DIVYANSH SHARMA
CST CORE SEM IV
ROLL NO - 46

DESIGN & ANALYSIS OF ALGORITHM

TUTORIAL - 2

Q1) what is the time complexity of the code & how ?

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

Ans. On the execution of while loop -

1st iteration, $i = 1$
2nd " $i = 1+2$
3rd " $i = 1+2+3$
4th " $i = 1+2+3+4$

∴ for i times, $i = (1+2+3+4 \ldots\ldots + i)$

this makes the series where sum $\Rightarrow i = \frac{i(i+1)}{2}$

now $i < n$ (for complexity to exist upper bound)

$\Rightarrow \frac{i^2+i}{2} < n$

$\Rightarrow i^2 < n$ (removing lower order)

$\rightarrow i = \sqrt{n} \Rightarrow$ **complexity = $O(\sqrt{n})$** ⟹ ans.

Q2) write recurrence relation for recursive fibonacci series function. solve the recurrence relation to get time complexity. what will be space complexity and why.

Ans) recursive f^n can be written as -

```
fibo(int n)
{ if n==c=1; —①
  return n;
 else
  return fibo(n-1) + fibo(n-2); — ②
```

from ① & ②

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

now $-T(n-2) \leq T(n-1)$

$\therefore T(n) = 2T(n-1) + n$ —③

① formula

for eq (iii), on substituting $n = \frac{n}{2}$ & so on we can obtain -

$$T(n) = 4T(n-2) + 3 \quad \text{— (iv)}$$

$$\Rightarrow T(n) = 8T(n-3) + 7 \quad \text{— (v)}.$$

rewriting eqn (v) - $T(n) = 2^k T(n-k) + (2^k - 1)$

$$n - k = 0 \rightarrow k = n$$

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

$$= 2^n - 1 \quad \text{(discarding } T(0) \text{ since its of lower order)}$$

$$\boxed{T(n) = O(2^n)} \Rightarrow \text{ans.}$$

Space complexity: space complexity of recursive functions is given by - (i) no of stack frames x memory per stack frame & (ii) the order of max depth of the binary tree for the function

recursive

for a n fibo() having n calls, the space complexity thus is equal to $\boxed{O(n)}$ ⇒ ans.

Q3) Write programs which have complexity

(i) $n(\log n)$ (ii) n^3 (iii) $\log(\log n)$

Ans. (i) $n \log n$: such algorithms implement $\log n$ n times - which occurs in Divide & conquer algorithms.

eg: Merge sort, Heap sort, Quick sort

(ii) n^3: Cubic complexity which is obtained during execution of triple nested loops.

eg: cubic / three variable equation problem programs.

(2) Ans.

(iii) $\log(\log n)$: eg - interpolation search.

The algorithm of interpolation search is an upgraded & more efficient of the binary search algorithms. It assumes that the values of the sorted array are uniformly distributed thus $(\log n)$ dataset division is further carried out log times
∴ increasing efficiency & decreasing complexity.

Q4). Solve: $T(n) = T(n/4) + T(n/2) + cn^2$

Ans). We can assume that $T(n/4) \leq T(n/2)$

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

⇒ applying MASTERS METHOD

$a = 2, b = 2,$

$c = \log_2^2 = 1$

$n^c = n^1 = n$

comparing n with f(n)

$\because n < n^2$

Complexity = $O(n^2)$. —ans.

Q5) 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) —①
  { //some O(1) task
}}}
```

Ans:

for i=1

j (inner loop) = 1+2+3+4 ... n times

for i=2

j = 1 ... 3 ... 5 ... 7 ..... $\frac{n}{2}$ times

for i=3

j = 1 ... 4 ... 7 ... 10 ... $\frac{n}{3}$ times

③ Ans.

∴ for $i = n$

$f = n$ times

∴ summation: $\Sigma = \{n + \frac{n}{2} + \frac{n}{3} + \dots n\}$.

$= n\{1 + \frac{1}{2} + \frac{1}{3} + \dots \frac{1}{n}\}$.

$= n(\log n)$ { series sum of $\log n$ }.

$\Rightarrow$ $\boxed{O(n) = O(n \log n)}$ ⟶ Ans.

Q6) what should be the time complexity of:

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

where k is a constant

Ans) - for first iteration of the loop: $i = 2^1$

" second " " " " " : $i = 2^k$

" third " " " " : $i = (2^k)^k = 2^{k^2}$

similarly, for n^{th} iteration : $i = 2^{k^i}$

ATQ: loop ends when $2^{k^i} = n$

on applying log on both sides:

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

$\Rightarrow k^i = \log n$

applying log again: $i \log k = \log n$

$\therefore i = \log_k (\log n)$

$\boxed{T(n) = O(\log_k \log n)}$ = Ans.

④

Q8). Arrange the fall in increasing order of rate of growth:

a) $n, n!, \log n, \log(\log n)$, root(n), $\log(n!)$, $n \log n$, $\log^2(n)$, 2^n, $2^{(2^n)}$ 4^n, n^2, 100

Ans). correct increasing order:

~~100 < log(logn) < log(n) <.~~

~~100 < log(log n) <~~

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

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

Ans). Correct increasing order:

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

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

Ans) correct increasing order:

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

⑤ Ans.