Tutorial 2

① What is the time complexity of below code eg. now.

```
void fun(int n)
{
 int j = 1, i = 0;
while (i < 7) {
 i = i + j;
 j++;
 }
}
```

1st time i = 1

2nd time i = 3 (i = 1 + 2)

3rd time i = 6 i = 1 + 2 + 3.

:

nth time $i = \frac{x(x+1)}{2}$ $= x^2 < n$

$x = sqrt(n)$

② Recurrence relation for recursive function

let T(0) = 1 * fib(n) = fib(n-1) + fib(n-2)

```
fib(n):
  if n <= 1
     return 1.
  return fib(n-1) + fib(n-2)
```

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

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

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

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

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

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

$= 2T(n-3) + 7c$

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

$n - 2K = 0 \Rightarrow \boxed{n = 2K}$ $\boxed{K = \frac{n}{2}}$

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

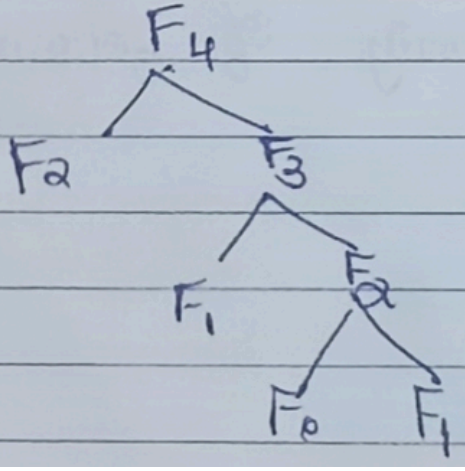
$= 2^n * 1 + 2^n c - c$

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

$\simeq 2^n$ // Constant can be ignored

$O(2^n)$

Space complexity the space is proportional to the maximum depth of the recursive tree.



Hence the Space Complexity of Fibonacci recursive is O(n)

(3) Write program which have complexity - $n\log n$, n^3, $\log(\log n)$

⇒ Merge sort = $n\log n$

For time complexity = n^3.

We can use three nested loops - $O(n^3)$

```
for (int i = 0; i<n; i++)
{
 for (int j = 0; j<n; j++)
 {
  for (int k = 0; k<n; k++)
 {
 // some O(1) expression
 }
}
}.
```

⇒ For time complexity - $\log(\log n)$

We can use the following function.

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

For time complexity - $n\log n$.

We can use the following function.

```
int fun(int n) {
for (i=1 ; i<=n ; i++)
{
 for(j=1 ; j<=n; j+=i
{
// Some O(1) expression
}
}.
```

Solve the following recurrence relation
$T(n) = T(n/4) + T(\frac{n}{2}) + cn^2$.

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

Using master's method $T(n) = aT(\frac{n}{b}) + f(n)$
$a \geq 1, b > 1, c = \log_b a$

Comparing n^c and $f(n)$
we get
$c - \log_2 2 = 1$.
$f(n) > n^c$.
$T(n) = \Theta(f(n))$
$\Rightarrow \Theta(n^2)$

(5) What is the time complexity of the following function:

```
int fun(int n) {
for (int i = 1; i <= n; i++)
{
for (int j = 1; j < n; j += i)
{ // Some O(1) task }}
```

Soln:- for i = 1 ⇒ j = 1, 2, 3, 4 ..... n (sum for n times)
for i = 2 ⇒ j = 1, 3, 5, ... (sum for n/2 times)
for i = 3 ⇒ j = 1, 4, 7 ... (sum for n/3 times)

$$T(n) = n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \dots$$

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

$$\Rightarrow \int_1^n \frac{dx}{x} \Rightarrow [\log x]_1^n$$

$$= n \log n$$

∴ The time complexity of following function is $n \log n$.

(6) What should be time complexity of following function.

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

where k is constant.

Solution: For first iteration $i = 2$

2nd iteration $= i = 2^k$.

3rd iteration $= i = (2^k)^k = 2^{k^2}$.

n^{th} iteration $i = 2^{k^i}$ depends at $2^{k^i} = n$.

apply log $\log n = \log 2^{k^i} \Rightarrow k^i = \log n$.

again apply log $\log(k^i) = \log n \Rightarrow i = \log_2(\log n)$.