Chapter 2

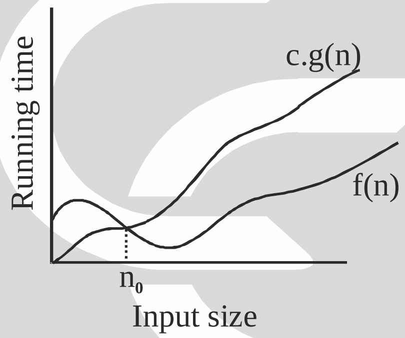
Asymptotic Notation

Goal : To simplify the analysis of running time by getting rid of "details" which may be affected by specific implementation and hardware.

The "Big Oh" (O-Notation) : It gives us the upper bound (Worst case behaviour) of the running time.
If f(n) and g(n) are two increasing functions on non-negative numbers then.

f(n) = O(g(n)) if there exists constant c and $n \geq n_0$ such that $f(n) \leq c \cdot g(n) \; \forall \; n \geq n_0$.

where c and n_0 are two positive constants and $c > 0$ & $n_0 \geq 1$



Example: If $f(n) = 60n^2 + 5n + 11$ then
$f(n) < 60n^2 + 5n^2 + 11n^2$

$\Rightarrow \quad f(n) < 76n^2$ where c = 76

$< 76\, n^2$.

$\therefore \quad n > n_0 = 1$

$\therefore \quad f(n) = O(n^2)$

Simple Rule: Pop lower order terms and constant factors.

e.g. $-5n^2 + 6n + 20$ is $O(n^2)$

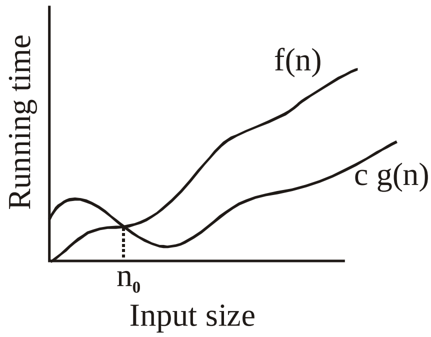
and $-5n^2 \log n + n + 7$ is $O(n^2 \log n)$

Omega Notation (Ω) : It gives the tight bound (Lower bound) on running time. If f(n) and g(n) are two increasing functions over non-negative numbers. Then

$f(n) = \Omega(g(n))$ if there exists a constant c and $n \geq n_0$

such that $f(n) \geq c\, g(n) \;\; \forall \; n \geq n_0$

and $c > 0$ & $n_0 \geq 1$



For example: $f(n) = 60n^2 + 5n + 11 > 60n^2 \ \forall \ n > 1$

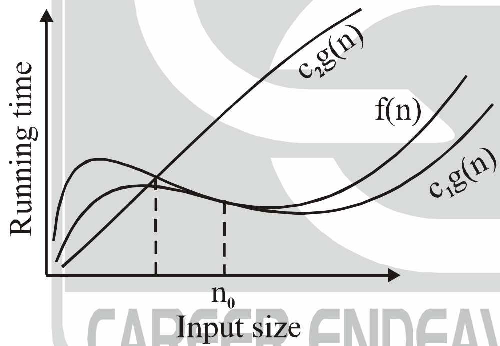
Therefore, $f(n) = \Omega(n^2)$ where c = 60 and $n > n_0 = 1$

Theta Notation (θ) : It gives the tight bound (Average case behaviour) on running time. If f(n) and g(n) are two increasing function defined over non-negative numbers then

$f(n) = \theta(g(n))$ if there exists constants c_1, c_2 and $n \geq n_0$ such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \ \forall \ n \geq n_0$$

f(n) is sandwidthed between $c_1 g(n)$ and $c_2 g(n)$



If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Then $f(n) = \theta(g(n))$ and vice-versa.

As we have, $f(n) = 60n^2 + 5n + 11$ and $f(n) = O(n^2)$ and $f(n) = \Omega(n^2)$

Therefore, $f(n) = \theta(n^2)$

Little O Notation (o) :

- $f(n) = o(g(n))$

$g(n)$ is strictly (do not consider equality) larger than $f(n)$

- $f(n) < g(n) \ \forall \ n \geq n_0$

- Graphical representation remain same.

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Little Omega (ω):

- $f(n) = \omega(g(n))$

$g(n)$ is strictly smaller than $f(n)$

- $f(n) > (g(n)) \; \forall \; n \geq n_0$
- Graphical representation remain same.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

Types of time complexitites:

(i) constant time complexity $\rightarrow$ $O(1)$

(ii) logarithmic time complexity $\rightarrow$ $O(\log n)$

(iii) linear time complexity $\rightarrow$ $O(n)$

(iv) quadratic $\rightarrow$ $O(n^2)$

(v) cubic $\rightarrow$ $O(n^3)$

(vi) polynomial $\rightarrow$ $O(n^k) \;\; k > 0$

(vii) exponential $\rightarrow$ $O(c^n) \;\; c > 1$

(1) $1 < \log\log n < \sqrt{\log n} < \log n$

(2) $n \log n < n\sqrt{n}$

(3) $2^n < n^n$

(4) $2^n < n^{\log n}$

(5) $n! < n^n$

(6) $n! < 2^n$

(7) $2^n < n! < n^n$

(8) $n > (\log)^{100}$

(9) $n < (\log n)^n$

(10) $n < (\log n)^{\log n} < (\log n)^n$

(11) $n^{\log_b{}^a} = a \log_b{}^n$

Note :
$\boxed{\begin{array}{l} n! = O(n^n) \text{ correct} \\ \text{n!=o}(n^n) \textit{ exactly correct} \end{array}}$

Properties of Asymptotic Notation:

(1) Relfexive properties:

(i) $f(n) = O(f(n))$

(ii) $f(n) = \Omega(f(n))$

(iii) $f(n) = \theta(f(n))$

(iv) $f(n) \neq o(f(n))$

(v) $f(n) \neq \omega(f(n))$

(2) Symmetric properties:

(i) if $f(n) = O(g(n))$ then $g(n) \neq O(f(n))$

(ii) if $f(n) = \Omega(g(n))$ then $g(n) \neq \Omega(f(n))$

(iii) if $f(n) = \theta(g(n))$ then $g(n) = \theta(f(n))$

(3) Transitive property:

(i) If $f(n) = O(g(n))$ && $g(n) = O(h(n))$ then $\boxed{f(n) = O(h(n))}$

(ii) If $f(n) = \Omega(g(n))$ && $g(n) = \Omega(h(m))$ then $f(n) = \Omega(h(n))$

(4) If $f(n) = O(g(n))$ then $h(n) \cdot f(n) = O(h(n) \cdot g(n))$

where, $h(n)$ is positive function.

(5) If $f(n) = O(g(n))$ $d(n) = O(h(n))$ then

(i) $f(n) + d(n) = \max(g(n) \cdot h(n)) = O(g(n) + h(n))$

(ii) $f(n) \cdot d(n) = O(g(n) \cdot h(n))$

Problem : Consider the following z functions.

$f(n) = O(g(n))$ && $g(n) \neq O(f(n))$

$g(n) = O(h(n))$ && $h(n) = O(g(n))$ True/False

(a) $f(n) + g(n) = O(h(n))$

(b) $f(n) = O(h(n))$

(c) $f(n) \cdot g(n) = O(h(n) \cdot h(n))$

(d) $h(n) - O(f(n))$

Find the Time complexity?

```
main ( )
{
    x = y + z;
    for (i = 1; i ≤ n; i + + )          (n time exeuction)
        x = y + z;
    for (i = 1; i ≤ n; i + +)
    {
```

for $\left(j=1;\ j\le\frac{n}{2};\ j++\right)$

{

$x=x+z;$ $\left(n\times\frac{n}{2}\text{ time excution}\equiv\frac{n^2}{2}\right)$

}

}

} so, $O(n^2)$.

Find complexity.

main ( )

{

$x=y+z$

for $(i=1;i\le n;i++)$

{

for $(j=1;\ j\le i;\ j++)$

{

$x=y+z$ $(1+2+3+\ldots\ldots+n)=\frac{n(n+1)}{2}$

}

}

} so, $O\left(n^2\right)$

Difference between Relative Analysis and Absolute Analysis

Relative Analysis :

(1) It is software and hardware dependent analysis.

(2) Exact answers

(3) Answer is changing from system to system.

Absolute analysis:

(1) It is software and hardware independent analysis.

(2) Approximate answers

(3) Answer will be same in all computer systems.

Note : Absolute analysis is the standard analysis in practice.

Absolute analysis : It is a determination of order of magnitude of a statement

Number of times a statement is executing.

Example :

main ( )

{

$x=y+z;\ \Rightarrow$ 1 time execution

O(1) or order of 1.

}

SOLVED PROBLEMS

1. Solve the recurrence equations
$T(n) = T(n-1) + n$
$T(1) = 1$

[GATE-1987 : 2 Marks]

Soln. By using substitution method we get following series:

$n + (n-1) + (n-2) + (n-3) \ldots 3 + 2 + 1$

Which is sum of 'n' natural numbers.

$$= \frac{n(n+1)}{2} \Rightarrow O(n^2)$$

2. What is the generating function G(z) for the sequence of Fibonacci numbers ?

[GATE-1987 : 2 Marks]

Soln. The generating function for the Fibonacci numbers G(z) is $G(z) = \dfrac{z}{1 - z - z^2}$.

3. $\sum_{1 \le k \le n} O(n)$, where O(n) stands for order n is

(a) $O(n)$ (b) $O(n^2)$ (c) $O(m^3)$ (d) $O(3n^2)$

[GATE-1993 : 2 Marks]

Ans. (b)

Soln. $$\sum_{1 \le k \le n} O(n) = O(1) + O(2) + O(3) + \ldots + O(n) = O\left(\frac{n(n+1)}{2}\right) = O(n^2)$$

4. Consider the following two functions:

$$g_1(n) = \begin{cases} n^3 & \text{for} \quad 0 \le n \le 10{,}000 \\ n^2 & \text{for} \quad n \ge 10{,}000 \end{cases} \qquad g_2(n) = \begin{cases} n & \text{for} \quad 0 \le n \le 100 \\ n^3 & \text{for} \quad n > 100 \end{cases}$$

Which of the following is true ?

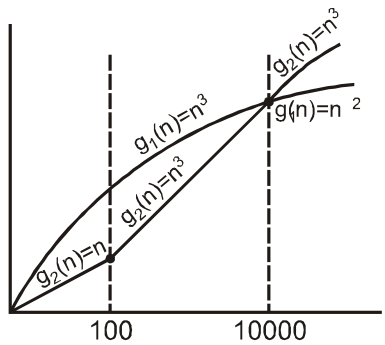
(a) $g_1(n)$ is $O(g_2(n))$ (b) $g_1(n)$ is $O(n^3)$

(c) $g_2(n)$ is $O(g_1(n))$ (d) $g_2(n)$ is $O(n)$

[GATE-1994 : 2 Marks]

Ans. (a)

Soln.



Therefore;

$n^2 \le n^3$ for $N \ge 10000$

$g_1(n) = O(g_2(n))$

Correct option is (a).

5. Which of the following is false?

(a) $100n \log n = O\left(\frac{n \log n}{100}\right)$

(b) $\sqrt{\log n} = O(\log \log n)$

(c) If $0 < x < y$ then $n^x = O(n^y)$

(d) $2n \neq O(nk)$

[GATE-1996 : 1 Mark]

Ans. (b, d)

Soln. (a) We know that

$f(n) = O(g(n))$ i.e., $f(n) \leq k.g(n)$

for k, some positive integers and $n > n_0$

$$100 \text{ n} \log n \leq 10000 \times \frac{n \log n}{100}$$

for k = 10000

$$\therefore \ 100 \text{ n} \log n = O\left(\frac{n \log n}{100}\right)$$

(b) $\sqrt{\log n} \leq 1 * \log \log n$

$\therefore \ \sqrt{\log n} \neq O(\log \log n)$

(c) $n^x \leq n^y$ as $0 < x < y$

$\therefore \ n^x = O(n^y)$

(d) $2n \leq kn$ for $k \geq 2$

$\therefore \ 2n = O(nk)$

6. The concatenation of two lists is to be performed in O(1) time. Which of the following implementations of a list should be used?

(a) singly linked list

(b) doubly linked list

(c) circular doubly linked list

(d) array implementation of list

[GATE-1997 : 1 Mark]

Ans. (c)

Soln. As list concatenation requires traversing at least one list to the end. So singly linked list and doubly linked requires O(n) time complexity whereas circular doubly linked list required O(1) time.

7. Let $f(n) = n^2 \log n$ and $g(n) = n(\log n)^{10}$ be two positive functions of n. Which of the following statements is correct ?

(a) $f(n) = O(g(n))$ and $g(n) \neq O(f(n))$

(b) $g(n) = O(f(n))$ and $f(n) \neq O(g(n))$

(c) $f(n) \neq O(g(n))$ and $g(n) \neq O(f(n))$

(d) $f(n) = O(g(n))$ and $g(n) = O(f(n))$

[GATE-2001 : 1 Mark]

Ans. (b)

Soln.

$f(n) = n^2 \log n$

$g(n) = n(\log n)^{10}$

$n(\log n)^{10} \leq n^2 \log n$

$\therefore \ g(n) = O(f(n))$

Whereas

$f(n) \neq O(g(n))$ because

$n^2 \log n \not\leq n(\log n)^{10}$

8. In the worst case, the number of comparisons needed to search singly linked list of length n for a given elements is

(a) $\log_2 n$ (b) n/2 (c) $\log_2 n - 1$ (d) n

[GATE-2002 : 1 Mark]

Ans. (d)

Soln. Worst case of searching singly linked list is when given element doesn't present at all in the singly linked list. Using linear search then require "n" comparisons in worst case.

9. Consider the following functions

$$f(n) = 3n^{\sqrt{n}} \qquad g(n) = 2^{\sqrt{n}\log_2 n} \qquad h(n) = n!$$

Which of the following is true?

(a) h(n) is O(f(n)) (b) h(n) is O(g(n)) (c) g(n) is not O(f(n)) (d) f(n) is O(g(n))

[GATE-2002 : 2 Marks]

Ans. (d)

Soln. $f(n) = 3n^{\sqrt{n}}$, $g(n) = 2^{\sqrt{n}\log_2 n}$, $h(n) = n! = O(n^n)$

$\Rightarrow g(n) = n^{\sqrt{n}\log_2 2}$ $[a^{\log b} = b^{\log a}]$

$\Rightarrow n^{\sqrt{n}}$

10. Consider the following algorithm for searching for a given number x in an insorted array A[1.....n] having n distinct values:

1. Choose an i uniformly at random from 1...n;
2. If A[i] = x then Stop else Goto 1;

Assuming that x is present on A, What is the expected number of comparisons made by the algorithm before it terminates?

(a) n (b) n – 1 (c) 2n (d) n/2

[GATE-2002 : 2 Marks]

Ans. (a)

Soln. Let expected number of comparisons be E. Value of E is sum of following expression for all the possible cases:

Case-I If A[i] is found in the first attempt,
Number of comparisons = 1
Probability = 1/n.

Case-II If A[i] if found in the second attempt,
Number of comparisons = 2
Probability $= \frac{(n-1)}{n} * \frac{1}{n}$

Case-III If A[i] is found in the third attempt,
Number of comparisons = 3
Probability $= \frac{(n-1)}{n} * \frac{(n-1)}{n} * \frac{1}{n}$

There are actually infinite such cases. So, we have following infinite series for E.

$$E = \frac{1}{n} + \frac{n-1}{n} * \frac{1}{n} * 2 + \frac{(n-1)}{n} * \frac{(n-1)}{n} * \frac{1}{n} * 3 + ... \qquad ...(i)$$

After multiplying equation (i) with $\frac{(n-1)}{n}$, we get

$$E = \frac{(n-1)}{n} = \frac{n-1}{n} * \frac{1}{n} + \frac{n-1}{n} * \frac{1}{n} * 2 + ... \qquad ...(ii)$$

Subtracting (ii) from (i), we get

$$\frac{E}{n} = \frac{1}{n} + \frac{n-1}{n} * \frac{1}{n} + \frac{n-1}{n} * \frac{n-1}{n} * \frac{1}{n} + \ldots.$$

The expression on right side is a G.P. with infinite elements.

So apply the sum formula $\left(\frac{a}{1-r}\right)$

$$\frac{E}{n} = \left(\frac{1}{n}\right) \Big/ \left(\frac{n-(n-1)}{n}\right) = 1$$

$E = n$

Correct option is (a).

11. The running time of the following algorithm Procedure A(n). If n < = 2 return (1) else return $\left(A\left(\lceil \sqrt{n} \rceil\right)\right)$; is best described by

(a) O(n) (b) O(log n) (c) O(log log n) (d) O(1)

[GATE-2002 : 2 Marks]

Ans. (c)

Soln. Recursive relation for procedure A(n) is

$T(n) = T\left(\sqrt{n}\right) + c1 \text{ if } n > 2$

Let $n = 2^m$

$\Rightarrow T(n) = T(2^m)$

$\Rightarrow T(2^m) = S(m)$

$\Rightarrow T(n) = S(m)$

$\Rightarrow T\left(\sqrt{n}\right) = T\left(2^{m/2}\right) = S\left(m/2\right)$

$T(n) = T\left(\sqrt{n}\right) + c1 \text{ if } n > 2$

$\therefore S(m) = S\left(\frac{m}{2}\right) + c1 = O(\log m)$

$= O(\log \log n) \quad [\because n = 2^m \Rightarrow m = \log_2 n]$

$\Rightarrow T(n) = S(m) = O(\log \log n)$

Correct option is (c).

12. Consider the following three claims

1. $(n+k)^m = \Theta(n^m)$, where k and m are constant
2. $2^{n+1} = O(2^n)$
3. $2^{2n+1} = O(2^n)$

Which of these claims are correct?

(a) 1 and 2 (b) 1 and 3 (c) 2 and 3 (d) 1, 2 and 3

[GATE-2003 : 1 Mark]

Ans. (a)

Soln. Consider each statement separately

I. $f(n) = (n+k)^m$

$\text{so, } f(n) = (1+n)^m$ (Assume k = 1 is constant)

$f(n) = 1 + {}^mC_1 n + {}^mC_2 n^2 + {}^mC_m n^m$

$f(n) = O(n^m)$

II. $f(n) = 2^{n+1}$

$f(n) = 2^n \cdot 2^1$

$f(n) = 2 \cdot 2^n$

$f(n) = O(2^n)$

III. $f(n) = 2^{2n+1}$

$f(n) = 2^{2n} \cdot 2^1$

$f(n) = 2 \cdot 2^{2n}$

$f(n) = O(2^{2n})$

Therefore I and II are correct.

13. Consider the following C function.

```
float f (float x, int y)
{
    float p, s; int i;
    for (s = 1, p = 1, i = 1; i < y; i++)
    {
        p* = x/i;
        s + = p;
    }
    return s; }
```

For large values of y, the return value of the function f best approximates

(a) x^y (b) e^x (c) $\ln(1+x)$ (d) x^x

[GATE-2003 : 1 Mark]

Ans. (b)

Soln. The given function f is not recursive, so consider the following iteration method.

i	p	s
	$p = p * \frac{x}{i}$	$s = s + p$
Initialize 1	1	1
1	$p = x$	$s = 1 + x$
2	$p = x \cdot \frac{x}{2}$	$s = 1 + x + \frac{x^2}{2}$
3	$p = \frac{x^2}{2} \cdot \frac{x}{4}$	$s = 1 + x + \frac{x^2}{2} + \frac{x^3}{6}$
4	$p = \frac{x^3}{6} \cdot \frac{x}{4}$	$s = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24}$
5	$p = \frac{x^4}{24} \cdot \frac{x}{5}$	$s = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120}$

For large value of y assume $y \to \infty$ then i also tends to infinite it means increment of for loop may tends to infinite. In the given function we choose y as a large integer but not infinite. The return value of the function f is s.

$$s = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \cdots + \infty$$

$$s = 1 + x + \frac{x^2}{!2} + \frac{x^3}{!3} + \frac{x^4}{!4} + \frac{x^5}{!5} + \cdots + \infty$$

$$s = e^x$$

14. The tighest lower bound on the number of comparisons, in the worst case, for comparisons-based sorting is of the order of

(a) n (b) n^2 (c) $n \log n$ (d) $n \log^2 n$

[GATE-2004 : 1 Mark]

Ans. (c)

Soln. Any decision tree that sorts n distinct elements has height at least $\log \underline{|n}$. So the tighest lower bound on the number of comparison based sorting is $\log \underline{|n}$ but from starling's approximation.

$$\underline{|n} = (n/e)^n$$

Taking log both sides

$$\log \underline{|n} = \log (n/e)^n$$

$$\log \underline{|n} = n \log (n/e)$$

$$\log \underline{|n} = n(\log n - \log e)$$

$$\log \underline{|n} = n(\log n - 1.44)$$

$$\log \underline{|n} = n \log n - 1.44\, n$$

So $\log \underline{|n} = O(n \log n)$

15. What does the following algorithm approximate? (Assume $m > 1, \varepsilon > 0$).

```
x = m;
y = 1;
while (x – y > ε)
{
    x = (x + y)/2;
    y = m/x;
}
    print (x);
```

(a) $\log m$ (b) m^2 (c) $m^{1/2}$ (d) $m^{1/3}$

[GATE-2004 : 2 Marks]

Ans. (c)

Soln. Let $x = m = 9$. The loop will be terminated when $x - y = 0$ or $x - y < 0$. Consider the following iteration for

$x = m = 9,\ y = 1$

$\mathbf{x - y > 0}$	$\Rightarrow$	$\mathbf{x = (x + y)/2,}$	$\mathbf{y = m/x}$
$9 - 1 = 8$		$x = 5.0,$	$y = 9/5.0 = 1.8$
$5.0 - 1.8 = 3.2$		$x = 3.4,$	$y = 9/3.4 = 2.6$
$3.4 - 2.6 = .80$		$x = 3.0,$	$y = 9/3.0 = 3.0$

$x - y = 3.0 - 3.0 = 0$, loop terminated

So, $m = 9$ then $x = 3$

$x = (m)^{1/2} = (9)^{1/2} \Rightarrow x = 3$

So the algorithm computes $x = m^{1/2}$.

16. Let A[1,...., n] be an array storing a bit (1 or 0) at each location, and f(m) is a function whose time complexity is Θ(m). Consider the following program fragment written in a C like language:

```
counter = 0;
for (i = 1; i < n; i++)
{
    if (A[i] = = 1) counter ++;
    else
    {
        f (counter);
        counter = 0;  }
}
```

The complexity of this program fragment is

(a) $\Omega(n^2)$
(b) $\Omega(n \log n)$ and $O(n^2)$
(c) $\Theta(n)$
(d) $O(n \log n)$

[GATE-2004 : 2 Marks]

Ans. (c)

Soln. The given code is

```
1.      Counter = 0;
2.  for (i = 1; i < = n; i++)
3.  {   if (A[i] = = 1) counter ++;
4.      else {f(counter); counter = 0;}
5.  }
```

The time complexity of the program fragment depends on the frequency (Number of steps) of line 3 nd 4. In line 4 the frequency depends on the variable counter and there is no increment in the counter variable which is initialize to 0, so f(0) then counter = 0 it means there is no cell in an array which having a bit 0, so all cells in the array contains 1. Consider the line 3 if (A[i] = = 1) counter ++; the value of i will be increases upto n so the value of counter will be n. Since n is the frequency of the line 3 and the frequency of line 4 is 0. So the time complexity of line 3 is O(n) on average n and f(0) = O(1) is the time complexity of line 4. So the time complexity of the program fragment is maximum of line 3 and 4 which is O(n) on average.

17. The time complexity of the following C function is (assume n > 0)

```
int recursive (int n)
{
    if (n = = 1)
        return (1);
    else
        return (recursive (n – 1) + recursive (n – 1));
}
```

(a) $O(n)$
(b) $O(n \log n)$
(c) $O(n^2)$
(d) $O(2^n)$

[GATE-2004 : 2 Marks]

Ans. (d)

Soln. The given C function is recursive. The best way to find the time complexity of recursive function is that convert the code (algorithm) into recursion equation and solution of the recursion equation is the time complexity of given algorithm.

```
1.  int recursive (int n) {
2.  if (n = = 1) return (1);
3.  else
4.  return (recursive (n – 1) + recursive (n – 1));
5.  }
```

Let recursive(n) = T(n)

According to line 4 the recursion equation is T(n) = T(n – 1) + T(n – 1), n > 1. So the complete recursion equations is

$T(n) = 1, n = 1$

$T(n) = T(n-1) + T(n-1), n > 1$

or $T(n) = 2T(n-1), n > 1$

$T(1) = 1 = 2^{\circ}$

$T(2) = 2T(1)\ 2.1 = 2^1$

$T(3) = 2T(2) = 2.2 = 2^2$

$T(4) = 2T(3) = 2.2^2 = 2^3$

$\vdots$

$T(n) = 2^{n-1}$

or $T(n) = 2^n \bullet 1/2$

So, $T(n) = O(2^n)$

18. The recurrence equation

$T(1) = 1$

$T(n) = 2T(n-1) + n, n \geq 2$

evaluates to

(a) $2^{n+1} - n - 2$ (b) $2^n - n$ (c) $2^{n+1} - 2n - 2$ (d) $2^n + n$

[GATE-2004 : 2 Marks]

Ans. (a)

Soln. $T(1) = 1$

$T(n) = 2T(n-1) + n \quad n \geq 2$

$T(2) = 2T(1) + 2 = 2.1 + 2 = 4$

$T(3) = 2T(2) + 3 = 2.4 + 3 = 11$

$T(4) = 2T(3) + 4 = 2.11 + 4 = 26$

$\vdots$

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

So $T(n) = 2^{n+1} - n - 2$

19. Let f(n), g(n) and h(n) be functions defined for positive integers such that f(n) = O(g(n), g(n)) ≠ O(f(n)), g(n) = O(h(n)), and h(n) = O(g(n)). Which one of the following statements is FALSE ?

(a) $f(n) + g(n) = O(h(n)) + h(n))$ (b) $f(n) = O(h(n))$

(c) $h(n) \neq O(f(n))$ (d) $f(n)h(n) \neq O(g(n)h(n))$

[GATE-2004 : 2 Marks]

Ans. (d)

Soln. We can verify as: $f \leq f$ BUT $g \not\leq f$. Therefore, f < g, Also g = h as g = O(h) and h = O(g). Therefore f < g and g = h

(a) $f(n) + g(n) = O(h(n)) + h(n))$ is true.

$\Rightarrow f + g = f + h < h + h$

(b) $f(n) = O(h(n))$ is true.

$\Rightarrow f < h$

(c) $h(n) \neq O(f(n))$ is true.

$\Rightarrow h \not< f$ is correct.

(d) $f(n)h(n) \neq O(g(n)h(n))$ is false.

$\Rightarrow f \cdot h < g \cdot h$ implies $fh = O(gh)$

20. The time complexity of computing the transitive closure of a binary relation on a set of n elements is known to be

(a) $O(n)$ (b) $O(n \log n)$ (c) $O(n^{3/2})$ (d) $O(n^3)$

[GATE-2005 : 1 Mark]