Tutorial-2

Name: Km Parshvi Sahu
Uni. Roll no: 1918035
Sec : D4

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

If i is increasing at the rate of j

→ If k is total no. of iterations, while loop terminals.

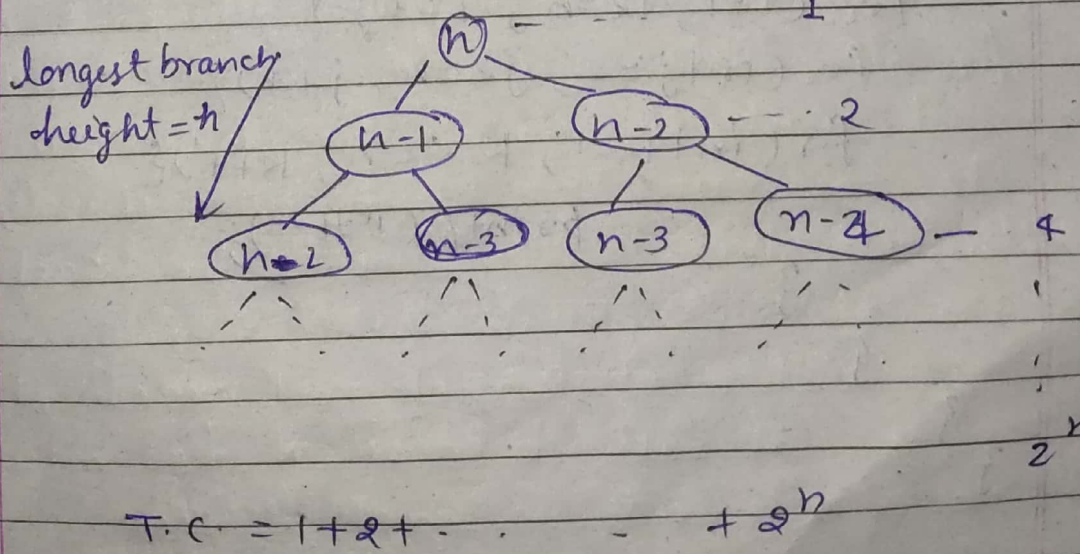
if, $0+1+\cdots+k = \frac{k(k+1)}{2} \to n$

T.C. → $O(\sqrt{n})$

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

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

Solving using tree method -



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

$a = 1 \quad r = 2 \quad S = \frac{a(r^{term} - 1)}{r - 1}$

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

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

Space complexity = $O(n)$ [∵ stack size now exceeds the depth of the call's tree shown above]

Ans3: Program with complexity =

i) $n \log n$ —

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

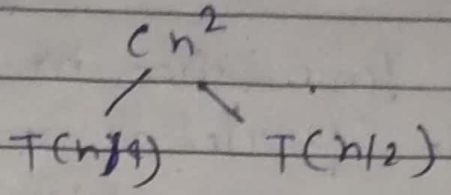
ii) n^3

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

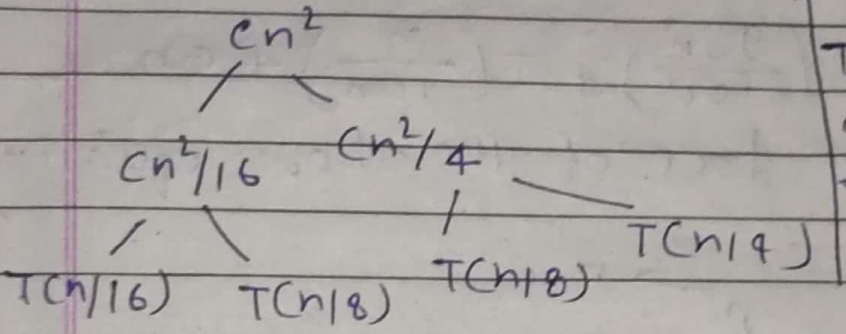
(iii) $\log(\log n)$ → for(int i=2; i<=n; i = pow(i,k)){ //O(1) expression }
also, interpolation search has this complexity

Answer4:- $T(n) = T(n/4) + T(n/2) + cn^2$

following is the initial recursion tree



on further breaking down,



To row the value of T(n) we need to calculate the sum of tree nodes level by level

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

$S_\infty = \frac{n^2}{1-5/16} \Rightarrow T.C = O(n^2)$

Ans5:- Same as ques 9
→ $O(n \log n)$

Ans6:- for(int i=2; i<=n; i pow(i,k))
{
//O(1) expression

In this case i takes value $2, 2^k, (2^k)^k, (2^{k^2})^k = 2^{k^3}$
The last sum must be less than or equal to n, we have,

$2^{k \log_k(\log(n))} = 2^{\log n} = n$, It's there

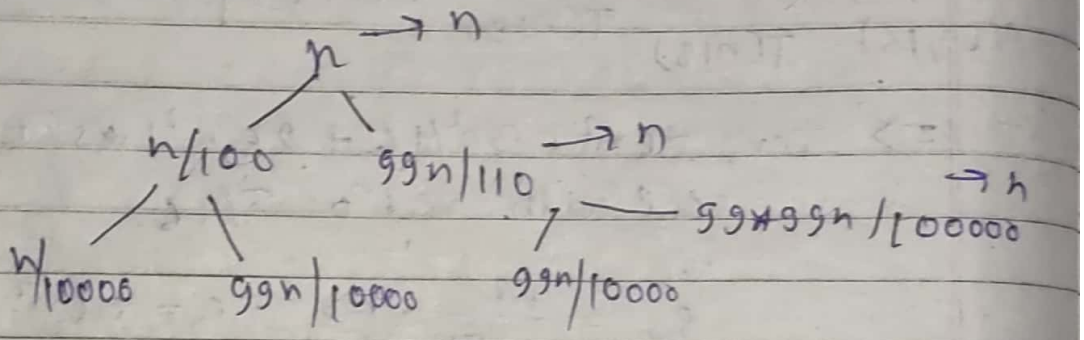
There are total $\log_k(\log(n))$ many iterations and each iteration takes constant amount of time to run,

∴ Total times complexity $= O(\log[\log n])$

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

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

Recursion tree of the above equation is,



We can see that initially, the cost is in for all levels this will follow until the left most branch of the tree reaches its base case (size 1) because the left most branch has least elements in each division so it will finish fruit.

The rightmost branch will reach its base case at last because it has maximum no. of elements in each division.

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

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

$$\rightarrow i = \frac{\log_{\frac{100}{99}} n}{\log_{\frac{100}{99}} \frac{100}{99}} \quad \left(\log_{\frac{100}{99}} n\right)$$

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

Thus,

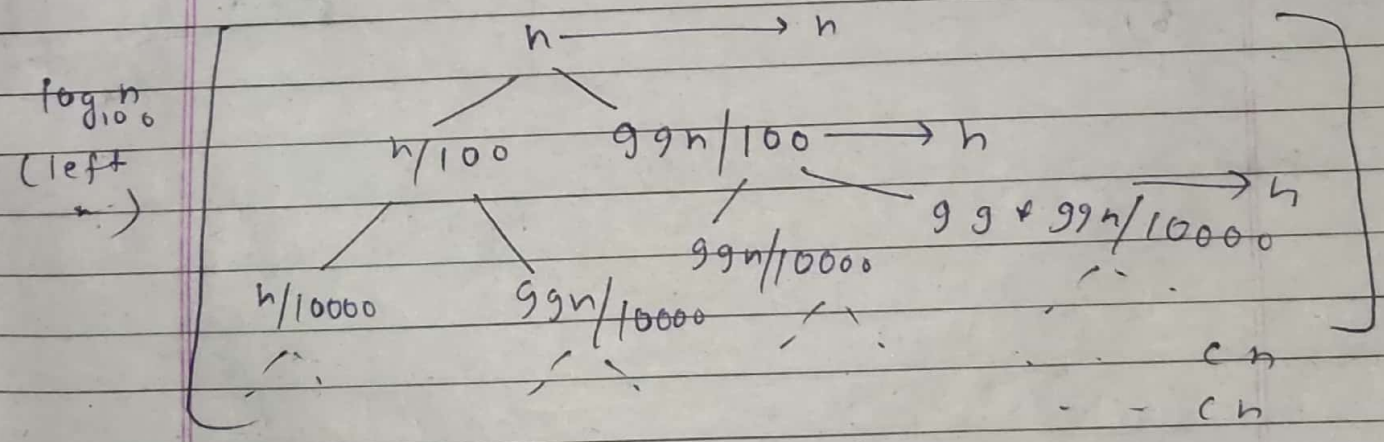
$$T(n) = \left(\frac{cn + cn + \cdots \text{(log}_{\frac{100}{99}} n + 1 \text{ times)} + 1 < (n)}{}\right) \leq$$

$$\left(\log_{\frac{100}{99}} n + 1\right) * cn$$

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

$$\left(\log_{\frac{100}{99}} n = \frac{\log_2 n}{\log_2 \frac{100}{99}}\right)$$ Ignoring constant term by $\log_2 \frac{100}{99}$.

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



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

$$100^x = n$$
$$\rightarrow x = \log_{100} n$$

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

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

Ans 8:- a) 9n creating order of rate of growth- $100, \log(\log n), \log x, \sqrt{n}, n, \log(n), n\log n, n^2, 2^n, 4^n, n!_0, 2^{2n}$

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

c) $96 < \log_8 n < \log_2 n < 5n < \log(n!) < n\log_6(n) < n\log_2 n < 8n^2 < 16n^3 < n! < 8^2 n$