Name - Khushi Saxena.
Btech (CSE), Sec-A
Roll No. - 1961084

DATE
PAGE No.

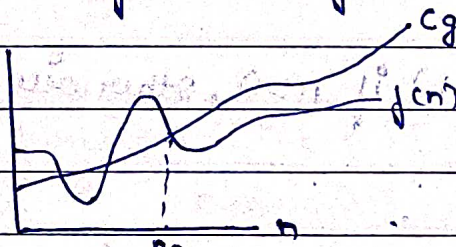
Design and Analysis of Algorithm
Assignment-1

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

→ Asymptotic notations are methods/languages using which we can define the running time of the algorithm based on input size.

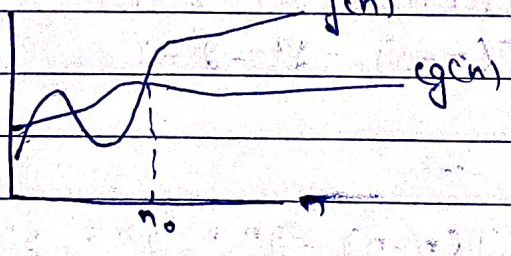
Types of Data Structure Asymptotic Notation -

1. Big-O Notation (O) represents the upper bound of the running time of an algorithm.



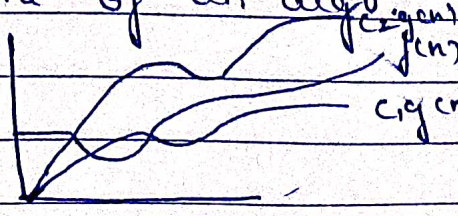
$0 \leq f(n) \leq cg(n)$

2. Omega Notation (Ω-notation) omega notation represe the lower bound of the running time of an algo



$0 \leq cg(n) \leq f(n)$

3. Theta Notation (Θ-notation) Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algo



$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$

DATE
PAGE No.

Small - O → upper bound on the growth rate of run time of an algorithm.

$$0 \leq f(n) \leq g(n)$$

Small - omega → denote the lower bound on the growth rate of runtime of an ~~g(n)~~ algorithm.

$$0 \leq g(n) < f(n).$$

2. for(i = 1 to n) { i = i * 2 }

$O(\log_2 n)$

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

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

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

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

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

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

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

$2^3 T(n-3) - 2^2 - 2^1 - 2^0$

- - -

- - -

$2^n T(n-n) - 2^{n-1} - 2^{n-2} - 2^{n-3}$

- - -

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

$T(n) = 1.$

$T(n) = O(1)$

DATE
PAGE No.

5. find the complexity:

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

loops → $1+2+3+\cdots+k = [k(k+1)/2] > n$

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

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

6.

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

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

7.

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

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

DATE
PAGE No.

(8) Time complexity of

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

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

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

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

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

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

for $T(n-k) = 1$

$k = n-1$

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

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

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

$T(n) = n^3$

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

(9) Time complexity of

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

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

(10) for the function n^k and a^n, what is the relation

→ n^k is $O(a^n)$

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

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

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

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

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

$k \approx \sqrt{n}$

(12) ~~time complexity of recursive fibonacci series is O(n).~~

Recurrence relation of fabanacci series is

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

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

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

$T(n) = 8T(n-6)+7$

$T(n) = 16T(n-8)+15$

$T(n) = 2^k T(n-2k) = T(0)$

$n = 2k$

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

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

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

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

hence space complexity of fabinci series is O(n) as it depends on height of recursive tree & it is equal to n in fibonacci series.

15. n(logn)

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

→ n^3

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

log(logn)

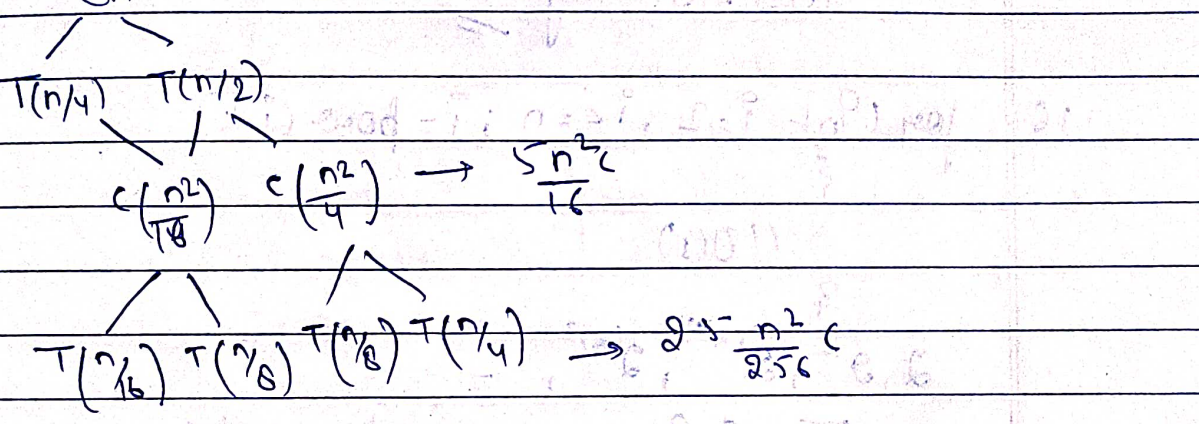
```
#include <bits/stdc++.h>
void fun (int n)
{
  if (n==2)
  return 1;
```

```
else
    fun(sqrt(n));
}
void main()
{
    fun(100);
}
```

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

$T(1) = 0$

$T(0) = 0$



$cn^2 \rightarrow cn^2$

$c\left(\frac{n^2}{16}\right)\ c\left(\frac{n^2}{4}\right) \rightarrow \frac{5n^2c}{16}$

$\rightarrow \frac{25n^2}{256}c$

$T(n)$ = cost of each level.

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

it is a G.P

with $a = n^2$

$r = \frac{5}{16}$

So sum of Sp

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

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

15- for (int i=0 to n)
{
for (int j=1 ; j<n ; j+=i)
{
O // O(i)
}
}

n, n/2, n/3, n/4 - - -

$K = \log_2 n$

$n(1, 1/2, 1/3, 1/4 - - - 1/n)$

$(n(\log n))$

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

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

$2, 2^k, 2^{(k^2)^2}, 2^{(k)^3}, - - -$

It G.P $a=2$, $r=2^k$, k^{th} term $= ar^{k-1}$

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

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

$k \log k = \log x$

$n = 2^x$

$\log_2 n = x \log 2$

$x = \log_2 n$

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

from (1)

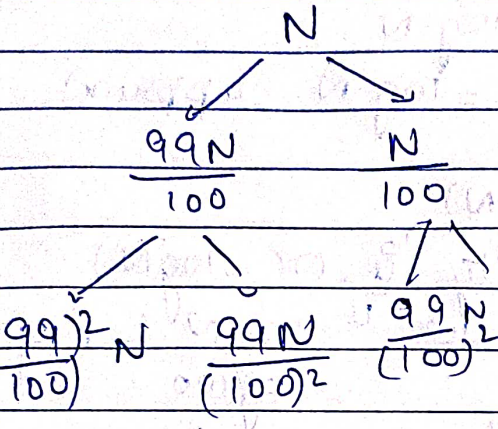
$k = \log(\log(n))$

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

17. hence pivot p is divided in 99%, b : 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.



$$N\left(\frac{(99)(99)}{100\times100} + \frac{99(1)}{100\times100}\right) + \frac{100}{100\times100}$$

$$= \frac{99}{100}N + \frac{N}{100}$$

$$= N$$

So cost of each level is N only.

Total cost = height * cost of each level

so for 1st stream $-N, \frac{99N}{100}, \left(\frac{99}{100}\right)^2 N, \ldots$

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

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

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

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

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

DATE
PAGE No.

height of 2nd stream

$$N, \frac{N}{100}, \frac{N}{(100)^2}, \frac{N}{(100)^3}$$

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

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

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

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

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

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

height of both entra 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 division is done more then height of tree will be more & and when division ratio is less then height is less.

Ans 18 - n, n!, logn, loglogn, sroot(n), nlog n, 2n, 2^n, 4^n, n^2, 100

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

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

3. $O(96) < O(\log_8(n)) < O(\log n) < O(\log_4(n!)) < O(n \log_4(n)) < O(n \log_2(n)) < O(5n) < O(8n^{\frac{2}{3}}) < O(7n^3) < O(n!) < O(8^{2n})$

19- Void linear search (int arr[], int n, int key)

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

20- Iterative Insertion sort

```
Void Insertion (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
  insertionsort (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 ~~the~~ be not optimal.

Other algorithm are off-line algorithms that are descussed in lectures.

Ans 1.

	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	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Quick sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(n)$
Heap sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$

22.

	Implace	Stable	Online sorting
B.S	Yes	Yes	No
S.S	Yes	No	No
I.S	Yes	Yes	Yes
M.S	No	Yes	No
Q.S	Yes	No	No
H.S	Yes	No	No

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
  }
}
```

→ T.C of Linear search = O(n)
S.C of Linear search = O(1)

→ T.C of Binary search = O(log n)
S.C of Binary search = O(n)

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