**TIME COMPLEXITY**

Let consider , we have 2 functions

Like f(n) and g(n) for constant c

after a particular value of constant k

* this shows Upper Bound for f(n)

Lets say f(n)=n+10

And g(n)=n

* g(n) >= c\*f(n)
* c\*n >= n+10
* n>= n+10 (for c=1) => not satisfied
  + 0 >= 10 (not possible for any value of n)
* 2\*n>= n+10 (for c=2) => not satisfied as (n should be atleast 10 else satisfied)
  + n>=10 (satisfied, possible for any value of n>=10)
    - so g(n)=2\*n , n>=10
      * where c=2,threshold k=10
    - after threshold => g(n) always greater than f(n)
* Case 2

Lets say f(n)=n^2+n+4

And g(n)=n^2

* g(n) >= c\*f(n)
* c\*(n^2) >= n^2+n+4
* 3\*n^2>= n^2+n+4 (for c=3) => satisfied
  + n>=2 (satisfied, possible for any value of n>=2)
    - so g(n)=3\*n^2 , n>=2
      * where c=3,threshold k=2
    - after threshold => g(n) always greater than f(n)

To write everything in one line

We will use Big-Oh notations

* f(n)=O(g(n))
  + Basically it checks the worst case of this function
    - i.e it shows the time complexity of worst case of your code
    - then it will take time either same or less than it

**ALGORITHM**

Suppose you are given 10 algorithm => how to find which one is best

WHICH ALGORITHM IS BEST?

An Algorithm which consumes less time and less complexity

How to figure out , which is best ?

* Just run on your system
  + i.e Experimental Analysis is not good to measure time complexity
  + Not good, as it depends on hardware specification and background processing and need to check for multiple test case
    - It is time taking and need variety of test case

We need a approach which is hardware independent, Environmental independent approach

Therefore we will use theoretical Approach

Main(){

Int n=\_\_\_\_\_\_\_

Int i= \_\_\_\_\_\_\_

While(i<=n)

{

Sop(i)

I=i=1

}

}

THEORITICAL ANALYSES:- time required to run each statement is k

Lets calculate

c1+c2+c3(n+1)+ c4n+ c5n+ c6n

(c3+c4+c5)n+ c1+ c2+ c3+c6

* f(n)=a\*n + b
* This is a function to calculate time complexity of a given algorithm
* Big – Oh of above statement f(n) is
  + - g(n)=n
    - f(n)<=cg(n)
    - a\*n + b<=(a+1)n
      * a\*n + b <=a\*n + n
      * b<=n(satisfied)
    - f(n)=O(g(n))
      * i.e time complexity of this function varies linearly with n

There could be any kind of upper bound like n^2 , n^3 etc.

But we prefer lowest of all upper bound

Case 3

Int a=2

Int b=3

Int c=a+b

Sop(c)

Time complexity is n^0\*a

O(n^0)=O(1)

this is time complexity of this code which means that ,In constant time, it will solve.

* O(n/2) is also considered as O(n)

Case 3

While(i<=n)

I=i\*2

=>>> 1 2 4 8 .. ….. 2^k

2^k=n

K=log2(n)

Case 4

While(n>0)

N, n/2^1. n/2^2, n/2^k

=>>>

n/2^k=1

2^k=n

K=log2(n)

How to calculate time complexity of recursion

Int Factorial(int n)

{

If(n==0)

Return 1;

R=n\*factorial(n-1)

Return r;

}

Lets consider recursion of factorial

T(n)=T(n-1)+c

T(n)= 1 , if n=0

T(n-1)+c, n>0

Lets find

T(n)=T(n-1)+c

T(n)=T(n-2)+2\*c

T(n)=T(n-3)+3\*c

T(n)=T(n-k)+k\*c

T(n)=1 +n\*c

Time Complexity of a factorial

T(n)=O(n)

TIME COMPLEXITY OF BINARY SEARCH

BS(){

If(l>r)

Return -1;

Mid=(l+r)/2

If(arr[mid]==elem)

Return mid;

Else if(elem < arr[mid-1])

R=BS( ARR,L,mid-1)

Else

R=BS( ARR,mid+1,r)

Return r;

}

T(n)= 1 , if n=0,1

T(n/2)+c, n>1

TIME COMPLEXITY CALCULATION

T(n)=T(n/2)+c

T(n)=T(n/2^2)+2\*c

T(n)=T(n/2^3)+3\*c

T(n)=T(n/2^k)+k\*c

T(n)=1 + c\*log(n)

Time Complexity is => O(log(n))