**How to Analyze an Algorithm**

**Algorithm to swap the contents of 2 variables**

Algorithm Exchange(a, b)

Begin

Tmp 🡨 a ; --------------------------- 1 (unit amount of time)

a 🡨 b; ------------------------------ 1

b 🡨 Tmp; ------------------------------ 1

End

Time function (or time complexity of algorithm): F(n)=3 , is constant value

F(n)= Θ(1)

Note: (Θ(1) is used to represent any constant/fixed value like 100, 200)

We are assuming that every single statement is taking unit amount of time even if it is complex statement. We will not consider clock time.

X=4 \*a + 9\*b

Since we do our analysis at shallow(basic) level, therefore, we assume that the above statement is taking unit amount or constant amount of time; although if we carefully look at this statement, we will see that there are four operations in total. However, we ignore the detail analysis. One can do detailed analysis up to the level of machine code.

**Algorithm to sum up all the elements of Array**

Algorithm sum(A, n)

Begin

s=0; --------------------------- 1

1 times n+1 n times

For ( i = 0; i< n; i++) ; --------------------------- n+1 times(bother about loop condition)

Begin

S= s + A[i]; --------------------------- n times

end

Return s; --------------------------- 1 time

End

Time function (or time complexity of algorithm): F(n)= 2n+3 F(n)= Θ(n)

**Algorithm to find mean of all the elements of the Array**

Algorithm sum(A, n)

Begin

s=0; --------------------------- 1

n times n+1 n times

For ( i = 0; i< n; i++) ; --------------------------- n+1 times(bother about loop condition)

Begin

S= s + A[i]; --------------------------- n times

End

mean= s/n --------------------------- 1 time

Return s; --------------------------- 1 time

End

Time function (or time complexity of algorithm): F(n)= 2n+4 F(n)= Θ(n) (Consider high order terms only and skip the rest of the constants and variables)

**Algorithm to sum up the two matrices.**

Algorithm sum(A, B, n) // A and B are square matrices of dimension nxn

Begin

For ( I= 0; I< n; I++) ; --------------------------- n+1

{

For ( J = 0; J< n; J++) ; -------------------------n x (n+1) =n2 +n

{

C[i, J] = A [I, J] + B[I, J]; -------------------------n x n

}

}

Time function (or time complexity of algorithm): F(n)= 2n2 + 2n +1 F(n)= Θ(n2)

**Algorithm for multiplication of two matrices.**

Algorithm sum(A, B, n) // A and B are square matrices of dimension nxn

Begin

For ( I= 0; I< n; I++) ; ---------------------------------n+1

{

For ( J = 0; J< n; J++) ; -----------------------------n x (n+1) =n2 +n

{

C[i, J]=0;

For ( K = 0; K< n; K++) ; -------------------------n x n x (n+1) =n3 +n2

{

C[i, J] = C[i, J] + A[I, K] x B[K, J]; -------------------------n x n x n = n3

}

}

}

Time function (or time complexity of algorithm): F(n)= 2n3 + 2n2 + 2n+1 F(n)= Θ(n3)

**Analysis Piece of Code:**

For ( I= n; I > 0; I--) ---------------------------------n+1 (we will ignore this)

{

Stmt; ---------------------------------n times

}

(In analysis, we consider number of times the loop body get executed, sometimes we will ignore the number of times the loop condition get executed).

Time function (or time complexity of algorithm): F(n)= 2n +1 F(n)= Θ(n)

**Analysis Piece of Code:**

For ( I= 0; I < n; I = I +2)

{

Stmt; ---------------------------------n/2 times

}

(In analysis, we consider number of times the loop body get executed, sometimes we will ignore the number of times the loop condition get executed).

Time function (or time complexity of algorithm): F(n)= n/2 F(n)= Θ(n)

Lets take n=8, loop condition (I < 8)will be true for i= 0, 2,4,6 and false for i=8; so the loop body will be executed 4 times, which is equal to 8/2 or n/2.

**Analysis Piece of Code:**

For ( I= 0; I < n; I = I +20)

{

Stmt; --

-------------------------------n/20 times

}

(In analysis, we consider number of times the loop body get executed, sometimes we will ignore the number of times the loop condition get executed).

Time function (or time complexity of algorithm): F(n)= n/20 F(n)= Θ(n)

**Analysis Piece of Code:**

For ( I= 0; I < n; I ++)

{

For ( J= 0; J < i; J++)

{

stmt;

}

We do not know how many times the loop body is executed because the condition in the inner most for loop is not straightforward. So, Lets analyze the algorithm by tracing

|  |  |  |  |
| --- | --- | --- | --- |
| I | J | Condition of inner most for loop  J < I | No of times loop body executed |
| 0 | 0 | false | 0 times  Sum up |
| 1 | 0 | true | 1 time |
| 1\* | false |
| 2 | 0 | true | 2 times |
| 1 | True |
| 2\* | false |
| 3 | 0 | True | 3 times |
| 1 | True |
| 2 | True |
| 3\* | false |
| **:** | **:** | **:** | **:** |
| n |  |  | n times |

We assume that the outer most for loop is executed up to n times, so sum up the no. of times the loop body get executed, that is 1+2+3………………n = n(n+1)/2

Time Complexity f(n)= n(n+1)/2 = (n2 + n) /2

F(n)= Θ(n2)

**Analysis Piece of Code:**

**P=0;**

For ( I= 1; P < = n; I ++)

{

P=P+I;

}

We do not know how much times the loop body is going to execute because the condition (I<=n) is missing. So, Lets analyze the algorithm by tracing

|  |  |  |  |
| --- | --- | --- | --- |
| I | P | P < =n | P = P + I |
| 1 | 0 | true | 0 +1 P=1 |
| 2 | 1 | True | 1+ 2 P=3 |
| 3 | 3 | True | 1+2 +3 P=6 |
| 4 | 6 | true | 1+2 +3+4 P=10 |
| : | : | : | : |
| :  :  : Assume that loop body is going to execute for k times, lets find K and loop will stop when P>n  : | | | |
| K |  |  | 1+2 +3+4 -----K |

Assume P > n (loop will terminate at this condition)

If K is the last iteration of the loop, then P = 1+2 +3+4 -----K = K(K+1)/2

K(K+1)/2 > n

Rougly assume K(K+1)/2 = K2

=> K2 > n => K > Γn

So Time Complexity= Θ (Γn)