19/08/19

Ch-1 (Introduction to DSA)

1.) Data → It is a collection of raw facts or set of values.

2.) Information → Useful data or data after processing is known as information.

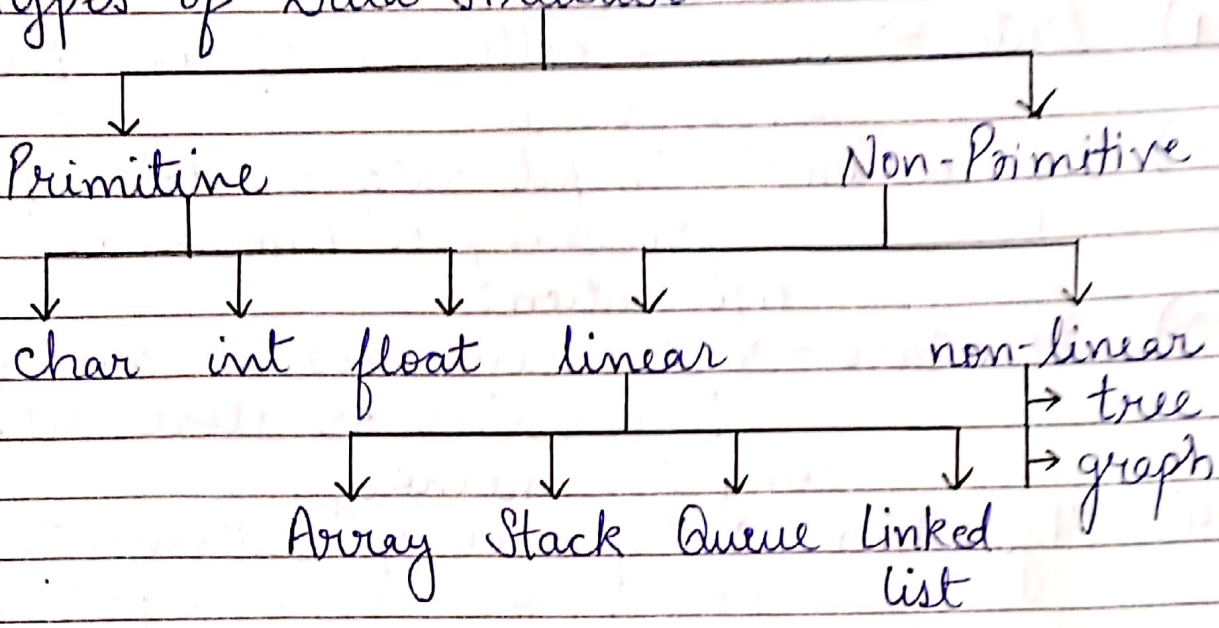
3.) Structure → It means how we are going to organise or store data in computer memory.

4.) Algorithm → Sequence of statements to perform or solve a particular task is known as algorithm. It describes only logic of all programming languages.

5.) Data Structure → It is a logical or a mathematical way to organise the data in a memory.

* Operations on Data Structure:
  (i) Creation
  (ii) Insertion
  (iii) Deletion
  (iv) Searching
  (v) Sorting
  (vi) Merging

★ Types of Data Structure:



★ Difference between linear & Non-linear:

Linear	Non-linear
Store data in sequential order.	Store data in non-sequential order.

★ Characteristics of a good algorithm:

1.) Input → some proper input for algorithm.
2.) Output → some proper output for algorithm.
3.) Definiteness → run for a particular way or clear
4.) Finiteness → run for particular steps or time
5.) Effectiveness → output should be desired.

★ Rules to write an algorithm:

1.) Begin / start
2.) End / Exit
3.) Variable should be in Capital.
4.) Use declare to declare a variable.

5.) if condition
then ___
___

else ___
___

6.) Repeat for i = ___ to ___

Ex: Write a program with the algorithm to find sum of elements in a given array.

Program:

```
#include <stdio.h>
#include<conio.h>
void main()
{
  int i, sum = 0, a[20], n;
  printf("Enter number of elements:");
  scanf("%d", &n);
  printf("Enter values in an array");
  for(i=0; i<n/i<=n-1; i++)
  {
    scanf("%d", &a[i]);
  }
  printf("The sum of elements are:");
  for(i=0; i<n; i++)
  {
    sum = sum + a[i];
  }
```

```
    printf("%d", sum);
    getch();
}
```

Algorithm:

Array - SUM(A, N)

// here A is an Array of N elements

Step 1: declare I, SUM;
Step 2: set SUM = 0;
Step 3: repeat step 4 for I = 0 to N-1
Step 4: set SUM = SUM + A[I]
Step 5: write SUM
Step 6: Exit

Note: i) // or [ ] is used as comments.
(ii) Start or Exit is not compulsory.
(iii) Set is used to initialize value to variable.
(iv) We can also write SUM with =, :, :-.
(v) for using loop we use repeat in algo.
(vi) To show output on the screen, we use print or write in algo.
(vii) To input the value to variable, we use read in algo.

* Analysis of Algorithm :

- Space Complexity
- Time Complexity
  - Best case
  - Average case
  - Worst case

* Time Complexity :

Array - SUM(A,N)	No. of steps	Time constant	Total Time
declare I, SUM;	1	C1	C1
set SUM = 0;	1	C2	C2
repeat step 4 for I=0 to N-1	n+1	C3	(n+1)C3
set SUM = SUM + A[I]	n	C4	(n)C4
write SUM / print SUM	1	C5	C5
exit	1	C6	C6

Total time :- $C1 + C2 + (n+1)C3 + (n)C4 + C5 + C6$

Assume there is C = 1,

$1 + 1 + n + 1 + n + 1 + 1$

$2n + 5$

Complexity order = n

Ex: Write an algorithm to perform addition between two matrices and also calculate time complexity of algorithm.

Algorithm:

Matrix Addition (A, B, N, M)

// here N = no. of rows, M = no. of columns

Step 1 : read A and B

Step 2 : declare C[n][n], I, J

Step 3 : repeat step 4 to 5 for I = 0 to N-1

Step 4 : repeat step 5 for J = 0 to M-1

Step 5 : set C[I][J] = A[I][J] + B[I][J]

Step 6 : write C[I][J]

Step 7 : exit

Time Complexity :-

No. Of Steps	Time Constant	Total Time
1	C_1	C_1
1	C_2	C_2
n+1	$C_3(n+1)$	$n+1(C_3)$
n(n+1)	C_4	$n(n+1)(C_4)$
n x m	C_5	$n \times m(C_5)$
1	C_6	C_6
1	C_7	C_7

Total Time :- $C1 + C2 + n+1(C3) + n(n+1)(C4) + n \times m(C5) + C6 + C7$

Assume there is C = 1,

$= 1 + 1 + n + 1(1) + n(n+1)(1) + n \times m(1) + 1 + 1$

$= nmC4 + nC4 + 5$

$= n + nm + nm + n$

$= 2nm + 2n + 5$

Complexity order = nm

Ex: Write an algorithm to multiply two matrices where size of first matrix is $n \times m$ & size of second matrix is $m \times p$. Prove that complexity of this algorithm is $n \times m \times p$.

Algorithm:

Matrix multiplication (A, B, N, M, P)

// here 1st Matrix, n = no. of rows, m = no. of columns

// here 2nd Matrix, m = no. of rows, p = no. of columns

Step 1: read A and B

Step 2: declare I, J, K, C[N][P]

Step 3: repeat step 4 to for I = 0 to N - 1

Step 4: repeat step 5 to for J = 0 to P - 1

Step 5: set C[I][J] = 0

Step 6: repeat step 7 for K = 0 to M - 1

Step 7: set C[I][J] = C[I][J] + A[I][K] * B[K][J]

Step 8: write C[ ][ ]

Step 9: exit

No. of Steps	time constant	Total Time
1	C_1	
1	C_2	
$n+1$	C_3	
$n(p+1)$	C_4	
$n \times p$	C_5	
$n(p \times m+1)$	C_6	
$n \times p \times m$	C_7	
1	C_8	
1	C_9	

* Searching :

To find out a position or a particular element in data is called Searching.
There are two methods for searching :-

(i) Linear Search
(ii) Binary Search

Linear Search	Binary Search
(i) Time complexity = O(n)	(i) Time Complexity = $O(\log_2 N)$
(ii) It is more complex.	(ii) It is less complex.
(iii) It can be applied in any type of array.	(iii) It can only be applied in sorted order.
(iv) It can also applied in 2D array.	(iv) It cannot be applied in 2D array.
(v) Lines of code is less.	(v) Lines of code is more.

(i) Linear Search :

Program

```
#include <stdio.h>
#include <conio.h>
void main ()
{
  int a[10], i, n, val, pos, flag = 0;
  printf ("Enter number of elements in array:");
  scanf ("%d", &n);
  printf ("Enter the values in an array:");
  for ( i = 0; i <n ; i++)
  {
```

```
    scanf("%d", &a[i]);
}
printf("Array is following : \n");
for(i=0; i<n; i++)
{
    printf("%d \n", a[i]);
}
printf("Enter value which you want to search");
scanf("%d", &val);
for(i=0; i<n; i++)
{
    if(a[i] == val)
    {
        flag = 1;
        pos = i;
        break;
    }
}
if(flag == 0)
{
    printf("Value not found");
}
else
{
    printf("Value found at %d position", pos+1);
}
getch();
}
```

(ii) Binary Search:

Program

```
#include <stdio.h>
#include <conio.h>
void main()
{
  int a[10], low, high, n, val, mid, flag=0,
     pos, i;
  printf("Enter number of elements in array:");
  scanf("%d", &n);
  printf("Enter the values in an array:");
  for(i=0; i<n; i++)
  {
    scanf("%d", &a[i]);
  }
  printf("Enter the value which you want to search");
  scanf("%d", &val);
  high = n-1;
  low = 0;
  while (low <= high)
  {
    mid = (low + high)/2;
    if (a[mid] == val)
    {
      pos = mid;
      flag = 1;
      break;
    }
```

```
        else if (val > a[mid])
        {
            low = mid + 1;
        }
        else
        {
            high = mid - 1;
        }
    }
    if (flag == 1)
    {
        printf("The value is at %d position", pos+1);
    }
    else
    {
        printf("Value not found");
    }
    getch();
}
```

* Sorting :

To arrange the data in a particular order is known as sorting.

There are some methods for sorting :-

(i) Simple
(ii) Bubble
(iii) Selection
(iv) Insertion
(v) Quick
(vi) Merge

(i) Bubble Sorting:

Program
(Here, n=5)

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

Note: Outer loop is used for processing and inner loop is used for comparison.

(ii) Selection Sorting:

Program

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

```
    {
        min = a[j];
        pos = j;
    }
}
temp = a[pos];
a[pos] = a[i];
a[i] = temp;
}
```

(iii) Insertion Sorting:

Algorithm

Insertion Sort (A, N)

Step 1: declare I, J, KEY
Step 2: repeat step 3 to 6 for I ← 1 to N-1
Step 3: set KEY = A[I]
J = I - 1
Step 4: repeat step 5 while J >= 0 && A[J] > KEY
Step 5: set A[J+1] = A[J]
J = J - 1
Step 6: set A[J+1] = KEY
Step 7: write A[ ]
Step 8: exit ( )

Program

```
for( i=1; i<n-1; i++)
```

```
{
    key = a[i];
    j = i-1;
    while (j>=0 && a[j]>key)
    {
        a[j+1] = a[j];
        j = j-1;
    }
    a[j+1] = key;
}
```

Note: Among bubble sort, insertion sort and selection sort, insertion sort is best.

Time Complexity Analysis

Steps	No. of steps	Time constant	Total Time
1	1	C_1	C_1
2	n	C_2	$C_2 n$
3	n-1	C_3	$C_3(n-1)$
4	$\sum_{i=1}^{n-1} ti$	C_4	$C_4 \sum_{i=1}^{n-1} ti$
5	$\sum_{i=1}^{n-1} (ti-1)$	C_5	$C_5 \sum_{i=1}^{n-1} (ti-1)$
6	n-1	C_6	$C_6(n-1)$
7	1	C_7	C_7
8	1	C_8	C_8

Note: ti = while loop

$$T(n) = C_1 + C_2 n + C_3(n-1) + C_4 \sum_{i=1}^{n-1} ti + C_5 \sum_{i=1}^{n-1} (ti-1) + C_6(n-1) + C_7 + C_8$$

Best case :-

$$= C_1 + C_2 n + C_3(n-1) + C_4(n-1) + C_5 \sum_{i=1}^{n-1} 0 + C_4 \sum_{i=1}^{n-1} t_i + C_6(n-1) + C_7 + C_8$$

$$= C_1 + C_2 n + C_3(n-1) + C_4(n-1) + C_6(n-1) + C_7 + C_8$$

$$= 4n - 1 \rightarrow O(n)$$

Note :
- Formula of Summation :-

$$\sum_{i=1}^{n-1} t_i = t_1 + t_2 + t_3 - - - t_{n-1}$$

$$= 1 + 1 + 1 - - - 1$$

- C_5 will be skip because of 0.
- In Best case, t_i will always 1.
- Sum of continuous terms : $\frac{n(n+1)}{2}$

Worst case :-

$$= C_1 + C_2 n + C_3(n-1) + C_4\left[\frac{n(n+1)}{2} - 1\right] + C_5\left[\frac{n(n+1)}{2} - 1\right] + C_6(n-1) + C_7 + C_8$$

$$= O(n^2)$$

★ Asymptotic Notations :

They are used to describe time complexity of an algorithm. It gives us a method for classifying functions according to their rate of growth.

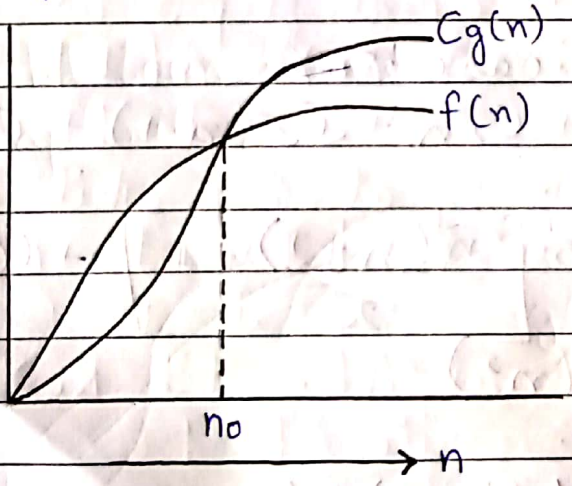
We need to develop a way to talk about rate of growth of functions so that we can compare algorithms. We are usually interested in the order of growth of the running time of an algorithm not in the exact running time. This is also referred as asymptotic running time. Asymptotic means tends to infinity. Eg: $O(n^2)$, $O(n)$

There are three types of asymptotic notations:-

(i) Big Oh (O)
(ii) Omega (Ω)
(iii) Theta (Θ)

(i) Big Oh (O)

It is upper bound and describes the worst case complexity.



$f(n) \leq cg(n)$
for all $n \geq n_0$, $c > 0$
then $f(n) = O(g(n))$

Ex: Prove that:

(a) $3n - 2 = O(n)$

here,

$f(n) = 3n - 2$

$g(n) = n$

Equation,

$f(n) \leq Cg(n)$

Let $n = 1$ and $C = 2$ (assumed values)

put in the above equation

$= 3n - 2 \leq Cn$

$= 3 \times 1 - 2 \leq 2 \times 1$

$= 3 - 2 \leq 2$

$= 1 \leq 2$

∴ Condition is True & satisfied

(b) $5n^2 + 3n - 2 = O(n^2)$

here,

$f(n) = 5n^2 + 3n - 2$

$g(n) = n^2$

Equation,

$f(n) \leq Cg(n)$

Let $n = 3$ and $C = 6$ (assumed values)

put in the above equation

$= 5n^2 + 3n - 2 \leq Cn^2$

$= 5 \times 9 + 3 \times 3 - 2 \leq 6 \times 9$

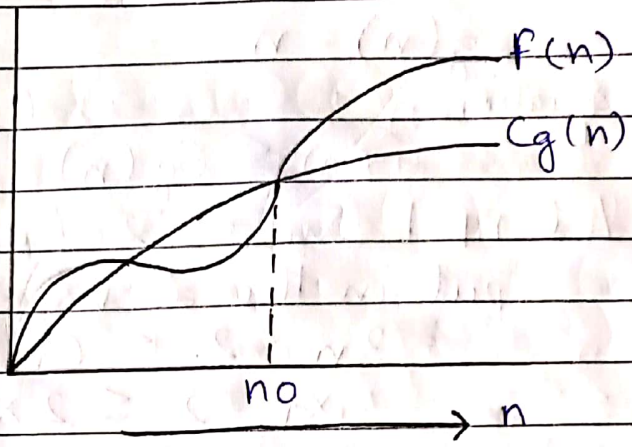
$= 45 + 9 - 2 \leq 54$

$= 52 \leq 54$

∴ Condition is True & satisfied

(ii) Omega (Ω)

It is lower bound and describes the best case complexity.



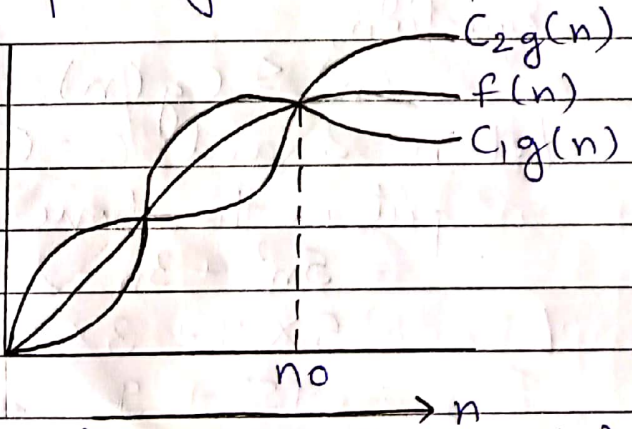
$f(n) \geq Cg(n)$

for all $n \geq no$ and $C < 0$

then $f(n) = \Omega(g(n))$

(iii) Theta (Θ)

It is tight bound and describes both the case complexity.



$C_1 g(n) \leq f(n) \leq C_2 g(n)$

for all $n \geq no$ and $C > 0$

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

Ex of Big oh (O):

(c) $4n + 3 = O(n)$

here,

$f(n) = 4n + 3$

$g(n) = n$

Equation,

$f(n) \leq cg(n)$

Let $n = 3/4$ and $C = 5$ (assumed values)

put in the above equation

$= 4n + 3 \leq 5n$

$= 4 \times 3 + 3 \leq 5 \times 3$

$= 12 + 3 \leq 15$

$= 15 \leq 15$

$4n + 3 \leq 5n$

$4 \times 4 + 3 \leq 5 \times 4$

$16 + 3 \leq 20$

$19 \leq 20$

∴ Condition is true & satisfied

$C \geq 5$

$n_0 \geq 3$

(d) $5n^2 + 4n + 3 = O(n^2)$

here,

$f(n) = 5n^2 + 4n + 3$

$g(n) = n^2$

Equation,

$f(n) \leq cg(n)$

Let $n = 1$ and $c = 13$ (assumed values)

put in the above equation

$= 5n^2 + 4n + 3 \leq 13n^2$

$= 5 \times 1 + 4 \times 1 + 3 \leq 13 \times 1$

$= 5 + 4 + 3 \leq 13$

$12 \leq 13$

∴ Condition is true & satisfied

$c \geq 13$
$n_0 \geq 1$

Ex of Omega (Ω) & Theta (Θ):

(a) $f(n) = 4n + 3 = \Omega(n)$

(b) $f(n) = 3n^2 - 5n + 3 = \Theta(n)$

(c) $f(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$

Sol(a): Here,

$f(n) = 4n + 3$

$g(n) = n$

Equation,

$f(n) \geq (g(n)$

Let

Sol (c) : $\frac{1}{2}n^2 - 3n$

here,

$f(n) = \frac{1}{2}n^2 - 3n$

$g(n) = n^2$ → $f(n) > 0$

Equation,

$C_1 g(n) \leq f(n) \leq C_2 g(n)$

Let $n = 1$ and $C_2 = \frac{1}{2}$ and $C_1 = \frac{1}{14}$

put in the above equation

$C_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq C_2 n^2$

$C_1 \leq \frac{1}{2}n^2 - 3n \leq C_2$

$\frac{1}{2} - \frac{3}{7} = \frac{7-6}{14} = \frac{1}{14}$ (C_1)

Q: Prove that (theta Θ):

(a) $f(n) = 5n^2 + 3n - 5 = \Theta(n^2)$

(b) $f(n) = 5n - 3 = \Theta(n)$

(c) $f(n) = 2n^3 + 5n^2 - 3 = \Theta(n^3)$

Ans (a) $C_1 n^2 \leq 5n^2 + 3n - 5 \leq C_2 n^2$

Let $n = 2$, $C_1 = 5$, $C_2 = 6$

$5 \times 4 \leq 5 \times 4 + 3 \times 2 - 5 \leq 6 \times 4$

$20 \leq 21 \leq 24$

(b) $C_1 n \leq 5n - 3 \leq C_2 n$

divide the equation with n,

$C_1 \leq 5 - \frac{3}{n} \leq C_2$

Let $n \geq 1$, $C_1 = 1/2$, $C_2 = 5$

$1/2 \leq 5 - \frac{3}{1} \leq 5$

$1/2 \leq 2 \leq 5$

(c) C

★ Pointers:

Pointer is a special variable i.e. used to store the address of another variable of similar datatype.

- Pointer to Pointer → It is a special variable i.e. used to store address of another pointer of similar datatype:

Ex:

```
void main ()
{
    int *ptr1, a = 10, b = 20;
    float *ptr2, c = 5.5;
    ptr1 = &a;
    printf("Value of a = %d", a);      // 10
    printf("Value of a = %d", *ptr1);  // 10
    printf("Value of a = %d", *(&a));  //10
    printf("Address of a = %d", &a);   // 2000
    printf("Address of a = %d", ptr1); //2000
    ptr1 = &b;
    printf("Value of b = %d", b);      // 20
    printf("Value of b = %d", *ptr1);  // 20
    ptr2 = &c;
    printf("Value of c = %f", c);      Error
    printf("Value of c = %f", *ptr2);  Error
    getch();                           becoz of
}                                      float datatype
```

c = 5.5, a = 10 (2000), b = 20 (3000)

Note: Multiple pointers can access a single

location at a time but a single pointer can access only one location.

★ Array Using Pointers :

Ex:
```
void main()
{
  int *p, a[5], i;
  p = a;          → address of first element → base address of array
  printf("Enter values in array:");
  for(i=0; i<5; i++)
  {
    scanf("%d", (p+i));     → It stores address
  }
  printf("Values in array:\n");
  for(i=0; i<5; i++)
  {
    printf("%d", *(p+i));     → It show values at that address
  }
  getch();
}
```

	0	1	2	3	4
a =	10	20	30	40	50

Note: (i) Pointer can be increment / decrement.
Eg: ptr1++;
which increases / decreases the address of array

(ii) Array is a collection of single datatypes values under the single variable name.

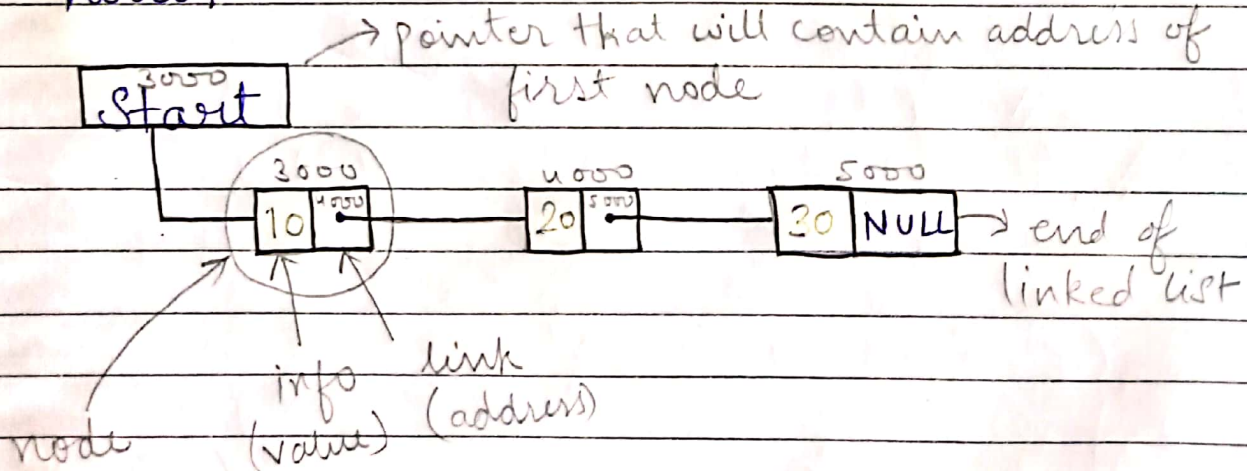
(iii) Addresses of array is continuous memory allocation like 2000, 2002, 2004, -- etc.
becoz of (int) type

★ Linked List :

- Problems in Array :-
  (i) Static Memory Allocation
  (ii) difficulty in insertion and deletion

- Advantages in Linked List :-
  (i) Dynamic Memory Allocation
  (ii) Direct insertion & deletion

- Linked List is a collection nodes where each node will contain two parts, first part will contain value and second part will contain address of another node.



Note: It is difficult to maintain because of more & more nodes and it goes only in one

direction and there is no direct access in linked list.

- Construction of Node :-

```
struct node
{
    int info;
    struct node *link;
};
```

datatype