CSE 221 (ALGORITHMS)

LAB 1

Submitted by Team: ALG0==FREAKS

Team Members:

Tasnia Zarin Shailee

ID: 19101145

sec: 01

Ishraq ahmed Esha

ID:19301261

sec: 01

Primality Testing

1. Naive approach:

```
bool prime [n]= {0};                    // O(1)
void isPrime(int n)                     // O(n^2)
{
        for(int i=2;i<=n;i++){          // O(n^2)
                int cnt = 0;
                for(int j=2;j<i;j++){
                        if(i%j==0)cnt++;
                }
        if(cnt==0)prime[i]=1;
        }
        for(int i=2;i<=n;i++){          // O(n)
                if(prime[i])
                        System.out.print(i+" ");
        }
}
```

Worst time complexity calculation:

$T(n)=O(1)+O(n^2)+O(n)$

$T(n)=O(n^2)$ [Answer]

2. Optimal Sieve

```
void sieveOfEratosthenes(int n)
{
        boolean prime[] = new boolean[n+1];    // O(1)
        for(int i=0;i<n;i++)                   //O(n)
                prime[i] = true;
        for(int p = 2; p<=sqrt(n); p++)        //O(n log logn)
        {
                // If prime[p] is not changed, then it is a prime
                if(prime[p] == true)
                {
                // Update all multiples of p
                for(int i = p*p; i <= n; i += p)
                        prime[i] = false;
                }
        }
        // Print all prime numbers
        for(int i = 2; i <= n; i++)            //O(n)
        {
                if(prime[i] == true)
                        System.out.print(i + " ");
        }
```

Worst time complexity calculation:

T(n)=O(1)+O(n)+O(n log logn)O(n)

T(n)=O(n log logn) [Answer]

Recursion Tree Time Complexity

1. $T(n) = T(n/2)+n-1,\ T(1) = 0$

We know Master Theorem, $T(n)=aT(n/b)+cn^k$

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

Here, $a=1$, $b=2$, $c=1$, $k=1$;

$b^k=2^1=2$; $a=1$;

$b^k > a$

We know that if $b^k > a$,

Then time complexity= $O(n^k)$

So, $O(n^1)=O(n)$

Worst time Complexity= O(n)

2. $T(n) = T(n-1)+n -1,\ T(1) = 0$

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

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

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

=……..

So, $T(n)=1+2+3+\ldots..+(n-3)+(n-2)+(n-1)$

$=n(n+1)/2$

$=n^2+n/2$

$=O(n^2)$

Worst time Complexity= $O(n^2)$

3. $T(n)=T(n/3)+2T(n/3)+n$

$=3T(n/3) +n$

Then, $3T(n/3)= 3^2T (n/3^2)+ 3 (n/3)$

$3^2T(n/3^2)= 3^3T (n/3^3)+3^2 (n/3^2)$

……..….

$3^kT(n/3^k)= 3^{k+1} T(n/3^{k+1})+ 3^k(n/3^k)$

Considering, $n/3^{k+1}=1$

So, $k+1= \log_3 n$

Adding,

$T(n)=3\log_3 n+(n+n+n+\ldots\ldots.+n)$

$T(n)= n+ (n* \log_3 n)$

$=O(n\log_3 n)$

Worst time Complexity=$O(n\log_3 n)$

4. $T(n)=2T(n/2) + n\text{^}2$

We know Master Theorem, $T(n)=aT(n/b)+cn^k$

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

Here, a=2, b=2, c=1, k=2;

$b^k=2^2=4$; a=2;

$b^k> a$

We know that if $b^k > a$,

Then time complexity= $O(n^k)$

So, $O(n^2)$

Worst time complexity= $O(n^2)$

Pseudocode to Coding

```
import java.util.Scanner;
public class Lab1{
 public static void main(String []args){
 Scanner sc= new Scanner(System.in);
int n=sc.nextInt();
int a=n;
int sum=0;
while (n>0){
int r=n%10;
sum=sum+r*r*r;
n=n/10;
 }
if(a==sum){
   System.out.println("Armstrong Number");
 }
 else{
   System.out.println("not an Armstrong Number");
  }
}
}
```