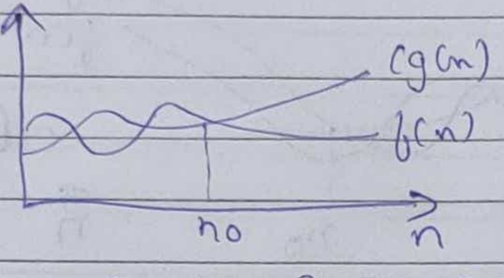
Page No.
Date 14 Aug 21

DAA assignment Name: Harshita Mehra
Roll no: 12 Sec: B

① What do you understand by Asymptotic notations. Define different Asymptotic notation with example.

Asymptotic notation are used to represent the complexities of algorithms for asymptotic analysis.
these notation are used for very large input

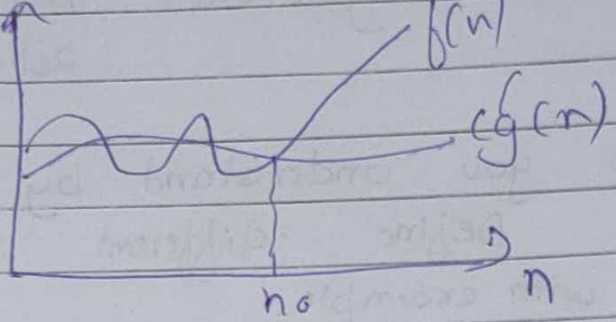
1 Big-oh (O).
It gives upper bound for a fxn f(n) to with a constant factor



eg: $O(n^2+3n) = O(n^2)$

2) Big omega notation (Ω)

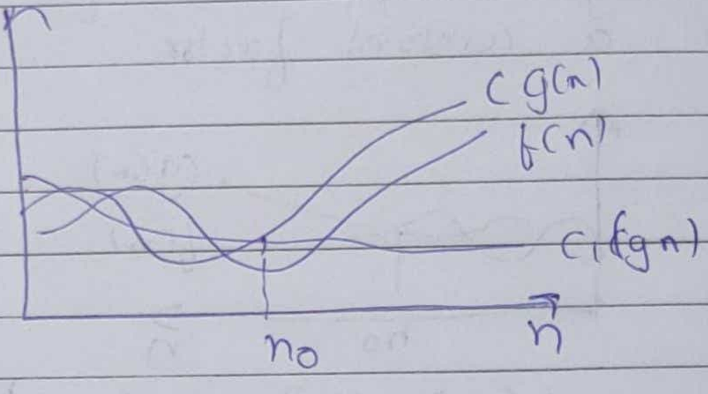
Gives us lower bound for a f(n) to within a constant factor.



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

3 Big theta Notation (θ)

it gives bound of fxn within a constant factor.



eg:- $\theta(n^2)$

Page No.
Date / /

2. What should be time complexity of
for (i=1 to n) {
i = i* 2;
}
$i = 1, 2, 4, 8, 16, 32, \ldots\ldots 2^k$
$2^k = n$
$\cdot\ k = \log n$
$T(n) = O(\log n)$

3. $T(n) = \{3T(n-1)$ if $n>0$, otherwise $1\}$
From backward substitution
$T(n) = 3T(n-1)$ — ①
$n = n-1$
Put in eq ①
$T(n-1) = 3T((n-1)-1) = 3T(n-2)$ — ②
Put $T(n-1)$ in eq ①
$T(n) = 3(3T(n-2))$
$T(n) = 9T(n-2)$ — ③
$n = n-2$
Put in eq ①
$T(n-2) = 3T((n-2)-1) = 3T(n-3)$ ④
Put eq $T(n-2)$ in eq ③
$T(n) = 9(3T(n-3))$
$= 27T(n-3)$ — ⑤
suppose $T(n) = 3^k T(n-k)$
$T(1) = 1$
$k = n-1$
$T(n) = 3^{n-1}(T(\cancel{n}-(\cancel{n}-1)))$
$T(n) = 3^{n-1}(T(1))$
$T(n) = 3^{n-1} = \frac{3^n}{3} =$
$T(n) = O(3^n)$

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

$T(n) = 2T(n-1) - 1$ — ①

$n = n-1$

Put in eq ①

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

$T(n-1) = 2T(n-2) - 1$ — ②

Put $T(n-1)$ in eq ①

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

$T(n) = 4T(n-2) - 2$ — ③

$n = n-2$

Put in eq ①

$T(n-2) = 2T(n-3) - 1$ — ④

Put $T(n-2)$ value in eq ③

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

$= 8T(n-3) - 3$

Suppose $= 2^k T(n-k) - 2^{k-1} - 2^{k-2} \cdots - 2^0$

$T(1) = 1$

$k = n-1$

$T(n) = 2^{n-1} T(1) - [\overbrace{2^0 + 2^1 + 2^2 + \cdots 2^{n-1}}^{AP}]$

$= 2^{n-1} - \left[\cancel{2}\left(\frac{-1 + 2^{n-1}}{\cancel{2}}\right)\right]$ (AP sum $= n/2$

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

$= 1$

$T(n) = 1$

$T(n) = O(1)$

5) Time complexity of

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

Sol: $O(\sqrt{n})$

6) Time complexity of

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

Sol $O(n)$

7) 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 ++;
    }
```

Sol: $O(n\log^2 n)$

8 Time complexity of

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

Sol. O(n)

9 Time complexity of -

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

Sol: O (n^2)

10 Time complexity of:
for the function, n^k, a^n what is asymptotic relationship b/w these fxns?

To assume for functions n^k and a^n what is the rotation

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

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

11) What is the time complexity of below code'

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

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

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

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

$k = \sqrt{n}$

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

(12) Write recurrence relation for the recursive fxn that ~~fx~~ prints Fibonacci series Solve the recurrence relation ~~re~~ to get time complexity of the program. What will be the space complexity of this program and why?

$T(n) = \{T(n-1) + T(n-2) + 1\}$
$= 2T(n-2) + 1$
$= 4T(n-4) + 3$
$= 8T(n-6) + 7$
$= 16T(n-8) + 15$

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

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

Space complexity $O(n)$ depends on height which is equal to n in fabinocci series.

13 Write program which have complexity - $n(\log n)$, ~~n^3~~ n^3, $\log(\log n)$

→ $n(\log n)$

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

→ n^3

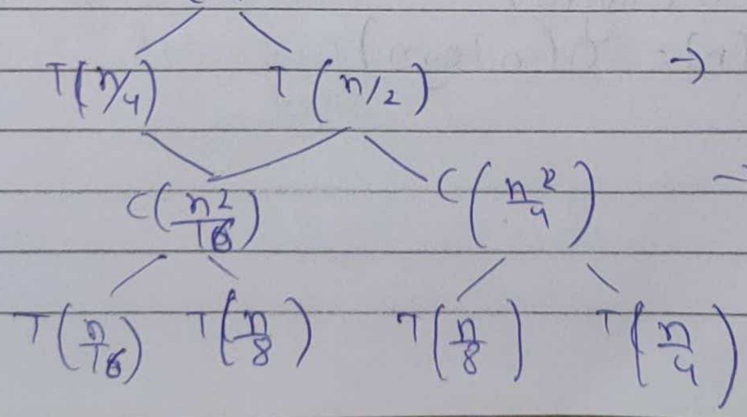
```
void main() {
    int n;
    cin>>n;
    for(int i=0; i<n; i++){
     for(int j=0; j<n; j++){
      for(int k=0; k<n; k++){
              n++;
    }}}
```

→ log (log n)

```
void fun(int n)
{   if (n==2)
            return 1;
    else
        fun(sqrt(n));
    }
void main()
{
    fun(100);
}
```

14 Solve the following recurre relation $T(n) = T(n/4) + T(\frac{n}{2}) + cn^2$

$T(n) = T(\frac{n}{4}) + T(\frac{n}{2}) + cn^2$



→ cn^2

→ $\frac{5n^2c}{16}$

→ $\frac{25n^2c}{256}$

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

it is a GP
with $a = n^2$
$r = \frac{5}{16}$

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

$= \frac{16cn^2}{11} = \frac{16cn^2}{11}$

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

15 What is the time complexity -

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

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

$k = \log_2 n$
$n\left(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots \frac{1}{n}\right)$
$(n(\log n))$
$T(n) = O(n \log n)$

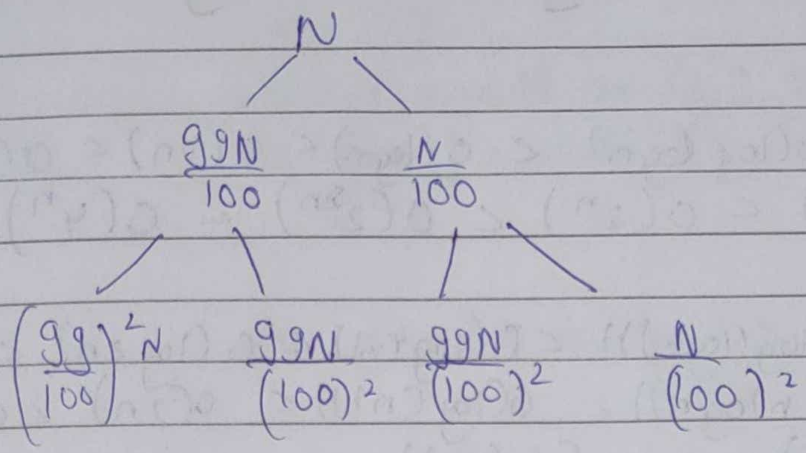
16 for(int i=2; i<n ; i= pow(i,k)){
// sometask

}

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

17) Hence pivot is divided in 99% & 1% so

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



so for first term = $N, \frac{99}{100}N, \left(\frac{99}{100}\right)^2 N, \ldots$

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

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

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

height of 2nd stream

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

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

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

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

15 Arrange the following in increasing order of rate of growth.

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

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

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

19 linear search pesudocode to search an element in a sorted array with minimum comparision

```
void LinearSearch(int arr[], int n, int key){
    for(i=0 to i<=n){
        if(arr[i]==key)
            cout<<"found";
        else
            continue;
    }
```

20 Write pseudo code for iterative and recursive insertion sort.

→ 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 turn out to be not optimal.
Other algorithm are offline algorithms that are discussed is lectures.

21 Complexity of all the sorting algorithms that has been discussed in lec.

	Time Complexity Best	Avg	worst	Space Complexity
Bubble Sort	O(n²)	O(n²)	O(n²)	O(1)
Selection Sort	O(n²)	O(n²)	O(n²)	O(1)
Insertion Sort	(n)	O(n²)	O(n²)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n²)	O(n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(1)

22 Divide all the sorting algo into place) stable / online sorting

	implace	Stable	Oniline Sorting
Bubble Sort	Yes	Yes	No
Selection Sort	Yes	No	No
Insertion Sort	Yes	Yes	No
Merge Sort	No	Yes	No
Quick Sort	Yes	No	No
Heap Sort	Yes	NO	NO

23 Write recurrence relation for binary search. Write Time and Space complexity of Linear & Binary Search.

Bineary 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 = $O(1)$

Binary Search
Time Complexity = $O(\log n)$
Space Complexity = $O(n)$.

24 What recurrence relation for binary recursive search.

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