Assignment-1

Q1. What do you understand by asymptotic notation define different asymptotic notations with example.

Ans1. Asymptotic notations are the mathematical notations used to describe the running time of an algo when the input tends towards the particular value or a limiting value.

There are mainly two asymptotic notations —

1. Big-O Notation
2. Big-Omega Notation

① Big-O Notation —

- It represents the upper bound of running time of an edge.
- This notation is called as super bound of the algo, or a worst case of an algo.
- $O(g(n)) = \{f(n)$: there exists +ve constants c and n_0 such that $0 \leq f(n) \leq g(n)$ for all $n \geq n$, where $c > 0$ and $n \geq n_0\}$
- Eg,

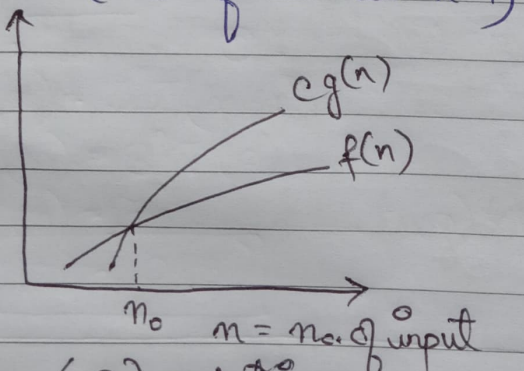
$f(n) = 3\log n + 100$

$g(n) = \log n$

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

$c = 150$ and $n > 2$

(undefined at $n = 1$)



② Big Omega (Ω) notation —

- It represents the lower bound of the running time of an edge.
- This notation is known as lower bound of an algo, or best case of an edge.
- $\Omega(g(n)) = \{f(n)$: there exist +ve constant c and n_0 such that $0 < c\,g(n) \leq f(n)\ \forall\ n, n > n_0$

• Eg-

$f(n) = 3n+2$

$cg(n) \leq f(n)$

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

$cn \leq 3n+2$

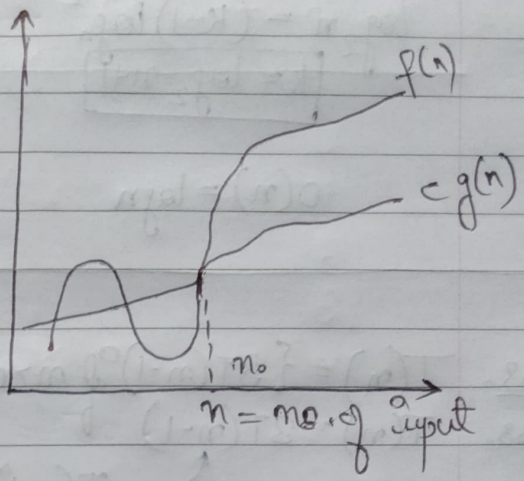
$cn - 3n \leq 2$

$n(c-3) \leq 2$

$\Rightarrow n \leq 2/c-3$

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

$\boxed{c = 4, n_0 = 2}$



② Theta (Θ) notation—

It enclose the function from above and below since it represents the upper and lower bound of running time of an algo.

This is known as tight bounds of an algo, an average case of algo.

$\Theta(g(n)) = \{ f(n)$: there exist +ve constant c_1, c_2 and n_0 such that $0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n) \forall n > n_0 \}$

• Eg—

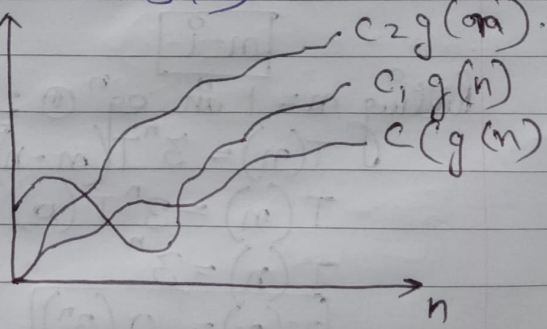
$f(n) = 5n^3 + 16n^2 + 3n + 8$

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

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

$\boxed{c_1 = 5, \ c_2 = 32, \ n_0 = 1}$

$f(n) \leftrightarrow \Theta(n^2)$



Ques. 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 $\ldots\ldots n$

$G_n = ar^{n-1}$

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

$n = 2^{k-1}$

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

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

$O(n) = \log n$

Ques3. $T(n) = \{3T(n-1)$ if $n>0$, otherwise $1\}$

Ans3. $T(n) = 3T(n-1)$

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

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

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

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

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

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

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

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

⋮

General form —

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

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

$n - i = 0$

$\boxed{n = i}$

Putting $n = i$ in eqn (1);

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

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

$T(n) = 3^n$ $\{T(0) = 1\}$

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

Ques4. $T(n) = \{2T(n-1) - 1$, if $n>0$ otherwise $1\}$

Ans4. $T(n) = 2T(n-1) - 1$

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

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

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

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

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

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

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

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

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

⋮

General form —

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

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

$n - i = 0$

$\boxed{n = i}$

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

$[T(0) = 1]$

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

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

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

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

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

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

Ques 5. What should be the T.C of :

```
int i=1, s=1;
while (3<=n)
{
  i++;
  3=3+i;
}printf ("#");
```

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	⋮

$T(n) = O(k)$

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

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

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

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

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

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

$2n = k(2 + k - 1)$

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

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

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

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

$k = \frac{1}{2}$

$T(n) = T(k)$

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

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

Ques6. T.C. of —

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

Ans6. Since, i is moving from 1 to $\sqrt{n}$ with like as growth so

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

Ques7. Time complexity of

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

Ans7. $O(n \log n \log n)$

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

Ques8. Time complexity of

```
function (int n) {
   if (n==1) return;
   for (i=1 to n)
```

```
{
    for (j=1 to n)
    {
        printf("k");
    }
}
function (n-1);
}
```

Ans8. $T(n) = T(n-1) + n^2 \cdot [T(n-1) = T(n-2) + (n-1)^2]$

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

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

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

⋮

General form —

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

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

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

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

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

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

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

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

Ques9. T.C of —

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

Ans9. $O(n\sqrt{n})$

Page No.:
Date:

Ques10. For the function n^k and a^n, what is the asymptotic relation between these function? Assume that $k \geq 1$ and $a > 1$ are constants. Find out the value of c and no. for what relation hold.

Ans10. If $c > 1$ then the exponential c^n for outgrows any term, so that answer is:

$$n^k \text{ is } O(c^n).$$

Ques11. What is the T.C of code and why?

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

Ans11. $i = 0, 1, 3, 6, 10, 15, \ldots$

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

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

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

Ques12. Write the recurrence relation for recursive function that prints fibonacci series. Solve recurrence relation to get T.C of program. what will be space complexity of this program and why?

Ans12. $T(n) = T(n-1) + T(n-2) + c$

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

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

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

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

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

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

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

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

⋮

General term —

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

$n - i = 0$

$\boxed{n = i}$

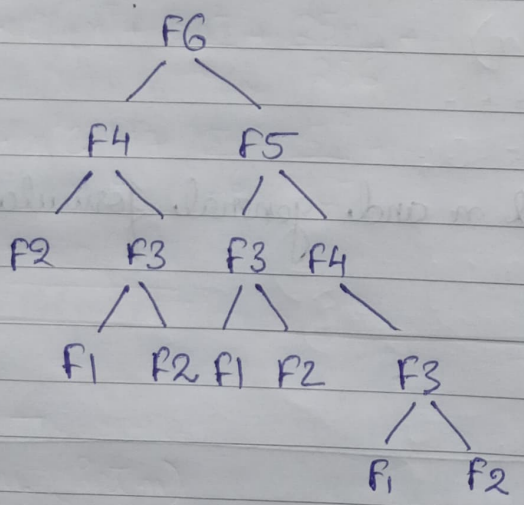
$T(n) = 2^n T(0) + (2^0 + 2^1 + 2^2 + \cdots + 2^{n-1})c$

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

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

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

Fib(6)



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

Ques 13. Write a program 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");
}
```

(2) 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 ("#");
     }
  }
}
```

(3) log (log n)

```
void
{
   bool prime [n+1];
   set ( 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;
}
```

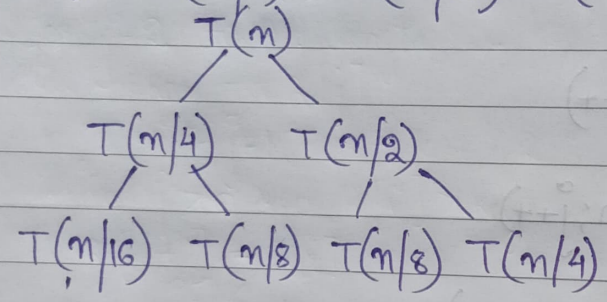
Ques14. $T(n) = T(n/4) + T(n/2) + cn^2$

Ans14. $T(1) = c$

$n = n/2$

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

$T(n) = T(n/4) + 2T(n/16) + c(n^2/16 + n^2/14 + n^2)$



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

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

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

Ques15. T.C of —

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

Page No.:
Date:

Ques15. 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 $= n + n/2 + n/3 + \cdots + n/2$
$= n\left(1 + \frac{1}{2} + 1/3 + \cdots + 1/n\right)$
$= n \log n$

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

Ques16. T.C of —

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

where k is a constant.

Ans16. $O(\log(\log n))$

Ques18. 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, 4^n, $n!$

(b) 1, $\log(\log_2(n))$, $\sqrt{\log n}$, $\log n$, $\log(2n)$, $\log(n!)$, $2\log(n)$, n, $2n$, $4n$, $n\log(n)$, n^2, $2(2^n)$, $n!$

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

Ques19. Write linear ~~sequ~~ 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.
```

Quesz0. Write pseudo code as------

Ansz0. Iterative methods 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 →

```
Insertion sort (A,n)
if n<=1
    return
Insertion sort (A,n-1)
    key = [n-1];
    j = m-2;
while j>=0 and A[j]&>key
A[j+1] = A[j]
    j = j-1
    A[j+1] = key
```

Insertion sort contain one i/p element per insertion and produces a partial solution without considering future element that's why it is called online sorting.

Other sorting algos that have been discussed in restore are

- Bubble sort
- Selection sort
- Quick sort
- Merge sort
- Heap sort
- Counting sort

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

Ques 22. Divide all sorting algo into

	In place	Stable	Online
Bubble Sort	Yes	Yes	Yes
Insertion Sort	Yes	Yes	Yes
Merge Sort	No	Yes	Yes
Quick sort	Yes	No	Yes
Selection sort	Yes	No	Yes
Heap sort	Yes	No	Yes
Count sort	No	Yes	Yes

Ques 23. Write recursive (iterative -----

Ans → 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 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 + i
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 $-O(1)$

Ques24. Write recurrence relation for binary recursive search.

Ans24. $T(n) = T(n/2) + 6$