Date
Page

Tutorial - 2

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

values 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 → $i = (1+2+3+\cdots+i) < n$

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

$$\Rightarrow i^2 < n$$

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

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

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 the 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 or result.

for $n > 1$

$T(n) = T(n-1) + T(n-2) + 1$ — (1)

Date

Page

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

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

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

Adding (2) in (1)

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

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

Using Backword substitution

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

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

$= 4 T(n-2) + 3$

We can substitute

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

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

General equation

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

for $T(0)$

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

Substituting values in (3)

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

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

Space Complexity → O(N)

Reason -
The function calls are executed 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

Date

Page

3(i)

O(n log n)

```
#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 pivot_ind = s + count;
    swap(arr[pivot_ind], arr[s]);
    int i = s, j = e;
    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 s, int e){
```

Date
Page

```
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, 4};
  int n = 5;
  quickSort(arr, 0, n-1);
  return 0;
}
```

O(log log n)-

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

  bool * nonPrime = new bool[n];
  nonPrime[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
}
```

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

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

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

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

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

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

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

$\therefore T(n) = \Theta(n^2)$

5 int fun(int n)

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

// Same O(1) task
} } }

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

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

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

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

$n \times \log n$

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

6 for (int i = 2; i <= n; i = pow(i, k))
{
// some O(1) expressions
}
with iterations
i take values
for 1st iteration → 2

for 2nd iteration → 2^k

for 3rd iteration → $(2^k)^k$

for n iteration → $2^{k \log(\log(n))}$

∵ last term must be less than or equal to n

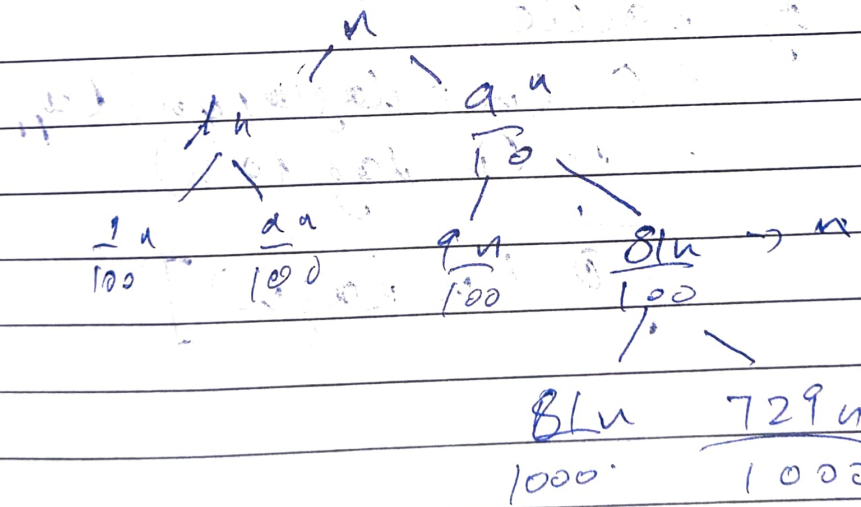
$2^{k \log_k(\log(n))} = 2^{\log n} = n$

Each iteration takes constant time.

∴ Total iteration = $\log_k(\log n)$

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

7.



If we split in this manner

Recurrance Relation

$T(n) = T(an) + T(bn) + O(n)$

When first branch is of size an i.e. $n/10$ & second one is $n/10$

Q6.

Solving the above using recursion tree approach calculating values

At 1st level, value = n

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

Value are same at all levels i.e n

Time complexity = summation of values

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

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

$\boxed{\Theta(n \log n)}$

Q8

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

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

Date

Page

$9\ 6 < \log_8(n) < n\log_6(n) < \log_2(n)$
$< n\log^2 2 \cdot (n) < \lg(n!) < 5^n < 8n^2$
$< 7n^3 < 10n < (8^4)^{2n}$