Aman Dwivedi

CST , 07

classmate
Date ______
Page ______

Tutorial - 2

1. What is the time complexity of blow code and how?

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

→ 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, $i=(1+2+3+4+\dots+i)$

sum ⇒ $i = \frac{i(i+1)}{2}$

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

⇒ $\frac{i^2+1}{2} < n$

⇒ $i^2 < n$ (remaining lower order)

⇒ $i = \sqrt{n}$

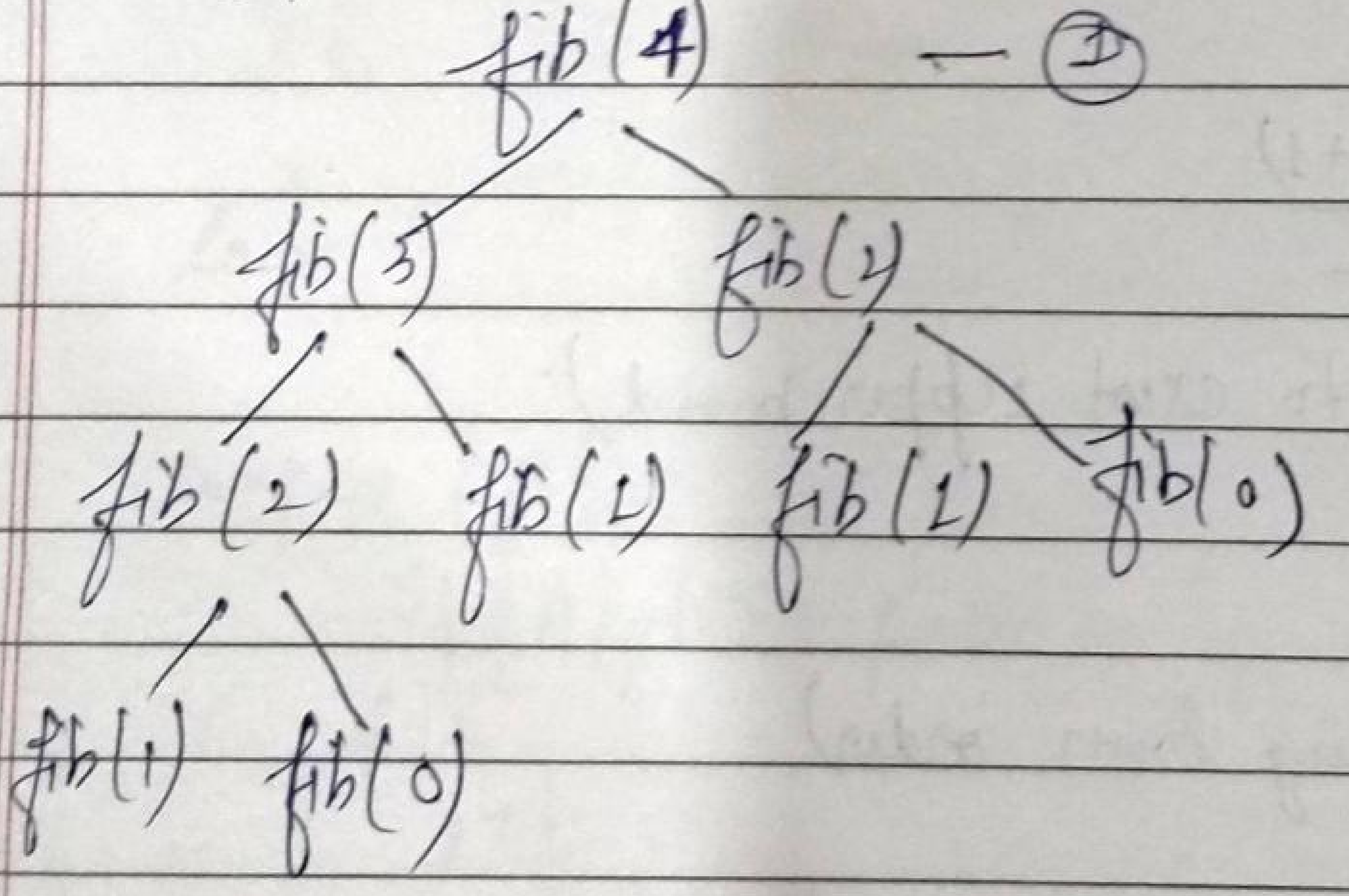
⇒ Complexity = $O(\sqrt{n})$

Scanned by TapScanner

2- Write Recurrence relation for Recursive function that prints fibonacci series. Solve the recurrence relation to get time complexity Of the program. What will be the space complexity of this program and why?

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

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



$T(n) = 1+2+4+8+ \ldots +n.$

$$T(n) = \frac{a(r^n - 1)}{r-1}$$

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

$T(n) = 2.2^n - 1$

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

Space complexity = O(1).

Recursive implementation doesn't store any values from and calculate every value from scratch, so space complexity is O(1).

3- Write program which have complexity
⇒ $n(\log n)$, n^3, $\log(\log n)$

→ Quick Sort :-

```
# define MAX 100
#include <stdio.h>
void quicksort(int[], int, int);
int main()
{
   int n, t=0;
   scanf("%d", &n);
   int A[n];
   for(int i=0; i<n; i++)
    {
      scanf("%d", &A[i]);
    }
   quicksort(A, t, n-1);
   for(int i=0; i<n; i++)
    {
      printf("%d", A[i]);
    }
}
void quicksort(int A[], int lb, int ub)
```

Scanned by TapScanner

```
{
 int i = lb, j = ub, Key = A[lb], t=0;
 if(lb >= ub)
 {
   Return;
 }
 while(i<=j)
 {
  while(Key >= A[i] && i<j)
  {
    i++;
  }
  while(Key < A[j])
  {
    j--;
  }
  if(i<j)
  {
    t = A[i];
   A[i] = A[j];
    A[j] = t;
  }
 }
A[lb] = A[j];
A[j] = Key;
Quicksort(A, lb, j-1);
Quicksort(A, j+1, ub);
}
```

Time Complexity : n(logn).

Scanned by TapScanner

3 variable equation solution

```
#include <stdio.h>
int main()
{
 int A[15];
 int p= 0;
for(int x = 0; x<n; x++)
  {
    for(int j= 0; j<n; j++)
    {
      for(int k=0; k<n; k++)
      {
        if(3*x + 9*j+8*k)
        {
         A[p] = x;
         A[p+1] = j;
         A[p+2] = k;
        }
      }
    }
  }
}
```

Time complexity : $O(n^3)$.

Scanned by TapScanner

when loop variable expands or shrinks:

```
#include <stdio.h>
int main()
{
  int n;
  scanf("%d",&n);
  int k=1;
  for(int i=2; i<=n; i=~~pow~~pow(i,k))
   {
     k++;
   }
}
```

Time complexity : O(log(logn)).

4- Solve the following recurrence ~~rto~~ relation.

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

→ $T(n/4) \leq T(n/2)$

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

=> Applying master's method.

$a = 2, b = 2$

$c = \log_2 2 = 1$

$n^c = n^1 = n$

comparing n with f(n)

$\because n < n^2$

Time complexity : $O(n^2)$

Scanned by TapScanner

5- What is the time complexity of following function:

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

→ for i=1, the inner loop is executed n times

for i=2, the inner loop is executed approximately n/2 times.

for i=3, the inner loop is exeated approximatily n/3 times.

⋮

for i=n, the inner loop is executed approximately n/n times.

So, the total time complexity of above:

alogorithm is (n + n/2 + n/3 + ... + n/n)

which becomes n* (1/1 + 1/2 + 1/3 + ... + 1/n)

The important thing about series is, it equal to O(nlog n).

Scanned by TapScanner

6- What should be the time complixity of:

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

where, k is a constant.

→ Here,

i takes value, $2, 2^{k}, (2^{k})^{k}, ((2^{k})^{k})^{k}$ ——

~~2^{k} log~~ $2^{k \cdot \log k(\log(n))}$

Last term must be equal to or less than n and we have $2^{k \log k(\log(n))} = 2^{\log(n)} = n$ which is equal to last term.

Total iteration = $\log_{k}(\log(n))$ which take contant amount of time to run.

So, time complixity is $O(\log(\log(n)))$.

8- Arrange 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^{2n}, 4^{n}, n^{2}, 100.

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

Scanned by TapScanner

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; $n\log(n)$.

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

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

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

Scanned by TapScanner