DATA STRUCTURE

A data structure is a particular way of storing and organizing the data in a computer memory so that data can be efficiently accessed.

TYPES OF DATA STRUCTURE:

1. Primitive data structure

2. Non-Primitive data structure

Primitive data structure:

Primitive data structures are those data structure which can store only single value.

Eg.int, float, char etc.

Non-Primitive data structure:

Non-primitive data structures are those data structure which are derived from primitive data structure.

Types of Non-Primitive data structure:

1. Linear data structure

2. Non-Linear data structure

Linear data structure:

Linear data structures are those data structures in which one value is connected to only one value or after one value there is only one value.

E.g. Arrays, Linked List, Stack, Queue etc.

Non-Linear data structure:

Non-Linear data structures are those data structures in which one value is connected to multiple values.

Eg. Graph, Trees, Heap etc.

ALGORITHMS

Algorithm is a blueprint of a program OR It is a step by step procedure for solving a computational problem.

$$\begin{tikzcd} & P \arrow[dl] \arrow[d] \arrow[dr] & & \\ S1 & S2 & S3 & \ldots\ldots\ldots\ldots Sn \end{tikzcd}$$

For any particular problem there can be multiple solutions. Let us consider there is a problem P for which there are multiple solutions S1, S2, S3 and let say Sn. These multiple solutions can be described in

simple English text language or they describe in simple sentence /words those simpler statement are known as Algorithm.

And then we analyse that particular algorithm on basis of 2 factors i.e Time and Space complexity. The algorithm which is most efficient in terms of time and space and then we implement in a programming language and make a suitable program.

ASYMPTOTIC NOTATIONS

Asymptotic notations are symbols which indicate the time complexity of an algorithm.

1. O (Big oh) [Worst case]

2. Ω (Omega) [Best case]

3. ϴ (Theta) [Average case]

# FORMULA :

1 + 2 + 3 + ………. + n = n(n+1)/2.

1^2 + 2^2 + 3^2 + ………. + n^2 = n(n+1)(2n+1)/6.

1^3 + 2^3 + 3^3 + …….. + n^3 = [n(n+1)/2]^2;

QUESTIONS:

```
1. Fun()
{
   int i,j,k,n;
   for(i=1;i<=n;i++)
  {
      For(j=1;j<=i^2;j++)
      {
          For(k=1;k<=n/2;k++)
          {
            print("NITESH");
          }
      }
  }
}
2.   Fun()
  {
  for(i=1 to n)
  {
      for(j=1 to n)
      {
        for(k=1 to I)
       {
         print("Nitesh");
       }
      }
  }
  }
```

CONCEPT:

1 . i = i * 2 [log2 n]

2 . i = i * 3 [log3 n]

3 . i = i * k [logk n]

3.

```
Fun()
{
  For(i=n/2;i<=n;i++)
 {
     For(j=1;j<=n/2;j++)
    {
       For(k=1;k<=n;k++)
      {
       print(“Nitesh”);
      }
    }
  }
}
```

ANS=O(n^3)

4.

```
Fun()
{
   For(i=n/2;i<=n;i++)
  {
     For(j=1;j<=n/2;j++)
    {
        For(k=1;k<=n;k=k*2)
       {
        printf(“Nitesh”);
       }
     }
   }
}
```

ANS: n/2*n/2*log n = O(n^2log n)

5.

```
Fun()
{
   For(i=1;i<=n;i=i*2)
  {
     For(j=n/2;j<=n;j++)
    {
        For(k=1;k<=n;k=k*2)
       {
        print(“Nitesh”);
```

```
      }
    }
  }
}
```

6.

```
int i=1, S=1 ;
while(S <= n)
{
i++ ;
S = S + i ;
print("Nitesh");
}
```

ANS : i = K , S = K(K+1)/2 = n(n+1)/2 = O(n^2)

i=1,2,3,4,K

S=1,3,6,10,k(k+1)/2

7.

A()

```
{
while(n>1)
{
n = n/2;
}
```

ANS : 2^1,2^2,2^3

n = 2,4,8

P = 1,2,3

2^k = n

log 2^k = log n

k log 2 = log n

k = log n

= O(log n)

8.

```
A()
{
int n = 2^2^k;
For(i=1,i<=n;i++)
{
j=2;
   while(j<=n)
   {
      j = j^2
    print("Nitesh");
   }
}
}
```

ANS :O(n log(log(n)))

# COMPARING COMPLEXITIES :

1.

A	B
O(n)	O(n^2)

ANS : A

2. O(n^2) > O(2^n)

ANS : log n^2 log 2^n

2 log n   n log 2

2 * log n   n

Eg. 2 * log 1024

2 * log 2^10

2 * 10 * log 2

= 20   = 10243

3.

n   <   log n

ANS : 1024   log 2^10

1024   10 * log 2

1024   10

4.

n   >   n log n

ANS :

1024   1024 * log 2^10

1024   1024 * 10

1024   10240

5.

log n^2 > log^2 n

ANS: 2 * log n (log n)^2

2 * log 2^10 [log 2^10]^2

2 * 10 (10)^2

= 20 = 100

CONCEPT :

O(1) < O(log n) < O(n) < O(n log n) < O(n^2)

< O(2^n) < O(n^3)

QUESTIONS:

1.

```
  A( Arr , n)
 {
  int  i;
  print("Nitesh");
 }
```

ANS : O(1)

2.

```
 A( Arr , n)
```

```
{
int i , j ;
Create brr[n][n];
}
```

ANS : O(n^2)

# STRUCTURE

Structure is the collection of similar and dissimilar data types in continuous memory allocation.

Eg. int, char, float, pointer etc. In a continuous block of memory we have one integer, character, float etc.

Syntax of structure:

```
struct Student
{
 int roll;
float per;
char grade;
};
```

Syntax of class:

```
class Nitesh
{
     int roll;
     float per;
     char grade;
}
```

Data members

# Program to implement structure:

```
#include<stdio.h>
#include<conio.h>
struct Student
{
int roll;
float per;
char grade;
};
int main()
{
struct Student s1;
s1.roll=10;
s1.per=85.4;
s1.grade='N';
printf("roll=%d\npercent=%f
\ngrade=%c",s1.roll,s1.per,s1.grade);
return 0;
}
```

# Difference between structure and class:

Structure	class
Structure is a collection of a Similar and dissimilar data in a continuous memory.	class is a collection of data members and member functions.
Structure are less secure.	class is more secure.
In structure we don't have access specifier.	In class we have access specifier like public,private etc.
We can't declare functions inside structure.	we can declare declare functions

inside class.

STACK

Stack is a linear list with the restriction that elements will be inserted and deleted only from one end of the stack called as the top of the stack. It is based on the principle of "last in first out". The operation for inserting an element in a stack is called as "push operation." And the operation to delete an element from the stack is called as "pop operation."

Eg. palindrome number or palindrome string.

Algorithm for PUSH operation:

```
Algorithm  PUSH ( Arr, Maxsize top, item):
Step 1: Check Overflow
If top >= Maxsize
Print (Stack Overflow)
Exit
Step 2:  top←top+1;
Step 3:  Arr[top]←item;
Step 4: Exit
```

Algorithm for POP operation:

```
Algorithm  POP(Arr, Max size, top,item)
Step 1:  Check for Underflow
if (Top = -1)
print(Stack Underflow )
return;
Step 2: item←Arr[top];
Step 3: top←top-1;
Step 4: return item;
Step 5: Exit
```

# Program to implement Stack :

```
#include<stdio.h>
#include<conio.h>
void push(struct stack*,int);
int pop(struct stack*);
```

```
struct stack
{
int arr[5];
int top;
};
int main()
{
int x,n,catch;
struct stack s;
s.top=-1;
while(1)
{
printf("Enter 1:push\n2:pop\n3:exit");
scanf("%d",&x);
if(x==1)
{
printf("Enter an element to push");
scanf("%d",&n);
push(&s,n);
 }
else if(x==2)
{
catch=pop(&s);
if(catch!=-999)
{
printf("Element deleted is %d",catch);
}
 }
else
{
exit(1);
}
return 0;
}
void push(struct stack *p, int n)
{
```

```
if (p→top==4)
{
printf("stack Overflow");
}
else
{
p→top=p→top+1;
p→arr[p→top]=n;
}
}
int pop(struct stack *p)
{
int z;
if(p→top==-1)
{
printf("Stack Underflow");
return -999;
}
```

```
else
{
z=p→arr[p→top];
p→top=p→top - 1;
return  z;
}
}
```

# QUEUE