**Introduction to Algorithms**

**By Abdul Bari**

**What is an Algorithm?**

* set of instructions to perform a computation

**What is a Program?**

* set of instructions to perform a computation
* implemented using a programming language.

**Difference between Algorithm and Program?**

|  |  |
| --- | --- |
| **Algorithm** | **Program** |
| Developed during Design Phase | Developed during Implementation Phase |
| Requires Domain Knowledge | Requires Programming Knowledge |
| Can be written in any language | Can be written in any programming language |
| Its H/W & OS Independent | Its H/W & OS Dependent |
| Analyze the Algorithm to check its functionality | Test the Program to check its functionality |

**Difference between Priori Analysis and Posteriori Testing?**

|  |  |
| --- | --- |
| **Priori Analysis** | **Posteriori Testing** |
| Done on the Algorithm | Done on the Program |
| Language and Hardware Independent | Language and Hardware Dependent |
| **Result: -** Get Time and Space Function | **Result: -** Watch Time and Memory (bytes) |

**Characteristics of Algorithm: -**

1. **Input: -** An algorithm can take 0 or more inputs
2. **Output:** - It returns atleast 1 output
3. **Definiteness: -** Every statement must give clear meaning (only one meaning)
4. **Finiteness: - It** must have finite set of statements means it should stop at some point (Duration must be finite)
5. **Effectiveness: -** Do not write unnecessary statements as a part of the algorithm means that the statements must serve some purpose as a part of the algorithm.

**How to write an Algorithm: -**

Algorithm Swap (a, b)

Begin

Temp<-a

a<-b

b<-Temp

End

**How to analyze an Algorithm: -**

1. **Time: -** After writing an algorithm we can compute how much time the algorithm took to compute the algorithm using time function
2. **Space: -** Similar to time, we can also compute how much space does the algorithm consume
3. **Network Consumption: -** we can also compute how much data is transferred
4. **Power Consumption: -** we can also compute how much power is being consumed
5. **CPU Registers Consumption: -** For system level programming, we can also compute how much CPU Registers are being consumed for the Algorithm execution

**Note: -** To analyze an algorithm, every statement in an algorithm takes 1 unit of time

**Time Analysis Example: -**

Algorithm Swap (a, b)

{

Temp=a ---------------1

a=b--------------------1

b=Temp----------------1

}

**Total Time= 3 units.**

**f(n)=3**

This was a very basic level of analysis and complex level analysis can also be done but just to maintain the granularity of the algorithm and for understanding purposes we are using every statement is equivalent to 1 unit.

For example,

X= 5\*a + 6\*b --------- 1

We have stated as 1 but at the program level it actually will require more time depending upon the complexity for the program, but here just for the understanding purposes, we have used 1 unit for all the statements.

**Space Analysis: -**

**Variables**

A----1

B----1

Temp---1

**Total= 3 variables used**

**S(n)=3**

**Both time and space functions resulted as constant and hence we denote it as O(1).**

**Frequency Count method**

**This algorithm is useful to determine the time complexity of an algorithm.**

**Algorithm Sum(a,n)**

**{**

**S=0**

**for (i=0;i<n;i++)**

**{**

**S=s+a[i]**

**}**

**Return s**

**}**

**As we know to calculate the time complexity, we assign 1 unit for each statement but if some statement is executed n number of times, we can use frequency count method to determine the**

**number of times the statement was executed as the factor that determines the time complexity**

**of the algorithm.**

**Time Complexity using Frequency Count Method Example:-**

**Algorithm I:-**

**Algorithm Sum(a,n)**

**{**

**S=0 ---------------------------- 1**

**for (i=0;i<n;i++) ---------------------- n+1**

**{**

**S=s+a[i] ---------------------------- n**

**}**

**Return s ---------------------------- 1**

**}**

**Total time taken= 2n+3**

**F(n)=2n+3**

**Degree of Polynomial = 1, hence Order of n= O(n)//**

**Space Complexity using Frequency Count Method Exampl:-**

**Variables**

**A--- n words**

**N---1 words**

**S----1 words**

**I-----1 words**

**Total:- n+3**

**S(n)= O(n)**

**We are denoting space as words because we do not know which datatype it will hold in the program to compute the bytes and hence we are specifying words//**

**Algorithm II:-**

**Finding the Sum of Matrices:-**

**Algorithm Add(A,B,n)**

**{**

**For(i=o;i<n:i++) ---------------- (n+1)**

**{**

**For(j=0;j<n;j++) ---------------- n\*(n+1)**

**{**

**C[I,j]=A[I,j]+ B[I,j]----------- n\*n**

**}**

**}**

**Total Time= 2n^2+2n+1**

**F(n)= O(n^2)//**

**Space Complexity:-**

**Variables**

**A---- n^2**

**B---- n^2**

**C---- n^2**

**N----1**

**I-----1**

**J-----1**

**S(n)= 3n^2 +3**

**O(n^2)**

**Algorithm III:-**

**Multiplication of two Matrices:-**

**Algorithm Multiply(A,B,n)**

**{**

**For(i=0;i<n;i++) ------------------------------------- (n+1)**

**{**

**For(j=0;j<n;j++) ------------------------------------- (n)\*(n+1)--- (n^2+n)**

**{**

**C[I,j]=0 -------------------------------------------- (n\*n)--- (n^2)**

**For(k=0;k<n;k++) ------------------------------ (n)\*(n)\*(n+1)---- (n^3+n^2)**

**{**

**C[I,j]=c[I,j]+A[I,k]\*B[k,j]----------------------(n\*n \*n)----- (n^3)**

**}**

**}**

**}**

**}**

**Time Complexity:-**

**Total Time= (2n^3)+ (3n^2)+(2n)+1**

**F(n)= O(n^3)//**

**Space Complexity:-**

**A---(n^2)**

**B---(n^2)**

**C---(n^2)**

**n--1**

**I--1**

**J--1**

**K—1**

**Total Space= (3n^2) +4//**

**S(n)= O(n^2)//**

**Few examples to analyze the time complexity of the algorithm: -**

**For(i=0;i<n;i++) ---- N+1**

**{**

**Stmt; ------ N**

**}**

**Time Complexity= 2n+1 = O(n)//**

**// We only need the statement which is inside the loop because that determines how many times the statement is executing and we can avoid the loop and hence O(n)//**

**2)**

**For (i=n;i>0;i--) ------ N+1**

**(**

**Stmt; ------ N**

**}**

**Time Complexity= N= O(n)//**

**// Even in this case where the loop starts from n and comes back decrementing still it executes n times and hence O(n)//**

1. **For (i=1;i<n;i=i+2)------- N+1**

**{**

**Stmt; ------------------- N/2**

**}**

**Time Complexity = N/2 = O(n)//**

**// Even in this case where we are decrementing by 2 so the statements get reduced to half so n/2 and hence the time complexity is O(n)//**

1. **For (i=1;i<n;i=i+20)------- N+1**

**{**

**Stmt; ------------------- N/20**

**}**

**Time Complexity = N/2 = O(n)//**

**// Even if you increment it by 20 we will get n/20 and finally when we check the degree of polynomial its n and hence it is O(n)//**

**For (i=0:i<n:i++) ----- (n+1)**

**{**

**For(j=0;j<n;j++)----- (n)\*(n+1)**

**{**

**Stmt;---------------- (n)\*(n)**

**}**

**}**

**Time Complexity = O(n^2)//**

**6)**

**For (i=0:i<n:i++)**

**{**

**For(j=0;j<i;j++)**

**{**

**Stmt;**

**}**

**}**

**For this case: -**

|  |  |  |
| --- | --- | --- |
| **i** | **j** | **No of times j (inner loop executed)** |
| **0** | **0, 0<0(F)** | **0** |
| **1** | **0,1, 0<1(T), 1<1(F)** | **1** |
| **2** | **0,1,2, 0<2(T), 1<2(T), 2<2(F)** | **2** |
| **3** | **0,1,2,3, 0<3(T), 1<3(T),2<3(T), 3<3(F)** | **3** |
| **n** |  | **n** |

**Total time for i=n**

**j=n**

**Total time for (no of times) = Sum (no of times)**

**= 1+2+3+4+…. n**

**= n(n+1)/2 (Sum of first n positive numbers)//**

**Time complexity= n(n+1)/2**

**= (n^2+n)/2**

**= O(n^2)//**

**Note: - to understand,**

**For 1st element(i-0), j executed 0 times**

**For 2nd element(i=1), j executed 1 times**

**For 3rd element(i=2), j executed 2 times**

**For 4th element(i=3), j executed 3 times**

**For nth element(i=4), j executed n times**

**Also, j always re-initializes from 0//**

**P=0**

**For (i=1; p<=n; p++)**

**{**

**p=p+i;**

**}**

|  |  |
| --- | --- |
| **i** | **P** |
| **1** | **0+1=1** |
| **2** | **1+2=3** |
| **3** | **1=2+3=6** |
| **4** | **1+2+3+4=10** |
| **k** | **1=2+3+4+…. k** |

**In this case,**

**I am not going till n so when i=1, p<=n so the conditions are different so for our understanding we assume that it executed k times and hence p went till k.**

**Assume it executes until p>n,**

**Therefore,**

**P= k(K+1)/2**

**K(K+1)/2>n**

**K^2>n**

**k>**

**Hence, the time complexity = //**