Ayush Agarwal
Sec I
Roll No 30

Tutorial 2

Q1 what is the Time complexity of below code & how

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

$$i = \overbrace{0, 1, 3, 6, 10, 15, 21 \dots n}^{k\text{th terms}}$$

let the sum of above k terms $= S_k$.

$S_k = 1+3+6+10+15+21 \dots T_{k-1}$ — (1)

$S_{k-1} = 1+3+6+10+15+21 \dots T_{k-1}$ — (2)

Subtracting (2) from (1).

$T_k = S_k - S_{k-1} = 1+2+3+4+5+6 \dots + k$

we have $T_k = n$

$1+2+3+4+5 \dots + k = n$

$\frac{k(k+1)}{2} = n \Rightarrow k^2 + k - 2n = 0$

$k = \frac{-1 \pm \sqrt{8n+1}}{2}$

taking only +ve value we get total no of times

The loop runs $i = k+1 = \frac{\sqrt{8n+1}}{2}$

Time complexity $T(n) = O\left(\frac{\sqrt{8n+1}}{2}\right) = O(\sqrt{n})$.

Ques2 write Recursive relation for recursive function that prints fibonacci series. Solve the recurrence relation to get time complexity of the prg. what will be space comp- of this & why?

Recursive function

```
int fib (int n)
{ if (n<=1)          — O(1) = c
    return n;
  return fibo(n-1) + f(n-2)   → T(n-1) + T(n-2)
}
```

Reccurence Relation

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

$T(n-1) \simeq T(n-2)$

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

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

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

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

$= 8T(n-3) + 7c.$

Generalising

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

Put $n-k=0$

$n=k$

Put $n=k$

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

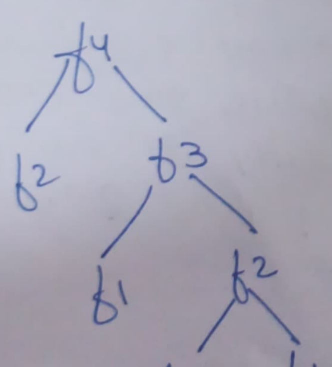
$2^n * 1 + 2^n c - c$

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

2^n

Time complexity $= O(2^n)$

Space complexity space is proportional to max depth of recursion tree.



Hence space complexity of fibbo recursion is $O(N)$

Ques 3 ~~write~~ prog which have complexities

(1) $(n \log n)$

```
for (i=1; i<=n; i++)
{
for (j=1; j<=n; j=j*2)
{
 sum=sum+j;
} }
```

(2) n^3

```
for (i=0; i<n; i++)
{ for (j=0; j<n; j++)
 { for (k=0; k<n; k++)
  { sum= sum+k;
}}}
```

(iii) $\log(\log n)$

```
for (i=1; i<n; i=i*2)
{
for (j=1; j<n; j=k*2)
{ sum=sum+j;
} }
```

Ques 4. Solve the Recurrence Relation $T(n) = T(n/4) + T(n/2) + cn^2$.

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

$T(n/4) \simeq T(n/2)$

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

as $a \geq 1$ and $b > 1$

using Master's Method

$T(n) = aT(\frac{n}{b}) + f(n)$

$c = \log_b a$

$c = \log_2 2 = 1$

$f(n) > n^c$

$T(n) = O(f(n))$

$= O(n^2)$

Ques 5 what is time complexity of follow func.

```
int func (int n)
{ for (int i = 1; i<=n; i++)
      for (int j = 1; j<n; j+=i)
         { O(1) }}}
```

for $i = 1$, j is $1, 2, 3, 4$ ——————.

for $j = 2$ j is $1, 3, 5,$ ------ upto $n/2$.

for $i = 3$, j is $1, 4, 7$ —— ——. upto $n/3$

$T(n) = n + n/2 + n/3$

$n(1 + \frac{1}{2} + \frac{1}{3} ---)$

$n\int_1^n dx/x = (\log x)_1^n$

Time complexity $\underline{n \log n}$

(6) what is Time complexity of

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

where k is const.

for 1st iteration $i = 2$

2nd " $i = 2k$

3rd " $i = (2k)^k = 2k^2$

n^{th} " $i = 2k^i$ loop ends at $2^i = n$

$\log n = \log_2 k^i$

$k^i = \log n$

$\underline{i = \log_e(\log n)}$

Ques 7 Recurrence relation when Quick sort repeatedly divides array into 2 parts of 99% & 1%. Derive the time complexity in this case. Show the recursion tree while deriving time complexity & find diff in heights of both the extreme parts. What do you understand by this analysis.

99 to 1 in quick sort
when pivot is chosen from front or end always.

so
$T(n) = T(99n/100) + T(n/100) + O(n)$
$T(n) = T(99n/100) + T(n/100) + O(n)$

$T(n)$

$T\left(\frac{99n}{100}\right)$ $\quad T(n/100)$

$T\left(\frac{99n}{100^2}\right)$ $\quad T\left(\frac{n}{100^2}\right)$

$T\left(\frac{(99)^2}{100^2} n\right)$ $\quad T\left(\frac{99n}{100^2}\right)$

$n = \left(\frac{99}{100}\right)^k$

$\log n = k \log(99/100)$

$TC = n * \log_{\frac{100}{99}}(n)$.

Ques8 Arrange foll. in order of inc order of rate of growth.

a) $100 < \log\log(n) < \log^2 n < \log n < \log n! < n < n\log n < n^2 < 2^n < 4^n < 2^n 2^n < n!$

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