Name:- Himanshu Joshi
Course:- B.Tech (CSE)
Section:- B

Date.
Page No.

Assignment 1

Q1. What do you understand by asymptotic notations. Define different Asymptotic notations with example.

A) 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 mainly 3 Asymptotic notations:-

(1) Big-O notation:-

- It represent the upper bound of running time of an algo.
- This notation is called as upper ~~bound~~ bound of the algo, or a worst case of an algo.
- $O(g(n)) = \{ f(n) :$ there exist positive constants c & n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$, where $c > 0$ and $n \geq n_0$.

- e.g.

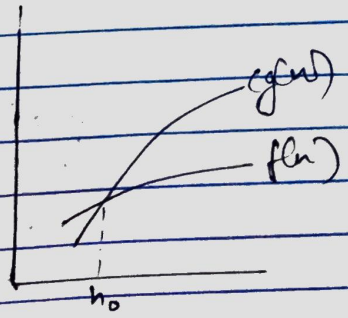
$f(n) = 3\log n + 100$

$g(n) = \log n$

$3\log n + 100 <= c(\log(n))$

$c = 150$ and $n \geq 2$

(undefined at $n = 1$)



(2) Big-Omega (Ω) notation:-

- It represent the lower bound of the running time of an algo.
- This notation is known as lower bound of an algo, or best case of an algo.

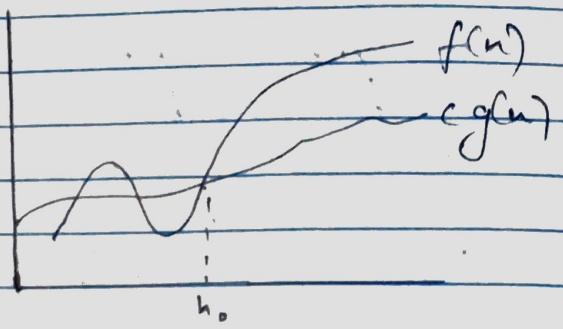
Date.
Page No.

- A function $f(n)$ is said to be $\Omega(g(n))$ iff there exist constant c and a constant n_0 such that

$$\Omega \leq cg(n) \leq f(n) \quad \forall \quad n \geq n_0$$

e.g. $n^3 + 1$

$\Omega(1)$.



(8) Theta (Θ) notation —

- Theta gives the tight upper and lower bound both.
- $f(n) = \Theta(g(n))$ iff
$cg(n) \leq f(n) \leq c_1 g(n)$ $\forall$ $n \geq \max(n_1, n_2)$
some constant c_1 and $c_2 > 0$.

e.g.

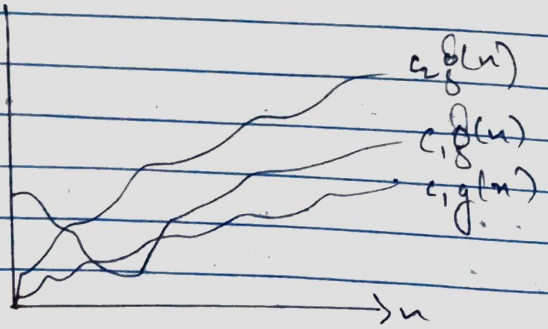
$f(n) = 5n^3 + 10n^2 + 2n + 5$

$5n^3 \leq 5n^3 + 10n^2 + 2n + 5 \leq (5 + 10 + 2 + 5)n^3$

$5n^3 \leq f(n) \leq 22n^3$

$c_1 = 5$, $c_2 = 22$, $n_0 = 1$

$f(n) \longleftrightarrow \Theta(n^3)$.



Date:
Page No.

Q1. What should be the time complexity

```
for (i=1 to n) { i = i*2 }
```

i = 1, 2, 4, 8, 16, ... , n { G.P }

$2^0, 2^1, 2^2, \ldots, 2^k$

$a = 1 \quad r = \frac{2}{1}, 2$

t_k (k^{th} term) $= ar^{k-1}$

$n = 1 \times 2^{k-1}$

$n = \frac{2^k}{2}$

$2n = 2^k$

$k = \log_2(2n) = \log_2(n)$

$O(n) = \log n$

Q2. $T(n) = \begin{cases} 3T(n-1) & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases}$

$T(n) = 3T(n-1) \longleftrightarrow T(n-1) = 3T(n-2)$

$T(n) = 3 \times 3T(n-2) \longleftrightarrow T(n-2) = 3T(n-3)$

$T(n) = 3 \times 3 \times 3T(n-3)$

$T(n) = 3^3 T(n-3) \longleftrightarrow T(n-3) = 3T(n-4)$

$T(n) = 3^3 \times 3T(n-4)$

$T(n) = 3^4 \times T(n-4)$

General form:—

$T(n) = 3^i T(n-i) \quad \ldots (i)$ [$T(0) = 1$]

$T(n-i) = T(0)$

$n - i = 0$

$n = i$

putting this in eq(i)

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

$T(n) = 3^n T(0)$

$T(n) = 3^n$

$\therefore \boxed{T(n) = O(3^n)}$

Q4 $T(n) = \begin{cases} 2T(n-1) - 1, & \text{if } n > 0, \text{ otherwise } 1 \end{cases}$

$T(n) = 2T(n-1) - 1 \leftarrow T(n-1) = 2T(n-2) - 1$

$T(n) = 2 \times (2T(n-2) - 1) - 1$

$T(n) = 2^2 T(n-2) - 3 \leftarrow T(n-2) = 2T(n-3) - 1$

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

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

$T(n-3) = 2T(n-4) - 1$

$T(n) = 2^3 (2T(n-4) - 1) - 2^2 - 2 - 1$

$T(n) = 2^4 T(n-4) - 2^3 - 2^2 - 2 - 1$

General form

$T(n) = 2^i T(n-i) - (2^{i-1} + 2^{i+2} + \dots + 1)$

$T(n-i) = T(0)$

$n - i = 0$

$n = i$

$T(n) = 2^n T(0) - (1 + 2 + 2^2 + 2^3 + \dots + 2^{n-1})$

$[T(0) = 1]$

$T(n) = 2^n - \left(\frac{2^{n-1} - 1}{2 - 1}\right)$

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

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

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

$\boxed{T(n) = O(2^n)}$

Date.
Page No.

Q5. What should be the time complexity of :

```
int i=1, s=1;
while (s<=n)
{
  i++;
  s=s+i;
  printf("#");
}
```

Ans)

No. of steps	S	i
0	0	1
1	1	2
2	3	3
3	6	4
4	10	5
⋮	⋮	⋮
K step	n	

$T(n) = O(k)$

$S = 0, 1, 3, 6, 10, \ldots\ldots n$

$S_n = 1 + 3 + 6 + 10 + 15 + \ldots\ldots + n$

$- \quad S_n = \quad 1 + 3 + 6 + 10 + \ldots\ldots - + (n-1) + n$

$0 = 1 + 2 + 3 + 4 + 5 + \ldots\ldots - n$

$n = 1 + 2 + 3 + 4 + 5 + \ldots\ldots$ K step

$n = \frac{k}{2}[2(1) + (k-1)1] = \frac{k}{2}[2 + k - 1]$

Date.
Page No.

$$2n = k^2 + k \Rightarrow 2n = \left(k + \frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2$$

$$2n + \left(\frac{1}{2}\right)^2 = \left(k + \frac{1}{2}\right)^2$$

$$k + 1/2 = \sqrt{2n + (1/2)^2}$$

$$k = \sqrt{2n + (1/2)^2} - 1/2$$

$$T(n) = T(k) = T\left(\sqrt{2n + (1/2)^2} - 1/2\right)$$

$$\boxed{T(n) = O\sqrt{n}}$$ ✗

Q6. Time Complexity of :-

```
void function (int n)
{
    int i, count = 0;
    for (i = 1 ; i*i <= n ; i++)
        count ++;
}
```

⇒ Since i is moving from 1 to $\sqrt{n}$ with linear growth so

$$\boxed{T(n) = O\sqrt{n}}$$

Date
Page No

97. Time complexity of :-

```
void function (int n)
{
  int i, j, k , count = 0;
  for (i = n/2 ; i <= n ; i++)
    for (j = 1; j <= n; i = j * 2)
    for (k = 1; k <= n; k = k * 2)
      count ++;
}
```

O(n log n log n)

O(n (log n)^2)

98. Time complexity of :-

```
function (int n)
  { if (n == 1) return;
    for (i = 1 to n)
    { for (j = 1 to n)
      {
        printf("*");}
    }
    function(n-1);}
```

$T(n) = T(n-1) + n^2$ $\quad [T(n-1) = T(n-2) + (n-1)^2]$

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

$[T(n-2) = T(n-3) + (n-2)^2]$

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

General form:-

$$T(n) = T(n-i) + n^2 + (n-1)^2 + (n-2)^2 + \dots + (n-i)^2$$

$$T(n-i) = T(1)$$

$$n = i+1$$

$$n-1 = i$$

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

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

$$T(n) = 1 + 1^2 + 2^2 + 3^2 + \dots + n^2$$

$$T(n) = \frac{n(n+1)(2n+1)}{6}$$

$$\boxed{T(n) = O(n^3)}$$

Q9. Time complexity of :-

```
void function (int n)
{
    for (i=0 to n)
    { for (j=1; j<=n; j=j+i)
        printf("#");
    }
}
```

→ $O(n\sqrt{n})$

Q10 For the function n^k and a^n, what is asymptotic relation between these function? Assume that $k \geq 1$ and $a > 1$ are constants. Find out the value of c and n_0 for what relation holds.

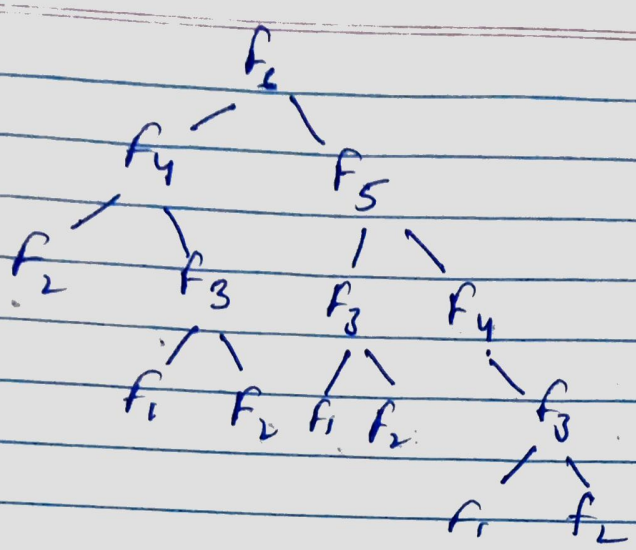
Ans) If $c > 1$ then the exponential c^n for out-grows any term.
So n^k is $O(c^n)$.

Q11 Write the recurrence relation for recursive function that prints fibonacci series. Solve recurrence relation to get Time complexity of program. What will be the space complexity of this program and why?

Ans)
$$T(n) = T(n-1) + T(n-2) + c$$
$$T(n-2) \approx T(n-1)$$
$$T(n) = 2T(n-1) + c$$
$$[T(n-1) = 2T(n-2) + c]$$
$$T(n) = 2(2T(n-2) + c) + c = 2^2 T(n-2) + 2c + c$$
$$[T(n-2) = 2T(n-3) + c]$$
$$T(n) = 2^3 (2T(n-3) + c) + 2c + c$$
$$T(n) = 2^3 T(n-3) + 2^2 c + 2c + c$$

General Term:-
$$T(n) = 2^i T(n-i) + (2^0 + 2^1 + 2^2 + \dots + 2^{i-1}) c$$
$$n - i = 0$$
$$n = i$$
$$T(n) = 2^n T(0) + (2^0 + 2^1 + 2^2 + \dots + 2^{n-1}) c$$
$$T(n) = 2^n (1) + \frac{2^0 (2^{n-1} - 1)}{2 - 1} c$$
$$T(n) = 2^n (1 + c) - c$$
$$T(n) = O(2^n)$$

Date.
Page No.



The max depth is proportional to N, hence the space complexity of fibonacci recursive is $O(n)$.

Q12 What is the Time complexity of code and why?

```
void function (int n) {
    int j=1; i=0;
    while (i<n) {
        i = i+j;
        j++; }}
```

⇒ $i = 0, 1, 3, 6, 10, 15, \ldots$

$j = 1, 2, 3, 4, 5, 6, \ldots$

So, i will go on till n and general formula for k^{th} term is $n = k\frac{(k+1)}{2}$

$\therefore T.C = O(\sqrt{n})$

Date.
Page No.

Q18 Write programs which have Time complexity:

1) n^3

```
void fun(int n)
{ int i,j,k;
  for (i=0; i<=n; i++)
  { for (j=0; j<=n; j++)
    {
      for (k=0; k<=n; k++)
      printf ("#"); }}}
```

2) $n \log n$

```
void fun()
{
  int i,j;
  for (i=1; i<=n; i++) {
    for (j=0; j<=n; j=j*2)
      printf ("#");
      printf ("\n"); }}
```

3) $\log(\log n)$

```
void fun (int n)
{ bool prime [n+1];
  memset (prime, true, sizeof (prime));
  for (int p=2; p*p <=n; p++)
    { if (prime[p] == true)
      {
        for (int i=p*p; i<=n; i+=p)
```

```
prime[i] = false;
}}
for (int p = 2; p <= n; p++)
    if (prime[p])
        cout << p << endl;
}
```

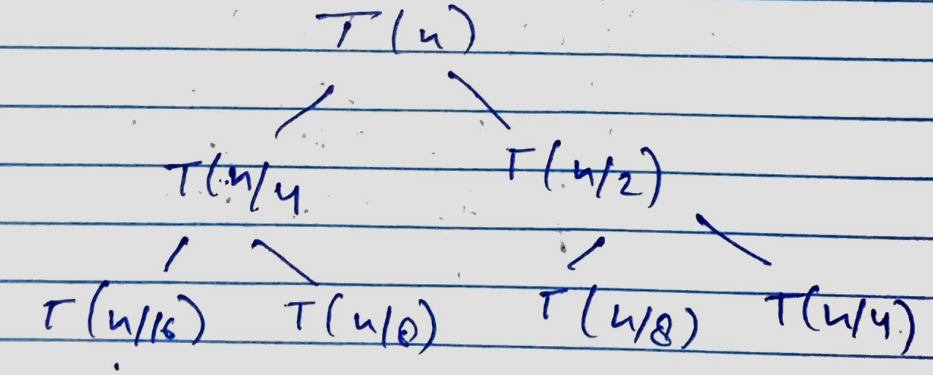
Q14 $T(n) = T(n/4) + T(n/2) + cn^2$

→ $T(1) = c$

putting $n = n/2$

$$T(n/2) = T(n/8) + T(n/4) + c(n^2/4)$$

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



$$T(n) = c\left[n^2 + \frac{5n^2}{16} + \frac{25n^2}{256} + \dots\right]$$

$$T(n) = n^2 c\left(1 + \frac{5}{16} + \frac{5^2}{16^2} + \dots\right)$$

$$\boxed{T(n) = O(n^2)}$$

Date.
Page No.

Q15 Time complexity of :-

```
int fun(int n)
{
    for (int i=1; i<=n; i++)
    {
        for (int j=1; j<n; j=j+i)
        {
            // Some O(1) task
        }
    }
}
```

for $i=1$, inner loop is executed n times
for $i=2$, inner loop is executed $n/2$ times
for $i=3$, inner loop is executed $n/3$ times
:
for $i=n$, inner loop is executed n/n times.

Total time $= n + n/2 + n/3 + \cdots + n/n$

$= n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + 1/n\right)$

$= n \log n$

$\boxed{T(n) = O(n \log n)}$

Q16 Time complexity of :-

```
for (int i=2; i<=n; i=pow(i,k))
{
    // Some O(1) expressions
}
```

where k is a constant.

$\boxed{O(\log(\log n))}$

Date.
Page No.

Q18 Arrange in increasing order of rate of growth.

a) $100, \log\log n, \log n, \sqrt{n}, n, n\log n, n^2, 2^n, 2^{2n}, 4^n, n!$

b) $1, \log(\log(n)), \sqrt{\log n}, \log n, \log(2n), \log(n!), 2\log(n), n, 2n, 4n, n\log(n), n^2, 2(2^n), n!$

c) $96, \log_8 n, \log n, \log(n!), 5n, n\log_6 n, n\log_2 n, 8n^2, 7n^3, 8^{2n}, n!$

Q19 Write linear search pseudo code.

→

```
Linear Search (A, key)
    comp ← 0, f ← 0
    for i = 1 to A.length
        comp ← comp + 1
        if A[i] == key
        print "Element found"
        f = 1
    if f == 0
        print "Element not found"
    print comp
```

Date.
Page No.

Q2 Write pseudo codes for

(1) Iterative method of Insertion sort →

```
Insertion Sort (A)
for j = 2 to A.length
    key = A[j]
    i = j-1
    while i>0 and A[i] > key
        A[i+1] = A[i]
        i = i-1
    A[i+1] = key.
```

(2) Recursive method →

```
Insertion Sort (A, n)
if n ≤ 1
    return
Insertion Sort (A, n-1)
    key = A[n-1];
    j = n-2;
while j ≥ 0 and A[j] > key
    A[j+1] = A[j]
    j = j-1
    A[j+1] = key.
```

→ Insertion sort consider one input element per iteration and produces a partial solution without considering future elements that's why it is called online sorting.

Date.
Page No.

Other sorting algos that have been discussed in lecture are :-

- Bubble Sort
- Selection Sort
- Quick Sort
- Merge Sort
- Heap Sort
- Counting Sort

Q.21 Complexity of all sorting :-

=>

	Best Case	Avg case	Worst case
Bubble Sort	$\Omega(N)$	$O(N^2)$	$O(N^2)$
Selection Sort	$\Omega(N^2)$	$O(N^2)$	$O(N^2)$
Insertion Sort	$\Omega(N)$	$O(N^2)$	$O(N^2)$
Merge Sort	$\Omega(N \log N)$	$O(N \log N)$	$O(N \log N)$
Heap Sort	$\Omega(N \log N)$	$O(N \log N)$	$O(N \log N)$
Quick Sort	$\Omega(N \log N)$	$O(N \log N)$	$O(N^2)$
Counting Sort	$\Omega(N+k)$	$O(N+k)$	$O(N+k)$.

Q.22 Divide all sorting algo sub.

=>

	Inplace	Stable	Online
Bubble Sort	Yes	Yes	Yes
Selection Sort	Yes	No	Yes
Insertion Sort	Yes	Yes	Yes
Merge Sort	No	Yes	Yes
Heap Sort	Yes	No	Yes
Quick Sort	Yes	No	Yes
Counting Sort	Yes	Yes	Yes

Q2 Write recursive, iterative :...

→ Linear Search →

```
Linear Search (A, key)
    found ← 0
    for i ← 1 to N
        if A[i] == key
            found ← 1
            print "Element found"
            break
    if found == 0
        print "Element Not found"
```

Time complexity - O(n)
Space complexity - O(1)

Binary Search (Iterative) →

```
Binary Search (A, beg, end, key)
    while beg ≤ end
        mid = beg + (end - beg)/2
        if mid == key
            return mid
        if A[mid] < key
            beg = mid + 1
        if A[mid] > key
            end = mid - 1
    return -1
```

Time Complexity = O(log₂n)
Space Complexity = O(1).

Date.
Page No.

Binary Search (Recursive) →

```
Binary Search (A, beg, end, key)
    if end > beg
        mid = (beg + end)/2
    if A[mid] == item
        return mid+1
    else if A[mid] < item
return Binary Search (A, mid+1, end, key)
    else
return Binary Search (A, beg, mid-1, end)
    return -1.
```

Time Complexity — $O(\log n)$
Space Complexity — $O(1)$

Q24 Write recurrence relation for binary recursive search.

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