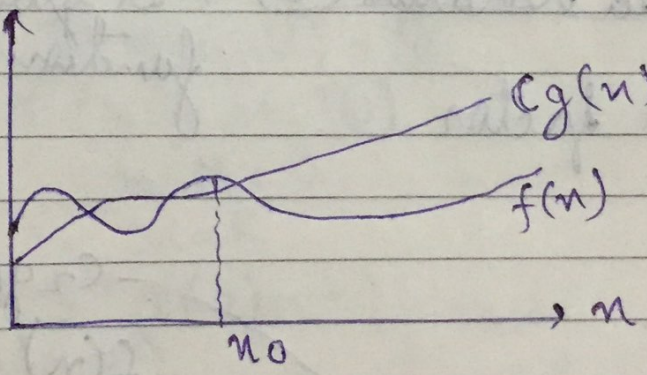
DAA Assignment

Name → Gaurav Pandey
Roll No → 1961050
Sec → A
Course → Btech (CSE)

Ans 1) Asymptotic notation are used to represent the complexities of algorithms for asymptotic analyses.

These notation are used for very large input

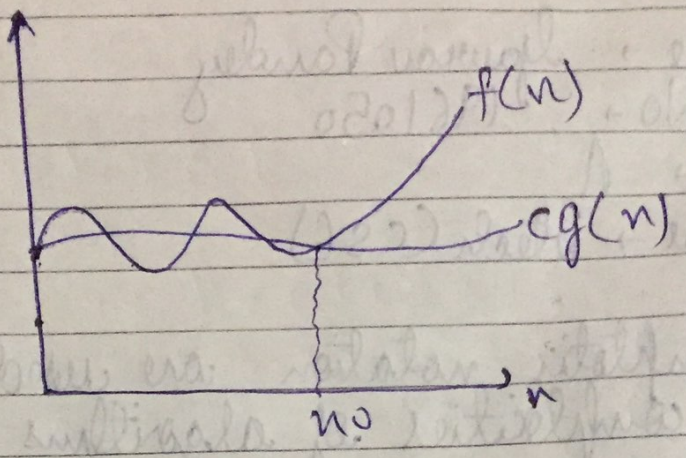
1) Big-oh (O) → It gives upper bound for a function f(n) to within a constant factor.



$f(n) \leq cg(n) \; \forall \; n \geq n_0 , c > 0$

eg → $O(n^2 + 3n) = O(n^2)$

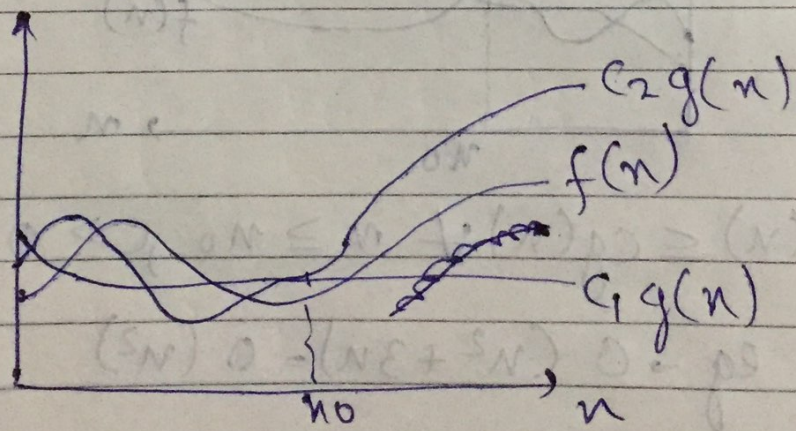
2. Big omega Notation (Ω) → Big Omega Notation (Ω) gives a lower bound for a f(n) to within a constant factor.



$\Omega(g(n)) = \{f(n)$: There exist (+ve) constant c & no such that
$0 \leq cg(n) \leq f(n) \; \forall \; n \geq no\}$

eg - $\Omega(n \log n)$

3) Big theta Notation (Θ) → It gives bound of function within a constant factor



$\Theta(g(n)) = \{f(n)$: There exist (+ve) constant
c1, c2 and no such that
$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \; \forall \; n \geq no\}$
eg - $\Theta(n^2)$

Ans 2)
```
for (i = 1 to n)
{
    i = i * 2;
}
```
1, 2, 4, 8, ... n

$T(n) = O(\log_2 n)$

Ans 3) $$T(n) = \begin{cases} 3T(n-1) & n>0 \\ 1 & n=0 \end{cases}$$

$T(n) = 3T(n-1)$ — ①

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

$T(n) = 9T(n-2)$ — ②

$T(n) = 3^3 T(n-3)$ — ③

$T(K) = 3^K T(n-K)$ — ④

for $T(n-K) = T(0)$

$n-K = 0$

$n = K$

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

$T(n) = 3^n$

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

Ans 4) $$T(n) = \begin{cases} 2T(n-1)-1 & , n>0 \\ 1 & n=0 \end{cases}$$

$T(n) = 2T(n-1) - 1$ — ①

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

$T(n) = 4T(n-2) - 1 - 2$ — ②

$T(n) = 8T(n-3) - (1+2+4)$ — ③

$$T(K) = 2^K T(n-K) - \underbrace{(1+2+4+\cdots 2^{K-1})}_{K \text{ terms}}$$

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

$$n = K$$

$$T(K) = 2^n T(0) - (1+2+4+\cdots) \quad (K \text{ terms})$$

Its a GP

$a = 1$

$r = 2$

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

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

$$T(n) = 1$$

$$T(n) = O(1)$$

Ans 5)

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

$1, 3, 6, 10, 15, \cdots \cdots n$

(K terms)

Its K^{th} term is $\frac{K(K+1)}{2} = n$

$K = \sqrt{n}$

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

Ans 6)
```
void function (int n) {
int i, count = 0;
for( int i = 1 ; i * i < n , i++)
        count++
}
```
$T(n) = O(\sqrt{n})$

Ans 7) $T(n) = O(n * \log_2 n * \log_2 n)$

$T(n) = O(n * (\log_2 n)^2)$

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

Ans 8)
```
function (int n) {                    T(n)
    if (n == 1) return;
for(i = 1 to n) {                     n^2
    for (j = 1 to n) {
        printf("*");
    }
}
function (n-3);                       T(n-3)
}
```

$T(n) = T(n-3) + n^2$ ——— (1)

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

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

$T(n) = T(n-5) + n^2 + (n-1)^2 + (n-2)^2$

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

for $T(n-k) = 1$
$k = n-1$

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

$T(n) = T(1) + (4^2 + 5^2 + \dots n^2)$

$T(n) = T(1) + \left(\frac{(n-3)(n-2)(2n-5)}{6}\right)$

$T(n) = 1 + \left(\frac{2n^3 + \dots}{6}\right)$

$T(n) = n^3$

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

Ans 9) void function(int n) {

```
for (i = 1 to n) {
    for (j = 1; j <= n; j = j + i)
        printf("*");
}
```

$i = 1$	n times	
$i = 2$	$1, 3, 5, \dots n$	n
$i = 3$	$1, 4, 7, \dots n$	$n/2$
⋮		
$i = n$	0	$n/3$

$$T(n) = \left(n + \frac{n}{2} + \frac{n}{3} \text{ ---} \right)$$

n times

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

Ans 10) for the functions n^k and a^n what is the relation

$k \geq 1$ & $a > 1$

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

Ans 11)

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

0, 3, 6, 10, 15 ---- n

to for this series is k terms

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

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

$$k \simeq \sqrt{n}$$

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

Ans 12) Recurrence relation of fabonacci series is

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

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

$T(n) = 8T(n-6) + 7$
$T(n) = 16T(n-8) + 15$
$T(n) = 2^k T(n-2k) + (2^k - 1)$

for $T(n-2k) = T(0)$
$n = 2k$
$k = \frac{n}{2}$

$T(n) = 2^{n/2} T(0) + (2^{n/2} - 1)$

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

Hence space complexity of fabonacci series is $O(n)$ as it depends on height of recursive tree & it is equal to n in fabonacci series.

Ans 3) $n(\log n)$

```
void fun() { for(int j = 0; j<n; i++){
    for(int i = 0; i<n; i = i*2)
    {
        print("*");
    }
}
}
```

```
void main()
{
fun();
}
```

→ n^3

```
#include <stdio.h>
void main()
{
int n;
cin >> n;
for(int i=0; i<n; i++){
for(int j=0; j<n; j++){
for(int k=0; k<n; k++){
x++;
}
}
}
}
```

→ log(log n)

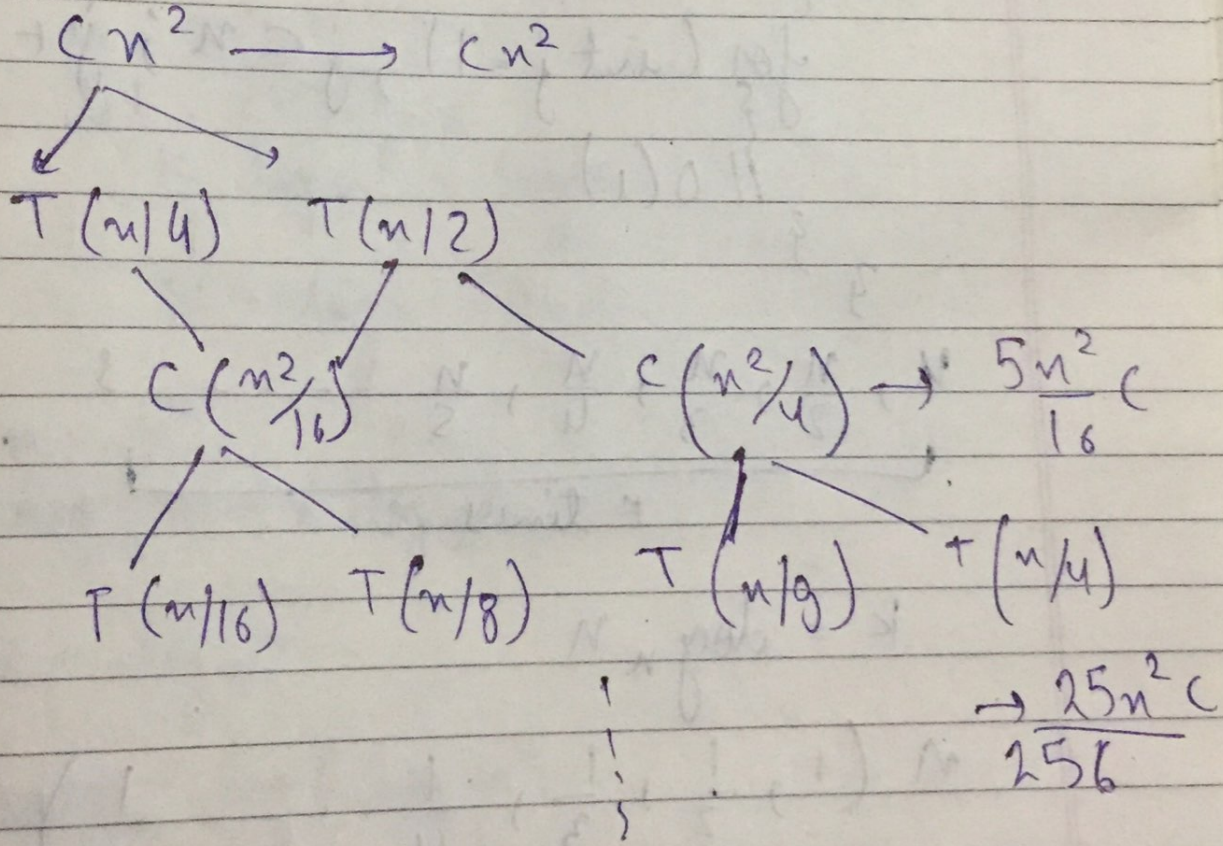
```
#include <bits/stdc++.h>
void fun(int n)
{
if (n==2)
return 1;
else
```

```
fun (dqrt & (n));
}
void main()
{
   fun (100);
}
```

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

$T(1) = c$

$T(0) = 0$



$T(n)$ = cost of each level

$$T(n) = cn^2 + \frac{5cn^2}{16} + \frac{25n^2}{256} + \text{----}$$

It is a G.P

with $a = n^2$

$r = 5/16$

to sum of sp.

$$T(n) = cn^2 / \left(\frac{1-5}{16}\right) = \frac{16cn^2}{11} = \frac{16cn^2}{11}$$

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

Ans 15) for (int i to n)

{

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

{

// O(1)

}

}

$$n, \frac{n}{2}, \frac{n}{3}, \frac{n}{4}, \frac{n}{5} \text{ ---- } 1$$

k times

$$k = \log_n n$$

$$n\left(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4} \text{ ---- } \frac{1}{n}\right)$$

$(n(\log n))$

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

Ans 6)

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

$2, 2^k, 2^{(k)^2}, 2^{k^3} \ldots\ldots n$

It is a GP

$a = 2$

$r = 2^k$

k^{th} term $= ar^{k-1}$

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

let $k^{(k-1)} = n$

$k \log_k k = \log n$

$k = \log n$ —— (1)

$n = 2^x$

$\log_2 n = x \log_2 2$

$x = \log_2 n$

$\log x = \log(\log n)$

from (1)

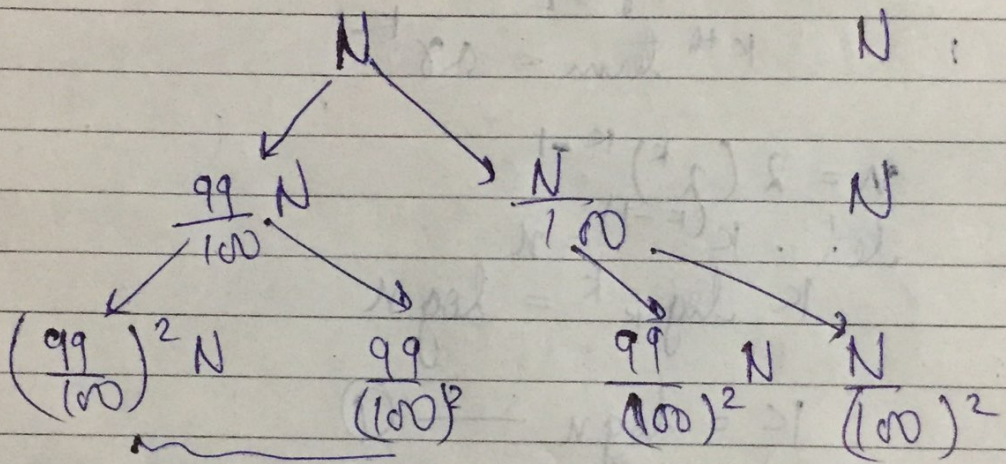
$k = \log(\log(n))$

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

Ans (7) Since pivot is divided in 99 % & 1 %.
so

$$T(n) = T\left(\frac{99}{100} N\right) + T\left(\frac{N}{100}\right) + N$$

Now as here we can use 2 extreme of a tree
when starting point is N



$$N\left(\frac{99(99)}{100\times 100} + \frac{99(1)}{100\times 100}\right) + \frac{100}{100\times 100} N = \frac{99}{100} N + \frac{N}{100}$$

$$= N$$

$$= N$$

so cost of each level is N only
Total cost = height * cost of each level
so for 1st term - $N, \frac{99}{100} N, \left(\frac{99}{100}\right)^2 N, \ldots$

$\left(\frac{99}{100}\right)^{h-1} N = 1$

$\left(\frac{99}{100}\right)^{h-1} = \frac{1}{N}$

$N = \left(\frac{100}{99}\right)^{h-1}$

$\log N = h \log(1)$

$h = \log N$ or $h = \frac{\log N}{\log(100/99)} + 1$

height of 2nd stream

$N, \frac{N}{100}, \frac{N}{(100)^2}, \frac{N}{(100)^3}, \text{---} \quad I$

$N \left(\frac{1}{100}\right)^{h-1} = 1$

$N = (100)^{h-1}$

$(n-1) \log 100 = \log N$

$n = \frac{\log N}{\log 100} + 1$ & $h = \log N$ (approx)

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

So time complexity is $O(N \log N)$

height of both extre is $\frac{\log N}{\log(100}$ + 1 of $\left(\frac{1}{100}\right)$

and $\frac{\log N}{\log\left(\frac{100}{99}\right)}$ + 1 of $\left(\frac{99}{100}\right)$

So we can conclude that if solution is done more than height of tree will be more & when division ratio is less then height is less.

Ans 18) (a) n, $n!$, $\log n$, $\log \log n$, root (n), $n \log^2 n$
2^n, 2^{2n}, 4^n, n^2, 100

Ans $O(100) < O(\log \log n) < O(\log n) < O(\sqrt{n})$
$< O(n) < O(n \log n) < O(n^2) < O(2^n) <$
$O(2^{2n}) < O(4^n)$

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

Ans $O(1) < O(\log(\log(n))) < O(\log(n)) < O(\log 2n)$
$< O(2 \log n) < O(n) < O(n \log(n)) < O(\log(n!))$
$< O(2n) < O(4n) < O(n^2) < O(n!)$
$< O(2(n^n))$

(C) 8^{n2n}, $\log_2 n$, $n \log_4(n)$, $n \log_2(n)$, $\log(n!)$, $n!$, $\log_8(n)$, 96, $8n^2$, $7n^3$, $5n$

Ans $O(96) < O(\log_8(n)) < O(\log_2 n) < O(\log(n))$
$< O(n \log_4(n)) < O(n \log_2(n)) < O(5n)$
$< O(8n^3) < O(7n^3) < O(n!) < O(8^{n2n})$

Ans 19)
```
void Linear Search (int arr[], int n, int key)
{
  for (i = 0 to i=n)
    if arr [i] == key
       cout << "found";
    else
    Continue
}
```

Ans 20) Iterative Insertion sort

```
Void Insertion sort (arr, n) {
  int i, temp, j
     for (i - 1 to n)
     {
       temp = arr[i]
          j = i - 1
  while j >=0 && arr[j] > temp
     {
       arr[j+1] = arr[j]
          j--
  }
```

```
arr[j+1] = temp
  }
}
```

Recursive Insertion sort

```
insertion sort (arr, n)
  {
   if n <=1
     return;
   insertion sort (arr, n-1);
    last = arr[n-1];
    j = n-2
    while (j >=0 and arr[j] > last)
     {
       arr[j+1] = arr[j]
           j--
     }
   arr[j+1] = last;
}
```

Insertion sort is called online sorting because it don't know the whole input it might make decision that later turns out to be not optimal.

Other algorithm are off-line algorithms that are discussed in lectures

Ans 21)

Time complexity

	Best	Avg	worst	Space
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion sort	(n)	$O(n^2)$	$O(n^2)$	$O(1)$
Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$ {due to recursion}
Quick sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(n)$
Heap sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$

Ans 22)

	inplace	stable	Online sorting
Bubble sort	yes	yes	No
Selection sort	yes	No	No
Insertion sort	yes	yes	yes
Merge sort	No	yes	No
Quick sort	yes	No	No
Heap sort	yes	No	No

Ans 23) Binary search (arr, int n, key)

```
{
  beg = 0
  end = n-1
  while (beg <= end)
  {
   mid = (beg + end) / 2

  if[arr [mid ] = = key ]
     found
  else if arr [mid ] < key
          beg = mid +1
  else
  end = mid - 1
  }
}
```

Time complexity of linear search - O(n)
Space complexity of linear search - O(1)

Time complexity of binary search - O(log n)
Space complexity of binary search - O(n)

Ans 24) $T(n) = \frac{T(n)}{2} + 1$