INDEX

NAME: Shraddha Markandey STD.: SEC.: ML

SCHOOL/COLLEGE: Graphic Era Deemed to be University ROLL No.: 80

E-mail. University Roll No - 2015281 MOBILE.:

S.No.	Date	Title	Page No.	Remarks
1	22.8.2021	Assignment-1	1-25	

Date
Page No.

..8.2021

Design and Analysis of Algorithms
Assignment-1

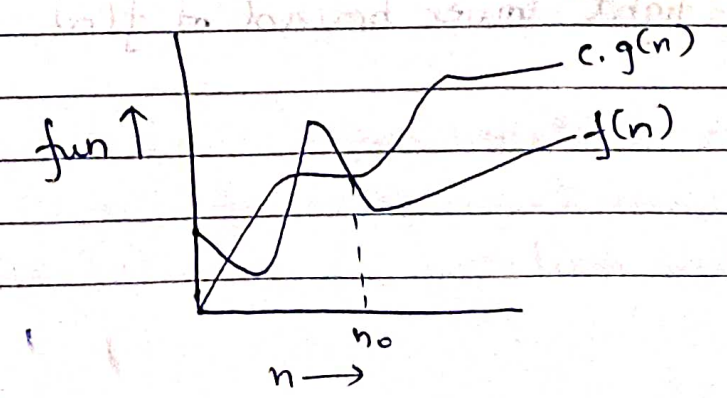
Q1 What do you understand by Asymptotic notations. Define different Asymptotic notation with examples.

Ans Asymptotic notations are methods / languages using which we can define the running time of the algorithm based on input size. The main idea of asymptotic analysis is to have a measure of the efficiency of algorithms that don't depend on machine-specific constants and doesn't require algorithms to be implemented and time taken by programs to be compared. Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis.

Various types of Asymptotic notations are -:

1 Big-O Notation

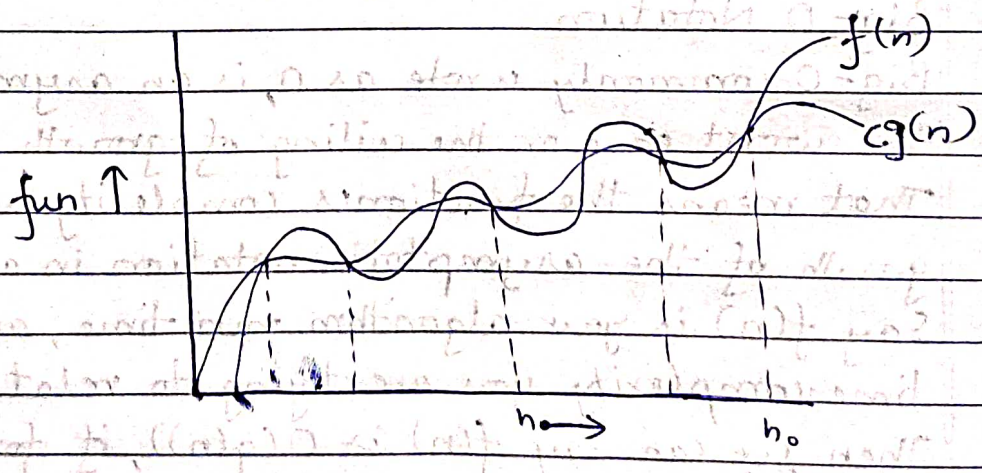
Big-O commonly wrote as O, is an asymptotic notation for the worst case or the ceiling of growth for a given function. That means the function's complexity will not cross the growth of the asymptotic notation in any case.
Say f(n) is your algorithm run time, and g(n) is an arbitary time complexity you are trying to relate to your algorithm. Then we can say f(n) is O(g(n)), if for some real constants c (c>0) and no, f(n)<=c g(n) for every input size n(n>=no)



Date
Page No

Example-
$f(n) = 3\log n + 100$
$g(n) = \log n$
Is $3\log n + 100 = O(\log n)$?
Is $3\log n + 100 <= c * \log n$
Let take $c = 200$
$3\log n + 100 <= 200 * \log n \quad \forall\ n > 2$ (undefined at $n = 1$)
Yes the definition of Big-O has been met therefore $f(n)$ is $O(g(n))$

2 Big-Omega Notation.
Big-Omega, commonly written as Ω, is an Asymptotic Notation for the best case, or a floor growth rate for a given function. It provides us with an asymptotic lower bound for the growth rate of run-time of an algorithm.
$f(n)$ is $\Omega(g(n))$, if for some real constants c ($c > 0$) and n_0 ($n_0 > 0$), $f(n)$ is $>= c\,g(n)$ for every input size n ($n >= n_0$)



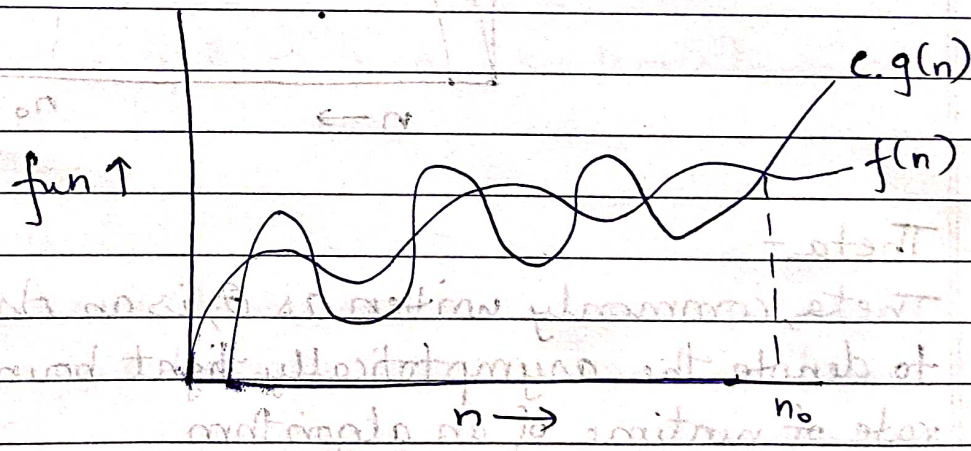
$g(n)$ is "tight" lower bound of $f(n)$

Date.
Page No.

3 Small-o Notation

Small-o, commonly written as o, is an Asymptotic Notation to denote the upper bound (that is not asymptotically tight) on the growth rate of run time of an algorithm.
$f(n)$ is $o(g(n))$, if for all real constants c ($c>0$) and some n_0 ($n_0>0$), $f(n) < c\,g(n)$ for every input size n ($n>n_0$)
The main difference between O-notation and o-notation ~~are~~ is that in $f(n)=O(g(n))$, the bound $f(n) <= g(n)$ holds for some constant $c>0$, but in $f(n)=o(g(n))$, the bound $f(n) < c.g(n)$ holds for all constants $c>0$. Intuitively, in o-notation, the function $f(n)$ becomes insignificant relative to $g(n)$ as n approaches infinity.

i.e $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

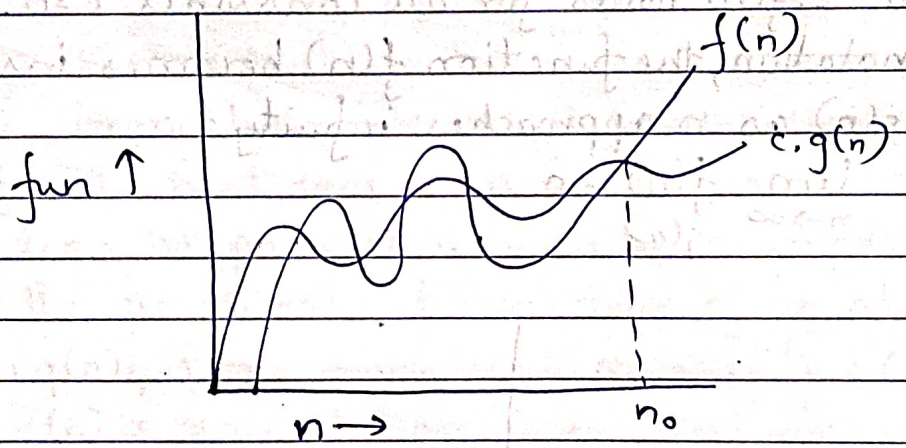


4 Small-omega Notation

Small-omega, commonly written as ω, is an Asymptotic Notation to denote the lower bound (that is not asymptotically tight) on the growth rate of runtime of an algorithm.
$f(n)$ is $\omega(g(n))$, if for all real constants c ($c>0$) and n_0 ($n_0>0$), $f(n) > c\,g(n)$ for every input size n ($n>n_0$). The main difference between Ω-notation and ~~ω~~ ω-notation

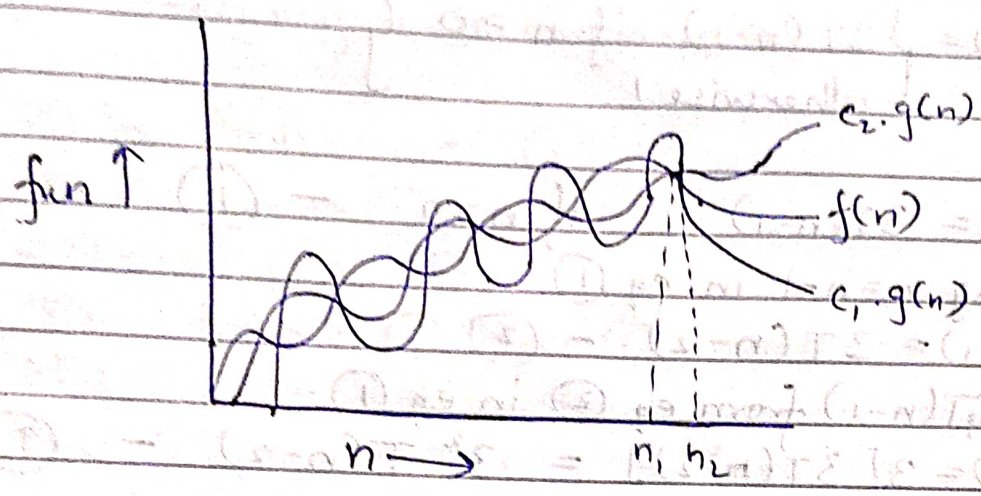
Date
Page No.

is that in $f(n) = \Omega(g(n))$, the bound $f(n) >= g(n)$ holds for some constant $c > 0$, but in $f(n) = \omega(g(n))$, the bound $f(n) > c\,g(n)$ holds for all constants $c > 0$.
Intuitively, in ω-notation, the function $f(n)$ becomes arbitrarily large relative to $g(n)$ as n approaches infinity.
ie. $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$



5 Theta -
Theta, commonly written as Θ, is an Asymptotic Notation to denote the asymptotically tight bound on the growth rate of runtime of an algorithm.
$f(n)$ is $\Theta(g(n))$, if for some real constants $c1$, $c2$ and n_0 ($c1 > 0$, $c2 > 0$, $n_0 > 0$), $c1 * g(n) < f(n) < c2 * g(n)$ for every input size n ($n >= n_0$).
$\therefore$ $f(n)$ is $\Theta(g(n))$ implies $f(n)$ is $O(g(n))$ as well as $f(n)$ is $\Omega(g(n))$

Date
Page No.



Q2 What should be time complexity of

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

Ans $\sum_{i=1,(i=i*2)}^{n} i$ = 1, 2, 4, 8, - - - n (k-terms)

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

Taking log on both sides

$$\log_2 2^{k-1} = \log_2 n$$

$$k-1 \log_2 2 = \log_2 n$$

$$k-1 = \log_2 n$$

$$k = \log_2 n + 1$$

Time Complexity = $O(\log n)$

Date
Page No.

Q3. $T(n) = \begin{cases} 3T(n-1) & \text{if } n > 0 \\ 1 & \text{otherwise} \end{cases}$

Ans $T(n) = 3T(n-1)$ if $n>0$ — (1)

Putting $n = n-1$ in eq (1)

$T(n-1) = 3T(n-2)$ — (2)

Putting $T(n-1)$ from eq (2) in eq (1)

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

Putting $n = n-2$ in eq (1)

$T(n-2) = 3T(n-3)$ — (4)

Putting $T(n-2)$ from eq (4) in eq (3)

$T(n) = 3^2[3T(n-3)] = 3^3 T(n-3)$ — (5)

$T(n) = 3^k T(n-k)$ — (6)

Base Case

$T(0) = 1$

So, $n-k = 0$

$n = k$

Putting $k = n$ in eq (6)

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

$= 3^n$

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

Date.
Page No.

Q4. $T(n) = \{2T(n-1)-1$ if $n>0$, otherwise $1\}$

Ans $T(n) = 2T(n-1)-1$ — (1)

Putting $n = n-1$ in eq (1)

$T(n-1) = 2T(n-2)-1$ — (2)

Putting $T(n-1)$ from eq (2) in eq (1)

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

$= 4T(n-2)-2-1 = 4T(n-2)-3$ — (3)

Putting $n = n-2$ in eq (1)

$T(n-2) = 2T(n-3)-1$ — (4)

Putting $T(n-2)$ from eq (4) in eq (3)

$T(n) = 4[2T(n-3)-1]-3 = 8T(n-3)-4-3 = 8T(n-3)-7$ — (4)

Putting $n = n-3$ in eq (1)

$T(n-3) = 2T(n-4)-1$ — (5)

Putting $T(n-3)$ from eq (5) in eq (4)

$T(n) = 8[2T(n-4)-1]-7 = 16T(n-4)-8-7 = 16T(n-4)-15$ — (6)

$T(n) = 2^k T(n-k) - (2^k-1)$ — (7)

Base Case

$T(0) = 1$

$n-k = 1$

$n = k$

Putting $k = n$ in eq (7)

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

$= 2^n - [2^n-1] = 2^n - 2^n + 1 = 1$

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

Date.
Page No.

Qs What should be time complexity of -

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

Ans After 1st iteration

$s = s+1$

After 2nd iteration

$s = s+1+2$

After 3rd iteration

$s = s+1+2+3$

After k iterations finally we have

$1+2+3+\cdots+k <= n$

So $\frac{k(k+1)}{2} = n$

$k(k+1) = 2n$

$k^2+k = 2n$

$k^2+k-2n = 0$

$$k = \frac{-1 \pm \sqrt{1-4\times1\times(-2n)}}{2\times1} = \frac{-1 \pm \sqrt{1+8n}}{2}$$

$$k = -\frac{1}{2} \pm \frac{\sqrt{1+8n}}{2}$$

Ignoring constants and coefficients we get time complexity as $O(\sqrt{n})$

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

Date.
Page No.

Q6 Time Complexity of -

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

Ans Time Complexity of count is O(1)

$i = 1, 2, 3, \cdots \sqrt{n}$

So the ~~tome~~ time complexity $T(n) = O(\sqrt{n})$ = Ans

Q7 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++
}
```

Ans $\sum_{i=n/2}^{n} \left(\sum_{j=1,(j=j*2)}^{n} \left(\sum_{k=1,(k=k*2)}^{n} 1 \right) \right)$

$= \sum_{i=n/2}^{n} \left(\sum_{j=1,(j=j*2)}^{n} \log n \right) = \sum_{i=n/2}^{n} (\log n)^2 = \left(\frac{n}{2}+1\right)(\log n)^2$

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

Date
Page No.

Q8 Time complexity of-

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

Ans Times complexity of nested loops = $\sum_{i=1}^{n} \sum_{j=1}^{n} 1$

$= \sum_{i=1}^{n} n = n^2$

$T(n) = T(n-3) + n^2$ — (1)

Putting $n = n-3$ in eq (1)

$T(n-3) = T(n-6) + (n-3)^2$ — (2)

Putting $T(n-3)$ from eq (2) in eq (1)

$T(n) = T(n-6) + (n-3)^2 + n^2$ — (3)

Putting $n = n-6$ in eq (1)

$T(n-6) = T(n-9) + (n-6)^2$ — (4)

Putting $T(n-6)$ from eq (4) in eq (3)

$T(n) = T(n-9) + (n-6)^2 + (n-3)^2 + n^2$

$T(n) = T(n-3k) + n^2 + (n-3)^2 + \dots + (n-3(k-1))^2$ — (5)

$T(1) = 0$

Date
Page No.

$n-3k=1$

$n-1=3k$

$k=\frac{n-1}{3}$

Putting value of k in eq (5)

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

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

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

$T(n)=T(n-3k)+(n-3(k-1))^2+(n-3(k-2))^2+\cdots+n^2$ — (5)

$n-3k=1$

$3k=n-1$

$k=\frac{n-1}{3}$

Putting value of k in eq (5)

$T(n)=T(1)+4^2+7^2+10^2+\cdots+n^2$

$=0+4^2+7^2+10^2+\cdots+n^2$

$=(3i+1)^2$

$$\sum_{i=1}^{\frac{n-1}{3}}(3i+1)^2=\sum_{i=1}^{\frac{n-1}{3}}9i^2+6i+1$$

$$=9\sum_{i=1}^{\frac{n-1}{3}}i^2+6\sum_{i=1}^{\frac{n-1}{3}}i+\sum_{i=1}^{\frac{n-1}{3}}1$$

$=9n^3+6n^2+n$

Time Complexity $=O(9n^3+6n^2+n)=O(n^3)$

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

Date.
Page No.

Q9 Time complexity of -

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

Ans $$\sum_{i=1}^{n}\left(\sum_{j=1,(j=j+i)}^{n} 1\right) = \sum_{i=1}^{n}\left(\frac{n-1+1}{i}\right)$$

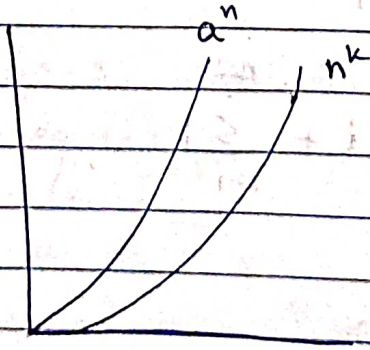
$$(n-1)\sum_{i=1}^{n} 1/i + \sum_{i=1}^{n} 1$$

$$(n-1)\log n + n$$

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

Q10 For the functions, n^k and a^n, what is the asymptotic relationship between these functions?
Assume that $k>=1$ and $a>1$ are constants. find out the value of c and n_0 for which relation holds.

Ans



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

$\therefore n^k \leq a^n . c \quad \forall\, c>0$ and $n \geq n_0$

Date
Page No.

Let $n = n_0$

$n_0^k \leq c \cdot a^{n_0}$

Say $k = a = 3$

$n_0^3 \leq c \cdot 3^{n_0} \quad \forall c \geq 1 \;\&\; n_0 \geq 1$

Q11 What is the time complexity of below code and why?

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

Ans If $n = 20$

j	i
1	0
2	1
3	3
4	6
5	10
6	15
7	21

$S = 0, 1, 3, 6, 10, 15, \ldots n$ — (1)

$S = 0, 1, 3, 6, 10, \ldots n$ — (2)

Subtract (2) from (1)

Date
Page No.

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

$n = 0 + 1 + 2 + 3 + \dots + k$

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

$n \simeq k^2$

$k = \sqrt{n}$

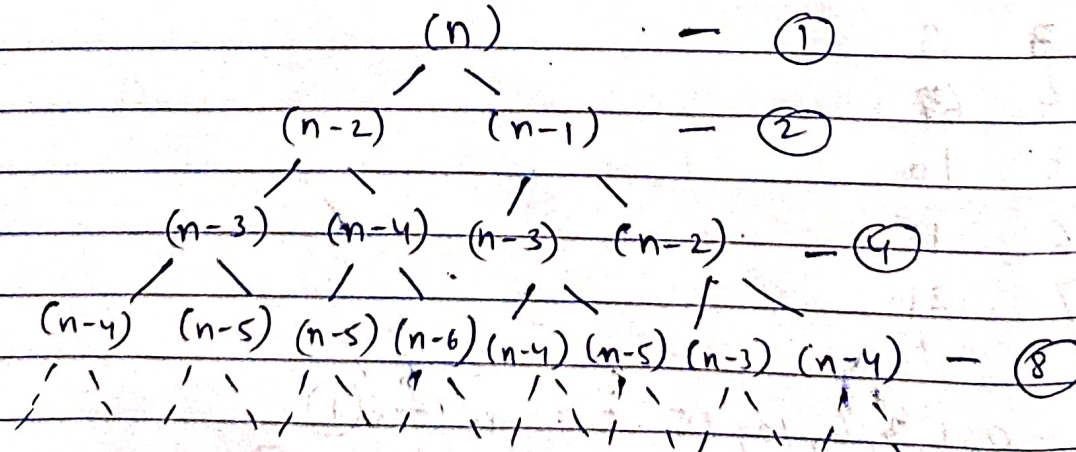
$T(n) = O(\sqrt{n})$

Q12 Write recurrence relation for the recursive function that prints fibonacci series. Solve the recurrence relation to get the time complexity of the program. What will be space complexity of this program and why?

Ans Program -

```
int fib(int n){
  if(n<=1)
     return n;
  return fib(n-1)+fib(n-2);
```

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



$= 2^n$

Date
Page No.

$T = 1 + 2 + 4 + \cdots + 2^n$

$a = 1$

$r = 2$

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

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

As we have only one statement to execute, we can draw a recursion tree, then height of the tree will be n, so the space complexity would be O(n) if we consider the stack size, else it will be constant O(1).

Q13 Write programs which have complexity - n(logn), n^3, log(logn)

Ans 1) Program which have complexity n(logn) -

```
sum=0
for(int i=1; i<=n; i*=2)
{
  for(int j=1; j<=n; j+=2)
  {
    sum+=j;
  }
}
```

2) Program which have complexity n^3 -

```
for(int i=1; i<=n; i++){
  for(int j=1; j<=n; j++){
    for(int k=1; k<=n; k++){
      print("*");
    }
  }
}
```

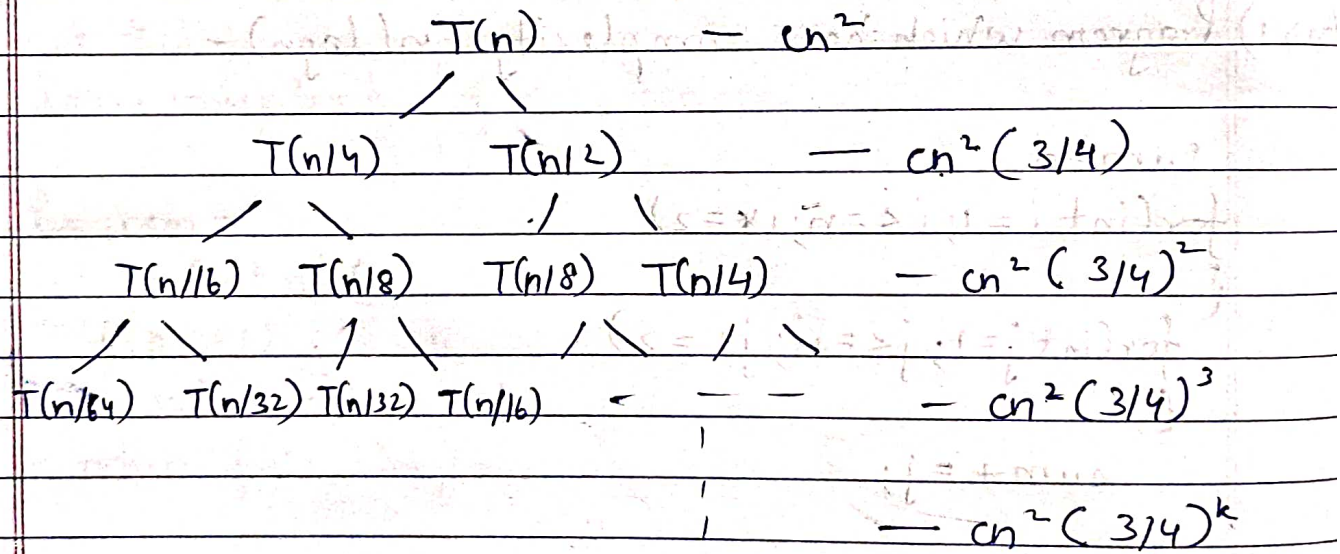
Date
Page No.

3) Program which have complexity log(logn)

```
int a=1, b=2;
while(a<=n){
    a*=b;
    b*=2;
}
```

Q14 Solve the following recurrence relation T(n)= T(n/4)+T(n/2) + cn^2

Ans



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

$$n = 2^k$$

$$k = \log n$$

$$T(n) = cn^2\left[1+\left(\frac{3}{4}\right)+\left(\frac{3}{4}\right)^2+\cdots\right]\left(\frac{3}{4}\right)^{\log n}$$

$$= cn^2(1)$$

$$= n^2$$

Date
Page No.

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

Q15 What is the time complexity of following function fun()?

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

Ans $T(n) = \sum_{i=1}^{n} \sum_{j=1,(j+=i)}^{n} 1 = \sum_{i=1}^{n} \frac{n-1}{i}$

$= (n-1)\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} - - - n\right) = n-1 \log n$

$T(n) = n \log n$

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

Q16 What should be the time complexity of

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

where, k is a constant

Ans $i = 2, 2^k, 2^{k^2}, 2^{k^3}, - - -$

$2^{k^x} = n$ $\quad \therefore k$ is constant

$\log(n) = k^x \log 2$

Date
Page No.

$\log n = k^x$

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

$x = \frac{\log(\log n)}{\log_k k} = \log(\log n)$

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

Q17 Write a recurrence relation when quick sort repeatedly divides the array into two parts of 99% and 1%. Derive the time complexity in this case. Show the recurrence tree while deriving the complexity and find the difference in heights of both the extreme parts. What do you understand by this analysis?

Ans $T(n) = T\left(\frac{99}{100} n\right) + \frac{n}{100}$ — (1)

$T(1) = 0$ — (2)

Putting $n = \frac{99}{100} n$ in eq (1)

$T\left(\frac{99}{100} n\right) = T\left(\left(\frac{99}{100}\right)^2 n\right) + \frac{n}{100}$ — (2)

Putting eq (2) in (1)

$T(n) = T\left(\left(\frac{99}{100}\right)^2 n\right) + \frac{2n}{100}$ — (3)

$T(n) = T\left(\left(\frac{99}{100}\right)^k n\right) + \frac{kn}{100}$ — (4)

Date.
Page No.

$$\left(\frac{99}{100}\right)^k n = 1$$

$$n = \left(\frac{100}{99}\right)^k$$

$$k = \log_{\frac{100}{99}} n = k \log_{\frac{100}{99}} \frac{100}{99} = \log_{\frac{100}{99}} n = k \quad -(5)$$

Putting $k = \log_{\frac{100}{99}} n$ in eq (4) we get

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

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

Q18 Arrange the following in increasing order of rate of growth

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

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

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

Ans a) $100 < \log\log n < \log n < \sqrt{n} < n < n\log n = \log(n!) < n^2 < 2^n < 2^{2n} < 4^n < n!$

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

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

Date
Page No.

Q19 Write linear search pseudo code to search an element in a sorted array with minimum comparisons.

Ans
```
int LinearSearch(int *arr, int n, int key){
  for(i=0 to n-1){
    if(arr[i]=key){
      return i;
    }
  }
  return -1;
}
```

Q20 Write pseudo code for iterative and recursive insertion sort. Insertion sort is called online sorting. Why? What about other sorting algorithms that has been discussed in lectures?

Ans

Iterative Insertion Sort

```
void insertionSort(int arr[], int n)
{
  int i, temp, j;
  for(i=1 to n){
    temp=arr[i];
    j=i-1;
    while(j>=0 AND arr[j]>temp){
      arr[j+1]=arr[j];
      j=j-1;
    }
    arr[j+1]=temp;
  }
}
```

Date
Page No.

Recursive Insertion Sort

```
void insertionSort(int arr[], int n)
{
    if(n<=1)
        return;
    insertionSort(arr, n-1);
    int last = arr[n-1];
    int j = n-2;
    while(j>=0 AND arr[j]>last)
    {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = last;
}
```

An online sorting is one that can process its input piece-by-piece in a serial fashion ie in the order that the input is fed to the algorithm, without having the entire input available from the beginning. Insertion sort considers one input element per iteration and produces a partial solution without considering future elements. Thus insertion sort is an online sorting. Whereas bubble sort, selection sort and merge sort are offline sorting as they require all inputs on which they can process the data for correct output ie these algorithms want all the input beforehand.

Date
Page No.

Q21 Complexity of all the sorting algorithms that has been discussed in lectures

Ans

	Algorithm	Time Complexity		
		Best Case	Average Case	Worst Case
1)	Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
2)	Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
3)	Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
4)	Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
5)	Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
6)	Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Q22 Divide all the sorting algorithms into in-place/stable/online sorting.

Ans

	Algorithm	In-place	Stable	Online
1	Bubble Sort	✓	✓	x
2	Selection Sort	✓	x	x
3	Insertion Sort	✓	✓	✓
4	Merge Sort	x	✓	x
5	Quick Sort	x	x	x
6	Heap Sort	✓	x	x

Q23 Write recursive/iterative pseudo code for binary search. What is the Time and Space complexity of linear and Binary Search (Recursive and Iterative)

Ans Iterative Binary Search

```
int BinarySearch(int arr[], int l, int r, int x)
{
    while (l <= r) {
```

Date
Page No.

```
        int m = (l+r)/2;
        if(arr[m] == x)
            return m;
        if(arr[m] < x)
            l = m+1;
        else
            r = m-1;
    }
    return -1;
}
```

Recursive Binary Search

```
int binarySearch(int arr[], int l, int r, int x)
{
    if(r >= l) {
        int mid = (l+r)/2;
    if  if(arr[mid] == x)
            return mid;
        else if(arr[mid] > x)
            return binarySearch(arr, l, mid-1, x);
        else
            return binarySearch(arr, mid+1, r, x);
    }
    return -1;
}
```

Iterative linear Search

Time Complexity

Best Case - O(1) Worst Case - O(n)

Average Case - O(n)

Date
Page No

Space Complexity
Best Case - O(1)
Average Case - O(1)
Worst Case - O(1)

Recursive linear Search
Time Complexity
Best Case - O(1)
Average Case - O(n)
Worst Case - O(n)

Space Complexity
Best Case - O(1)
Average Case - O(n)
Worst Case - O(n)

Iterative Binary Search
Time Complexity
Best Case - O(1)
Average Case - O(logn)
Worst Case - O(logn)

Space Complexity
Best Case - O(1)
Average Case - O(1)
Worst Case - O(1)

Recursive Binary Search
Time Complexity
Best Case - O(1)
Average Case - O(logn)
Worst Case - O(logn)

Date
Page No.

Space Complexity

Best Case – O(1)

Average Case – O(log n)

Worst Case – O(log n)

Q24 Write recurrence relation for binary recursive search

Ans Recurrence relation for binary recursive search is

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

∴ The time complexity of binary recursive search is O(log n)