UNIT-1

DESIGN AND ANALYSIS OF ALGORITHMS

DEFINITION: It is a combination of sequence of finite steps to solve a particular problem or it is a step by step method to solve a particular a problem.

Characteristics of Algorithm:

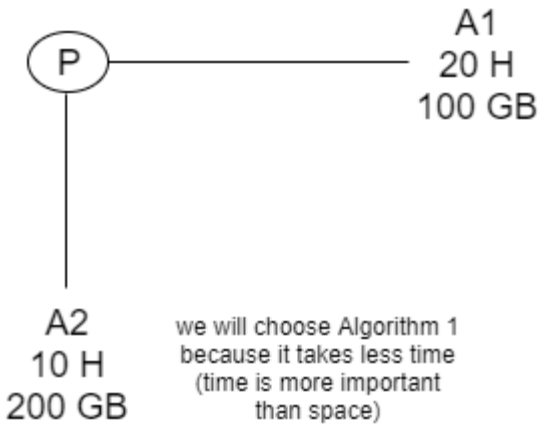
1. Input : Algorithm must contain '0' or more input.
2. Output : Algorithm must contain '1' or more output.
3. Finiteness : Algorithm must complete after finite no. of steps.
4. Definiteness : The step of the algorithm must be defined precisely or clearly unambiguous .
5. Effectiveness : Every statement in the algorithm should perform some operation.

Steps Required to Construct an Algorithm

1. Problem Definition (Knowing the problem clearly)
2. Design Algorithm (Divide and Conquer, Greedy Technique and Dynamic Programming)
3. Flowchart
4. Verification
5. Implementation/Coding
6. Analyzing Algorithm

Analysis of Algorithm : To solve a particular problem if at all more than one algorithm is possible, best one will be decided by analysis based on two factors:

1. **Time (CPU Time):** The amount of time required by the algorithm is called Time Complexity.
2. **Space (Main Memory space):** The amount of space/memory required by the algorithm is called as space complexity.



Calculating Time Complexity:

There are two types of Algorithms

Iterative

```
A( )
{
 For( i=1 to n )
   Max(a,b)
 }
```

If there is no function calling and if it having a for loop or while loop then it is called Iterative Algorithm.

Recursion

```
A( )
{
If( )
A(n/2)
}
```

If A is calling itself then it is called Recursion.

Note:

1. Any program that could be written using iteration could be written using Recursion and vice-versa.
2. In order to analyze , such iterative program, we have to count number of times a loop is going to get executed.
3. **In case, algorithm does not contain either iteration or recursion then it means that there is no dependency of the running time on the input size. Whatever is the input size, running time is constant [O(1)].**

Iteration: Order of magnitude of statement- how many times that a statement is executed.

1.

```
main()
{
  x+y+z=1;
}
```

Explanation:
Here, order of magnitude of statement (how many times that a statement is executed)
Time Complexity = O(1)

2.

```
main()
{
  a=b+c;          order of magnitude =1
  for(i=1;i<=n;i++)
 {
   x=y+z;          order of magnitude=n
 }
}
```

Explanation:
Time Complexity = n+1 (1 is smaller as compared to n, neglect 1) = O(n)

i=1	i=2	i=3	i=4	i=5	……	i=n
x=y+z	x=y+z	x=y+z	x=y+z	x=y+z	……	x=y+z

3.

```
main()
{
 for(i=1;i<=n;i++)
 {
  for(k=1;k<=n;k++)
  {
    x=y+z;     order of magnitude  = n*n = n^2
  }
 }
}
```

Explanation:

Outer loop is going to run n times and for every value of i, k will run n times.
So total (x=y+z) will be executed n^2 times.
Time Complexity = $O(n^2)$

i=1	k=1	x=y+z
	k=2	x=y+z
	k=3	x=y+z
	.	.
	.	.
	k=n	x=y+z
i=2	k=1	x=y+z
	k=2	x=y+z
	k=3	x=y+z
	.	.
	k=n	x=y+z
i=3	k=1	x=y+z
	k=2	x=y+z
	k=3	x=y+z
	.	.
	k=n	x=y+z
.	.	.
i=n	k=1	x=y+z
	k=2	x=y+z
	k=3	x=y+z
	.	.
	k=n	x=y+z

4.

```
main()
{
 x=y+z ;      order of magnitude = 1
  for(i=1;i<=n;i++)
  {
    x=y+z ;     order of magnitude = n
  }
  for(i=1;i<=n;i++)
  {
    for(k=1;k<=n;k++)
    {
      x=y+z;     order of magnitude  = n*n = n^2
    }
  }
}
```

Explanation:

Time Complexity = n^2+n+1 (n & 1 is smaller as compared to n^2, neglect 1 & n)

= $O(n^2)$

5.

```
main()
{
  int i=1, s=1;
  While(S<=n)
  {
     i++;
     s=s+i;
     Printf("vishal");
  }
}
```

i=1	i=2	i=3	i=4	i=5	……	i=n
s=1	s=3	s=6	s=10	s=15	……	S=k(k+1)/2
First natural no	Sum of 1st 2 natural nos	Sum of 1st 3 natural nos	Sum of 1st 4 natural nos	Sum of 1st 5 natural nos		Sum of 1st n natural nos

Explanation:

Here, increment of i is linear but s is dependent on i.

The entire loop is going to stop whenever s reaches a point where s value is greater than n (s>n).

Suppose after k iteration, s reaches greater than n

Then s=$\frac{k(k+1)}{2}$ (sum of 1st n natural nos)

For loop has to stop:

$\frac{k(k+1)}{2}$>n

$\frac{k^2+k}{2}$>n

k=O($\sqrt{n}$)

Time Complexity : O($\sqrt{n}$)

6.

```
main()
{
 int i , j, k;
 for(i=1;i<=n;i++)       ……n times
 {
  for(j=1;j<=i;j++)  ……..depends on i
  {
   for(k=1;k<=100;k++)     ……always 100 (constant)
   {
    Printf("Vishal");
   }
  }
 }
```

i=1	i=2	i=3	i=4	……	i=n
j=1	j=2 times	j=3 times	j=4 times	……	j=n times
k= 1*100 times	k=2*100 times	k= 3*100 times	k= 4*100 times	……..	k= n*100 times
Vishal (100)	Vishal (2*100)	Vishal (3*100)	Vishal (4*100)	…….	Vishal (n*100)

Explanation:

The outer loop is running for i=1 to n, total n times.

The inner loop is running from j=1 to i, this loop depends on i.

The innermost loop k, run from k=1 to100, this loop is independent of i & j, it's a constant (100).

Total no of this printf will be executed = 100 + 2*100 + 3*100 + 4*100 + ..+ n*100

= 100(1+2+3+4+5+……+n)

= $100\frac{n(n+1)}{2}$

= $n^2 + n$

= $O(n^2)$

7.

```
main()
{
 int i , j, k,n=10;
 for(i=1;i<=n;i++)        ……n times
 {
  for(j=1;j<=i^2;j++)    ……..depends on i
  {
   for(k=1;k<=n/2;k++)   ……for n =10, it will always execute 5(n/2) times
   {
    Printf("Vishal");
   }
  }
 }
```

i=1	i=2	i=3	i=4	…	i=n
j=1	$j=2^2$=4 times	$j=3^2$ = 9 times	$j=4^2$=16 times	…	$j=n^2$ times
k= n/2*1 times	k= n/2*4 times	k= n/2*9 times	k= n/2*16 times	…	k= $n/2*n^2$ times
Vishal (n/2*1)	Vishal (n/2*4)	Vishal (n/2*9)	Vishal (n/2*16)	…	Vishal ($n/2*n^2$)

Explanation:

Total no of times printf will be executed = $\frac{n}{2}*1+\frac{n}{2}*4+\frac{n}{2}*9+\cdots+\frac{n}{2}*n^2$

= $\frac{n}{2}(1+4+9+\cdots+n^2)$

= $\frac{n}{2}(\frac{n(n+1)(2n+1)}{6})$

= $O(n^4)$

8.

```
main()
{
 for(i=1;i<=n;i=i*2)  i*3 ……..n times
 {
   Printf("Vishal");
 }
}
```

Explanation:

i is going from 1 to n but this time i is going faster in powers of 2 (not linear)

$2^k = n$

$k * \log_2 2 = \log_2 n$

$k = \log_2 n$

Time Complexity = $O(\log_2 n)$

If it will be incremented i*3 then T.C.=$O(\log_3 n)$

i=1	i=2	i=4	i=8	i=16	……	i=n
2^0	2^1	2^2	2^3	2^4	……	2^k

9.

```
main()
{
 int i , j, k, n=10;
 for(i=n/2; i<=n; i++)       ……..n/2 times
 {
  for(j=1; j<=n/2; j++)    ……..n/2 times
  {
   for(k=1; k<=n; k*2)     ……log2 n
   {
    Printf("Vishal");
   }
  }
 }
```

Explanation:

Total Time Complexity = $\frac{n}{2} * \frac{n}{2} * \log_2 n$ = $O(n^2 \log_2 n)$

10.

```
main()
{
 int i , j, k,n;
 for( i = n/2; i < = n; i++)        ……..n/2 times
 {
  for(j=1; j < =n; j*2)    ……..log2 n  times
  {
   for(k = 1; k < = n;  k*2)     ……log2 n times
   {
    Printf("Vishal");
   }
  }
 }
```

Explanation:

Total Time Complexity = $\frac{n}{2} * \log_2 n * \log_2 n$

= O[n($\log_2 n$) ^{2}]

11.

Assume n >= 2

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

Explanation:

Everytime value of n is divided by 2. Therefore, Time complexity = O($\log_2 n$)

Assume n >= 2

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

Explanation:

Everytime value of n is divided by 3. Therefore, Time Comlexity = O($\log_3 n$)

Assume n >= 2

```
main ()
{
 while(n>1)
 {
   n = n/k
 }
}
```

Explanation

Everytime value of n is divided by k. Therefore, Time Comlexity = O($\log_k n$)

12.

```
main()
{
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j+i)
  {
   Printf("Vishal");
  }
 }
}
```

i=1	i=2	i=3	i=4	…	i=n
j=1 to n	j=1, 3, 5,….n	j=1, 4,7,…n	j=1,5,9,….n	…	j=1 to n times
n times	n/2 times	n/3 times	n/4 times	…	n/n times
Vishal (n times)	Vishal (n/2 times)	Vishal (n/3 times)	Vishal (n/4 times)	…	Vishal (n/n)

Explanation:

Here, j is dependent on i

Time Complexity = $n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \cdots + \frac{n}{n}$

$= n(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}$

$= O(n \log n)$

13.

```
main()
{
 int n = 2^2^k
 for(i=1; i<=n; i++)
  {
    j=2;
   while(j<=n)
    {
      j = j^2;
      printf("Vishal");
    }
  } }
```

k=1	k=2	k=3	…	k=k
n=2	n=4	n=16	…	$n=2^k$
j=2	j=2,4	j=2,4,16	…	j=2,4,…..(k+1)
Vishal (n*2 times)	Vishal (n*3 times)	Vishal (n*4 times)	…	Vishal (n*(k+1))
For k=1, 2 times	For k=2, 3 times	For k=3, 4 times		For k=k, (k+1) times

Explanation:

$n = 2^{2\wedge k}$

$\log n = 2^k$

loglogn = k

Time Complexity = O(n*($loglogn + 1$))

14.

```
main()
{
 i=1;
  while(i<=n)
  {
    i = i+10;
    i = i+20;
   }
 }
```

Explanation:

i = i+10;

i = i+20;

Total increment, i=i+30;

So, every time i value is incremented by 30.

Therefore, Time Complexity = O(n+30) (30 is constant and very small as compared to n)

Time Complexity = O(n)

15.

```
main()
{
  while(n>=1)
  {
    n = n-10;
  }
}
```

Explanation:

n = n-10;

Decrement, n=n-10;

So, every time i value is decremented by 10.

Therefore, Time Complexity = O(n-10) (10 is constant and very small as compared to n)

Time Complexity = O(n)

16.

```
main()
{
  i=1;
  While(i<=n)
  {
    i = 2*i;
  }
}
```

Explanation:

i = 2*i;

So, every time i value is multiplied by 2 .

Therefore, Time Complexity = $O(\log_2 n)$

```
main()
{
  i=1;
  While(i<=n)
  {
    i = 3*i;
  }
}
```

Explanation:

i = 3*i;

So, every time i value is multiplied by 3 .

Therefore, Time Complexity = $O(\log_3 n)$

17.

```
main()
{
  i=1;
  While(i<=n)
  {
    i = 2*i;
    i = 3*i;
   }
}
```

Explanation:

i = 2*i;

i = 3*i;

Total increment = 2*3 = 6

So, every time i value is multiplied by 6 .

Therefore, Time Complexity = O($\log_6 n$)

18.

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

Explanation:

n = $\frac{n}{2}$

Every time value is divided by 2

Therefore, Time Complexity = O($\log_2 n$)

19.

```
main()
{
  i = 2;
   While(i<=n)
   {
      i = i^2
    }
}
```

Explanation:

$i = i^2$

Every time value is divided by 2

Therefore, Time Complexity = $O(\log_2 n)$