DAA

Tutorial 2.

Geetesh Kumar Dimri (Sec: M)
Roll no = 08 Sem - 4 th

(1) what is the time Complexity of below code & why how ?

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

Ans value of i is incremented as follows.

1 , 1+2 , 1+2+3 , 1+2+3+4 , 1+2+3+4 . . . K times

after Kth operation i becomes greater than n & loop terminated.

Hence, K is the number of operations & hence determines the time complexity.

$$1 + 2 + 3 + \cdots + K \geq n$$

$$\frac{K(K+1)}{2} > n$$

$K = O(n)$.

(2) Write recurrence relation for the recursive fn that prints fibonacci series. Solve recurrence

relation to get time complexity of the Program.
what will be the space complexity of Program
& why ?

Ans Recurrence relation for fibonacci series.

$f(n) = f(n-1) + f(n-2)$.

& $f(0) = 0$ and $f(1) = 1$

```
int fib (n)
{
    if (n <= 1)
        return n;          — O(1)

    return fib(n-1)+fib(n-2); → T(n-1) + T(n-2)
}
```

$T(n) = T(n-1) + T(n-2) + O(1)$ → C

for lower bound : approximate $T(n-1) \sim T(n-2)$

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

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

$= 8T(n-6) + 7C$

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

put, $n - 2K = 0 \Rightarrow K = n/2$

$= 2^{n/2} + (2^{n/2} - 1) C$

$= 2^{n/2} (1 + C) - C$

$\therefore T(n) = 2^{\wedge}(n/2)$

for upper bound. approximate $T(n-2) \approx T(n-1)$

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

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

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

put $n-k = 0$

$k = n$.

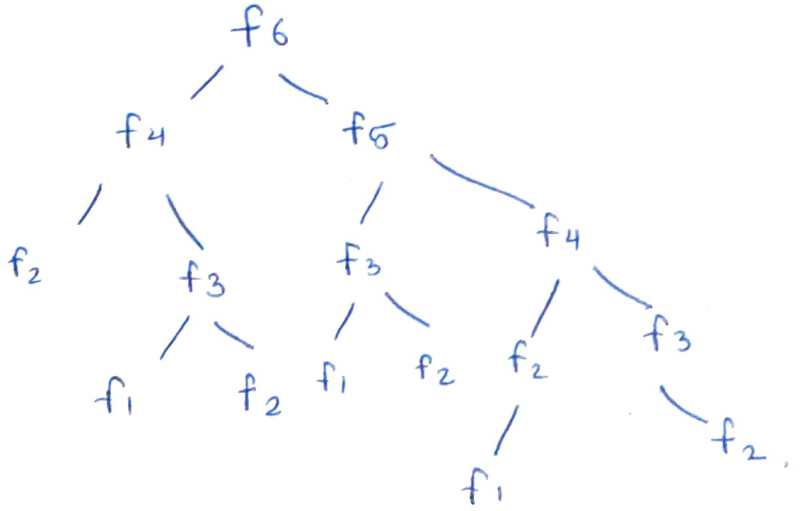
$$T(n) = 2^n + (2^n - 1)c$$

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

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

Now, space complexity.

for f6: Recursion tree is,



from the tree shows that maximum depth is directly proportional to n.

& maximum depth is directly proportional to space required.

Hence, Space complexity is $O(n)$.

(3). Write programs which have complexity $n(\log n)$, n^3, $\log(\log n)$.

↳ for $n \log n$ complexity

```
void func1 (int n){
    for (int i=0; i<n; i++)                    → O(n)
        for (int j=1; j<n; j*=2)               → O(log n)
            // some O(1) expression.
}
```

↳ for n^3 complexity.

```
void func2 (int n)
{
    for (int i=0; i<n; i++)                    → O(n)
        for (int j=0; j<n; j++)                → O(n)
            for (int k=0; k<n; k++)            → O(n)
                // some O(1) expression
}
```

↳ for $\log(\log n)$

```
void func3 (int n)
{
    int i=2;
    while (i<n)
    {   i = pow(i, 2)
    }
}
```

$2^{2^k} = n$

$2^k = \log_2 n$

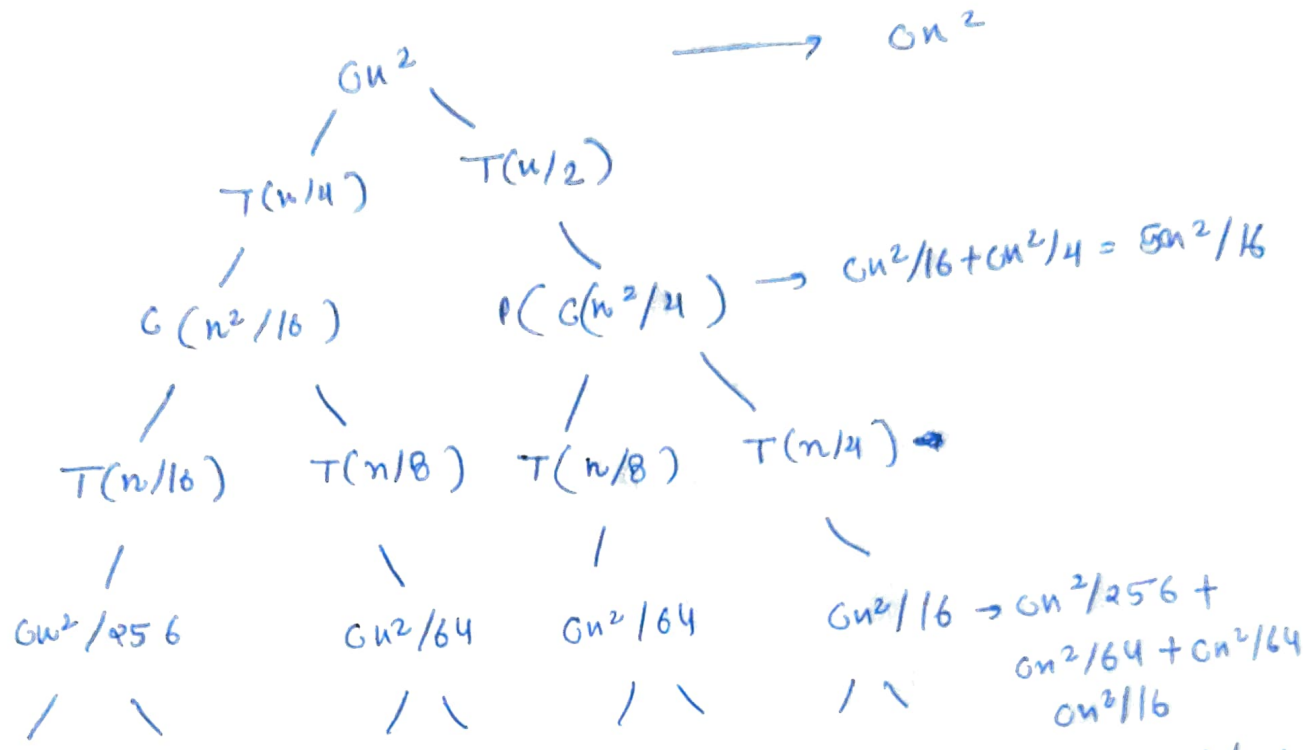
$k = \log(\log_2 n)$

$\therefore T(n) = O(\log \log n)$.

(4). Solve recurrence relation.

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

Ans The initial recursion tree is.



$T(n) = cn^2 (1 + 5/16 + 25/256 \ldots\ldots)$

The ratio for the G.P is $5/16 < 1$

Hence, we get a G.P with infinite sum & is decreasing. ∴ Sum is constant (K)

$T(n) = cn^2 (K)$

$= Kcn^2$

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

(5) what is time complexity of following function fun()?

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

Ans for first outer loop : (i=1)
inner loop works n times.

for second outer loop : (i=2)
inner loop works n/2 times.

for third outer (i=3)
inner loop works n/3 times.

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

$= n\left(1 + 1/2 + 1/3 \cdots + 1/n\right)$

$T(n) = n \log n$

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

(6) what should be time complexity of

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

where, k is a constant.

Ans for first iteration $i = 2$

for second ... $i = 2^K$

for third ... $i = 2^{K^2}$

$\vdots$

$i = 2^{K^C}$

$2^{K^C} = n$

$\log_2 (2^{K^C}) = \log_2 n$

$K^C = \log_2 n$

$C \log_K K = \log_K \log_2 n$

$C = \log_K \log_2 n.$

$\therefore T(n) = O(\log_K \log_2 n).$

(7). Write a recurrence relation when quick sort repeatedly divides the array in to two parts of 99% & 1%. Derive the time Complexity in this case. Show recursion tree while deriving time complexity & find the difference in heights of both the extreme parts. what do you understand by this analysis?

Ans In general, for quick sort's time Complexity is.

$T(n) = T(K) + T(n-K-1) + n.$

k is number of elements smaller than pivot

Now, when Quicksort divides array into 99% & 1%.

$$T(n) = T(0) + T(n-1) + \Theta(n)$$

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

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

$$= T(n-K) + (n-K+1) + \cdots n-1+n.$$

Put $n-K = 0 \Rightarrow K = n.$

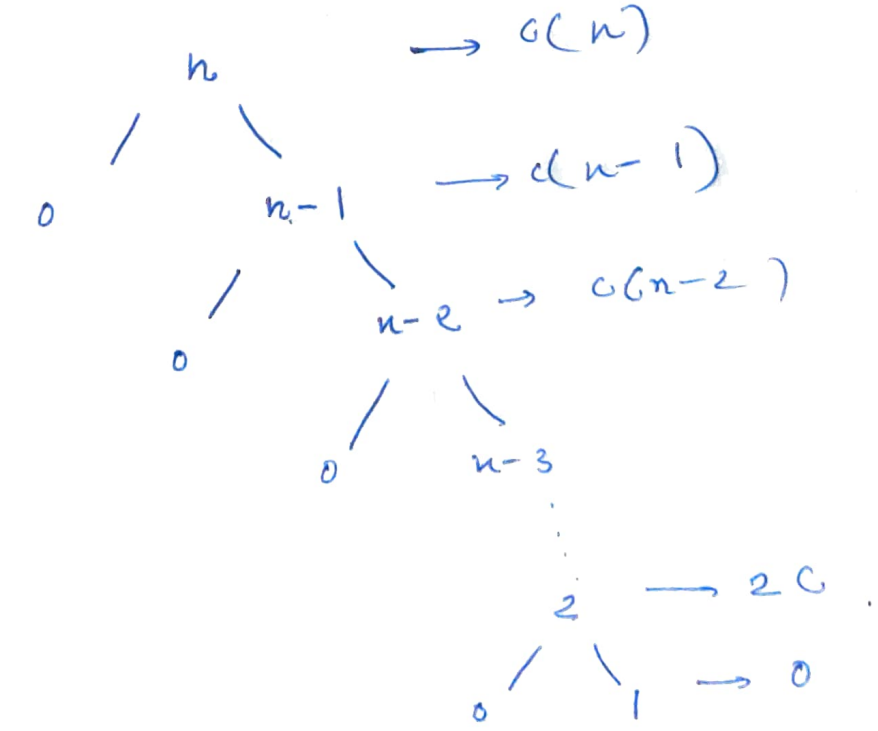
$$= T(0) + T(1) + 2 + \cdots n-2 + n.$$

$$= 1+2+\cdots n-2 + n-1 + n$$

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

$T(n) = O(n^2)$ for worst case.

Tree:



$$c\left(\frac{n(n+1)}{2} - 1\right) = O(n^2).$$

∴ difference in heights of both extreme parts is proportional to number of elements in this case.

from above analysis, we found that quicksort performs worst when array is sorted or when pivot divides in 0 & n-1 parts. It can be eliminated using Randomized Quick sort which will have better time complexity than $O(n^2)$.

3) Arrange following in increasing order of rate of growth.

a) n, $n!$, $\log n$, $\log\log n$, $\text{sqrt}(n)$, $\log(n!)$, $n\log n$, $\log^2(n)$, 2^n, 2^{2^n}, 4^n, n^2, 100.

$O(100) < O(\log^* \log n) < O(\log n) < O(\log^2 n) < O(n) < O(\log(n!)) < O(n\log n) < O(n^2) < O(2^n) < O(4^n) < O(n!) < O(2^{2^n})$.

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

$O(1) < O(\log\log(n)) < O(\sqrt{\log(n)}) < O(\log(n)) \leq O(2\log(n)) = O(\log(2n)) < O(2n) = O(n) = O(4n) < \log(n!)$

$< O(n \log n) < O(n^2) < O(2^{n+1}) < O(n!)$.

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

$O(96) < O(\log_8 n) < O(\log_2 n) < O(5n) < O(\log(n!))$

$< O(n \log_8 n) < O(n \log_2 n) < O(8n^2) < O(7n^3)$

$< O(8^{2n}) < O(n!)$.