TUTORIAL 2

Q1] What is the time complexity of below code & why?

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

Ans

j	i	
1	0	→ given
2	1	1
3	3	1+2
4	6	1+2+3
5	10	1+2+3+4
⋮	⋮	

Loop will run i<n times

So $(1+2+3+\_\_\_\_+i) < n$

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

$$i^2 < n$$

$$i = \sqrt{n}$$

Time complexity = $O(\sqrt{n})$

Q2] Write recurrance relation for the recursive function that prints Fibonacci series. Solve the recurrance relation to get time complexity of the program. what will be the space complexity of this program and why?

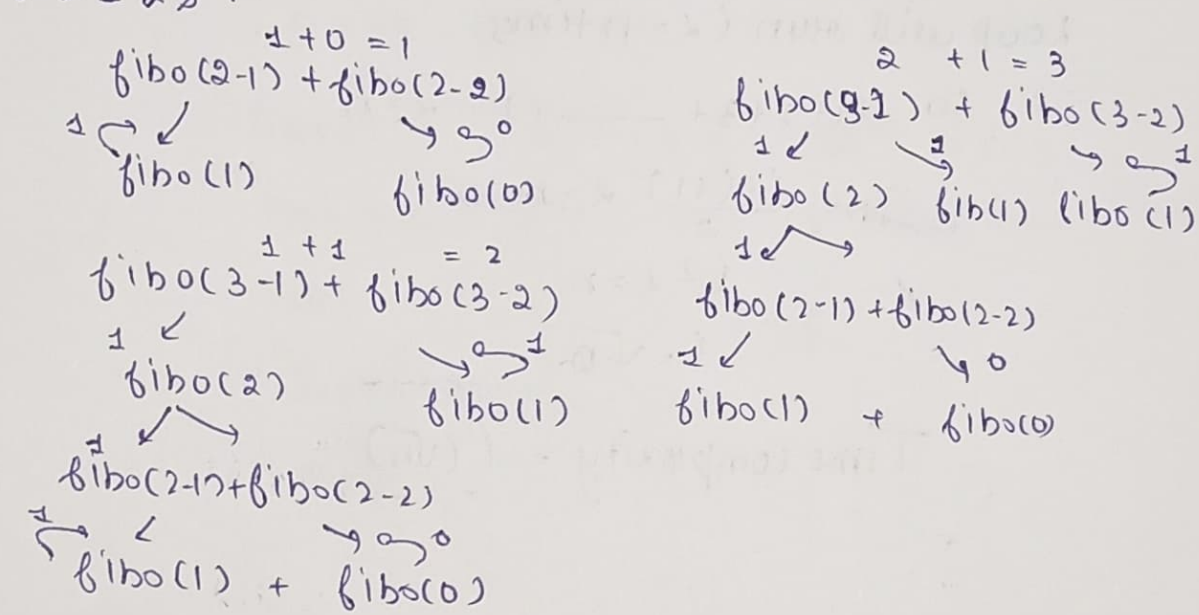
Ans
```
int fibo(int n)
{
    if (n<=1)
        return n;
    else
        return (fibo(n-1) + fibo(n-2));
}

int main()
{
    int n;
    cin>>n;
    for(int i=0; i<n; i++)
        cout << fibo(i);
}
```

let n = 5

i = 0 1 2 3 4

O/P = 0 1 1 2 3



Time complexity

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

Let $T(n-1) \approx T(n-2)$

$T(n) = 2T(n-2) + c$ — (1)

$T(n-2) = 2T(n-4) + c$ — (2)

Put eq (2) in eq (1)

$T(n) = 2[2T(n-4) + c] + c$

$T(n) = 2 \cdot 2\, T(n-4) + 2c + c$ — (3)

$T(n-4) = 2T(n-6) + c$ — (4)

Put eq (4) in eq (3)

$T(n) = 2 \cdot 2 \cdot [2\, T(n-6) + c] + 2c + c$

$T(n) = 2 \cdot 2 \cdot 2 \cdot T(n-6) + 2 \cdot 2c + 2c + c$ — (5)

On comparing eq (1), (3) & (5)

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

$n - 2k = 0$

$n = 2k \Rightarrow k = n/2$

$T(n) = 2^{n/2} \cdot T(n - 2 \cdot \frac{n}{2}) + (2^{n/2} - 1)c$

$T(n) = 2^{n/2} \cdot T(n-n) + (2^{n/2} - 1)c$

$T(n) = 2^{n/2} * 1 + 2^{n/2} c - c$

$T(n) = 2^{n/2}[1 + c] - c$

$T(n) = 2^{n/2}$

$T(n) = T(n-1) + T(n-2) + c$ —

Let $T(n-2) \approx T(n-1)$

$T(n) = 2T(n-1) + c$ — (1)

$T(n-1) = 2T(n-2) + c$ — (2)

Put eq (2) in eq (1)

$T(n) = 2[2 \cdot T(n-2) + c] + c$

$T(n) = 2 \cdot 2 \cdot T(n-2) + 2c + c$ — (3)

$T(n-2) = 2T(n-3) + c$ — (4)

Put eq (4) in eq (3)

$T(n) = 2 \cdot 2 \cdot [2 \cdot T(n-3) + c] + 2c + c$

$T(n) = 2 \cdot 2 \cdot 2 \cdot T(n-3) + 2 \cdot 2 \cdot c + 2c + c$ — (5)

On comparing eq 1, 3, 5

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

$n - k = 0$

$n = k$

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

$= 2^n + 2^n c - c$

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

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

Q2] Write programs which have complexity
$n \log n$, n^3, $\log(\log n)$

Ans $n \log n$

Merge sort complexity is $n \log n$ so there is the merge sort ↓

```
void mergesort (int arr[], int s, int e)
{
    if (s > e)
        return;

    int mid = (s+e)/2;
    mergesort (arr, mid+1, e);
    mergesort (arr, s, mid);
    merge (arr, s, e);
}

void merge (int arr[], int s, int e)
{
    int mid;
    mid = (s+e)/2;
    int leng1 = mid-s+1;
    int leng2 = e-mid;

    int * first = new int [leng1];
    int * second = new int [leng2];

    int k = s;
```

```
for(int i=0; i<leng1 ; i++)
    first[i]= arr[k++];

for(int i=0; i<leng2 ; i++)
    second[i] = arr[k++];

int index1 = 0
int index2 = 0
 k = s;
while (index1 < leng1 && index2 < leng2)
 {
    if (first[index1] < second[index2])
      arr[k++] = first[index1++];
    else
      arr[k++] = second[index2++];
 }

while (index1 < leng1)
 {
   arr[k++] = first[index1++]
 }
while (index2 < leng2)
 {
   arr[k++] = second[index2++];
  }
delete [] first;
delete [] second;
}
```

Q3] Consider the recurrence relation

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

Ans $T(n/2) > T(n/4)$

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

Applying master's theorem

$$T(n) <= 2T(n/2) + cn^2 \quad \text{if } a=2, b=2, c = \log_b a \Rightarrow c = 1$$

$$f(n) = cn^2$$

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

Time complexity $T(n) = \Theta(n^2)$

Q-1 What is the time complexity of following function fun()?

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

Ans For $i=1$ $j = 1, 2, 3, \ldots n$ runs for n times

For $i=2$ $j = 1, 3, 5, \ldots n$ n/2 times

For $i=3$ $j = 1, 4, 7, \ldots$ n/3 times.

$$T(n) = n + n/2 + n/3 + \ldots 1$$

$$= n\left[1 + \frac{1}{2} + \frac{1}{3} + \ldots 1\right]$$

$$\int_1^n \frac{1}{x}\,dx$$

$$n[\log x]_1'$$

$$n \log n$$

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

Ans] For first time / iteration $i = 2$

Second time / iteration $i = 2^k$

third time $i = (2^k)^k = 2^{k^2}$

!

i^{th} time $i = 2^{k^i}$ loop ends when $2^{k^i} = n$

taking log both sides

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

$$k^i = \log n$$

Again taking log both sides

$$i \log k = \log(\log n)$$

$$i = \log(\log n) \quad \{\text{as } k \text{ is constant}\}$$

$$T(n) = \log(\log n)$$ Ans

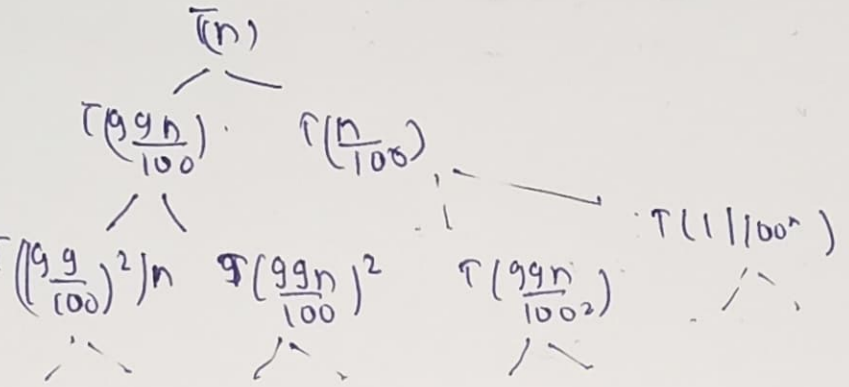
Q7) Write a recurrence relation when quick sort repeadetly divides the array into 2 parts of 99:1 & 1:1. Derive the time complexity in this case. Show the recursion tree while deriving time complexity & find the different in heights of both the extreme parts. what do you understand by analysis?

Ans 99 to 1 in Quick sort

~~when pivot~~ is at

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

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



$$\frac{n}{(99/100)^K} = 1$$

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

$$\log n = K \log \frac{99}{100}$$

$$K = \frac{\log n}{\log\left(\frac{99}{100}\right)} \Rightarrow TC = n * \log_{\frac{100}{99}}(n)$$

2.

Q8| Arrange the following in increasing order of rate of growth:

(a) Ans

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

(b) Ans

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

(c) Ans $96 < \log n < \log 2n < \sqrt{n} < n \log n < n(\log n) < \log(n!)$
$< 8n^2 < 4n^2 < n! < 8^{2n}.$