TUTORIAL :- 2

Q1 What is the time complexity of below code and How.

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

k-term

Soln: $i = 0, 1, 3, 6, 10, 15, \ldots\ldots + T_k$ — (i)

$S_{k-1} = 1+3+6+10+15+21+\ldots\ldots+T_{k-1}$ — (ii)

$S_{k+1} = 1+3+6+10+15+21\ldots\ldots+T_{k-1}$

Sub (ii) from (i)

$T_k = S_k - S_{k-1} = 1+2+3+4+5+6+\ldots\ldots k$

we have $T_k = n$.

$1+2+3+4+5+\ldots\ldots+k = n$

$$\frac{k(k+1)}{2} = n \Rightarrow k^2+k-2n=0$$

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

Taking only (+ve) value we get total no of times the loop runs for

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

$T(c)$; $T(n) = O\left(\frac{\sqrt{2n+1}^2}{2}\right) = O(n)$

~~Q2 Write Recurrence~~

Q2 Write Recurrence rel n for the recursive fun that prints fibonacci Series solve the recurrence rel n to get complexity of the prog what will be the Space complexity of prog

Soln:

Recursive fun.

```
int fib(int n)
{ if(n<=1) → O(1).
    return n;
  return fib(n-1) + fib(n-2) → T(n-1)+T(n-2)
}
```

Recurrance rel n $T(n) = T(n-1) + T(n-2) + c$

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

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

$T(n-2) = 2*(2T(n-2-2)+c)+c$

$4T(n-2)+3c$

$T(n-4) = 2*(4T(n-2)+3c)+c$

$8T(n-3)+7c$

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

Put $n-k=0$ $n=k$

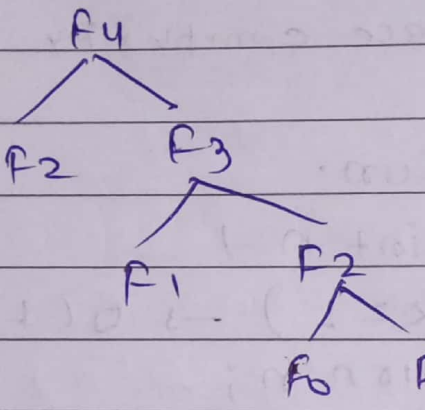
$T(n) = 2^n * T(0) + (2^n-1)c$

$= 2^n * 1 + 2^n c - c$

$= 2^n (1+c) - c = \Theta 2^n$

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

Space Complexity: Space is proportional to the maximum depth of recursion tree.



Hence space complexity of fibonacci recursion is O(N).

Q3 Write a program which have complexity

a. Soln:

1. $n(\log n)$

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

2. $\log n (\log n)$

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

Q4. Solve Recurrence Rel^n

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

$$T(n/4) \cong T(n/2)$$

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

as $a \geq 1$ & $b > 1$

∴ using master's method:

$$T(n) = aT(n/b) + F(n)$$

$$c = \log_b a$$

$$c = \log_2 2 = 1$$

$$F(n) > n^c$$

$$T(n) = O(F(n))$$

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

Q Write a reccurance Rel^n when quick sort repeatedly divide the array in two parts of 99 % and 1 %. derive the Time complexity in this case. Show the recursion tree while deriving time complexity & Find in height of both the extreme parts what understand a—

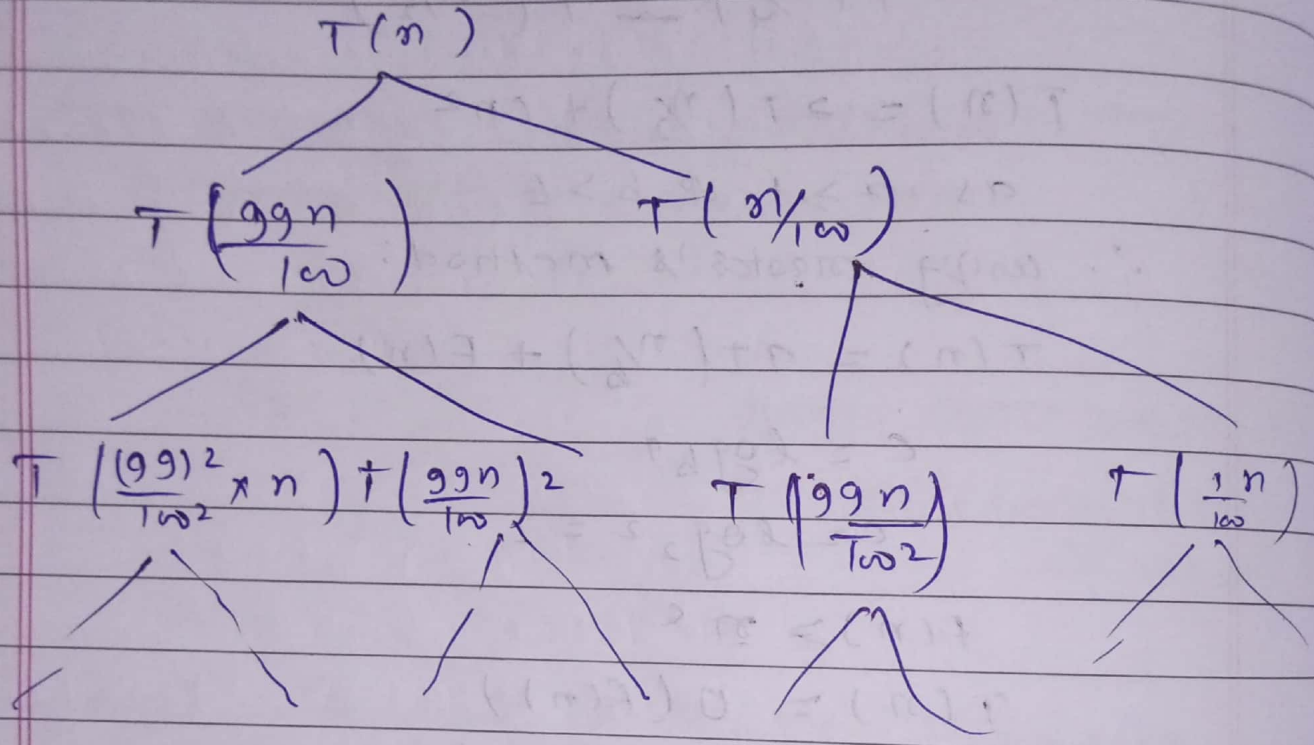
Soln: 99 to 1 in quick sort-

When pivot is where from front or end always.

So

$$T(n) = T(99n/100) + T(n/100) + O(n)$$

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



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

$$\log n = k \log 99/100$$

$$k = \log_n \frac{100}{99}$$

$$Tc = n * \log_{\frac{100}{99}}(n)$$

Q Arrange the following in increasing order of rate of growth.

Soln:

a. $1 < \log\log(n) < \log^2 n < \log n < \log n! < n < n\log n < n^2 < 2^n < 4^n < 2^{(2^n)} < n!$

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