

## Assignment 1

Q1. What do you understand by asymptotic notations.

Define different Asymptotic Notations with example.

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 represents 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.

•  $O(g(n)) = \{ f(n) : \text{there exist positive constants } c, n_0 \text{ such that } 0 \leq f(n) \leq g(n) \text{ for all } n \geq n_0, \text{ where } c \geq 0 \text{ and } n \geq n_0 \}$

e.g.

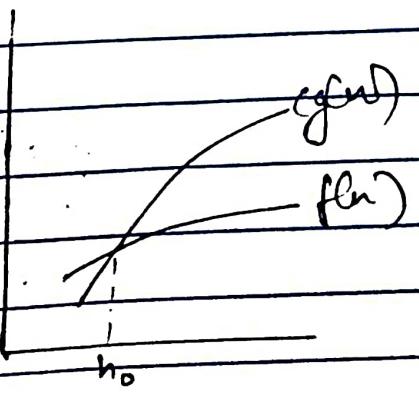
$$f(n) = 3\log n + 100$$

$$g(n) = 108n$$

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

$$c = 150 \text{ and } n \geq 1$$

(undefined at  $n=1$ )



(2) Big - Omega ( $\Omega$ ) notation:

• It represents the lower bound of the running time of an algo.

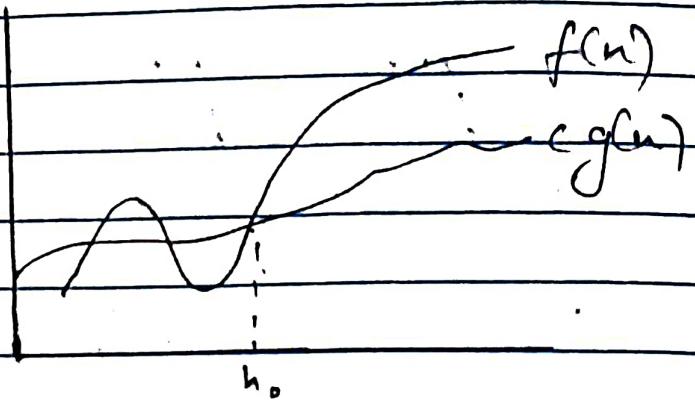
• This notation is known as lower bound of an algo, or best case of an algo.

- A function  $f(n)$  is said to be  $\mathcal{O}(g(n))$  iff there exist constant  $C$  and a constant  $n_0$  such that

$$\mathcal{O} \leq cg(n) \leq f(n) \quad \forall n \geq n_0$$

e.g.  $n^3 + 1$

$\mathcal{O}(1)$ .



### (8) Theta ( $\Theta$ ) notation —

- Theta gives the tight upper and lower bound both.

$f(n) = \Theta(g(n))$  iff  
 $c_1 g(n) \leq f(n) \leq c_2 g(n)$  if  $n \geq \max(n_1, n_2)$ .  
 some constant  $c_1$  and  $c_2 > 0$ .

e.g.

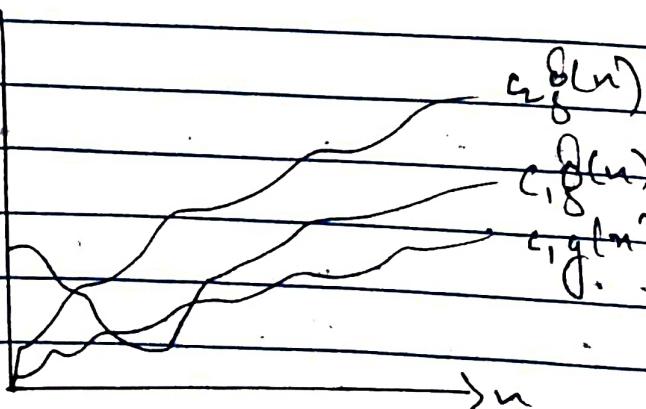
$$f(n) = 5n^3 + 10n^2 + 2n + 5$$

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

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

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

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



Q2. What should be the time complexity  
for  $(i=1 \text{ to } n) \{ i = i * 2\}$

$$i = 1, 2, 4, 8, 16, \dots, n \quad \{ \text{G.P} \}$$
$$2^0, 2^1, 2^2, \dots, 2^k;$$

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

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

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

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

$$\therefore n = 2^k$$

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

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

Q3.  $T(n) = \begin{cases} 3T(n-1) & \text{if } n > 0 \\ \text{otherwise } 1 \end{cases}$

$$T(n) = 3T(n-1) \leftrightarrow T(m-1) = 3T(m-2)$$

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

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

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

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

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

General form:-

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

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

$$n-i = b$$

$$n = p$$

putting this in eq(i)

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

$$T(n) \leq 3^n T(0)$$

$$T(n) \leq 3^n$$

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

Q4

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

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

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

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

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

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

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

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

General form

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

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

$$n-i=0$$

$$n=9$$

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

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

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

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

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

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

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

Q5. What should be the time complexity of -

Put  $i=1, s=1;$

while ( $s \leq n$ )

{

$i++;$

$s = s + i;$

printf ("#");

}

No. of steps

s

i

0

0

1

1

1

2

2

3

3

3

6

4

4

10

5

:

k step

n

$$T(n) = O(k)$$

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

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

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

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

$$n = 1 + 2 + 3 + 4 + 5 + \dots \quad k \text{ step}$$

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

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

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

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

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

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

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

Q6. Time Complexity of :-

void function (int n)

    int i, count = 0;

    for (i=1; i <= n; i++)

        count++;

}

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

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

Q7. 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++;

}

•  $O(n \log n \log n)$

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

Q8. Time complexity of :-

function (int n)

{ if (n == 1) return;

    for (i = 1 to n)

        { for (j = 1 to n)

            {

                printf("\*"); }

}

    function(n - 1); }

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

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

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

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

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

General form

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

$$n = i+1$$

$$n - i = 1$$

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

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

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

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

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

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 functions? Assume that  $k \geq 1$  and  $a \geq 1$  are constants. Find out the value of  $c$  and  $n_0$  for what relation holds.

Q11 If  $c > 1$  then the exponential  $c^n$  grows faster than 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?

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

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

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

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

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

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

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

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

General Term:-

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

$$n-i=0$$

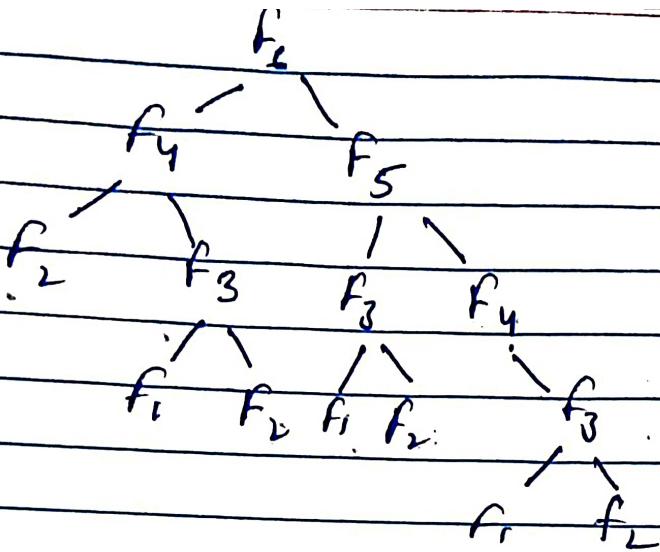
$$n=0$$

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

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

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

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



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++; } }

→

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

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

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

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

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 ("%d"); } } }.

2)  $n \log n$

→ void fun()  
  { int i, j;  
  for (i = 1; i <= n; i++)  
    { for (j = 0; j <= n; j++)  
        { printf ("%d");  
        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;

else

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

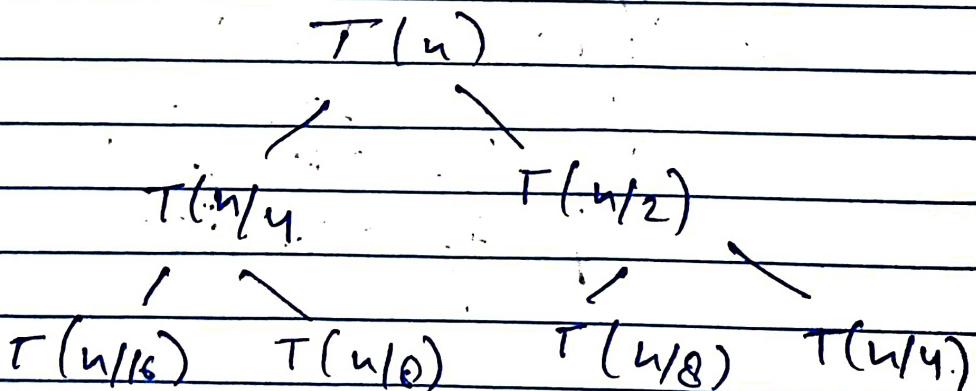
→

$$T(1) = C$$

putting  $n = n/2$

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

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



$$T(n) = C \left[ n^2 + \frac{5n^2}{16} + \frac{28n^2}{256} + \dots \right]$$

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

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

Q15 Time complexity of :-

Put fun(put n)

{  
for (int i=1; i<n; i++) -

{ for (int j=i; j<n; j+=2)

} // Some O(r) 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  $1/n$  times.

$$\begin{aligned} \text{Total time} &= n + n/2 + n/3 + \dots + 1/n \\ &= n \left( 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} \right) \end{aligned}$$

=  $n \log n$

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

Q16

Time complexity of :-

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

}

// some O(1) expressions

}

where k is a constant.

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

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

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

Q19

Write Linear Search pseudo code.

3)

'Linear Search (A, key)'

'comp  $\leftarrow 0$ , f  $\leftarrow 0$ '

'for i = 1 to A.length'

'comp  $\leftarrow comp + 1$ '

'if A[i] == key'

'print "Element found"'

'f = i'

'if f == 0'

'print "Element not found"'

'print comp'

Q1 Write pseudo code for

(1) Iterative method of Insertion sort →

Insertion Sort (A)

for  $j \geq 2$  to A.length

key = A[j]

i = j - 1

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

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

$i = i - 1$

$A[i+1] = key$

(2) Recursive method →

Insertion Sort (A, n)

if  $n \leq 1$

return

Insertion Sort (A, n-1)

key = A[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 consider one input element per iteration and produces a partial solution without considering future elements that's why it is called online sorting.

Other sorting algs 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	$\Omega(n)$	$\Theta(n^2)$	$\Omega(n^2)$
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$\Omega(n^2)$
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$\Omega(n^2)$
Merge Sort	$\Omega(n \log n)$	$\Theta(n \log n)$	$\Omega(n \log n)$
Heap Sort	$\Omega(n \log n)$	$\Theta(n \log n)$	$\Omega(n \log n)$
Quick Sort	$\Omega(n \log n)$	$\Theta(n \log n)$	$\Omega(n)$
Counting Sort	$\Omega(nk)$	$\Theta(n+k)$	$\Omega(n+k)$

Q.22

Divide all sorting algo into

	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

Time complexity = ..

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<sub>2</sub>n)

Space complexity = O(1)

Binary Search (Recursive) →

Binary Search (A, beg, end, key)

if end > beg

$$\text{mid} = \lceil (\text{beg} + \text{end}) / 2 \rceil$$

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 recursive relation for binary search:

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