Section: CST-SPL-2
Roll No: 02 , University Roll No: 2017547
Name: Sandeep Singh

Tutorial 2

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

1st i = 1
2nd i = 1 + 2
3rd i = 1 + 2 + 3
4th i = 1 + 2 + 3 + 4
for ith i = (1 + 2 + 3 + ... i) < n

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

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

$$= i < \sqrt{n}$$

Time Complexity $= \sqrt{n}$

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

Recurrance Relation: ⇒

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

let $T(n)$ denote time complexity of $F(n)$ in $F(n-1)$ and $F(n-2)$ time will be $T(n-1)$ and $T(n-2)$. We have one more addition to sum our results

for $n > 1$

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

for $n = 0$ and $n = 1$, no addition occurs

$\therefore T(0) = T(1) = 0$

let $T(n-1) \approx T(n-2) \longrightarrow (2)$

Adding (2) in (1)

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

using backward substitution.

$$T(n-1) = 2T(n-2) + 1$$
$$T(n) = 2[2T(n-2)+1] + 1$$
$$= 4T(n-2) + 1 + 2$$

we can substitute

$$T(n-2) = 2T(n-3) + 1$$
$$T(n) = 8T(n-3) + 7$$

General equation of above relation:

$$T(n) = 2^k \times T(n-k) + 2^k - 1 \longrightarrow (3)$$

for T(0)

$n-1 = 0 \Rightarrow k = n$

substituting values in (3)

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

$\Rightarrow 2^n + 2^n - 1$

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

Space Complexity $= O(N)$

Reason: The function calls are executed sequentially. Sequencial execution guarantee that the stack size will never exceed the depth of calls for first F(n-1) it will create N stack

Q3 (i) O(nlogn) :-

```
#include <iostream>
using namespace std;
int partition(int arr[], int s, int e)
{   int pivot = arr[s];
    int count = 0;
  for(int i=s; i<=e; i++)
   { if (arr[i] <= pivot)
        count++;
   }
   int pivat_ind = s+count;
  Swap (arr[pivatind], arr[s]);
```

```
int i=s, j=e;
while(i< pivatind && j > pivat-ind)
{ while( arr[i] <= pivot)
        i++;
  while( arr[j > pivot)
        j--;
  if(i<pivatind && j>pivatind)
  { swap( arr[i++], arr[j--]);
  }
  return pivotind;
  }
Void quick (int arr[], int s, inte)
{ if (s==e)
      return;
  int p = partition( arr, s, e);
   quicksort( arr, s, p-1);
   quicksort( arr, p+1, e);
}
int main()
{  int  arr[] = {6, 8, 5, 2, 1};
    int n = 5;
   quicksort (arr, 0, n-1);
    return 0;
}
```

(ii) $O(N^3)$

```
int main()
{  int n = 10;
for (int i=0; i<n; i++)
{ for(int j=0; j<n; j++)
  { for(int k=0; k<n; k++)
    {   printf(" *");
    }
  }
}
return 0;
}
```

(iii) $O(\log \log n)$?

```
int CountPrimes(int n){
    if(n<2)
        return 0;
    bool * nonprime = new bool [n];
    non-prime [1] : true;
    int numnonprime = 1;
for(int i=2; i<n; i++)
{ if (nonprime[i])
        continue;
    int j = i*2;
    while(j<n) { if (!nonprime[j])
                { nonprime[j] = true;
                  numnonprime ++; }
```

→

```
        j += i;
    }
}
return (n-1) - numnonprime;
}
```

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

using masters theorem

We can assume $T(n/2) >= T(n/4)$

equation can be rewritten as

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

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

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

also $T(n) >= cn^2 \Rightarrow T(n) >= O(n^2)$

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

$\therefore T(n) = O(n^2)$ and $T(n) = \Omega(n^2)$

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

Q:5.

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

for i = 1 inner loop is executed n times
for i = 2 inner loop is executed n/2 times
for i = 3 inner loop is executed n/3 times.

It is forming a series

$$\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \cdots \frac{n}{n}$$

$$n\left[\frac{1}{n} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots \frac{1}{n}\right]$$

$$n \sum_{k=1} \frac{1}{k}$$

$$\Rightarrow n \log n$$

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

⑥ `for(int i=2; i<=n; i=pow(i,k))`
`{`
`// some O(1) expressions`
`}`

with iteration :-

~~i = k~~

for 1st iteration → 2
for 2nd iteration → 2^k
for 3rd iteration → $(2^k)^k$
for n iteration → $2^{k \log k (\log n)}$

∴ last term must be less than or equal to n

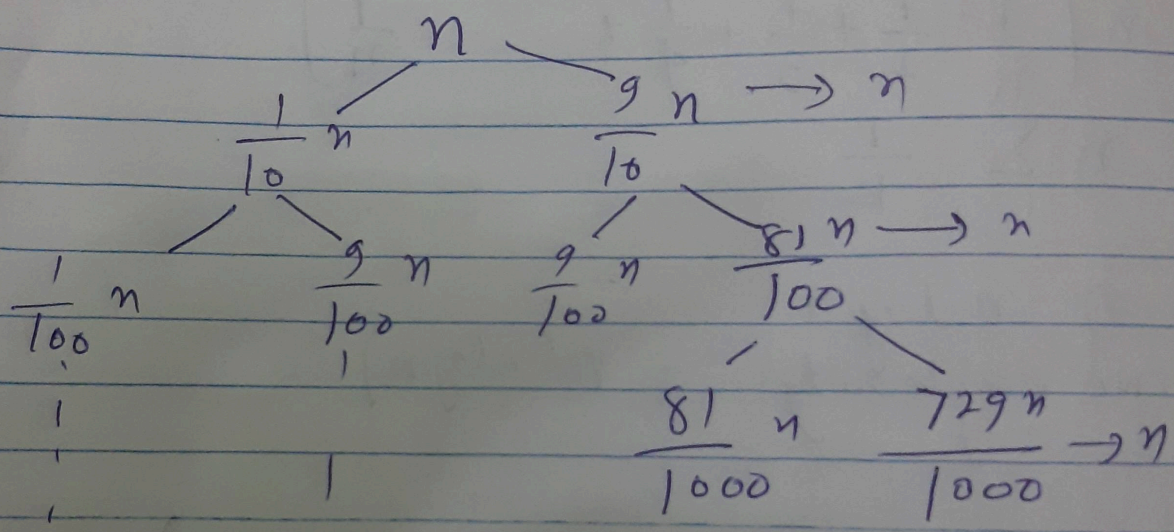
$2^{k \log_{1/k}(\log(n))} = 2^{\log n} = n$

each iteration takes constant time

∴ total iteration $= \log_{1/k}(\log(n))$

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

ANS NO: ⇒ 7



If we split in this manner

Recurrance Relation

$T(n) = T(\frac{9n}{10}) + T(\frac{n}{10}) + O(n)$

when first branch is of size $9n/10$ and second one is $n/10$

showing the above using recursion the approach calculating values.

at ist level, value = n

At 2nd level, value = $\frac{9n}{10} + \frac{n}{10} = n$

Values remains same at all levels

i.e = n

time complexity = summation of values

$= O(n \times \log\log n)$ (Upper bound)

$= \Omega(n \log n)$ lower bound

$\Rightarrow \Theta(n \log n)$ ans

ANS NO: (8)

(a) $100 < \log(\log 1) < \log(n) < \sqrt{n} < n < n\log(n) < \log^2(n) < \log(n!) < n^2 < 2^n < n! < 4^n < 2^{2^n}$

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

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