Name- Mukul Dev Madic
Roll no- 42
Semester - V

Date.....

Design and Analysis of Algorithm

Assignment-1

Q1⇒ What do you understand by Asymptotic notation? Define different between Asympotic notation with examples.

Ans1: Asymptotic Notations:
Methods/language using which we can define the running time of an algorithm based on input size.

Different Asymptotic Notations are as follows-

1. Big O: for the worst case or the ceiling of growth for a given function.
→ Function's complexity will not cross the growth of the symptotic notation in any case.
→ $F(n) = O(g(n))$
example: $F(n) = 3\log n + 100$
$g(n) = \log n$

2. Big Omega: for the best case, or a floor growth rate for a given function
$F(n) = \Omega g(n)$ implies that $g(n)$ is tight lower bound of $f(n)$
→ $F(n)$ will never perform better than $g(n)$ after a threshold n_0
example: $F(n) = n(\log(n))$

Spiral

Date.....

f(n) is >= c g(n) ∀ n >= n0

3. Theta : Theta gives us tight upper and lower bound both

example: f(n) = θ(g(n))

then c1 * g(n) < f(n) < c2 * g(n) ∀ n ≥ n0

where (c1>0, c2>0, n0>0)

$3n^3 + 6n^2 + 6000 = \theta(n^3)$

4. Small Omega: TO denote lower bound (that is not asymptotically tight) on the growth rate of runtime of an algorithm.

f(n) = ω(g(n))

f(n) = c·g(n) ∀ n > n0

example: 4n + 8 ∈ ω(1)

5. Small oh: Notation used to describe an upper bound that cannot be tight

f(n) = o(g(n))

0 ∈ f(n) < c* g(n) where c>0

for integer constant n0 ≥ 1

example: 7n + 8 ∈ O(n²)

Spiral

Date.....

Que-2 What should be the time complexity of

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

Time complexity = $O(\log_2 n)$

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

Given, $T(0) = 1$

By ~~Backward~~ forward substitution

$T(1) = 3T(1-1) = 3T(0) = 3$

$T(2) = 3T(2-1) = 3T(1) = 3 \times 3$

$T(3) = 3T(3-1) = 3T(2) = 3 \times 3 \times 3$

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

$T(n) = 3 \times 3 \times 3 \times \ldots$ n times

Time complexity = $O(3^n)$

Que-4 $T(n) = \begin{cases} 2T(n-1)+1 & \text{if } n>0, \text{ otherwise } 1 \end{cases}$

Given, $T(0) = 1$

By Forward substitution

$T(1) = [2T(1-1)+1] = 2T(0)+1 = 2+1 = 1$

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

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

$T(n) = 1$

Time complexity = $O(1)$

Spiral

Date.....

Q.5 What should be the time complexity of

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

Ans 5:

while (s<=n) → s = 1+2+3+ ... + upto k terms

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

$k^2 + k = 2n$

$k^2 = 2n - k$ → as k is constant in Ans ignore it

$k^2 = 2n$

$k = \sqrt{2n}$

$k = \sqrt{n}$

Time complexity = $O(\sqrt{n})$

Que-6 Time complexity of

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

Ans-6 loop i will run till $\sqrt{n}$ as $i^2 < n$

Date

Time complexity = $O(\sqrt{n})$

Que-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++;
            }
        }
    }
}
```

Ans-7: i = loop 1 will run = $O(n/2)$

j = loop 2 will run = $O(\log n)$

k = loop 3 will run = $O(\log n)$

Time complexity = $O(n/2) * O(\log n) * O(\log n)$

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

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

Que-8 Time complexity of:

```
function (int n)
{
    if (n == 1)
    return;
```

Spiral

Date.....

```
For (i=1 to n)
{
    Printf ("*");
}
Function (n-3);
}
```

Ans- B

$n - 3k = 1$

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

For each k, "*" will print $(n-k)*(n-k)$ times

$TC = \left(\frac{n-1}{3}\right)(n-k)(n-k)$

$= \left(\frac{n-1}{3}\right)\left(n-\left(\frac{n-1}{3}\right)\right)\left(n-\left(\frac{n-1}{3}\right)\right)$

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

$= \left(\frac{n-1}{3}\right)\left(\frac{4n^2+1+4n}{9}\right)$

$= \frac{4n^3+4n^2+n-4n^2-4n-1}{27}$

Time complexity = $O(n^3)$

Here, all the constant are ignored and retaining the highest order term

Spiral

Date

Que 9 Time complexity of

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

Ans 9. Here

```
for (j=1; j<n; j=j+i)
        TC = O(√n)
for (i=1 to n)
        TC = O(n)
```

Total time complexity = $O(n\sqrt{n})$

Que 10 For the functions n^k and a^n, what is the asymptotic relationship between these functions?
Assume that $k \geq 1$ and $a > 1$ are constants
Find out the value of c and n_0 for which relation holds.

Ans 10 $n^k = O(a^n)$

Que 11 Time complexity of

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

Spiral

Date.....

```
    i=i+1;
    j++;
  }
}
```

Ans:

i	j	time
0	1	1
1	2	1
3	3	1
6	4	1
10	5	1

Total time = 1+3+6+10+ ... + k terms = n

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

$$k^2 + k = 2n$$

$$k^2 = n$$

$$k = \sqrt{n}$$

total number of steps = k

Time complexity of each step = O(1)

Total TC = O(k*1)

= O(√n)

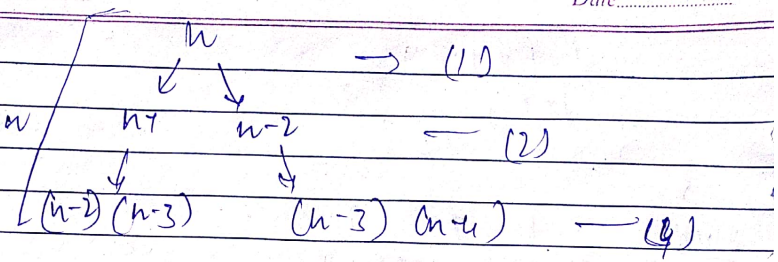
Ques 2 Write recurrence relation for the recursive function that print fibonnaci series. Solve the recurrence relation to get time complexity what will be the space complexity and why?

Ans-1 Recurrence relation T(n) = T(n-1) + T(n-2) + 1

Solve by recurrence tree)

Spiral

Date.....



→ (1)

— (2)

— (3)

Ans Recurrence relation:

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

depth of tree is n

Thus successive stack will be get given then space complexity O(n)

Total steps = O(n)

Total number of steps = 2^n

$1 + 2 + 4 + 8 + \dots + 2^n$

$S_{nn} = \frac{a(r^n - 1)}{r - 1} = \frac{2^{n+1} - 1}{2 - 1}$

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

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

Que-13 Time complexity of (write code)

```
i. n log n
int main()
{
    int count = 0
    for (int i = 0; i < n; i++)
```

Spiral

Date.....

```
for (int i=0 ; i<n ; i=i*2)
    count++;
}
```

2. n^3

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

Que.14. Solve the following Recurrence Relation

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

Recurrence tree : cn^2

$T(n/4)$ $T(n/2)$ → $c(\frac{n^2}{16})$ $c(\frac{n^2}{4})$

$T(n/16)$ $T(n/8)$ $T(n/8)$ $T(n/4)$ → $c(n^2/256)$ $c(\frac{n^2}{64})$

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

$TC = O(n^2/(1-5)/16)) = O(n^2)$

Spiral

Date.....

Que-15 Time complexity of

```
int Fun(int n)
    for(int i=1; i<=n; i++)          → O(n)
    {
        for(int j=1; j<n; j=j*i)     → O(√n)
        {
            // some O(1) task
        }
    }
```

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

Que-16 Time complexity of

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

where, k is constant

Ans-16 $i = 2^k, 2^{k^2}, 2^{k^3}, \ldots, (2^k)^f = n$

$= 2^{k^f} = 2^{\log_2 n}$

Total number of iteration = $\log_k \log n$

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

Spiral

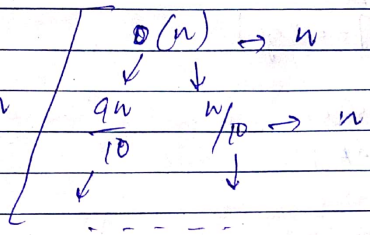
Date.....

Que.17 Given, quick sort divide array into two parts of 99% and 1%

$\Rightarrow \frac{9n}{10} + \frac{n}{10}$ (supporting size of array = n)

Recurrence Relation = $T(n) = T(9n/10) + T(n/10) + O(n)$

Solving by the method



No. of iterations = $\log_{10/9} n$

TC = $O(n * \log_{10/9} n)$ taking longer branch

= $O(n \log n)$

Que.18 Arrange in increasing order

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

b. $1 < \lg \log n < \sqrt{\log n} < \log n < \log n < 2 \log n < \log n! < n^2 < n \log n < 2^n < 4^n < n^2 < 2^{n!} < n!$

Spiral

Date.....

Q $96 < \log n < \log n < n\log n < n \log n < \log(n!)$
$< 5n < 8n^2 < 7n^3 < n! < 8^{2n}$

Que-19 Write linear search pseudocode to search an element in a sorted array with minimum comparison

```
int linearsearch (int arr[], int n, int key)
{
    for i=1 to n
      if (arr[i] == key)
      {
          Print i;
          break;
      }
}
```

Que-20 Iterative insertion sort

```
void insertionsort (int arr[], int n)
{
    for i=1 to n
    int value ← arr[i];
    int j ← i;
    while j>0 and arr[j-1] < value
    {
        arr[j] ← arr[j-1];
        j--;
    }
    arr[j] ← value;
}
```

Date.....

Recursive

```
void insertionsort (int arr[], int i, int n)
{
    int value ← arr[i];
    int j ← i;

    while j > 0 and arr[j-1] > value
    {
        arr[j] ← arr[j-1];
        j--;
    }
    arr[j] ← value;
    if (i <= n)
    {
        insertionsort (arr, i+1, n);
    }
}
```

Insertion sort is an online sorting algorithm since it can sort a list as it receives it. In all other algorithm we need all elements to be provided to the algorithm before applying it.

Que: Complexity of sorting algorithm that has been discussed in lectures

Insertion sort → $O(n^2)$
Bubble sort → $O(n^2)$
Selection sort → $O(n^2)$

Spiral

Date.....

Que-22 Divide all sorting algorithm into in Inplace/stable/online sorting

	Inplace	stable	Online
Insertion	✓	✗	✓
Selection	✓	✗	✗
Bubble	✓	✓	✗
Quick	✓	✗	✗
Merge	✗	✓	✗

Que-23

```
int binarysearch (int arr[], int n, int key)
{
    int l ← arr[0];
    int r ← arr[n-1];
    while (l <= r)
    {
        mid = ⌈(l+(r-1))/2⌉
        if (mid == key)
            return mid;
        else if (mid > key)
            r ← mid-1;
        else
            l ← mid+1;
    }
    return -1;
}
```

Spiral

Date.....

→ Time complexity linear search : O(n)
(Iterative)
space complexity : O(1)

→ Time complexity linear search : O(n)
(Recursive)
space complexity : O(n)

→ Time complexity Binary search : O(logn)
(Iterative)
space complexity : O(1)

→ Time complexity Binary search : O(logn)
(Recursive)
space complexity : O(logn)

Que-4 Write recursive relation for binary recursive search

Ans-4 $T(n) = T(n/2) + 1$