Tutorial -2

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

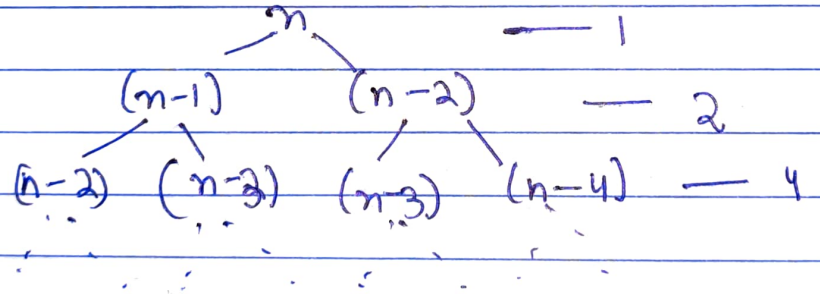
i is increasing at the rate of j.

$0 + 1 + \cdots + k = \frac{k(k+1)}{2} > n$

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

2. The recurrence relation for the recursive method of fibonacci series is -

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



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

$= \underbrace{\qquad\qquad}_{G.P.}$

$\text{Sum} = 2^{n+1}$

$T.C = O(2^{n+1}) = O(2^n)$

Space complexity $= O(n)$

Date
Page No.

3. Program with complexities

(i) $n \log n$ -

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

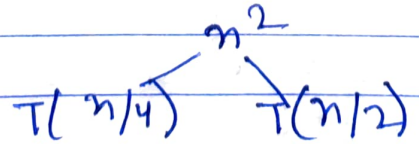
(ii) n^3 -

```
Void fun ( int n){
for( int i= 1;  i<=n ; i++){
  for( int j=1 ; j<=n ; j++){
     for ( int k=1 ; k<=n ; k++){
          printf("#");
}}}
```

(iii) $\log(\log n)$

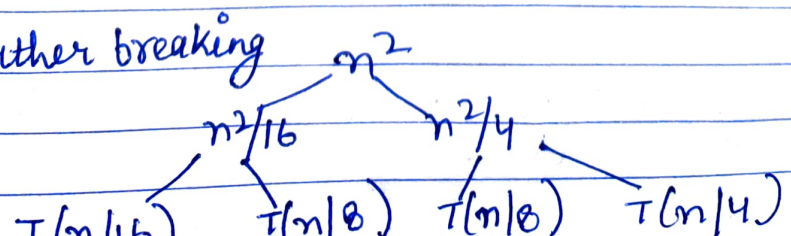
```
for ( int i= 2; i<=n ; i= pow(i,k)) { printf("#"); }
```

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



Initial recursion tree.

On further breaking



$T(n) = n^2 + 5n^2/16 + 25n^2/256 + \cdots$

Sum of GP $= \frac{n^2}{1-5/16} = \frac{n^2}{11/16}$

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

5. $O(n \log n)$

6.
```
for (int i=2; i<=n; i=pow(i,k))
{ // O(1)
}
```

In this case i takes value $2, 2^k, (2^k)^k = 2^{k^2}, 2^{k^3}$

$2^{k^{\log_k(\log n)}}$

The last term must be less than equal to n.

$2^{k^{\log_k(\log n)}} = 2^{\log n} = n$, Its True.

There are total $\log_k(\log n)$ iterations and each iteration takes constant amount of time to run.

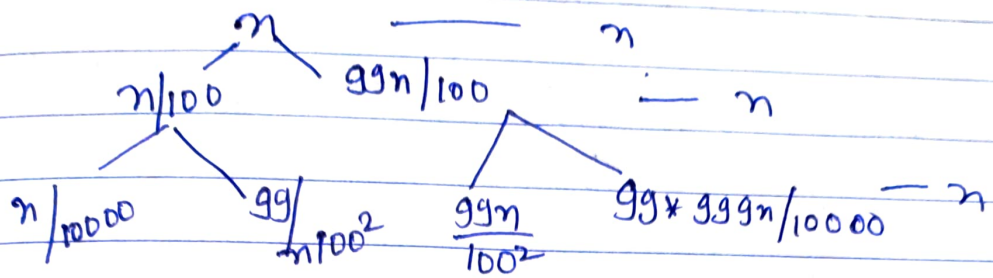
Total T.C $= O(\log(\log n))$

7. The running time when in quick sort when the partition is putting 99% of elements on one side and 1% of elements on another in each repetition.

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

Date
Page No.

Recursion Tree



We can see that initially, the cost is in on all four levels, This will follow until the left most branch of the tree reaches its best case because the left most branch has least elements in each division, so it will finish first.

At level i, the rightmost node has $n * \left(\frac{99}{100}\right)^i$ elements, for the last level,

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

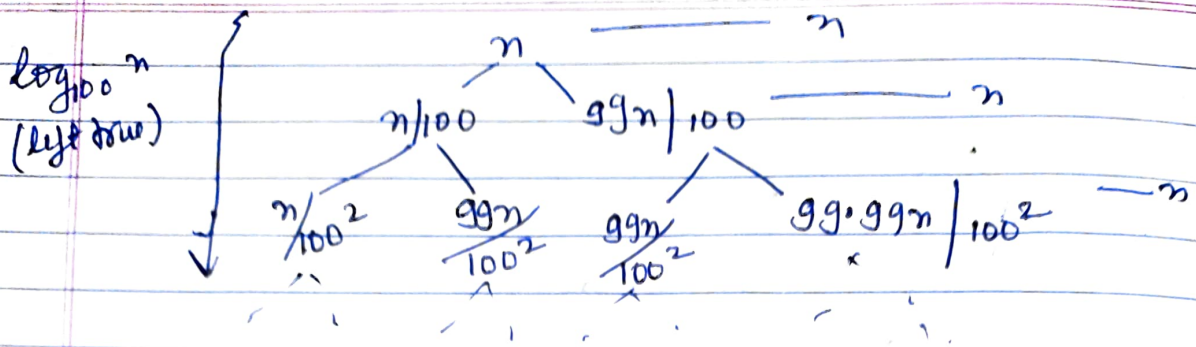
$$i = \log_{\frac{100}{99}} n$$

So there are total $\left(\log_{\frac{100}{99}} n\right) + 1$ levels

This, $T(n) = \left(\frac{cn + cn + \cdots \text{ } cn}{\log_{\frac{100}{99}} n + 1 \text{ times}}\right) < \left(\log_{\frac{100}{99}} n + 1\right) cn$

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

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



Starting with subproblem of size 1 & multiplying it by 100 until we reach relation

$100^k = n$

$k = \log_{100} n$

Right child is $\frac{99}{100}$ of size of nodes above it. Each parent is $\frac{100}{99}$ times the size of right child.

8. Increasing order of rate of growth,

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

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

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