Name - AMARJIT SRIPESH
CST (core) 09

Tutorial -2

(1) 1st time → i = 1
2nd time → i = 1+2
3rd time → i = 1+2+3
For ith time → i = (1+2+3+... i) < n

$\Rightarrow \frac{i(i+1)}{2} < n$

$\Rightarrow i^2 < n$
$i = \sqrt{n}$

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

(2) Recurrence relation for fibonacci series
$F(n) = F(n-1) + F(n-2)$
let it be of time complexity $T(n)$
$T(n) = T(n-1) + T(n-2) + 1$ — (1)

For n=0 & n=1 no addition occurs
$\therefore T(1) = T(0) = 0$

Let $T(n-1) \approx T(n-2)$ — (2)
From (1) & (2)
$T(n) = T(n-1) + T(n-1) + 1$
$= 2T(n-1) + 1$

Using Backward Substitution
$\therefore T(n-1) = 2 \times T(n-2) + 1$
$T(n) = 2 \times [2 \times T(n-2) + 1] + 1 = 4 \times T(n-2) + 3$ — (3)

Now $n = n-2$
$T(n-2) = 2 \times T(n-3) + 1$ — (4)

From (3) & (4)
$T(n) = 8 \times T(n-3) + 7$

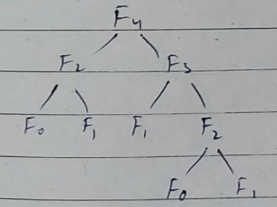
$\therefore T(n) = 2^k \times T(n-k) + (2^k - 1)$ — (5)

For T(0)
$n-k = 0 \Rightarrow k = n$

$T(n) = 2^n \times T(0) + 2^n - 1$
$= 2^n + 2^n - 1$

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

The space is proportional to the maximum dept of the recurrence tree



$\therefore$ space complexity = $O(N)$

③ ~~Merge sort~~

time complexity - O(n log n)

```
#include <iostream>
using namespace std;
void print(int l, int h)
{   for(int i=h-l ; i>=0 ; i--)
    {  cout<<"*" ; }
    cout<<endl;
}
void loop (int l, int h)
{   if (l<h)
    {   int m=(l+h)/2;
        loop(l,m);
        loop(m+1, h);
        print( l, h);
    }
}
int main()
{   int a;
    cin>>a;
    loop(0, a-1);
    return 0;
}
```

time complexity - $O(n^3)$

```
#include <iostream>
using namespace std;
void main()
{   int a;
    cin>>a;
    for(int i=0 ; i<n ; i++)
    for(int j=0 ; j<n ; j++)
    for(int k=0 ; k=n ; k++)
      cout<<"*";
}
```

time complexity - O(log(log n))

function with the O(log(log n))

```
int countPri(int n)
{  if(n<2) return 0;
   bool nonPrime[n];
   nonPrime[1] = 1;
   int nononPrime = 1;
   for(int i=2; i<n; i++)
   {  if(nonPrime[i])
      continue;
      int j=i*2;
      while(j<n)                    // O(log(log n))
      {  if(!nonPrime[j])
         {  nonPrime[j] = 1;
            nononPrime++;
         }
         j=j+i;
      }
   }
   return (n-1) - nononPrime;
}
```

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

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

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

using master's method

$a \geq 1 \quad b > 1 \quad c = \log_b a$

n^c & $f(n)$

$c = \log_2 2 = 1$

$f(n) > n^c$

$T(n) \leq O(f(n))$

$\therefore O(n^2)$

(5) for $i=1$ j runs for n times

$i=2$ j runs for $n/2$ times

$i=3$ j runs for $n/3$ times

$\therefore T(n) = n + n/2 + n/3 + \dots$

$= n(1 + 1/2 + 1/3 + \dots)$

$= n \int_1^n \frac{1}{n}$

$= n \int_1^n \frac{du}{u}$

$= n \log n$

$\therefore$ Time complexity - $O(n \log n)$

(6) For i^{st} iteration it runs for 2^k times

2^{nd} iteration it runs for 2^{k^2} times

3^{rd} iteration it runs for 2^{k^3} times

For n^{th} iteration it run for 2^{k^i} times

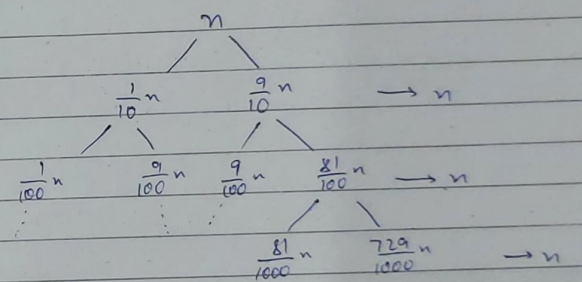
$\therefore \log n = \log 2^{k^i}$

$\log n = k^i$

$\log \log n = i$

$\therefore$ time complexity - $O(\log(\log(n)))$

(7)



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

(8) At 1^{st} level, value = n

At 2^{nd} level, value = $\frac{n}{10} + \frac{9n}{10} = \frac{10n}{10} = n$

value remains same at all levels

Time complexity = $O(n \log n)$

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

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

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