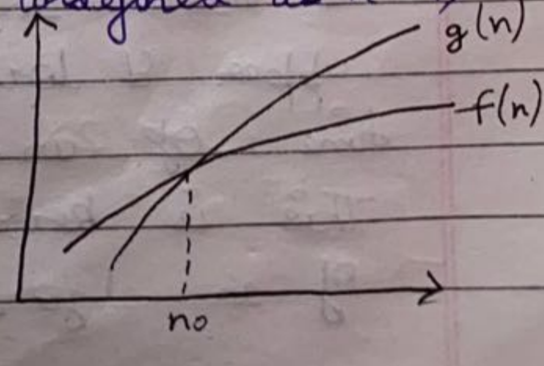
Name - Sachin Bhatt
Roll no. - 2061735
Sec - B

Assignment - 1

Ans 1 ⇒ Asymptotic notations are the mathematical notations used to describe 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

a) 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 positive constants c & n_0 such that $0 < f(n) \le cg(n)$ for all $n \ge n_0$, where $c > 0$ & $n \ge n_0$.

e.g.,
$$f(n) = 3 \log n + 100$$
$$g(n) = \log n$$
$$3 \log n + 100 <= c * \log(n)$$
$$c = 1 < 0 \text{ \& } n > 2$$
(undefined at $n = 1$)



(b) Big omega (Ω) notation ⇒

It represents 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.

$\Omega(g(n)) = \{ f(n)$: there exist (+)ve constant c & n_0 such that $0 \le c g(n) \le f(n) \; \forall n, n \ge n_0$

e.g.,

$f(n) = 3n+2$

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

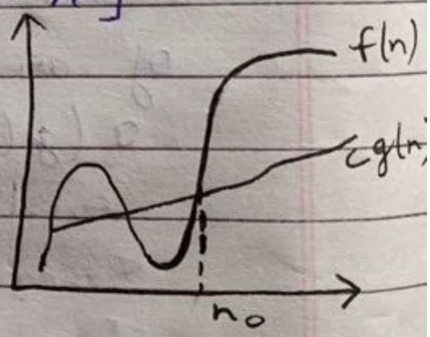
[c = constant, $g(n) = n$]

$cn \le 3n+2$

$cn - 3n \le 2$

$n(c-3) \le 2$

$n \le \frac{2}{c-3}$



if we assume $c = 4$, then $n_0 = 2$

$c = 4$, $n_0 = 2$

(c) Theta (θ) notation ⇒

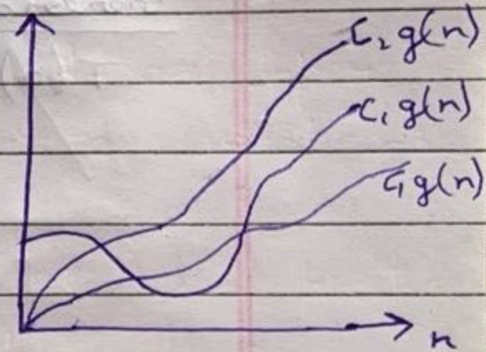
It enclose the function from above & below. Since, it represents the upper & lower bound of running time of an algo.

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

e.g.,

$$f(n) = 5n^3 + 16n^2 + 3n + 8$$
$$5n^3 \le (n^3 + 16n^2 + 3n + 8$$
$$5n^3 \le (5 + 16 + 3 + 8) n^3$$
$$5n^3 \le f(n) \le 32n^3$$
$$c_1 = 5, \; c_2 = 32, \; n_0 = 1$$
$$f(n) \leftrightarrow \theta(n^3)$$



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

$$a_n = a r^{n-1}$$
$$a_n = 1(2)^{k-1}$$
$$n = 2^{k-1}$$
$$\log_2 n = (k-1) \log_2 2$$
$$k = \log_2 n + 1$$
$$O(n) = \log n$$

Ans 3 ⇒ $T(n) = 3T(n-1)$

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

⋮

<u>General form</u> –

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

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

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

<u>Ans</u> 4 ⇒ $T(n) = 2T(n-1) - 1$

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

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

$\vdots$

<u>General form</u> -

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

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

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

$$[T(0) = 1]$$

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

Ans 5 ⇒

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	

$T(n) = O(k)$

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

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

$\cancel{S_n} \quad 1 + 3 + 6 + 10 + \ldots + (n-1) + n$

$-\quad - \quad - \quad - \quad - \quad - \quad - \quad -$

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

$n = 1 + 2 + 3 + 4 + \ldots$ k step

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

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

$2n = k^2 + k$

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

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

$(k + \frac{1}{2})^2$

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

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

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

Ans 6 ⇒ Since, i is moving from 1 to $\sqrt{n}$ with linear growth so
$$T(n) = O(\sqrt{n})$$

Ans 7 ⇒ $O(n \log n \log n)$
$$O(n(\log n)^2$$

Ans 8 ⇒ $T(n) = T(n-1) + n^2$
$$T(n) = T(n-2) + n^2 + (n-1)^2$$
$$T(n) = T(n-3) + n^2 + (n-1)^2 + (n-2)^2$$
$$\vdots$$

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

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

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

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

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

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

Ans 9 ⇒ $O(n\sqrt{n})$

Ans 10 ⇒ If $c > 1$ then the exponential c^n for outgrows any term, so that answer is :

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

Ans 12 ⇒ $T(n) = T(n-1) + T(n-2) + c$

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

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

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

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

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

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

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

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

$\vdots$

General Term –

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

$$n - i = 0$$

$$n = i$$

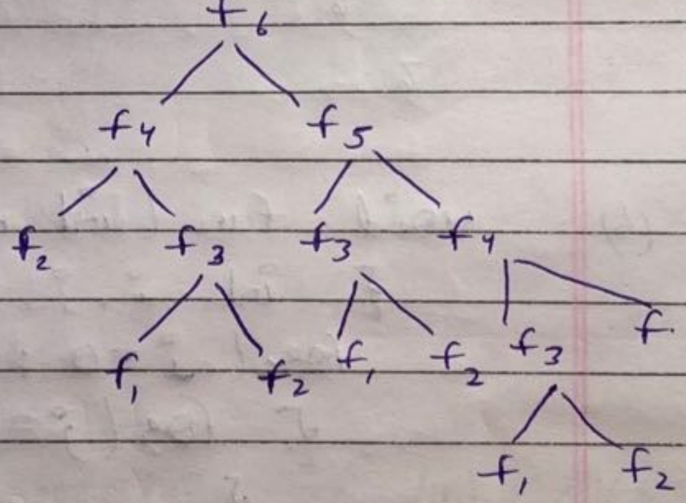
$$T(n) = 2^n\, T(0) + (2^0 + 2^1 + 2^2 + \dots 2^{n-1})c$$

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

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

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

fib. (6)



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

Ans 11 ⇒ $i = 0, 1, 3, 6, 10, 15, \dots$

$j = 0, 1, 2, 3, 4, 5, 6, \dots$

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

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

Ans 13 ⇒ void fun()

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

(b)

```
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("*");
    }
  }
}
```

(c)

```
void sieveofErdosthenes(int n)
{
    bool prime[n+1];
  memset(prime, true, sizeof
```

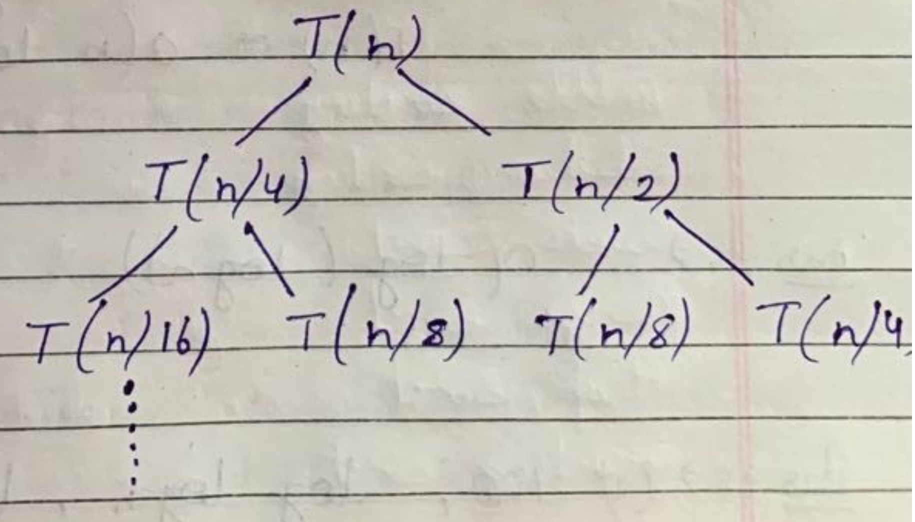
```
(prime)) ;
for (int p=2 ; p*p <=n ; p++)
  {
    if (prime [p] = true)
      {
    for (int i = p*p ; i<=n ;
            i+ = p)
      prime [i] = false ;
    }
  }
    for (int p=2 ; p <=n ; p++)
      if (prime [p] )
          cout << p << endl ;
      }
```

Ans 4 ⇒ $T(1) = c$

$n = n/2$

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

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



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

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

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

Ans 15 → 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 $= n + n/2 + n/3 + \ldots n/2$

$= n(1 + 1/2 + 1/3 + \ldots 1/n)$

$= n \log n$

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

Ans 16 → $O(\log(\log n))$

⋮

Ans 18 → (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) $96, \log_8 n, \log_2 n, \log_3(n!), 5n, n\log_6 n, n\log_2 n, 8n^2, 7n^3, 8^{2n}, n!$

Ans 19 ⇒ 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
```

Ans 20 ⇒ 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]
```

$i = i - 1$

$A[i+1] = Key$

Recursive Method of insertion sort

INSERTION_SORT (A, n)

if $n \leq 1$

return

INSERTION_SORT (A, n-1)

Key = Arr[n-1] ;

$j = n-2$;

while $j \geq 0$ and $A[j] > key$

$A[j+1] = A[j]$

$j = j - 1$

$A[j+1] = Key$

Insertion sort considers one input element per iteration & produces a partial solution without considering future elements thats why it is called Online sorting

Other sorting algorithm that have been discussed in lectures are -

Bubble sorting
Selection sorting
Quick sort
Merge sort
Heap sort
Counting sort

Ans 21 ⇒

	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	$\Omega(N\log N)$	$\theta(N\log N)$	$O(N^2)$
Counting sort	$\Omega(N+K)$	$\theta(N+K)$	$O(N+K)$

Ans 22 ⇒

	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

Ans 23 ⇒ 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 - $O(\log_2 n)$
Space complexity - $O(1)$

Binary Search (Recursion)

```
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 - $\theta(\log n)$
Space complexity - $O(1)$

Ans 24 ⇒ $T(n) = T(n/2) + c$