Advanced Algorithms & Data Structures

Session -12

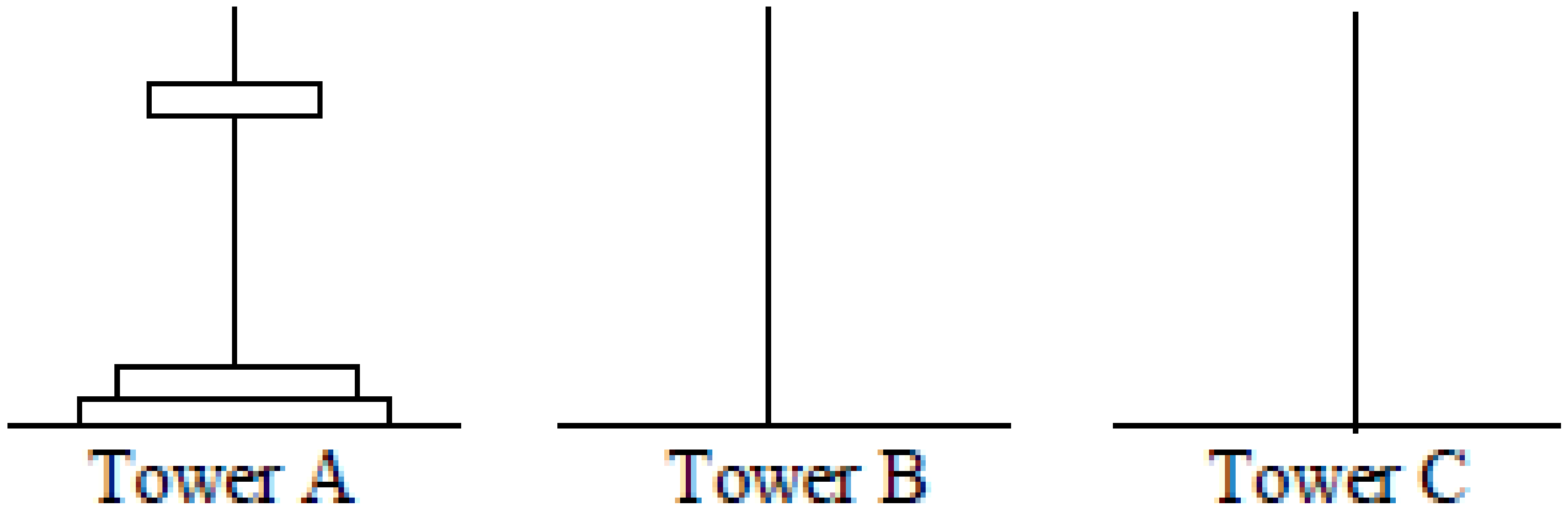
Performance Analysis of recursive and non-recursive algorithms

Recursive Algorithms:

- A recursive function is a function that is defined in terms of itself. Similarly an algorithm is said to be recursive if the same algorithm is invoked in the body. An algorithm that calls itself is direct recursive.

- Algorithm A is said to be indirect recursive if it calls another algorithm which in turn calls A. These recursive mechanisms are extremely powerful, and also it can express a complex process very clearly.

Example: Towers of Hanoi problem



```
Algorithm TowersOfHanoi(n, x, y, z)
{
    If (n ≥ 1) then
    {
        TowersOfHanoi(n-1, x, z, y);
        Write("move top disk from tower" , x, "to top of tower", y);
        TowersOfHanoi(n-1, z, y, x);
    }
}
```

Performance Analysis:

There are two major criteria for judging algorithms that have a more direct relationship to performance.

computing time(Time complexity)

storage requirement (Space complexity)

Time Complexity :

Is the amount of time it needs to run to completion

- The time, $T(P)$, taken by a program P, is the sum of its compile time C and its run (or execution) time, $T_P(I)$.

$$T(P)=C+T_P(I)$$

- The compile time does not depend on the instance characteristics.
- We will concentrate on estimating run time $T_p(I)$.

Methods to compute Time Complexity

1) Step Count Method

2) Tabular Method

- **Step Count Method**
  - – Introduce global variable count into programs with initial value zero.
    - Statements to increment count by the appropriate amount are introduced into the program.
    - The value of the count by the time program terminates is the number steps taken by the program.

Method-I: Introduce variable count

EX:- 1) Iterative sum of n numbers

```
Algorithm sum(a,  n)
{
 s:=0;

  for i:=1 to n do
        {
      s:=s+a[i];

        }

    return s;
  }
```

EX:- 2) Recursive sum of n numbers

```
Algorithm RSum(a,n)
{
        // for the if conditional
        if(n ≤ 0) then
        {
                return 0;

        }
        else
        {
                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.
- We obtain the following recursive formula for above (RSum) algorithm.

$$t_{RSum}(n) = \begin{cases} 2 & \text{If } n=0 \\ 2 + t_{RSum}(n-1) & \text{If } n>0 \end{cases}$$

One way of solving such recursive formula is by using substitution method.

$t_{RSum}(n) = 2+t_{RSum}(n-1)$

$= 2+2+t_{RSum}(n-2)$

$= 2(2)+t_{RSum}(n-2)$

$= 2+2+2+_{tRSum}(n-3)$

$\mathbf{= 3(2)+t_{RSum}(n-3)}$

:

:

$= n(2) +t_{RSum}(n-n)$

$=2n+t_{RSum}(0)$

$= 2n+2$

The step count for Rsum is 2n+

Tabular method

- Determine the total number of steps contributed by each statement per execution × frequency

- Add up the contribution of all statements

Method-II: Tabular method

- EX:- 1) Iterative sum of n numbers

Statement	s/e	frequency	Total steps
Algorithm sum(a, n)	0	--	0
{	0	--	0
s:=0 ;	1	1	1
for i:=1 to n docc	1	n+1	n+1
s:=s+a[i];	1	n	n
return s;	1	1	1
}	0	--	0
Total			2n+3

EX:- 2) Recursive sum of n numbers

- EX:- 2) Recursive sum of n numbers

Statement	s/e	Frequency		Total steps	
		n=0	n>0	n=0	n>0
Algorithm RSum(a,n)	0	--	--	0	0
{	0	--	--	0	0
if(n ≤ 0) then	1	1	1	1	1
return 0;	1	1	0	1	0
else					
return Rsum(a,n-1)+a[n] ;	1+x	0	1	0	1+x
}	0	--	--	0	0
Total				2	2+x

$x = t_{RSum}(n-1)$

Space Complexity

- Amount of computer memory that is required during program execution as a function of input size.

OR

- Space complexity is the amount of memory it needs to run to completion

Space=Fixed part +Variable part

Fixed: varies from problem to problem. Includes space needed for storing instructions variables , constants and structured variables.[arrays , struct].

Variable: varies from program to program. Includes space needed for stack and for structured variables that are dynamically allocated during run time.

$$S(P)=C+SP(I)$$

Fixed Space Requirements (C)

Independent of the characteristics of the inputs and outputs

- instruction space(space for code)
- space for simple variables, fixed-size structured variable, constants

Variable Space Requirements (SP (I))

depend on the instance characteristic I

- number, size, values of inputs and outputs associated with I
- recursive stack space, formal parameters, local variables, return address

Example1

```
Algorithm abc(a,b,c)
{
        return a+b+b*c+(a+b-c)/(a+b)+4.0;
}
```

Problem instance characterized by the specific values of a,b and c.

If we assume one word (4 bytes) is adequate to store the values of each a, b, and c , then the space needed by abc is independent of the instance characteristics.

Therefore, $S_{abc}(\text{ instance characteristics})=0$

Example2

```
Algorithm sum(a,n)
{
        s:=0;
  for i:=1 to n do
            s:=s+a[i];
         return s;
}
```

The amount of space needed depends on the value of n.

Therefore, $S_{sum}(n) >= (n+3)$

Example3

```
Algorithm RSum(a,n)
{
        if(n ≤ 0) then return 0;
        else return RSum(a,n-1)+a[n];
}
```

Total no.of recursive calls n, therefore **$S_{RSum}(n) >= 3(n+1)$**

Find time and space complexity for the following algorithm

```
Algorithm Add(a, b, c, m, n)
{
  for i:=1 to m do
  {
                           for j:=1 to n do
                           {
                               c[i,j]:=a[i,j]+b[i,j];
                           }
  }
}
```

SAMPLE QUESTIONS

- Define Performance Analysis
- Differentiate recursive and non-recursive algorithms
- Provide an example algorithm that uses recursion.
- Explain in detail about Towers of Hanoi problem
- Solve Towers of Hanoi problem using 3 disk
- Solve Towers of Hanoi problem using 5 disk
- Write an algorithm for Towers of Hanoi problem using recursion
- Write an algorithm for Towers of Hanoi problem using iterative method
- Explain in detail about step-count method

SAMPLE QUESTIONS Cont..

- Explain in detail about tabular method
- Why is it important to consider the step count of an algorithm when analyzing its efficiency
- Discuss the role of data structures in determining the step count of an algorithm.
- Explain the concept of nested loops and their impact on the step count of an algorithm
- Explain the process of creating a table using the tabular method to analyze an algorithm.
- Discuss the relationship between the tabular method and the Big O notation.