

## Tutorial - 2

So<sup>n</sup>, when while loop executes -

At first Pass  $i=1$

2<sup>nd</sup> Pass  $i=1+2$

3<sup>rd</sup> pass  $i=1+2+3$

Similarly 4<sup>th</sup>  $i=1+2+3+4$

..... n<sup>th</sup>  $i=1+2+3+\dots+n$

For i<sup>th</sup> time  $i=(1+2+3+4+\dots+i) < n$

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

$$= \left( \frac{i^2}{2} + \frac{i}{2} \right) < n$$

ignoring  $\frac{i}{2}$  &  $\frac{1}{2}$

After neglecting, we left with

$$= i^2 < n$$

$$= i < \sqrt{n}$$

Hence the time complexity is  $O(\sqrt{n})$

So<sup>n</sup> 2.

int recfib (int n) {

if ( $n \leq 1$ )

return n;

else

return recfib(n-1) + recfib(n-2);

}

Time complexity!

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

when  $n=0 \& n=1$

$$\text{i.e., } T(0) = T(1) = 0$$

For  $T(n)$ ,

$$\text{Here } T(n-2) \approx T(n-1)$$

on substituting the value of  $T(n-1) = T(n-2)$   
into  $T(n)$ ,

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 1 \\ &= 2 * T(n-1) + 1 \end{aligned}$$

on substituting,

$$T(n) = 2 * [2 * T(n-2) + 1] + 1$$

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

$$T(n-2) = 2 * T(n-3) + 1$$

$$T(n) = 2 * [2 * [2 * T(n-3) + 1] + 1] + 1$$

$$T(n) = 8 * T(n-3) + 7$$

1

↓

$$T(n) = 16 * T(n-4) + 15$$

Similarly for  $k^{\text{th}}$  term

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

$$T(n) \propto n-k=0$$

$$n=k$$

$$\begin{aligned} \text{Hence, } T(n) &= 2^n * T(0) + (2^n - 1) \\ &= (2^n + 2^n - 1) \end{aligned}$$

So, time complexity is  $O(2^n)$

Space complexity! -

Here  $n$  are the no. of entries in a stack  
& for each function call one,

So Space complexity for each case (call)  
is 1, i.e.,  $O(1)$

& for  $n$  no. of cases,  $\leq n$   
i.e.  $O(n)$

Sol<sup>n</sup> 3. For (int  $i=0$ ;  $i < n$ ;  $i++$ ) {

    for (int  $j=0$ ;  $j < n$ ;  $j=j+2$ )  
        {

$O(1)$  // statement -

}

}

$O(n \log n)$

For (int  $i=0$ ;  $i < n$ ;  $i++$ ) {

    for (int  $j=0$ ;  $j < n$ ;  $j++$ ) {

        for (int  $k=0$ ;  $k < n$ ;  $k++$ ) {

$O(1)$

        // (Statement)

}

}

$O(n^3)$

```
for(int i=0; i<n; i+=i/2) {
    for(int j=0; j<n; j+=j/2)
```

$\sum_{i=1}^{O(1)} \{ \xrightarrow{\text{Statement}} \}$

$\sum_{i=1}^{O(\log(\log n))} \{ \xrightarrow{\text{Statement}} \}$

4.  $T(n) = T(n/4) + T(n/2) + cn^2$   
on removing  $T(n/4)$  as smaller term

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

On applying Master's theorem on RHS

~~$T(n) < P$~~   $a=0, b=2$   $k=2, P=0,$   
 $\log_b a = \log_2 0 = 0.$

$$0 < 2 \quad i.e \quad \log_b a < k$$

$$P > 0$$

$$O(n^k \log^P n)$$

$$\stackrel{so}{\sim} O(n^2 \log^0 n)$$

$$\underline{O(n^2)}$$

P2

~~Ans~~ time complexity of the function  
fun() is  $O(n \log n)$

for  $i=1$ , inner loop executed  $n$  times.

for  $i=2$ , inner loop executed  $n/2$  times.

for  $i=3$ , inner loop executed  $n/3$  times.

for  $i=n$ , inner loop executed  $n/n$  times.

so, Complexity is like as.

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

which becomes like  $\sum_{i=1}^n \frac{1}{i}$

forms  $\text{H.P.} \rightarrow \left( \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} \right) -$

Particular term time complexity is  $(\log n)$

So for total 3 loops time complexity is  $O(n(\log n))$ .

Ques. for (int i=2; i<n; i=pow(i, k)) {

11 O(1) — expansions.

}

'k' is constant.

it takes the value like  $2, 2^k, 2^{k^2}, 2^{k^3}, \dots, 2^{k \log_{2}(n)}$   
last term must be less than or equal to  $n$ .

$O(\log_k(\log(n)))$

~~so 10  
8(a)~~  $10 \leq n < n! < \log n < \log(n!) < \frac{\log(n!)}{n} < \log(\log n) < n \log n <$   
 $\log^2 n < 2^n < 4^n < 2^{(2^n)} < n^2$

~~8(b)~~  $10 \leq \log n < \log(n!) < \log(\log n) < n < n! < n \log n <$   
 $\log^2 n < 2^n < 4^n < 2^{(2^n)} < n^2$

(b)  $1 < \log(n) < \log(n!) < \log(\log n) <$

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

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