

Assignment 1

Q81. What do you understand by a asymptotic notation. Define different asymptotic notation with example.

Ans → 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 notations: -

(i) 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.
- $O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ \& } n_0 \text{ such that } 0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0, \text{ where } c > 0 \text{ \& } n \geq n_0\}$

• Eg,

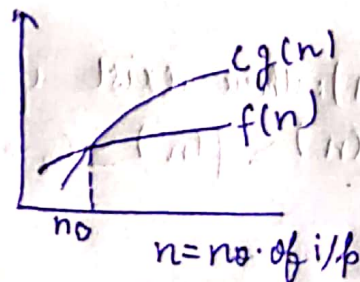
$$f(n) = 3 \log n + 100$$

$$g(n) = \log n$$

$$3 \log n + 100 \leq c \cdot \log(n)$$

$$c = 1 < 0 \text{ \& } n > 2$$

(undefined at $n=1$)



(ii) 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.
- $\Omega(g(n)) = \{f(n) : \text{there exist positive constant } c \text{ \& } n_0 \text{ such that } 0 \leq g(n) \leq f(n) \forall n, n \geq n_0\}$

• e.g.,

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

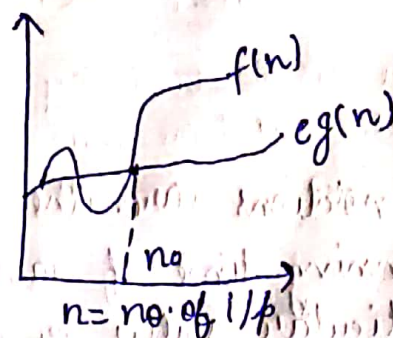
$$c g(n) \leq f(n)$$

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

$$c n \leq 3n + 2$$

$$cn - 3n \leq 2$$

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



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

$$c = 4, n_0 = 2$$

(III) Theta (θ) notation

• It enclose the funⁿ 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.

• $\theta(g(n)) = \{f(n) : \text{there exist positive constant } c_1, c_2 \text{ \& } n_0 \text{ such that } 0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n) \forall n > n_0\}$

• e.g.,

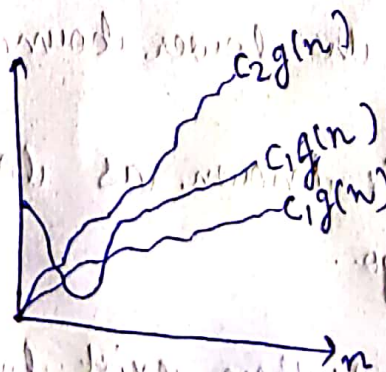
$$f(n) = 5n^3 + 16n^2 + 3n + 8$$

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

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

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

$$f(n) \leftrightarrow \theta(n^3)$$



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

Ans $\rightarrow i=2, 4, 6, 10, \dots, k^{\text{th}} \text{ term} \dots, n$

$$a_n = a_{n-1}$$

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

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

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

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

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

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

Ans $\rightarrow T(n) = 3T(n-1)$

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

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

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

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

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

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

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

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

⋮

General form: -

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

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

$$n-i = 0$$

$$\boxed{n=i}$$

Putting $n=i$ in eqⁿ (i) ;

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

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

$$[T(0) = 1 \text{ given}]$$

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

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

Q84) $T(n) = \begin{cases} 2T(n-1) - 1, & \text{if } n > 0, \\ \text{otherwise } 1 \end{cases}$

Ans →

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

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

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

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

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

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

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

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

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

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

⋮

general form: -

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

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

$$n-i = 0$$

$$\boxed{n=i}$$

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

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

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

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

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

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

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

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

Q85) What should be the T.C of:-

with $i = 1, s = 1;$

while ($s \leq n$)

{

$i++;$

$s = s + i;$

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 steps	n	

$$T(n) = O(K)$$

$$s = 0, 1, 3, 6, 10, \dots, n$$

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

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

$$- - - - -$$

$$0 = 1 + 2 + 3 + 4 + 5 + \dots + n$$

$$n = 1 + 2 + 3 + 4 + \dots + K \text{ steps}$$

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

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

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

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

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

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

$$T(n) = T(K)$$

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

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

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
do,

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

Q87) 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++;  
}
```

Ans $\rightarrow O(n \log n \log n)$

$$\boxed{O(n(\log n)^2)}$$

Q88) Time complexity of
function (int n)

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

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

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

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

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

General Term:-

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

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

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

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

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

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

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

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

Qs 9. T.C of:

void function (int n)

{

for (i=0 to n) {

for (j=1; j <= n; j=j+i)

printf("#");

}

Ans $\rightarrow O(n\sqrt{n})$

Qs 10) For the funⁿ n^k & a^n , what is the asymptotic relation b/w these function? Assume that $k \geq 1$ & $a > 1$ are constants.

Find out the value of c & n_0 for what relation holds.

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

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

Q11. 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 → $i = 0, 1, 3, 6, 10, 15, \dots$

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

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

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

Q12. 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 → $T(n) = T(n-1) + T(n-2) + C$
 $T(n-2) \approx T(n-1)$

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

$$\uparrow T(n-1) = 2T(n-2) + C$$

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

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

$$\uparrow T(n-2) = 2T(n-3) + C$$

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

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

⋮

General Term: -

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

$$\begin{matrix} n-i=0 \\ \boxed{n=i} \end{matrix}$$

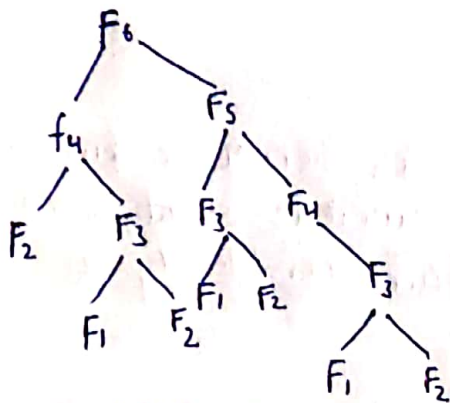
$$T(n) = 2^n T(0) + (2^0 + 2^1 + 2^2 + \dots + 2^{n-1}) C$$

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

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

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

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: -

1) $n \log n$

```

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;
```

```
}
```

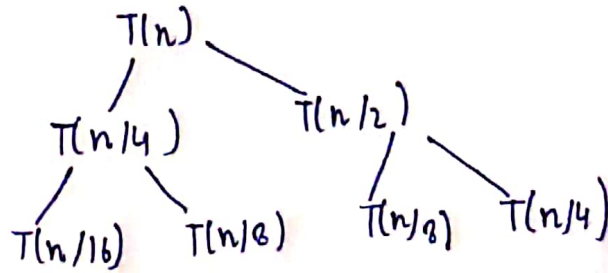
Q814) $T(n) = T(n/4) + T(n/2) + cn^2$

Ans → $T(1) = C$

$n = n/2$

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

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



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

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

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

Q.15) T.C :-

```

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

$$\text{Total time} = n + n/2 + n/3 + \dots + n/n$$

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

$$= n \log n$$

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

Q816) T.C of:-

for (int i=2; i <= n; i = pow(i, K))

{

// some O(1) expressions

}

where, K is a constant

Ans → $O(\log(\log n))$

Q818) Arrange in inc. order of rate of growth

(a) 100, $\log \log n$, $\log n$, \sqrt{n} , n , $n \log n$,
 n^2 , 2^n , 2^{2^n} , 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, $\log_8 n$, $\log_2 n$, $\log(n!)$, $5n$, $n \log_6 n$, $n \log_2 n$, $6n^2$, $7n^3$, 8^{2n} , $n!$

Q819) Write Linear search pseudo code ----

Ans → LinearSearch(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

Q820) Write pseudocode for ----

Ans → Iterative Method of Insertion Sort -

InsertSort(A)

for $j = 2$ to $A.length$

Key = $A[j]$

$i = j - 1$

while $i > 0$ & $A[i] > \text{Key}$

$A[i+1] = A[i]$

$i = i - 1$

$A[i+1] = \text{Key}$

Recursive Method \rightarrow

Insertionsort (A, n)

if $n \leq 1$

return

Insertionsort ($A, n-1$)

Key = $A[n-1]$

$j = n - 2$

while $j \geq 0$ and $A[j] > \text{Key}$

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

$j = j - 1$

$A[j+1] = \text{Key}$

\rightarrow 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
- Selection sort
- Quick sort
- Merge sort
- Heap sort
- Counting sort

Q21) Complexity of all sorting -----

Ans:-

	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	$\Omega(N \log N)$	$\Theta(N \log N)$	$\Theta(N \log N)$
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	$\Omega(N+K)$	$\Theta(N+K)$	$O(N+K)$

Q822) Divide all sorting algo into -----

Ans→

	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

Q823) Write recursive iterative -----

Ans→ 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) →

BinarySearch (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) →

BinarySearch (A, beg, end, Key)

if end > beg

mid = (beg + end) / 2

if A[mid] == item

return mid + 1

else if A[mid] < item

return BinarySearch (A, mid + 1, end, Key)

else

return BinarySearch (A, beg, mid - 1, end)

return -1

Time Complexity - $O(\log n)$

Space Complexity - $O(1)$

Q24) Write recurrence relation for binary recursive search

Ans $\rightarrow T(n) = T(n/2) + C.$