Page No.
Date: / /

Tutorial No. 1

Q.1. Define Algorithm. Explain different characteristics of it.

Algorithm - An algorithm is a finite set of instructions that, if fallowed, accomplishes a particular task.

Also, algorithm is defined as, it is a well defined set of computational statements that takes 0 or more inputs and produce at least one output or solution of the problem within a finite amount of time.

Different characteristics of algorithm are as follow:

1. Input - zero or more quantities are externally supplied.

2. Output - At least one quantity is produced.

3. Definiteness - Each instruction is clear and unambiguous.

4. Finiteness - If we trace out the instructions of an algorithm then for all cases, the algorithm terminates after a finite number of steps.

5. Effectiveness - Every instruction must be basic enough to be carried out in principle, by a person using only pencil and paper. It is not enough that each operation be definite as in 3 it also must be feasible.

Q.2. Explain briefly specifications of algorithm

Specifications of algorithm are as follows:

1. Comment - It is used to describe statement of program.
   // ——, /* —— */

2. Block - { } - It is the body of looping statement. Function if, whileloop are written in the block.

3. Identifier - It is the name given to the variable. There are some rules for identifiers. Some are as follows:

i. You cannot use keywords as identifiers.

ii. The first letter of an identifier should be either a letter or an underscore

iii. A valid identifier can have letters, digits and underscore

4. Assignment statement

The assignment statement is indicated by placing equal(=) between the variable (in left hand side) and expression or value(s) (in right hand side). For example, addition of a and b is assigned in variable a

a = a+b

5. Boolean

It species either 0 or 1 or true or false.

6. Array - Array is used in algorithm with the help of square bracket

7. Looping statement

These statements are used when there is a need of some statements to be executed number of times. These statements also called iterative control statements.

```
a) while (condition) do
   {
   statement(s)
   }
```

when condition is true, statement within while loop executes. condition is false, while loop executes.

```
b) for variable = start value to final value step increment
        value do
     {
       statement (s)
     }
```

```
c) repeat
     {
      statement(s)
     } until (condition)
```

This repeat loop is executed until the condition is false and exited when condition becomes true.

8. Conditional operators

If statement has one of the following two forms:

```
a) if condition
   then
   statement (s)
b) if condition
   then
   statement (s)
   else
   statement (s).
```

In general case statement has the form:

```
select case (expression)
     case value 1 : statement (s)
     case value 2 : statement(s)
          :
```

case value n : statement (s)
default :

9. Read & write operation
To read / write text file, image file, video file using read & write operation.

10 Algorithm Name of Algorithm (parameter list)
```
{ - - - -
  - - - -
  - - -
}
```

Q.3 what is mean by time complexity and space complexity?

Time complexity

1. It is amount of computer time required by an algorithm to provide a solution or result
2. It is the sum of compile time and run time. The compile time does not depend on the instance characteristics
3. There are two methods to find time complexity
   - Introduce new variable count
   - Frequency table

Space complexity

1. It is the amount of storage space required by algorithm to produce solution or result
2. It is a fixed part that is indepedent of the characteristics of the inputs and outputs.
3. This part typically includes the instruction space, space for simple variables and fixed-size component variables, space for constants, etc.
4. The space requirement S(P) of any program P is written as S(p) = C + Sp where c is a constant

Determine time complexity of following algorithm :

```
Algorithm Sum (P,n)             ————————   0
{
  for (i=n; i>=1; i/2)          ————————   log n
  {
   printf("%d", i);             ————————   log n
  }                                        ——————
}                                          2 log n
```

Time complexity - O(log n)

~~Space complexity~~ -

Determine space complexity of following algorithm

```
 factorial (N)
 {
  if (N<=1)
  {
   return 1;
  }
  else
  {
   return (N * factorial (N-1));
  }
 }
```

~~Time complexity~~ -

Space complexity - O(n)

Q.4 Explain Asymptotic notations with examples

Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

There are 3 types to denote asymptotic notation.

1. Big-O Notation (O-notation)

Big-O notation represents the upper bound of the running time of an algorithm. Thus, it gives the worst-case complexity of an algorithm

$O(g(n)) = \{ f(n)$: there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0 \}$

e.g. If $f(n) = 2n+3$

$2n+3 \le 10n$ for $n=1$

$5 \le \underset{c}{\underset{\uparrow}{10}}\,\underset{g(n)}{\underset{\uparrow}{n}}$

2. Omega Notation (Ω-notation)

Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm

$\Omega(g(n)) = \{ f(n)$: there exist positive constants c and n_0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0 \}$

e.g. If $f(n) = 2n+3$

$2n+3 \ge \log n$ for $n=1$

$5 \ge 0$

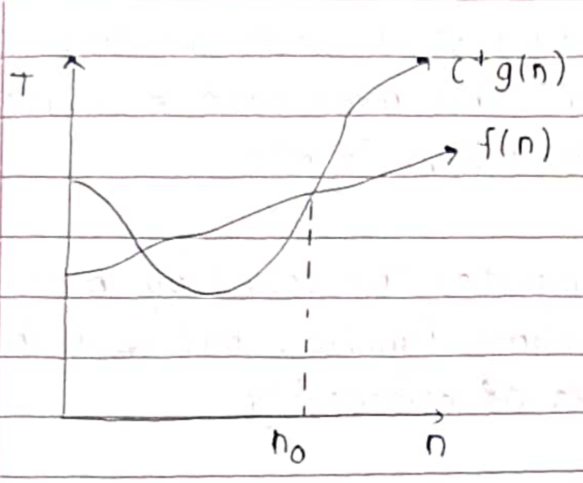
3. Theta Notation (Θ-notation)

Theta notation encloses the function from above and below since it represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm

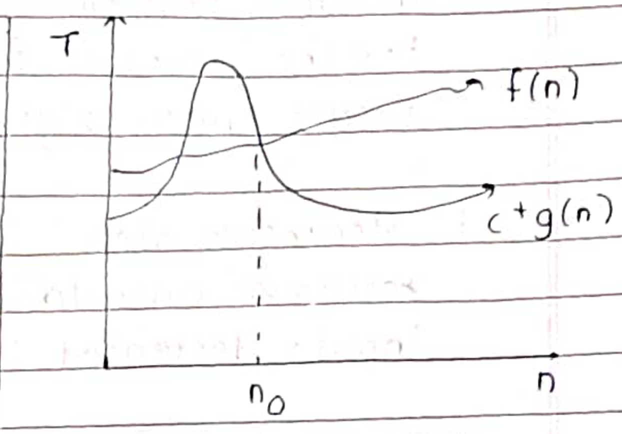
$\Theta(g(n)) = \{ f(n)$: there exist positive constants c_1, c_2 and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0 \}$

consider $f(n) = 2n+3$

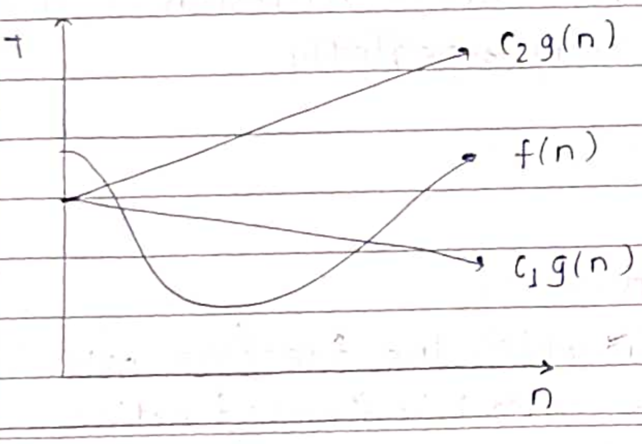
$\underset{c_1}{1} \cdot \underset{g(n)}{\log n} \le \underset{f(n)}{2n+3} \le \underset{c_2}{n} \, \underset{g(n)}{\log n}$



The graph of Big-O notation



The graph of Ω notation



The graph of Θ notation

Q.5. Explain what is meant by recursive algorithm & write two examples of it (Algorithm of Fibonacci series & Tower of Hanoi.)

1. A recursive algorithm calls itself with smaller input values and returns the result for the current input by carrying out basic operations on the returned value for the smaller inputs.
2. If a problem can be solved by applying solutions to smaller versions of the same problem and the smaller

versions shrink to readily solvable instances, then the problem can be solved using a recursive algorithm

3. There are two cases :

i. Base case : It is nothing more than the simplest instance problem consisting of a condition that terminates the recursive function. This base case evaluates the result when a given condition is met.

ii. Recursive step : It computes the result by making recursive calls to the same function but with the inputs decreased in size or complexity.

4. There are two types of recursion :

i. Direct Recursion :

A function is called direct recursion if it calls itself in its function body repeatedly

```
A() {
     A()
   }
```

ii. Indirect Recursion :

The recursion in which the function calls itself via another function is called indirect

```
A() {
     B()
  }
B() {
     A()
  }
```

Fibonacci series Algorithm

The fibonacci sequence is the series number :
0, 1, 1, 2, 3, 5, 8, 13, 21, - - - -

The next number is found by adding up the two numbers before it.

```
void fibonacci (int n)
{
    int a[n+1];
    a[0] = 0;
    a[1] = 1;
    if ( n = = 1 || n = = 2)
        return 1
    else
        return ( fibonacci (n-1) + fibonacci (n-2))
}
```

Tower of Hanoi Algorithm

```
Algorithm Tower-of-Hanoi (n, x, y, z)
{
  if (n ≥ 1) then
  {
    Tower-of-Hanoi ( n-1, x, z, y )
    move ( n, x, y, z )
    Tower-of-Hanoi (n-1, z, y, x )
  }
}
```

Q.6. Solve following recurrence relation using substitute method.

1 $T(n) = 4T(n/2) + n^2$

$T(n) = 4T(n/2) + n^2$ ——— (1)

$T(n/2) = 4T(n/4) + (n/2)^2$

$T(n/2) = 4T(n/4) + (n/2)^2$ ——— (2)

$T(n/4) = 4T(n/8) + (n/4)^2$ ——— (3)

$T(n) = 4\left[4T(n/4) + (n/2)^2\right] + n^2$

$= 16T(n/4) + 4(n^2/4) + n^2$

$$T(n) = 16\,T(n/4) + 2n^2$$

$$= 16\left[4T(n/8) + (n/4)^2\right] + 2n^2$$

$$= 64\,T(n/8) + 3n^2$$

$$= 4^3\,T(n/2^3) + 3n^2$$

$$= 4^K\,T(n/2^K) + Kn^2$$

Let $\frac{n}{2^K} = 1$

$$n = 2^K$$

$$\log_2 n = K$$

$$T(n) = 4^{\log_2 n}\,T\left(\frac{n}{2^{\log_2 n}}\right) + \log_2 n \cdot n^2$$

$$T(n) = 4^{\log n}\,T(1) + \log n \cdot n^2$$

$$= O(n^2)$$

2. $T(n) = T(n-1) + n$

$$T(n) = T(n-1) + n \quad \text{——— (1)}$$

$$T(n-1) = T(n-2) + n-1 \quad \text{——— (2)}$$

$$T(n-2) = T(n-3) + n-2 \quad \text{——— (3)}$$

Substitute eq (2) in eq (1)

$$T(n) = T(n-2) + n-1+n$$

$$T(n) = T(n-2) + 2n-1$$

From eq (3) and eq (2)

$$T(n) = T(n-3) + n-2 + 2n-1$$

$$T(n) = T(n-3) + 3n-3$$

$$T(n) = n + (n-1) + (n-2) + \cdots + 3 + 2 + 1$$

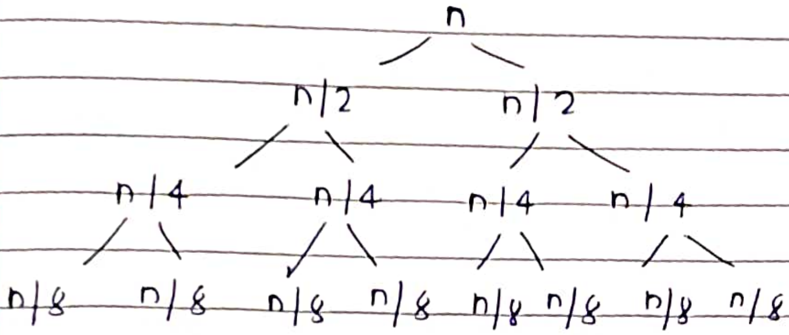
$$T(n) = \frac{n(n+1)}{2}$$

$$T(n) = \frac{n^2+n}{2}$$

$= o(n^2)$

Q.7. Solve following recurrence relation using recursive tree method

1. $T(n) = 2T(n/2) + n$



for n values

$\log 2^n$

$= \log 2^n * cn$ → constant

$= o(n \log_2 n)$