COURSE HANDOUT

Course Code	ACSC13
Course Name	Design and Analysis of Algorithms
Class / Semester	IV SEM
Section	A-SECTION
Name of the Department	CSE-CYBER SECURITY
Employee ID	IARE11023
Employee Name	Dr K RAJENDRA PRASAD
Topic Covered	Time complexity and Examples
Course Outcome/s	Compute the time and analyse performance of algorithm in terms of time
Handout Number	9
Date	3 April, 2023

Content about topic covered: Time complexity and Examples

Time complexity:

The time complexity of an algorithm is the amount of computer time it needs to run to completion.

The time T(P) taken by a program P is the sum of the compile time and the run (or execution)time. The compile time does not depend on the instance characteristics.

Therefore, the time complexity of an algorithm is determined by the number of steps it requires to compute the function for which it was constructed.

There are two methods we can figure out how many steps a program needs to take in order to solve a specific problem.

1. Introduction of global variable called count.
2. To build a table in which we list the total number of steps contributed by each statement.

Examples

1. Count method:

Eg 1. Algorithm abc(a, b, c)

{

```
        return a + b + b * c + (a + b - c)/(a + b) + 4.0;
        count = count +1;
    }
```

The time complexity is O(1).

Eg 2.

```
Algorithm Sum(a,n)
{
    s:=0.0; count = count +1;
    for i :=1to n do
    {
        count = count +1;   // for 'for loop'
        s :=s+ a[i]; count = count +1;
    }
    count = count +1;       // for last time of 'for loop'
    count = count +1;       // for the return statement
    return s;
}
```

So, the total steps required = 2 n + 3;

Time complexity = O(n)

Eg 3.

```
Algorithm   RSum(a,n)
{
    count = count +1;       // for the if condition
    if (n ≤ 0) then
    {
        count = count +1;       //for the return statement
        return 0.0;
    }
```

```
        else
        {
                count = count +1;        //function invocation
                return RSum (a,n-1)+ a[n];
        }
}
```

When analyzing a recursive program for its step count, we often obtain a recursive formula for the step count, for example

$$tRsum(n) = \begin{cases} 2 & if\ n = 0 \\ 2 + \text{tRsum}(n-1) & if\ n > 0 \end{cases}$$

$$\begin{aligned} t_{Rsum}(n) &= 2 + t_{Rsum}(n-1) \\ &= 2 + 2 + t_{Rsum}(n-2) \\ &= 2(2) + t_{Rsum}(n-2) \\ &= 2(2) + 2 + t_{Rsum}(n-3) \\ &= 2(3) + t_{Rsum}(n-3) \\ &\vdots \\ &= 2(n) + t_{Rsum}(n-n) \\ &= 2(n) + t_{Rsum}(0) \\ &= 2n + 2 \end{aligned}$$

So, the time complexity = O(n).

2. Table Method

An algorithm's step count is computed by first calculating the total number of times (frequency) that each statement is executed overall, as well as the number of steps required for its execution (s/e). The s/e of a statement is the amount by which the count differs from the original value as a result of the statement's execution. These two numbers are multiplied to determine the contribution of each statement as a whole. The total number of steps for the entire algorithm can be found by totalling the from each statement.

Eg 1. Sum of elements of an array.

Statement	s/e	Frequency	Total steps
1. Algorithm Sum(a, n)	0	-	0
2. {	0	-	0
3. s:=0.0;	1	1	1
4. for i :=1 to n do	1	n + 1	n +1
5. s:=s+ a[i];	1	n	n
6. return s;	1	1	1
7. }	0	-	0
Total			2 n + 3

Time complexity = O (n).

Eg 2. Matrix multiplication.

Statement	s/e	Frequency	Total steps
1. Algorithm Add (a, b, c, m, n)	0	-	0
2. {	0	-	0
3. for i :=1 to m do	1	m +1	m +1
4. for j :=1 to n do	1	m(n+1)	mn + m
5. C[i,j] := a[i,j] +b[i,j];	1	mn	mn
6.	0	-	0
Total			2mn +2m +1

Time complexity = O (mn).