COMPLEXITY ANALYSIS

POSTERIORI AND PRIORI ANALYSIS

- If the problem is having more than one solution or algorithm then the best one is decided by the analysis based on two factors.

1. CPU Time (Time Complexity)
2. Main memory space (Space Complexity)

Time complexity of an algorithm can be calculated by using two methods:

1. Posteriori Analysis
2. Priori Analysis

A Posteriori analysis	A priori analysis
Posteriori analysis is a relative analysis.	Priori analysis is an absolute analysis.
It is dependent on language of compiler and type of hardware.	It is independent of language of compiler and types of hardware.
It will give exact answer.	It will give approximate answer.
It doesn’t use asymptotic notations to represent the time complexity of an algorithm.	It uses the asymptotic notations to represent how much time the algorithm will take in order to complete its execution.
The time complexity of an algorithm using a posteriori analysis differ from system to system.	The time complexity of an algorithm using a priori analysis is same for every system.
If the time taken by the algorithm is less, then the credit will go to compiler and hardware.	If the program running faster, credit goes to the programmer.

1. Relative is always in proportion to a whole. Absolute is the total of all existence.
2. Relative is dependent while absolute is independent.
3. Absolute is capable of being thought of or conceived by itself alone.

1)

```
For(i=0; i<n;i++)
            {
            statement;
            }
```

// Incremented Loop

2)

```
For(i=n; i>0;i--)
            {
            statement;
            }
```

// Decrementing Loop

3)

```
For(i=0; i<n;i+2)
            {
            statement;
            }
```

4)

```
For(i=0; i<n; i++)
{
   for(j=0; j<n;j++)
     {
       statement;
      }
}
```

5)

```
For(i=0; i<n; i++)
{
   for(j=0; j<i;j++)
     {
       statement;
      }
}
```

1+2+3…=n(n+1)/2………..O(n^2)

i	j	Statement Execute
0	0	NA
1	O 1	1
2	0 1 2	2
3	0,1,2,3	3 and soon

6)

```
P=0;
For(i=1; p<=n; i++)
  {
    p=p+i;
   }
```

i	P
1	0+1=1
2	1+2=3
3	1+2+3=6
4	…………
K	1+2+3….k

It is repeating for sometimes, but not for n times.
Also, here I is not less than n, but p>n
Assume that p>n, it will stop
P=k(k+1)/2………………k(k+1)/2>n…………
k^2>n…..
k>root(n)………………O(root(n))

7)

```
For(i=1; i<n; i*2)
   {
     Statement;
      }
```

i=1, 2, 2^2,2^3……………2^k
Assume i>n, it terminates
Since i=2^k
2^k >=n
2^k=n
K= $\log_2 n$
O($\log_2 n$)

N=8……it will continue 3 times
N=10----it will continue 4 times, but $\log_2 10$ =3.2
We will use Floor or ceiling symbol.

8)

```
 for(i=n; i>=1; i=i/2)
   {
   sta;
  }
```

i=n, ……n/2……n/2*2……..n/2^k

Assume i<1, it will terminate

n/2^k <1

n/2^k = 1

n=2^k

k= $\log_2 n$

O($\log_2 n$)

9)

```
For(i=0; i*i<n; i++)
 {
  St;
 }
```

i*i<n
i*i>=n
i^2 = n
i= root(n)

10)

```
For(i=0; i<n; i++)
   {
    St;-----------------O(n)
   }
For(j=0; j<n; j++)
   {
    st;------------------O(n)
   }
```

O(n)

11)

```
p=o;
    for(i=1; i<n; i=i*2)
      {
        p++; -----------log n …….P=log n
       }
   for(j=1; j<p; j= j*2)
       {
       state;………log p
        }
log log n
```

12)

```
For(i=0; i<n; i++)----------------n
 {
    for(j=1; j<n; j=j*2)-----------n*log n
      {
         st;……………………….n*log n
       }
   }
2nlog n= n log n
```

TIPS TO REMEMBER

- For(i=0; i<n; i++)--O(n)
- For(i=0; i<n; i+2)--n/2…….O(n)
- For(i=n; i>1; i--)…………………………………….. O(n)
- For(i=1; i<n; i=i*2) ……………………………….O($\log_2$ n)
- For(i=1; i<n; i=i*3) ………………………………….O($\log_3$n)
- For(i=1; i<n; i=i/2)………………………………….O($\log_2$ n)

ANALYSIS OF IF AND WHILE

1)

```
i=0; …………………...1
While(i<n)………...n+1
{
Statement;……….n
i++;  ……………...n
}
```

2)

```
a=1;
While(a<b);
{
Statement;
a=a*2;
}
```

A=1
1*2=2
2*2=4
.
.
. $k=2^k$
It will terminate a>=b
$a=2^k$
$2^k>=b$
$2^k=b$
$K=\log_2 b$

3)

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

4)

```
i=1;
K=1;
While(k<n)
{
statement;
K=k+i;
i++;
}
```

i	K
1	1
2	1+1=2
3	2+2=4
. . . K	2+2+3+4---m)=m(m+1)/2 k>=n m^2>n m=root(n)

5)

```
While(m!=n)
{
If(m>n)
   m=m-n;
Else
   n=n-m;
}
```

```
Algorithm test(n)
{
        if(n<5)
        { printf("%d", n);}
        else
        {       for(i=0; i<n;i++)
                        { printf("%d", i);
                        }
        }
}
```

SPACE COMPLEXITY

Addition of 2 No.

```
Function add(n1, n2)

{

Sum =n1+n2;

Return;

}
```

Lets, integer takes 4 Bytes
n1 takes 4 bytes
n2 takes 4 bytes
Sum = 4 bytes

And some space i.e Auxiliary space = 4 bytes

Total =16 bytes
O(1) i.e Constant space

Sum of all elements in an array

```
function sum of numbers(arr[ ], n)
{sum = 0;
For (i=0 to n)
    {
    Sum= sum +arr[ ]
    }
Print(sum)
}
```

Arr= n*4
Sum = 4 bytes
i= 4 bytes
Auxiliary space = 4 bytes
Total space= 4n+12 bytes
= 4n +c
O(n)………….Linear space complexity