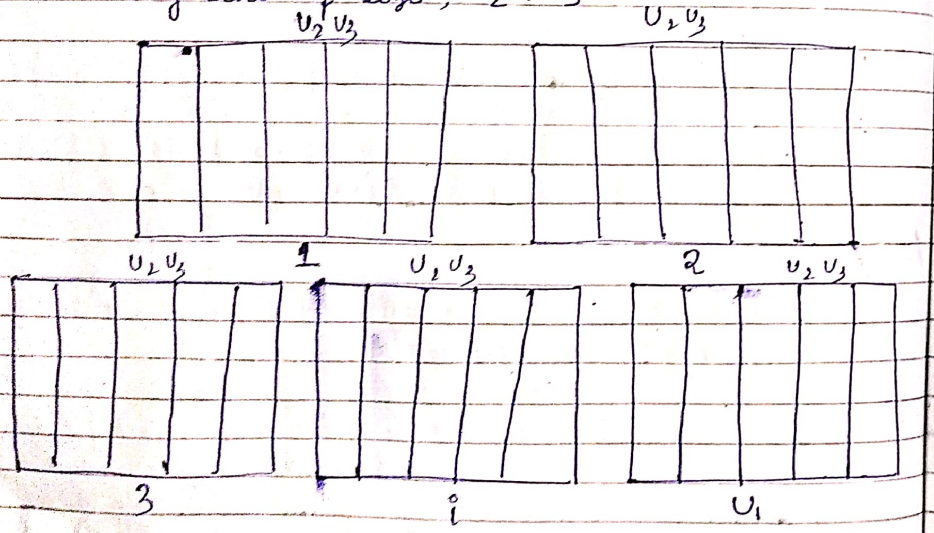
Assumption

Let say first index is 1,

So, $A[1:U_1, 1:U_2, 1:U_3]$

total no. of slice can be cut upto $U_1$.

Every slice of size, $U_2 * U_3$




Address of 1st element

$A[1,1,1] = \alpha$

$A[2,1,1] = \alpha + U_2 * U_3$

$A[3,1,1] = \alpha + 2\,U_2 * U_3$

$A[i,1,1] = \alpha + (i-1)\,U_2 * U_3$

Let us expand i array: we already know the address of 1st element, what will be the address of 1st element of 2nd row, it will be address of 1st row 1st element + $U_3$

$A[i,2,1] = \alpha + (i-1)\,U_2 * U_3 + U_3$

$A[i,3,1] = \alpha + (i-1)\,U_2 * U_3 + 2U_3$

Similarly address of 1st element of $j^{th}$ row.

$A[i,j,1] = \alpha + (i-1)\,U_2 * U_3 + (j-1)\,U_3$

$A[i,j,K] = \alpha + (i-1)\,U_2 * U_3 + (j-1)\,U_3 + (K-1)$

Now remove assumptions, then i is on 1st dimension side, j is for 2nd dimension & K for 3rd dimension and every element is acquiring n byte for storage

$A[i,j,K] = \alpha + [(i-1)\,U_2 * U_3 + (j-1)\,U_3 + (K-1)] * n$

i will be replaced $i-L_1+1$, j by $j-L_2+1$, K by $K-L_3+1$

$$\boxed{A[i,j,K] = \alpha + [(i-L_1)\,U_2 * U_3 + (j-L_2)\,U_3 + (K-L_3)] * n}$$

$A[i,j,K] = \alpha + [(i-L_1)(U_2-L_2+1)(U_3-L_3+1) + (j-L_2)(U_3-L_3+1) + (K-L_3)] * w$

So, then replacing, $i-L_1 = E_1$

$j-L_2 = E_2$

$K-L_3 = E_3$

$\alpha$ = Base Address

$U_2 - L_2 + 1 = L_2$

$U_3 - L_3 + 1 = L_3$

$n = w$

$A[i,j,K] =$ Base Address $+ [((E_1 L_2 + E_2) L_3 + E_3)] * w$

$A[m_1,m_2,m_3,m_4,m_5] = \alpha + [(m_1-L_1)(U_2-L_2+1)*(U_3-L_3+1)*(U_4-L_4+1)*(U_5-L_4+1) + (m_2-L_2)(U_3-L_3+1)*(U_4-L_4+1)(U_5-L_5+1) + (m_3-L_3)(U_4-L_4+1)(U_5-L_5+1) + (m_4-L_4)(U_5-L_5+1) + (m_5-L_5)] * n$

$A[m_1,m_2,m_3,m_4,m_5] =$ Base Address $+ [E_1 L_2 L_3 L_4 L_5 + E_2 L_3 L_4 L_5 + E_3 L_4 L_5 + E_5 L_5 + E_6] * w$

Column major Array 3D

$A[i,j,K] = B.A + w[(K-L_3)(U_2-L_2+1)(U_1-L_1+1) + (j-L_2)(U_1-L_1+1) + [i-L_1]]$

Q. Given a 3D array $A[2:8, -4:1, 6:10]$, B.A = 200

Words per cell = 4

Calculate address of $A[5,-1,8]$.

a) Row major order

b) Column major order

a) $\alpha = 200$, $i=5$, $j=-1$, $K=8$

$L_1 = 2$ $U_1 = 8$

$L_2 = -4$ $U_2 = 1$ $w = 4$

$L_3 = 6$ $U_3 = 10$

$A[5,-1,8] = 200 + [(5-2)(1+4+1)(10-6+1) + (-1+4)(10-6+1) + (8-6)] * 4$

$= 200 + [3*6*5 + 3*5 + 2] * 4$

$= 200 + [90 + 15 + 2] * 4$

$= 200 + 428$

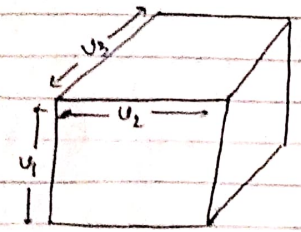
$= 628$

b) $A[5,-1,8] = 200 + 4[2*6*7 + 3*7 + 3]$

$= 200 + 4[84 + 21 + 3]$

$= 632$

## 3-Dimensional Array (Row Major Order)




Array is : $A[L_1:U_1, L_2:U_2, L_3:U_3]$

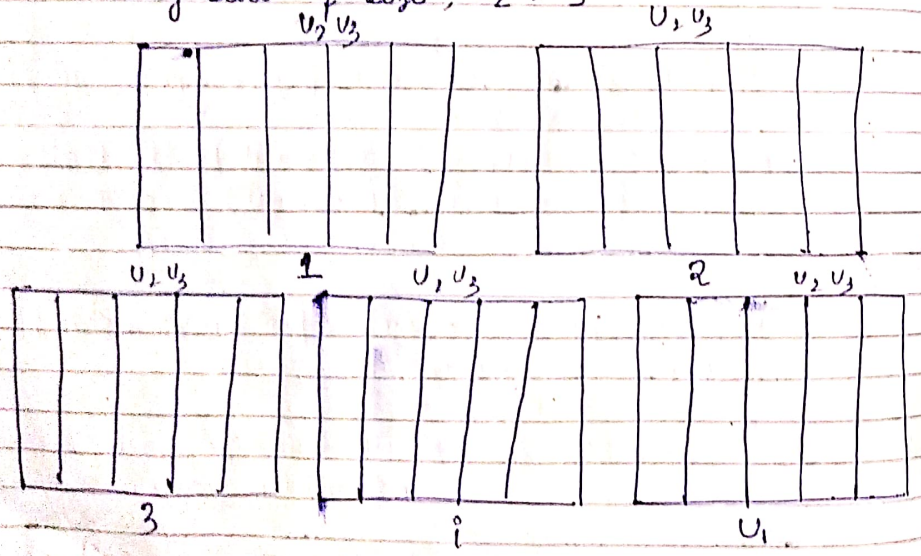
Assumption

Let say first index is 1.

So, $A[1:U_1, 1:U_2, 1:U_3]$

total no. of slice can be cut upto $U_1$.

Every slice of size, $U_2 * U_3$




Address of 1st element $\sim$ $A[1,1,1] = \alpha$

$A[2,1,1] = \alpha + U_2 * U_3$

$A[3,1,1] = \alpha + 2\ U_2 * U_3$

$A[i,1,1] = \alpha + (i-1)\ U_2 * U_3$

Let us expand i array: we already know the address of 1st element, what will be the address of 1st element of 2nd row, it will be address of 1st row 1st element + $U_3$

$A[i,2,1] = \alpha + (i-1)\ U_2 * U_3 + U_3$

$A[i,3,1] = \alpha + (i-1)\ U_2 * U_3 + 2U_3$

Similarly address of 1st element of $j^{th}$ row

$A[i,j,1] = \alpha + (i-1)\ U_2 * U_3 + (j-1)\ U_3$

$A[i,j,K] = \alpha + (i-1)\ U_2 * U_3 + (j-1)\ U_3 + (K-1)$

Now remove assumptions, then i is on 1st dimension side, j is for 2nd dimension & K for 3rd dimension and every element is acquiring n byte for storage.

$A[i,j,K] = \alpha + [(i-1)\ U_2 * U_3 + (j-1)\ U_3 + (K-1)] * n$

i will be replaced $i-L_1+1$, j by $j-L_2+1$, K by $K-L_3+1$

$$\boxed{A[i,j,K] = \alpha + [(i-L_1)\ U_2 * U_3 + (j-L_2)\ U_3 + (K-L_3)] * n}$$

$$A[i,j,K] = \alpha + [(i-L_1)(U_2-L_2+1)(U_3-L_3+1) + (j-L_2)(U_3-L_3+1) + (K-L_3)]*w$$

So, then replacing, $i-L_1 = E_1$

$j-L_2 = E_2$

$K-L_3 = E_3$

$\alpha$ = Base Address

$U_2-L_2+1 = L_2$

$U_3-L_3+1 = L_3$

$n = w$

$A[i,j,K]$ = Base Address + $[((E_1 L_2 + E_2) L_3 + E_3)] * w$

$A[m_1,m_2,m_3,m_4,m_5] = \alpha + [(m_1-L_1)(U_2-L_2+1)*(U_3-L_3+1)*(U_4-L_4+1)*(U_5-L_4+1) + (m_2-L_2)(U_3-L_3+1)*(U_4-L_4+1)(U_5-L_5+1) + (m_3-L_3)(U_4-L_4+1)(U_5-L_5+1) + (m_4-L_4)(U_5-L_5+1) + (m_5-L_5)] * n$

$A[m_1,m_2,m_3,m_4,m_5]$ = Base Address + $[E_1 L_2 L_3 L_4 L_5 + E_2 L_3 L_4 L_5 + E_3 L_4 L_5 + E_5 L_5 + E_6] * w$

Column major Array 3D

$A[i,j,K] = B.A + w[(K-L_3)(U_2-L_2+1)(U_1-L_1+1) + (j-L_2)(U_1-L_1+1) + (i-L_1)]$

Q. Given a 3D array $A[2:8, -4:1, 6:10]$, B.A = 200

Words per cell = 4

Calculate address of $A[5,-1,8]$.

a) Row major order

b) Column major order

a) $\alpha = 200$, $i=5$, $j=-1$, $K=8$

$L_1 = 2$ $U_1 = 8$

$L_2 = -4$ $U_2 = 1$ $w = 4$

$L_3 = 6$ $U_3 = 10$

$A[5,-1,8] = 200 + [(5-2)(1+4+1)(10-6+1) + (-1+4)(10-6+1) + (8-6)] * 4$

$= 200 + [3*6*5 + 3*5 + 2] * 4$

$= 200 + [90 + 15 + 2] * 4$

$= 200 + 428$

$= 628$

b) $A[5,-1,8] = 200 + 4[2*6*7 + 3*7 + 3]$

$= 200 + 4[84 + 21 + 3]$

$= 632$

* Linear Search

linearSearch ( arr[ ], n, key )

1. set i=1, loc=0
2. Repeat steps 3 to 4 while(i<=n)
3. if A[i] == key
   loc = i;
   break;
4. else
   i++
5. if loc = 0
   Print : key not found
6. Else
   Key found at loc.
7. Exit.

* Binary Search

Binary Search (arr, n, key)

① Set LB=1, UB=n, loc=0, mid = $\left[\frac{UB+LB}{2}\right]$

② Repeat step ③, ④, ⑤, ⑥ while (LB ≤ UB)

③ if A[mid] == key
   loc = mid
   break;

④ else if
   key > A[mid]
   set LB = mid+1

⑤ else
   set UB = mid-1

⑥ mid = $\left[\frac{LB+UB}{2}\right]$ (End of while)

⑦ if loc = 0
   print ( Element not found)

⑧ Else
   print (Element found)

⑨ Exit

* Hashing

H : K ⟶ L

Hash function

- Division method
- mid square method
- folding method

$H^{(K)} = K \bmod m$

m = largest prime number of length of element

indexing from 1
then $H^{(K)} = K \bmod m + 1$

Ex:- 21, 35, 40, 18, 32

$H : K \rightarrow L$

$m = 5$,

$H^{(K)} = K \bmod m$
$= 21 \bmod 5 = 1$
$= 35 \bmod 5 = 0$
$= 40 \bmod 5 = 80$ → if collision happen we do linear probing
$= 18 \bmod 5 = 3$
$= 32 \bmod 5 = 2$ (// linear probing)

| | |
|---|---|
| 35 | 0 |
| 21 | 1 |
| 40 | 2 |
| 18 | 3 |
| 32 | 4 |
| | 5 |
| | 6 |

2 Suppose a class consist of 68 students assign a 4 digit student no. Suppose L (address of array) consist of 100. Apply a Hash func$^n$ to each of the 3205, 7148, 2345

3205, 7148, 2345

$H^{(K)} = K \bmod n$
$= 3205 \bmod 97 = 04$

| | |
|---|---|
| | 00 |
| | |
| 3205 | 04 |
| | |
| | |
| | |
| 2345 | 17 |
| | |
| | |
| 7148 | 67 |

Q Consider inserting the key 26, 37, 59, 76, 65, 86 into a hash table of size $m = 11$ using linear probing, consider the primary hash func$^n$ $H^{(K)} = K \bmod m$

$H^{(K)} = K \bmod n$
for $(m = 7)$
26, 37, 59, 76, 65, 86

$= 26 \bmod 7 = 5$
$= 37 \bmod 7 = 2$
$= 59 \bmod 7 = 4$
$= 76 \bmod 7 = 6$
$= 65 \bmod 7 = 2$ (// Linear probing)
$= 86 \bmod 7 = 2$

| | |
|---|---|
| | 0 |
| | 1 |
| 37 | 2 |
| 65 | 3 |
| 59 | 4 |
| 26 | 5 |
| 76 | 6 |
| 86 | 7 |
| | 8 |
| | 9 |
| | 10 |

for $(m = 11)$
26, 37, 59, 76, 65, 86

$= 26 \bmod 11 = 4$
$= 37 \bmod 11 = 4$
$= 59 \bmod 11 = 4$
$= 76 \bmod 11 = 10$
$= 65 \bmod 11 = 10$
$= 86 \bmod 11 = 9$

| | |
|---|---|
| 65 | 0 |
| | 1 |
| | 2 |
| | 3 |
| 26 | 4 |
| 37 | 5 |
| 59 | 6 |
| | 7 |
| | 8 |
| 86 | 9 |
| 76 | 10 |

* Mid square method

3205, 7148, 2345

(1) Square the number and take the middle no.

$(3205)^2 = 10272025$ Address → 72

$(7148)^2 = 51093904$ 93

$(2345)^2 = 5499025$ 99

* Folding Method

keys: 3205, 7148, 2345

(1) Break into two parts and add the numbers.

32 + 05 = 37
71 + 48 = 119 → discard either left or right digit
23 + 45 = 68

Remote Folding :- In this we break in two parts and then add reverse the no. and do operation same as Folding method.

123456789 → Address Rang: 000 - 999
we break it into three parts 123, 456, 789

* Collision Resolution Technique
Open addressing → linear Probing $(v+i)\%m$

Q Key = 3, 2, 9, 6, 11, 13, 7, 12

$h(K) = 2K + 3$
$m = 10$

Use division method and open addressing to store these values

| | |
|---|---|
| 0 | 9 13 |
| 1 | 9 |
| 2 | 12 |
| 3 | |
| 4 | |
| 5 | 6 |
| 6 | 11 |
| 7 | 7 |
| 8 | 7 |
| 9 | 3 |

| Key | location (x) | Probe |
|---|---|---|
| 3 | $(2*3+3)\%10 = 9$ | 1 |
| 2 | $(2*3+3)\%10 = 7$ | 1 |
| 9 | $(9*2+3)\%10 = 1$ | 1 |
| 6 | $(6*2+3)\%10 = 5$ | 1 |
| 11 | $(11*2+3)\%10 = 5$ | 2 |
| 13 | $(13*2+3)\%10 = 9$ | 2 |
| 7 | $(7*2+3)\%10 = 7$ | 2 |
| 12 | $(12*2+3)\%10 = 7$ | 6 |

(11) $(5+0)\%10 = 5$
$(5+1)\%10 = 6$

(13) $(9+0)\%10 = 9$
$(9+1)\%10 = 0$

(7) $(7+0)\%10 = 7$
$(7+1)\%10 = 8$

(12) $(7+0)\%10 = 7$
$(7+1)\%10 = 8$
$(7+2)\%10 = 9$
$(7+3)\%10 = 0$
$(7+4)\%10 = 1$
$(7+5)\%10 = 2$

avg. successful probe = $\frac{1+1+1+1+2+2+2+6}{8}$

$\Rightarrow 2$

avg. unsuccessful probe $\frac{9+3+2+1+1+9+8+7+6+5}{10}$

$\Rightarrow 4.6$

Q Use division method and quadratic probing to

Key : 3, 2, 9, 6, 11, 13, 7, 12

$h(K) = 2K+3$ , $m = 10$

| | |
|---|---|
| 0 | 13 |
| 1 | 9 |
| 2 | |
| 3 | 12 |
| 4 | |
| 5 | 6 |
| 6 | 11 |
| 7 | 2 |
| 8 | 7 |
| 9 | 3 |

| Key | location | Probes |
|---|---|---|
| 3 | $(2\times3+3)\%10 = 9$ | 1 |
| 2 | $(2\times2+3)\%10 = 7$ | 1 |
| 9 | $(2\times9+3)\%10 = 1$ | 1 |
| 6 | $(2\times6+3)\%10 = 5$ | 1 |
| 11 | $(2\times11+3)\%10 = 5$ | 2 |
| 13 | $(2\times13+3)\%10 = 9$ | 2 |
| 7 | $(2\times7+3)\%10 = 7$ | 2 |
| 12 | $(2\times12+3)\%10 = 7$ | 7 |

(11) $(5+0)\%10 = 5$
$(5+1)\%10 = 6$

(13) $(9+0)\%10 = 9$
$(9+1)\%10 = 0$

(7) $(7+0)\%10 = 7$
$(7+1)\%10 = 8$

(12) $(7+0)\%10 = 7$
$(7+1)\%10 = 8$
$(7+4)\%10 = 1$
$(7+9)\%10 = 6$
$(7+16)\%10 = 3$

Q Key : 3, 2, 9, 6, 11, 13, 7, 12

$h(K) = 2K+3$ , $m = 10$

Use division method and double Hashing technique to insert this element where $h_2(K) = 3K+1$, insert Ki at first free cell location $(v + v*i)\%m$

$v = h_2(k)\%m$

Key location (Li) Probes v

Q Suppose the Table T has 11 memory location T[1], T[2], T[3] ..... T[11]
and suppose the
~~use division method~~ : A, B, C, D, E, X, Y, Z
H(K) : 4 8 2 11 4 11 5 1

| X | C | Z | A | E | Y | | B | | | D |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

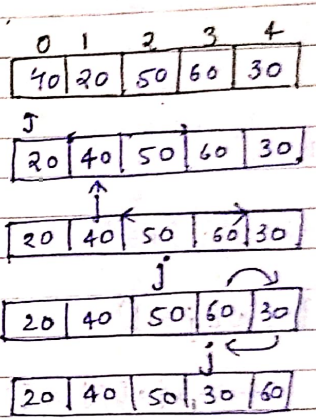
avg. successful Probe : $\frac{1+1+1+1+2+2+2+3}{8}$

$= \frac{13}{8}$

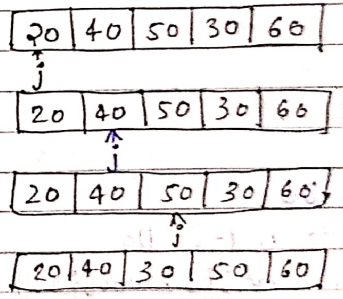
avg. unsuccessful Probe = $\frac{7+6+5+4+3+2+1+2+1+1+1}{11}$.

* Bubble sorting

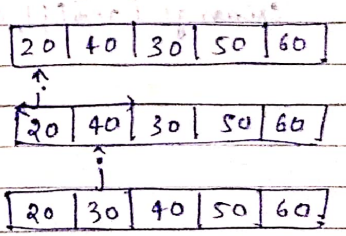
Pass 1
i=0




Pass 2
i=1




Pass 3
i=2



Pass 4

i=3

| 20 | 30 | 40 | 50 | 60 |
|---|---|---|---|---|

no exchange

```
for(i=0 ; i<n-1 ; i++)
{
    for(j=0 ; j<n-1-i ; j++)
    {
        if (a[j] > a[j+1])
        {
            swap(a[j], a[j+1]);
        }
    }
}
```

```
Bubble_Sort (arr, n)
Begin:
    FOR i=1 to n-1 DO
        FOR j=1 to n-1-i Do
            IF arr[j] > arr[j+1]
                exchange (arr[j], arr[j+1])
End:
```

Recursion: - It is defined as defining anything in terms of itself.

There are two Basic properties of Recursion.

① Base Call
② Recursive Procedure

→ There must be a certain criteria called Base Criteria for which the func$^n$ does not call itself

→ Each time the procedure call itself (directly or indirectly) it must be closure to the base criteria

Types of Recursion

① Direct Recursion →

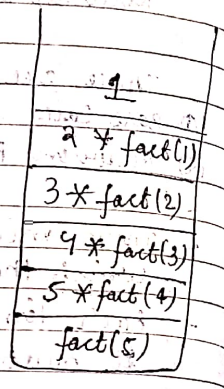
```
abc()
{
    ...
    abc();
}
```

② Indirect Recursion

```
abc()
{
    ...
    xyz();
}
xyz()
{
    ...
    abc();
}
```

Factorial of a no. using Recursion

```
int factorial (int n)
{
    if (n==1)    //n==0
        return 1;
    else
        return (n * factorial(n-1));
}
```




factorial(5) = 120

5 * factorial(4)

4 * factorial(3)

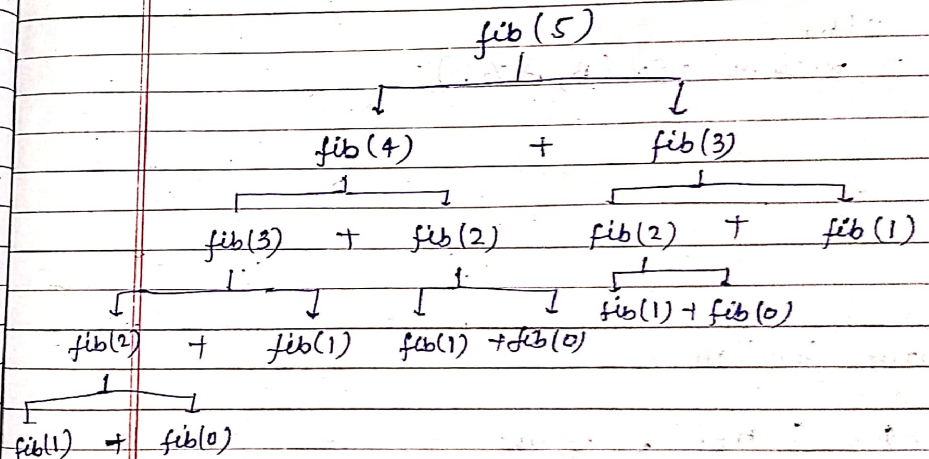
3 * factorial(2)

2 * factorial(1)

* fibonacci ~~series~~ Term

n^th fibonacci Term

```
int fib(int n)
{
    if (n==0)
        return 0;
    else if (n==2)
        return 1;
    else
        return (fib(n-1) + fib(n-2));
}
```



* $GCD(A,B) = \begin{cases} GCD(B,A), & \text{if } A<B \\ A, & \text{if } B=0 \\ GCD(B, A\%B), & \text{if } A \geq B \end{cases}$

Find GCD(50, 20) GCD (20, 25)

```
int gcd( int A , int B)
{
    if (B==0)
        Return A;
    else
        return gcd (B, A%B);
}
```

* Ackermann function

$$A(m,n) = \begin{cases} n+1, & \text{if } m=0 \\ A(m-1,1), & \text{if } m \neq 0 \text{ but } n=0 \\ A[m-1, A(m,n-1)], & \text{if } m \neq 0 \text{ and } n \neq 0 \end{cases}$$

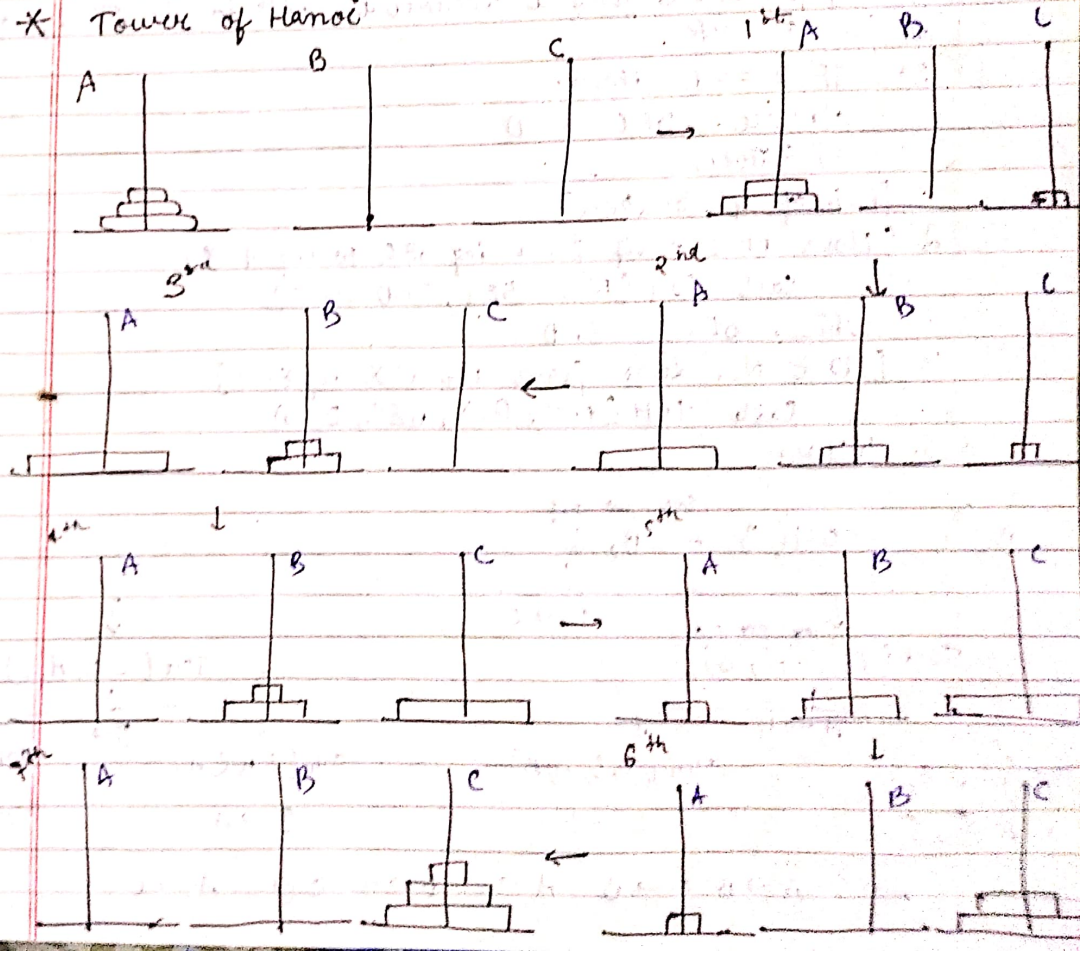
Solve A(1,3)

1) A(1,3) = A(0, A(1,2)) => A(0, A(0, A(1,1)))

A(0, A(0, A(0, A(0,1)))) <= A(0, A(0, A(0, A(1,0))))

A(0, A(0, A(0, A(0,2)))) => A(0, A(0,3)) => A(0,4) = 5

* Tower of Hanoi



function → $2^n - 1$ , complexity → $O(2^n)$

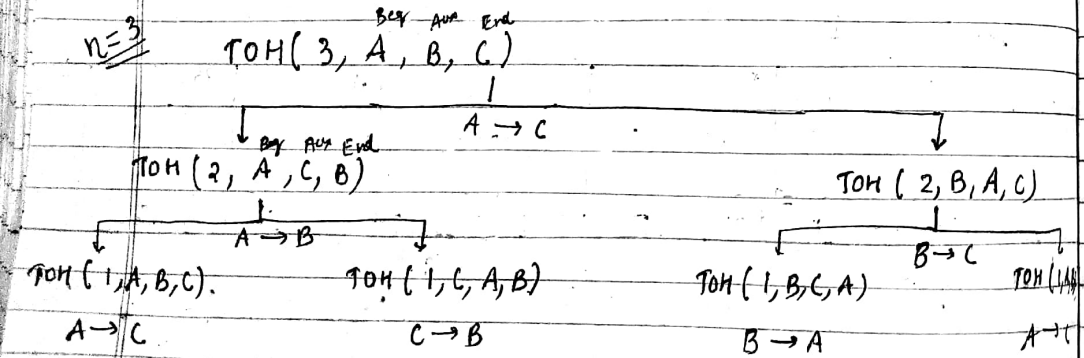
Algorithm

TOH ( N, BEG, AUX, END)

// This procedure gives a recursive sol$^n$ to the TOH puzzle for n disk.

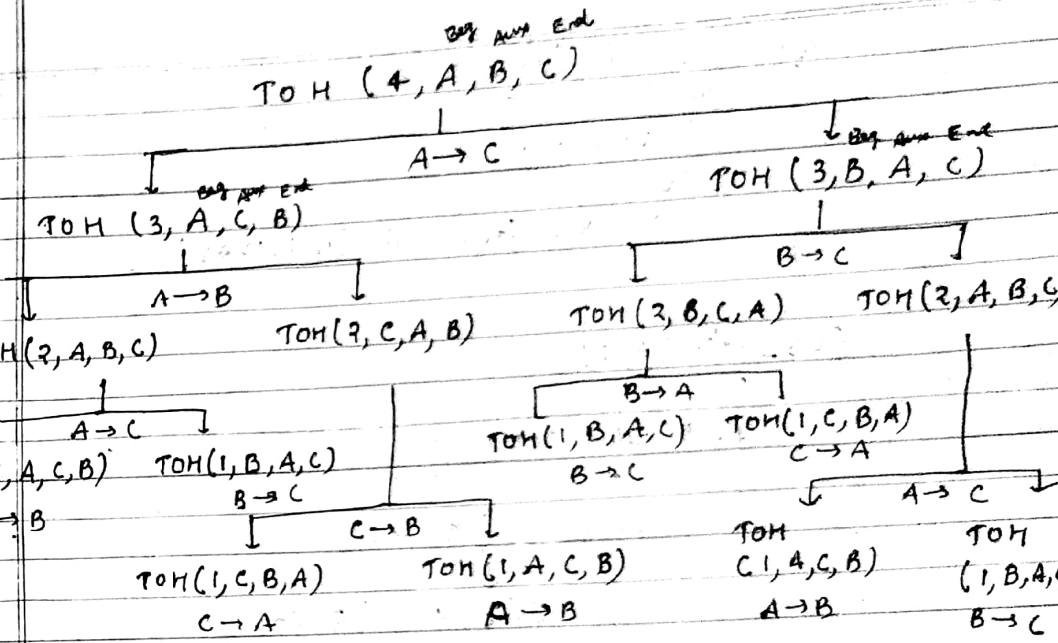
1. IF N == 1 then:
   a) write : BEG - END
   b) Return
   [End of if structure]
2. [Move N-1 disks from Peg BEG to Peg AUX]
   Call TOH (N-1, BEG, END, AUX)
3. Write : BEG → END
4. [MOVE N-1 disks from Peg AUX, Peg END]
   Call TOH (N-1, AUX, BEG, END)
5. Return

n=3




A→C A→B C→B A→C B→A B→C A→C

n=4




A→B, A→C, B→C, A→B, C→A, C→B, A→B, A→C
B→C, B→A, C→A, B→C, A→B, A→C, B→C

* Tail Recursion :- It is a special case of Recursion in which the last operation on Recursion is

```
int fact_tail (int n, int r)
{   if (n == 1)
        return r
    else
        return fact_tail (n-1, n*r)
}
```

fact_tail (5, 1)
↓
fact tail (4, 5)
↓
fact_tail (3, 20)
↓
fact_tail (2, 60)
↓
fact_tail (1, 120) = 120

```
int fib( int n, int next, int r)
{
    if (n==1)
        return r
    else
        fib_tail (n-1, next+r, next)
}
```

fact (4, 1, 0)
↓
fib (3, 1, 1)
↓
fib (2, 2, 1)
↓
fib (1, 3, 2)
↓
3

* Selection sort

| 0 | 1 | 2 | 3 | 4 | |
|---|---|---|---|---|---|
| 40 | 20 | 50 | 60 | 30 | a[0] > a[1] swap |
| 20 | 40 | 50 | 60 | 30 | No swap |
| 20 | 40 | 50 | 60 | 30 | No swap |
| | | | | | |

Pass 1

Selection_Sort(A)

1. n ← length[A]
2) for j ← 1 to n-1
3) smallest ← j
4) for i ← j+1 to n
5) if A[i] < A[smallest]
6) then smallest ← i
}
7) Exchange (A[j], A[smallest])

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 50 | 20 | 10 | 40 | 30 |

↑ j

Pass 1

n=5, j=1, smallest = 1
i = 2
20 < 50 ✓ smallest = 2
i = 3
10 < 20 ✓ smallest = 3
i = 4
40 < 10 ✗ no change
i = 5
30 < 10 ✗ no change

Pass 2

| 10 | 20 | 50 | 40 | 30 |
|---|---|---|---|---|

j = 2, smallest = 2
i = 3
50 < 20 ✗ no change
40 < 20 ✗ no change
30 < 20 ✗ no change

Pass 3

| 10 | 20 | 50 | 40 | 30 |
|---|---|---|---|---|

j = 3, smallest = 3
i = 4
40 < 50 ✓, smallest = 4

| 10 | 20 | 50 | 40 | 30 |
|---|---|---|---|---|

↑ j ↑ smallest ↑ i

i = 5, 30 < 40
smallest = 5

| 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|

Pass 4

| 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|

j = 4, smallest = 4
i = 5
50 < 40 no change

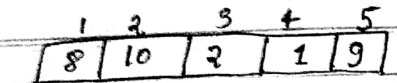
Hence, array is sorted

5
5
10
10
40
40
50

* Insertion sort

Insertion_Sort(A)

1) for j ← 2 to length(A)
2) do key ← A[j]
3) // Insert A[j] into sorted sequence A[1 ... j-1]
4) i ← j-1
5) while i > 0 and A[i] > key
do A[i+1] ← A[i]
i ← i-1
6) A[i+1] ← key



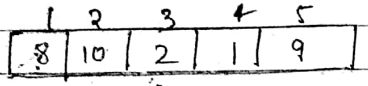

Key = 10

i=1, j=2

while 1>0 and 8>10

→ A[2] ← A[1] X

A[2] ← Key




Key = 2

i=2

while 2>0 and 10>2

→ A[3] ← A[2]

8 ← ~~key~~

i=1


1>0 and 8>2

A[2] ← A[1]

* Quick Sort (First element as Pivot)

```
Partition (A, lb, ub)
{
   start = lb
   end = ub
   pivot = A[lb]
while ( start < end)
{
  while ( a [start] <= pivot)
   {
      start ++
   }
  while ( a [end] > pivot)
     {
        end --
     }
   if (start < end)
   {
      exchange (A[start], A[end])
   }
   ~~return end;~~
}
 exchange ( A[lb], A[end])
}
 return end;
}
```

Pivot = 15

| 15 | 22 | 30 | 10 | 15 | 64 | 1 | 3 | 9 | 2 |
|---|---|---|---|---|---|---|---|---|---|
| | i | | | | | | | | j |
| 15 | 2 | 30 | 10 | 15 | 64 | 1 | 3 | 9 | 22 |
| | i | i | | | | | | j | j |
| 15 | 2 | 9 | 10 | 15 | 64 | 1 | 3 | 30 | 22 |
| | | i | i | i | | | j | j | |
| 15 | 2 | 9 | 10 | 3 | 64 | 1 | 15 | 30 | 22 |
| | | | i | i | j | j | | | |
| 15 | 2 | 9 | 10 | 3 | 1 | 64 | 15 | 30 | 22 |
| | | | | j | i | | | | |
| 1 | 2 | 9 | 10 | 3 | 15 | 64 | 15 | 30 | 22 |

```
QuickSort (A, lb, ub)
{
    If (lb < ub)
    {
        end = Partition (A, lb, ub)
        QuickSort (A, lb, end-1)
        QuickSort (A, end+1, ub)
    }
}
```

Pivot

| 7 | 6 | 10 | 5 | 9 | 2 | 1 | 15 | 7 |
|---|---|---|---|---|---|---|---|---|
| i | i | | | | | | | |

| 7 | 6 | 10 | 5 | 9 | 2 | 10 | 15 | 7 |
|---|---|---|---|---|---|---|---|---|
| | i | i | i | j | j | j | | |
| 7 | 6 | 1 | 5 | 2 | 9 | 10 | 15 | 7 |
| | | | i | i | | | | |
| | | | | j | i | | | |
| 2 | 6 | 1 | 5 | 7 | 9 | 10 | 15 | 7 |

partition I

partition II

1) start < end  1 < 9

7 <= 7

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| 2 | 6 | 7 | 5 | 1 | 7 | 9 | 15 | 10 |

Algorithm when pivot is last element

Partition (A, p, r)

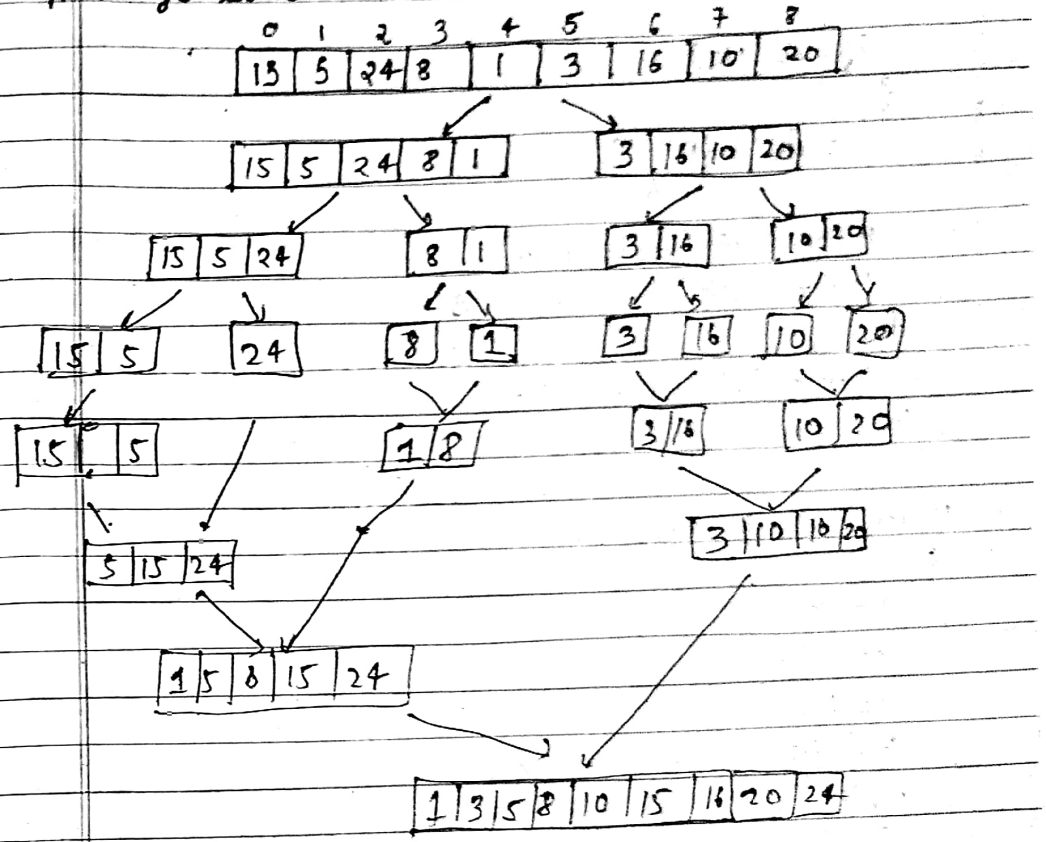
(p: lb, r: ub)

(x: Pivot)

1) A x ← A[r]
2) i ← p-1
3) for j ← p to r-1
4) do if A[j] ≤ x
5) then i ← i+1
6) exchange A[i] ↔ A[j]
7) Exchange A[i+1] ↔ A[r]
8) Return i+1

```
QuickSort (A, P, r)
{
    if (P<r)
    {
      q = Partition (A, P, r)
      QuickSort(A, P, q-1)
      QuickSort (A, q+1, r)
    }
}
```

* Merge sort



```
Merge sort (A, lb, UB)
{
    if (lb < Ub)
    {
        mid = lb + UB
              2
    Merge sort (A, lb, mid)
    Merge sort (A, mid+1, UB)
    }
    Merge (A, lb, mid, UB)
}

Merge (A, lb, mid, UB)
{
    i = lb
    j = mid + 1
    K = lb
    b [High - low + 1] or b[n]
while (i <= mid && j <= UB)
{
    if (A[i] < A[j])
    {
        b[K] = A[i]
        i++;
        K++;
    }
else
    {   b[K] = A[j]
        j++;
        K++;
    }
}
while (i <= mid)
    {
        b[K] = A[i]
        i++;
        K++;
    }
while (j <= UB)
    {
        b[K] = A[j]
        j++;
        K++
    }
```