DESIGN AND ANALYSIS OF ALGORITHM

Name- Priyanshu Ahlawat

Class - CST SPL 1

UNIVERSITY ROLL NO.- 2017527

CLASS ROLL NO.- 12

TUTORIAL 2

Ans 1

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

Values often execution

1st time → i = 1
2nd time → i = 1+2
3rd time → i = 1+2+3
4th time → i = 1+2+3+4
For i^{th} time → $i = (1+2+3+\dots+i) < n$

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

$$\Rightarrow i^2 < n$$

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

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

Ans 2 RECURRENCE RELATION –

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

Let $T(n)$ denote the time complexity of $F(n)$

For $F(n-1)$ and $F(n-2)$ time will be $T(n-1)$ and $T(n-2)$. We have one more addition to sum our results. For $n > 1$

$$T(n) = T(n-1) + T(n-2) + 1 \quad \text{(1)}$$

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

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

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

Putting (2) in (1)

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

Using Backward Substitution

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

$$T(n) = 2 \times [2 \times T(n-2) + 1] + 1 = 4 \times T(n-2) + 3$$

We can substitute $T(n-2) = 2 \times T(n-3) + 1$

$\Rightarrow T(n) = 8 \times T(n-3) + 7$

General equation -

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

For $T(0)$

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

Substituting values in (3).

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

$$\boxed{T(n) = O(2^n)} \quad \underline{\text{Ans}}$$

Space Complexity -> $O(N)$

Reason -

The function calls are execute sequentially. Sequential execution guarantees that the stack size will never exceed the depth of calls. For first $F(n-1)$ it will create N stack frames, the other $F(n-2)$ will create $N/2$. So the largest is N.

Ans 3 O(nlogn):-

```
#include <iostream>
using namespace std;

int partition(int arr[], int start, int end)
{
  int pivot = arr[start];
  int count = 0;

  for(int i= start+1; i<=end ; i++) {
     if(arr[i] <= pivot) count++;
  }

  int pivot_ind = start + count;
  swap ( arr[pivot_ind], arr[start]);

  int i= start , j= end ;

  while (i<pivot_ind && j > pivot_ind) {
   while (arr[i] <= pivot){ i++; }

   while ( arr[j] > pivot ){ j--; }

   if ( i< pivot_ind && j> pivot_ind ) {
       swap (arr[i++], arr[j--]);
     }
  }
   return pivot_ind ;
}
```

```
void quick (int arr[], int start, int end) {
if (start >= end) return;

int p = Partition (arr, start, end);
quicksort (arr, start, p-1);
quicksort (arr, p+1, end);
}

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;

  boolean[] nonprime = new boolean[n];

  nonprime[1] = true;

  int numNonPrimes = 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;
  }
```

Ans 4

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

Using Master's Theorem

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

Equation can be rewritten as

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

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

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

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

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

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

$\boxed{T(n) = O(n^2)}$ Ans

Ans 5

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

Date __/__/____ Page No.:

For $i=2$, 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 -

$$\Rightarrow n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n}$$

$$\Rightarrow n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n}\right)$$

$$\Rightarrow n \times \sum_{k=2}^{n} \frac{1}{k}$$

$$\Rightarrow n < x \log n$$

Time complexity = $O(n \log n)$ Ans

Ans 6

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

With iterations -

i take values
for 1st iteration → 2
for 2nd iteration → 2^k
for 3rd iteration → $(2^k)^k$
:
for n iterations → $2^{k^{\log_k(\log(n))}}$

Date __/__/____

Page No.: _____

∵ last term must be less than or equal to n

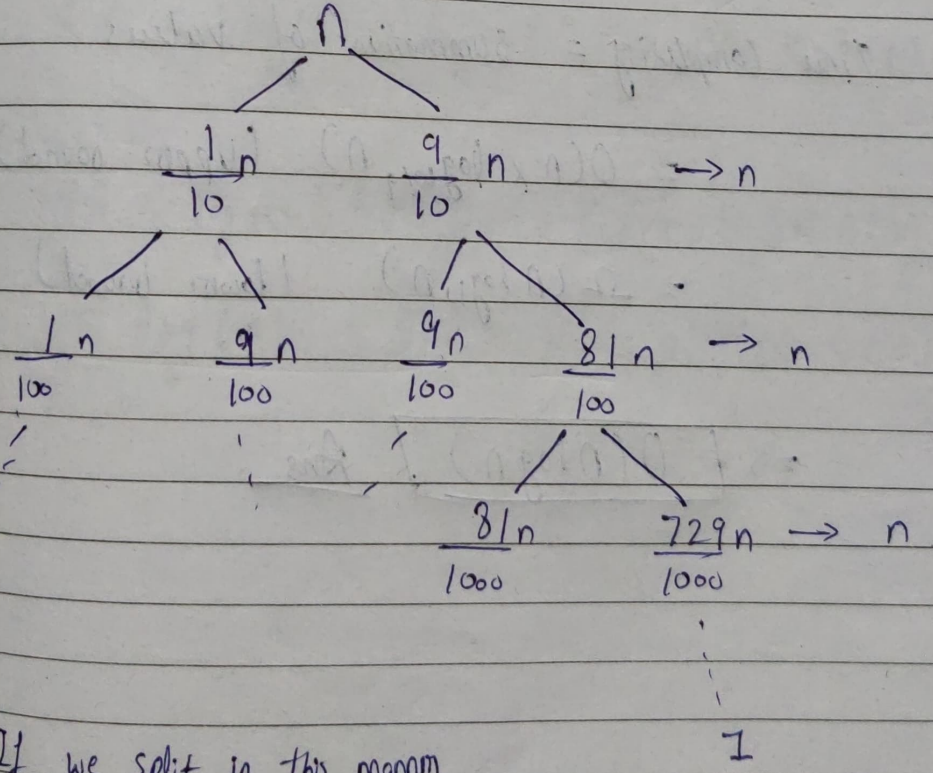
$$2^{k^{\log_k(\log(n))}} = 2^{\log n} = n$$

Each iteration takes constant time.

∴ Total Iteration = $\log_k(\log(n))$

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

Ans 7



If we split in this manner

Recurrence Relation — $T(n) = T(9n/10) + T(n/10) + O(n)$

Date __/__/____ Page No.

where first branch is of size $\frac{9n}{10}$ and second one is $\frac{n}{10}$

Solving the above using recursion tree approach calculating values

At 1st level, Value = n

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

Value remains same at all levels i.e n

Time complexity = Summation of values

$= O(n \times \log_{10/9} n)$ (Upper bound)

$= \Omega(n \log_{10} n)$ (lower bound)

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

Ans 8

(a) $100 < \log(\log n) < \log(n) < \sqrt{n} < n < n\log(n) <$

$\log^2(n) < \log(\underline{n}) < n^2 < 2^n < \underline{n} < 4^n < 2^{2^n}$

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

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

(c) $96 < \log_8(n) < n\log_6(n) < \log_2(n) < n\log_2(n)$

$< \log(n!) < 5n < 8n^2 < 7n^3 < \underline{n} < (8)^{2n}$