Name- Ayush Singh Pujara
Roll no:- 19140011
Subject- DAA
College - B. Tech (C.S.E)

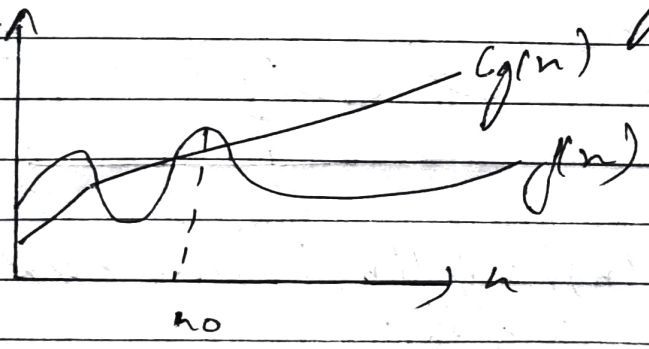
Assignment - 1

Ans-1 Asymptotic notation are used to represent the complexities of algorithms for asymptotic analysis.

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

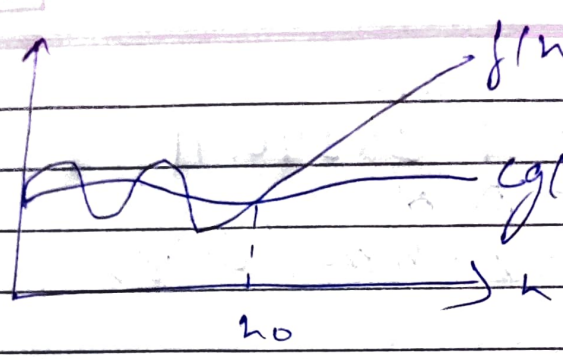


$f(n) \leq c \cdot g(n) \quad \forall \; n \geq n_0 \;, c > 0$

eg- $o(n^2 + 3n) = O(n^2)$

(2) Big-omega Notation (Ω)

Big-omega (Ω) notation gives a lower bound for $f(n)$ to within a constant factor

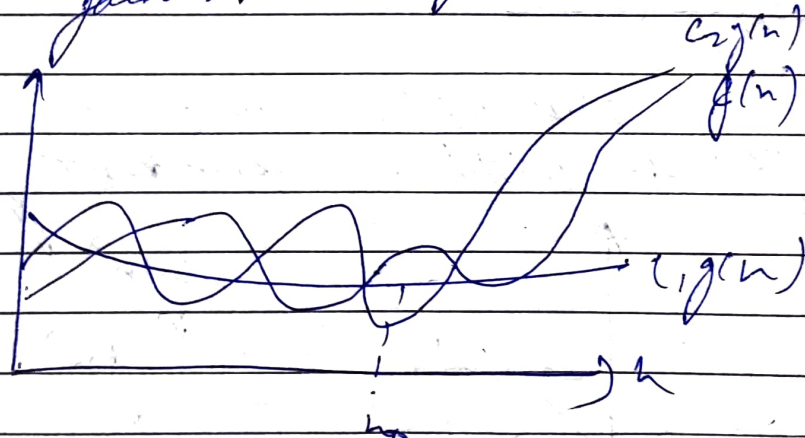


$\Omega(g(n)) = \{ f(n)$; There exist the constant c and n_0. such that $0 \leq cg(n) \leq f(n)$ $\forall n \geq n_0 \}$

eg $= n(n \log n)$

(3) Big theta notation (Θ)

It gives bound of function within a constant factor.



$\Theta(g(n)) = \{ f(n)$; There exist +ve constant c_1, c_2 and n_0.
such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \}$

eg $= \Theta(n^2)$

Ans 2

```
for(i = 1 to n)
{
    for(j = i; j <= 2; j++)
    {
```

1, 2, 4, 8, ---- n

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

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

$T(n) = 3T(n-1) \quad -(1)$

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

$T(n) = 9T(n-2) \quad -(2)$

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

$T(n) = 3^3 T(n-k) \quad -(4)$

put $T(n-k) = T(0)$

$n - k = 0$

$n = k$

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

$T(n) = 3^n$

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

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

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

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

$T(n) = 4T(n-2) - 1 - 2 \quad -(2)$

$T(n) = 8T(n-3) - (1+2+4) \quad -(3)$

$T(n) = 2^n T(n-k) - (1+2+4+\cdots 2^{k-1})$ (k terms)

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

$n = k$

$T(n) = 2^n T(0) - (1+2+4+\cdots)$ (k terms)

Here a (2-1)^0

a = 1

r = 2

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

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

$T(n) = 1$

$T(n) = O(1)$

Ans-5

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

1, 3, 6, 10, 15, ---- n

← k terms →

If k^{th} terms 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)

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

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

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

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

for $T(n-k) = 1$

$k = n-1$

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

$(n-3)$ terms.

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

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

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

$T(n) = n^3$

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

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 --- n n

i = 3 : 1, 4, 7 --- n n/2

⋮

i = n : 0

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

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

Ans-10 For the functions n^k and a^n, what is the relation.

$k \geq 1$ and $a > 1$

relation is $n^k = O(a^n)$

Ans-11

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

F. No.
Dt.

$\underbrace{0, 3, 6, 10, 15, \ldots\ldots n}_{k \text{ terms}}$

to for this series, is

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

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

$$n \approx k^2$$

$$k \approx \sqrt{n}$$

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

Ans-12 Recurrence relation of fabonacci series

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

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

$$T(n) = 4T(n-4) + 3$$
$$T(n) = 8T(n-6) + 7$$
$$T(n) = 16T(n-8) + 15$$

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

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 fabinacci series is $O(n)$ as it depends on height of recursive tree it is equal to n in this

Ans-15

```
for (int i to n)
{
    for(int j=1, j<n ; j+=i)
    {
        O(1)
    }
}
```

$n, \frac{n}{2}, \frac{n}{3}, \frac{n}{4}, \frac{n}{5}, \dots\dots 1$ (k times)

$k = \log_2 n$

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

$(n(\log n))$

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

Ans-16

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

$2, 2^k, 2^{k^2}, 2^{k^3} \dots\dots n$

A.P. $a = 2$

$r = 2^k$

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

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

let $k(k-1) = x$

$k \log_2 k = \log x$

$k = \log x - (1)$

$n = 2^k$

$\log_2 n = k \log_2 2$

$k = \log_2 n$

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

from (1)

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

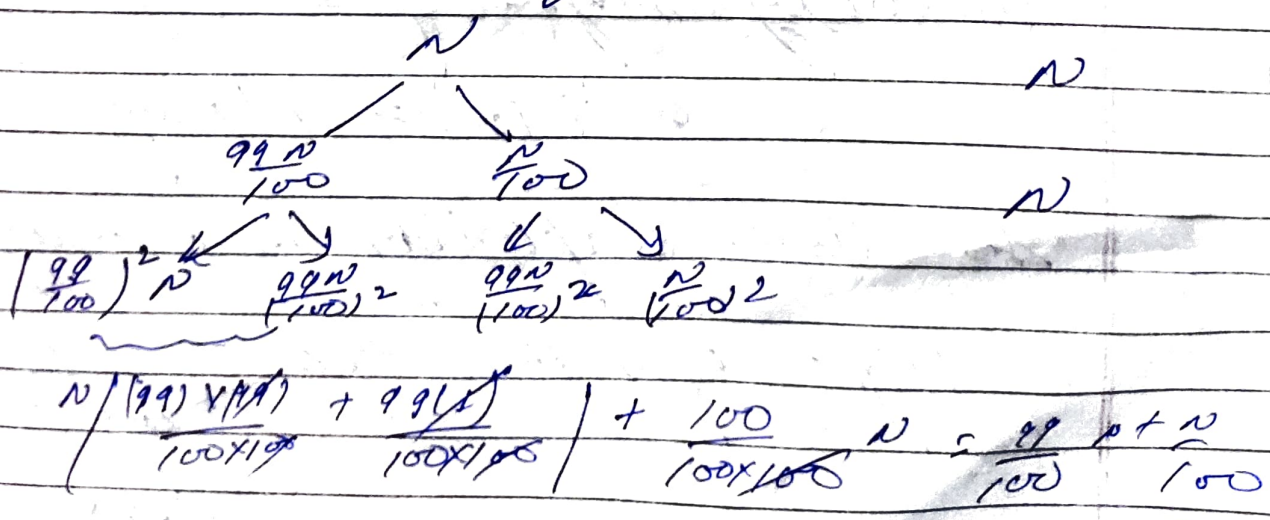
$T(n) = O(\log(\log(n)))$

Ans-17 hence pivot is divided in 99% and 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 tree.

where starting point is n



to cost of each level is N only.

Total cost of ~~each level~~ is T

= height * cost of each level.

to get last term = $N + \frac{99N}{100} + \left(\frac{99}{100}\right)^2 N$ ----

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

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

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

$$\log N = h \log\left(\frac{1}{1}\right)$$

$$h = \log N \text{ or } \gamma$$

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

similarly height of 2nd term -

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

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

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

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

so we can conclude that if division is done more than height of tree will be boost and when division ratio is less than height is less.

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 algorithm that are discussed in lectures.

Ans-21)

	Time complexity			space
	Best	Avg	worst	
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 binary search - O(n)

Space complexity of binary search - O(1)

Time complexity of Binary search = O(log n)

Space complexity of Binary search = O(n)

Ans-24

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