Siddhi Shankar
2002517
SPL-2

Date
Page No.

# Tutorial - 2

1)

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

Value after 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+... i) < n

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

$$\Rightarrow i^2 < n$$

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

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

Siddhi Shankar
2002517
DAA-2

Date.
Page No.

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 $n > 1$

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

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

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

Using Backward Substitution

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

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

Siddhi Shankar
2002517
SP2

Date
Page No.

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

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

General Equation -

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

For $T(0)$

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

Substituting values in (3)

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

$= 2^n + 2^n - 1$

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

3 $O(n \log n)$ -

```
#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++;
}
```

Siddhi Shankar
2002517
DPL-2

Date.
Page No.

```
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;
}
```

Siddhi Thankar
2002157
SPL-2

Date
Page No.

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 Clog(Clogn):-

```
int CountPrimes(int n) {
    if (n<2) return 0;

boolean[]    nonprime = new boolean[n];

non prime[1] = true;

    int numNonprimes = 1;

    for (int i=2; i<n; i++) {
        if (nonprime[i]) continue;
```

Siddhi Shankar
2002512
SEC2

Date.
Page No.

```
int j = i*2;
while (j<n) {
  if (!nonprime[j]) {
    nonprime[j] = true;
    numnonprime++;
  }
  j+ = i;
}
}
return (n-1) - numNonPrime;
}
```

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

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

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

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

Sridam Thankaz
2000215172
SPL-2

Date.
Page No.

5 for i = 1 → j = 1, 2, 3, 4, ... n (sum for n times)
for i = 2 → j = 1, 3, 5, ... (sum for n/2 times)
for i = 3 → j = 1, 4, 7, ... (sum for n/3 times)

$T(n) = n + n/2 + n/3 + n/4 + \dots$
$n(1 + 1/2 + 1/3 + 1/4 + \dots)$

$n \sum_{1}^{n} \frac{1}{i} \Rightarrow n \int_{1}^{n} \frac{dx}{x} \Rightarrow [\log x]_1^n$

$= n \log n$

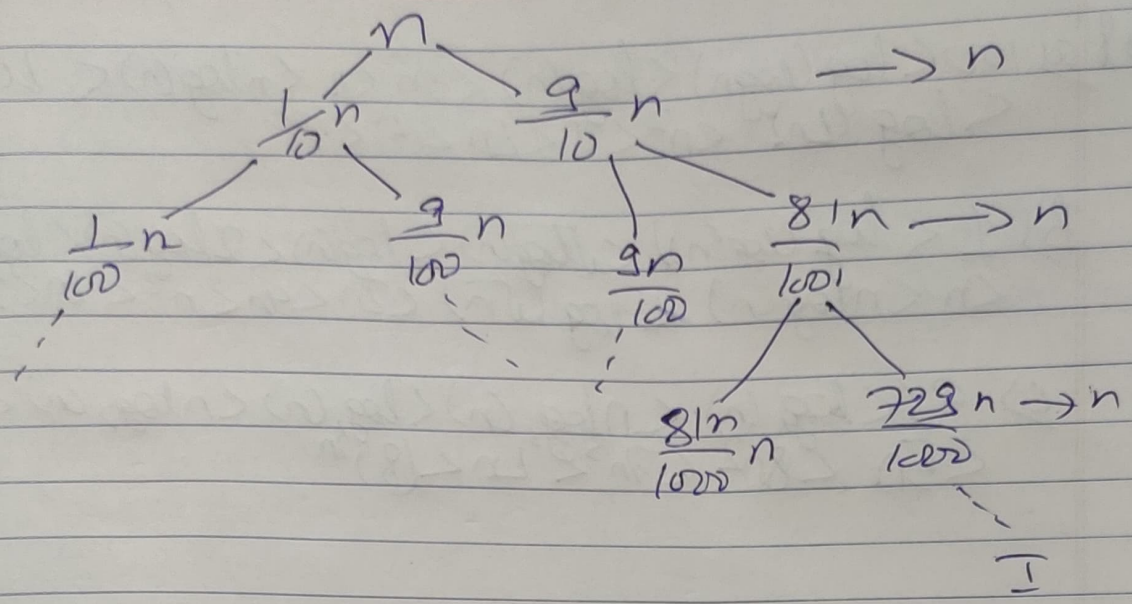
The time complexity function is $n \log n$

6 for first iteration $i = 2$
2nd iteration $i = 2^{k}$
3rd iteration $i = (2^{k})^{k} = 2^{k^2}$
⋮
$n^{th}$ iteration $i = 2^{k^i}$ loop ends at $2^{k^i} = n$
apply log $\log n = \log 2^{k^i} \Rightarrow k^i = \log n$
again apply log $\log(k^i) = \log n \Rightarrow i = \log_k(\log n)$

Siddhi Shankar
2002157
SDL-2

Date.
Page No.

7




If we split in this manner

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

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

Siddhi Thankar
2021517
DPL-2.

Date
Page No.

Ans 8 a) $100 < \log(\log n) < \log(n) < \sqrt{n} < n < n\log(n) < \log^2(n)$
$< \log(n^2) < n^2 < 2^n < 4^n < 2^{2n}$

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

c) $96 < \log_8(n) < n\log_8(n) < \log_4(n) < n\log_2(n) < \log(n!)$
$< 5n < 8n^2 < 7n^3 < 2^{4n} < (8)^{2n}$