Name - Jaya Mahar
Course - BTech (CSE)
Sec - B

Assignment 1

Qs 1. What do you understand by a symphotic notation. Define diff. asymptotic notation with example

Asymptotic notation are the mathematical notations used to discribe the running time of an algo when the i/p tends towards a particular value or a limiting value.
There are mainly 3 asymptotic notation.

Big - O - notation

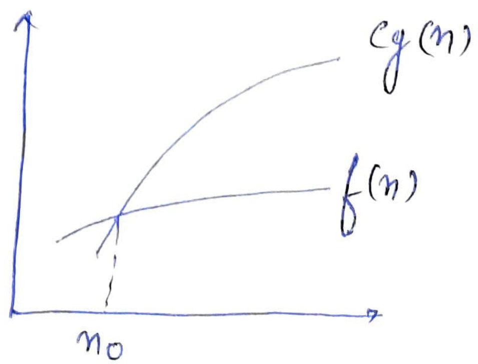
- It represents the upper bound of running time of an algo.
- This notation is called as upper bound of the algo or a worst case of an algo.

$O(g(n)) = \{ f(n)$: there exist +ve constant c & n_0 such that

$0 \le f(n) \le c\,g(n)$ for all $n \ge n_0$, where $c > 0$ & $n \geq n_0$

eg:

$f(n) = 3 \log n + 100$
$g(n) = \log n$
$3 \log n + 100 <= (c * \log(n)$
$c = 1 < 0$ & $n > 2$
(undefined at $n = 1$)



n = no of i/p

(ii) Big omega (Ω) notation

- It represents the lower bound of the running time of an algorithm
- This notation is known as lower bound of an algo or best case of an algo
- $\Omega(g(n)) = \{ f(n)$: there exist positive constant c & n_0 such that
  $0 \le c\,g(n) \le f(n) \ \forall n, \ n \ge n_0$

eg.

$f(n) = 3n+2$

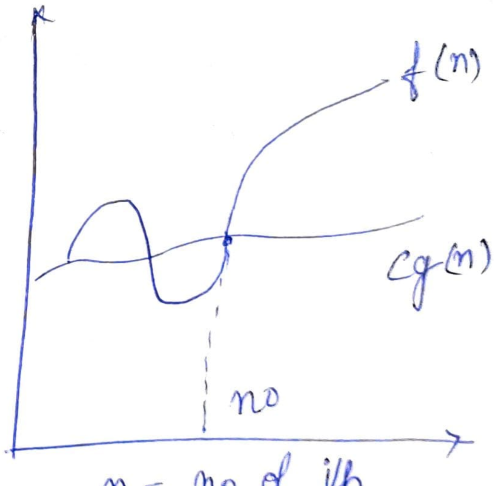
$g(n) \le f(n)$

$[c = \text{constant}, \ g(n) = n]$

$cn \le 3n+2$

$cn - 3n \le 2$

$n(c-3) \le 2 \Rightarrow n \le \frac{2}{c-3}$



If we assume $c = 4$

then $n_0 = 2$

$c = 4, n_0 = 2$

(III) Theta (Θ) notation

- It enclose the function from above & below since, It represents the upper & lower bound of running time of an algo.
- This is known as tight bounds of an algo, or an average case of algo.
- $\Theta(g(n)) = \{ f(n)$: there exist positive constant c_1 & c_2 & n_0 such that

$$0 \le c_1 * g(n) \le f(n) \le c_2 * g(n) \ \forall\ n > n_0$$

eg :-

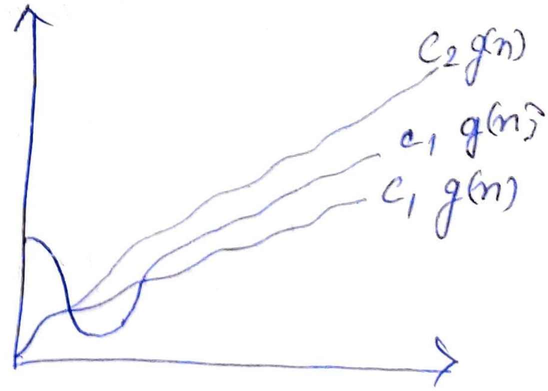
$$f(n) = 5n^3 + 16n^2 + 3n + 8$$

$$5n^3 \le (n^3 + 16n^2 + 3n + 8 \le (5+16+3+8)\, n^3$$

$$5n^3 \le f(n) \le 32n^3$$

$c_1 = 5$, $c_2 = 32$, $n_0 = 1$

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



Qs2. What should be the time complexity:
for $(i=1$ to $n)$ $\{i=i*2\}$

Ans→ $i=2, 4, 8, 16, \ldots\ldots K^{th}$ term $\ldots\ldots n$

$$a_n = ar^{n-1}$$

$$a_n = 1(2)^{K-1}$$

$$n = 2^{K-1}$$

$$\log_2 n = (K-1)\log_2 2$$

$$\boxed{K = \log_2 n + 1}$$

$$O(n) = \log n$$

Qs3? $T(n) = \{3T(n-1)$ if $n>0$, otherwise $1\}$

Ans→ $T(n) = 3T(n-1)$

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

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

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

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

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

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

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

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

⋮

General form:-

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

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

$n-i = 0$

$\boxed{n=i}$

Putting $n=i$ in eq^n (i) ;

$$T(n) = 3^n T(n-n)$$
$$T(n) = 3^n T(0) \qquad [T(0)=1 \text{ given}]$$
$$T(n) = 3^n$$
$$\boxed{T(n) = O(3^n)}$$

Q54) $T(n) = \{2T(n-1)-1, \text{ if } n>0, \text{ otherwise } 1\}$

Ans → $T(n) = 2T(n-1)-1$

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

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

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

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

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

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

⋮

general form:-

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

$$T(n-i) = T(0)$$
$$n-i=0$$
$$\boxed{n=i}$$

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

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

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

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

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

Q8) What should be the T.C of:-

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

Ans→

No. of steps (K)	s	i
0	0	1
1	1	2
2	3	3
3	6	4
4	10	5
5	15	6
6	21	7
|	|	|
K step	n	

$T(n) = O(K)$

$s = 0, 1, 3, 6, 10, ------ n$

$S_n = 1 + 3 + 6 + 10 + 15 + ----- + n$

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

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

$n = 1+2+3+4+ \text{-----} K$ step

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

$2n = K[2+K-1]$

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

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

$K + \frac{1}{2} = \sqrt{2n+(\frac{1}{2})^2}$

$K = \sqrt{2n+(\frac{1}{2})^2} - \frac{1}{2}$

$T(n) = T(K)$

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

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

Q86) T.C of:-

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

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

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

7. 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 ; j = j * 2)
  for (k = 1 ; k <= n ; k = k * 2)
  count ++ ;

}
```

$O(n \log n \log n)$

$O(n (\log n)^2)$

8. Time Complexity of

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

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

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

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

⋮

general term

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

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

$n = i+1 \Rightarrow \boxed{n-1=i}$

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

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

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

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

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

Qs9. T.C of:

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

Ans→ $O(n\sqrt{n})$

Qs10) For the funⁿ n^k & a^n, what is the asymptotic relation b/w these function? Assume that $K>=1$ & $a>1$ are constants. Find out the value of c & n_0 for what relation holds.

Ans→ If $c>1$ then the exponential c^n for outgrows any term, so that answer is:

n^k is $O(c^n)$

Qs11. What is the T.C of code & why?

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

Ans → $i = 0, 1, 3, 6, 10, 15, \ldots$

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

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

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

Qs12. Write the recurrence relation for recursive function that prints fibonacci series. Solve recurrence relation to get T.C of program. What will be the space complexity of this program & why?

Ans → $T(n) = T(n-1) + T(n-2) + c$

$T(n-2) \approx T(n-1)$

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

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

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

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

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

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

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

|
|
|

general Term:-

$$T(n) = 2^i\, T(n-i) + (2^0 + 2^1 + 2^2 + \ldots 2^{i-1})\, c$$

$$n - i = 0$$

$$\boxed{n = i}$$

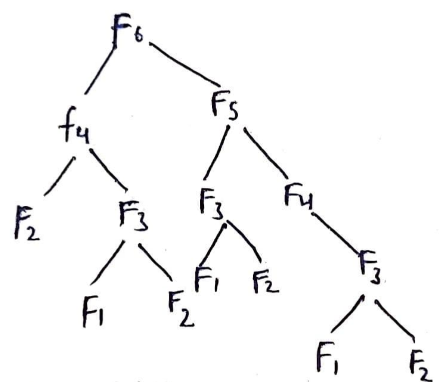
$$T(n) = 2^n\, T(0) + (2^0 + 2^1 + 2^2 + \ldots 2^{n-1})\, c$$

$$T(n) = 2^n (1) + 2^0 \frac{(2^{n-1} - 1)}{2-1}\, c$$

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

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

fib (6)



The max. depth is proportional to N, hence the space comp. of Fibonacci recursive is O(n).

Qs13) Write programs which have T.C :-

(i) 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"); }}
```

(II) 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 ("#");
     }
   }
}
```

(III) $\log(\log(n))$

```
void fun (int n)
{
  bool prime [n+1];
  memset (prime, true, size of (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 ;
}

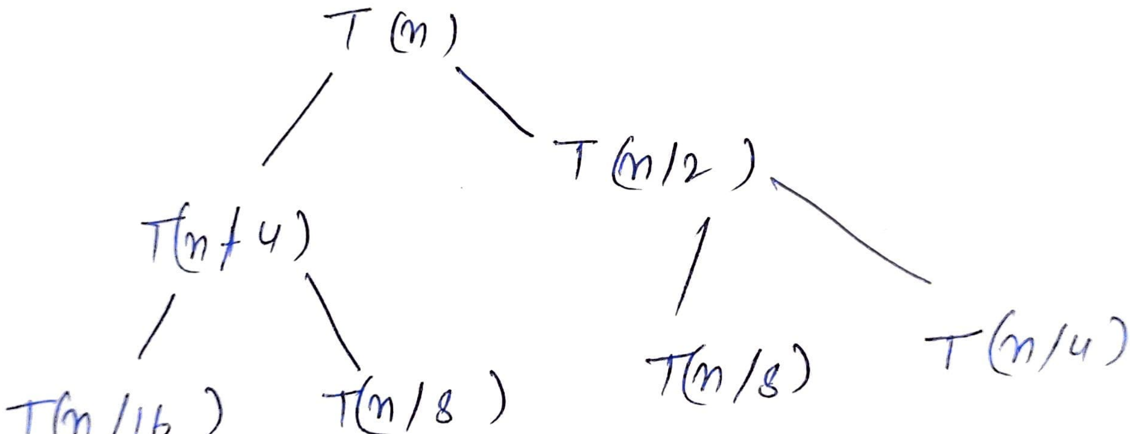
14. $T(n) = T(n/4) + T(n/2) + Cn^2$

$T(1) = C$

$n = n/2$

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

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



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

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

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

Qs15) T.C of:-

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

Ans→ 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 $= n + n/2 + n/3 + \text{------} n/2$

$= n(1 + 1/2 + 1/3 + \text{----} 1/n)$

$= n \log n$

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

16. TC of :-

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

where k is a constant

$O(\log(\log n))$

18. Arrange in asc. order of rate of growth.

(a) 100, log log n, log n, root 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, $\log_8 n$, $\log_2 n$, $\log(n!)$, 5n, $n \log_6 n$, $n \log_2 n$, $8n^2$, $7n^3$, 8^{2n}, n!

19. 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
```

20. Write pseudocode for ...

Iterative method of Insertion Sort -

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

Recursive Method →

```
Insertion Sort (A,n)
 if n ≤ 1
   return
 Insertion Sort (A, n-1)
 key = [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 Considers one i/p element per iteration & produces a partial Solution without Considering future elements that's why it is Called online sorting.

Other Sorting algos that have been discussed in lecture are

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

21. Complexity of all Sorting - - -

	Best Case	Average case	Worst Case
Bubble Sort	$\Omega(N)$	$\Theta(N^2)$	$O(N^2)$
Selection Sort	$\Omega(N^2)$	$\Theta(N^2)$	$O(N^2)$
Insertion Sort	$\Omega(N)$	$\Theta(N^2)$	$O(N^2)$
Merge Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	$O(N \log N)$
Heap Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	$O(N \log N)$
Quick Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	$O(N^2 \cancel{\log N})$
Counting sort	$\Omega(N+k)$	$\Theta(N+k)$	$O(N+k)$

22. Divide all sorting algo into - - -

	In Place	Stable	Online
Bubble Sort	yes	yes	yes
Insertion Sort	yes	yes	yes
Selection Sort	yes	No	yes
Merge Sort	No	yes	yes
Quick Sort	yes	no	yes
Heap Sort	yes	no	yes
Count Sort	No	yes	yes

23. Write recursive iterative - - - - -

Linear Search →

LinearSearch (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 - $O(\log_2 n)$
Space Complexity - $O(1)$

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 - $O(1)$

24. Write recurrence relation for binary recursive search

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