Anshika Semwal
CST 26

Page No.:
Date:

Tutorial-2

Q1- What is the time complexity of below code & how.

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

Time complexity O(sqrt n)

1st time i = 1
2nd time i = 3 i = 1 + 2
3rd time i = 6 i = 1 + 2 + 3
!
nth time $i = \frac{n(n+1)}{2}$

$x^2 < n$

$x = sqrt(n)$

Page No.:
Date: YOUVA

2. Write recurrence relation for recursive function that prints Fibonacci series. Solve the recurrence relation to get complexity of the program. What will be space complexity of program & why?

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)
```

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

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

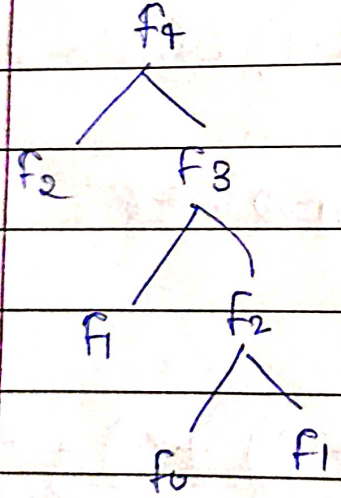
$T(n-4) = 2^2 * (4T(n-4) + 3C) + C$
$= 8T(n-3) + 7C$
$= 2^k \times T(n-k) + (2^k - 1)C$

$n - k = 0$
$\Rightarrow k = n$

$$T(n) = 2^n * T(0) + (2^n - 1) \cdot c$$
$$= 2^n * 1 + 2^n c - c$$
$$= 2^n (1 + c) - c$$
$$\approx 2^n \quad \text{// constants can be ignored}$$

$O(2^n)$

Space complexity :-
The space is proportional to the maximum depth of the recursion tree



Hence the space complexity of fibonacci recursion is in O(N)

Date: YOUVA

3. Write program which have complexity nlogn

Sd Merge sort - nlogn

⇒ 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) expressions
        }
    }
  }
```

⇒ for time complexity - log(logn)
We can use the following function

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

Where k : constant

→ for time complexity nlogn

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 }}
```

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

Using master's method

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

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

comparing n^c & $f(n)$

we get $c = \log_2 2 = 1$

$$f(n) > n^c$$

$$T(n) = \Theta(f(n))$$

$$= \Theta(n^2)$$

Date: YOUVA

(8) for $i=1 \rightarrow j = 1, 2, 3, 4 \ldots n$ (run n times)
for $i=2 \rightarrow j = 1, 3, 5 \ldots$ (run for $n/2$ times)
for $i=3 \rightarrow j = 1, 4, 7 \ldots$ (run for $n/3$ times)

$T(n) = n + n/2 + n/3 + n/4 + \ldots$

$n/1 + 1/2 + 1/3 + 1/4 + \ldots$

$$n\int_1^n \frac{1}{x} \Rightarrow n\int_1^n \frac{dx}{x} = \log x \Big|_1^n$$

$= n \log n$

∴ Time complexity of following function is $n \log n$

(6) for first iteration $i = 2$
2nd iteration $i = 2^{\wedge}k$
$i = (2^k)^k = 2^{k^2}$
⋮
n^{th} iteration $i = 2^{k^l}$ loop ends at
$2^i = n$

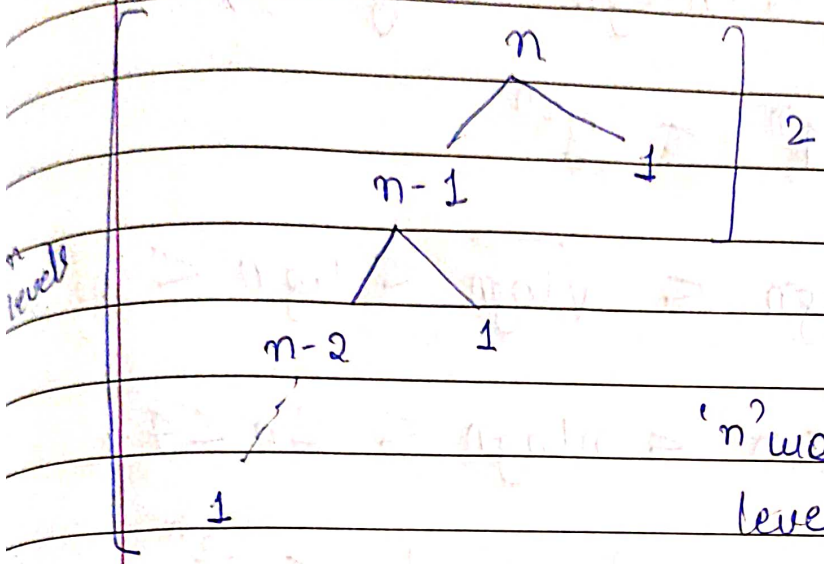
Apply log $\log n = \log 2^{k^l} \Rightarrow k^l = \log n$

again apply log $\log(k^l) = \log n$

$i = \log_* (\log n)$

7) Given algo divides array in 99% & 1% part

$\therefore T(n) = T(n-1) + O(1)$



'n' work is done at each level for merging

$$T(n) = \left(T(n-1) + T(n-2) + \dots T(1) + O(1) \right) \times n$$

$$= n \times n$$

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

lowest height = 2
highest " = n

$\therefore$ diff = $n - 2$ $\quad n > 1$.

The given algorithm produces linear result.

8. Considering for large value of n

(a) $100 < \log\log n < \log n < (\log n)^2 <$

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

$< 2^n < 4^n < 2^{2^n}$

(b) $1 < \log\log n < \sqrt{\log n} < \log n < \log$

$< 2\log n < n < n\log n < 2n < 4n$

$< \log(n!) < n^2 < n! < 2^{2^n}$

(c) $96 < \log_8 n < \log 2n < 5n < \log$

$< n\log_2 n < \log(n!) < 8n^2 < 7n^3$

$< n! < 8^{2n}$