Exercise – Analysis of Algorithms

1. Count the number of operations and give the Big-O characterizations for the following Java code segments:

```
public class IntOps
{

        public static void main(String[] args)
        {
                int a = Integer.parseInt(args[0]);
                int b = Integer.parseInt(args[1]);
                int sum  = a + b;
                int prod = a * b;
                int quot = a / b;
                int rem  = a % b;

                System.out.println(a + " + " + b + " = " + sum);
                System.out.println(a + " * " + b + " = " + prod);
                System.out.println(a + " / " + b + " = " + quot);
                System.out.println(a + " % " + b + " = " + rem);
                System.out.print(a + " = " + quot + " * ");
                System.out.println(b + " + " + rem);

        }
}
```

```
public class Flip
{

        public static void main(String[] args)
        {
                // Math.random() returns a value between 0.0 and 1.0
                // so it is heads or tails 50% of the time
                if (Math.random() < 0.5)
                        System.out.println("Heads");
                else
                        System.out.println("Tails");
        }
}
```

```java
public class NHellos
{
   public static void main(String[] args)
   {
        int N = Integer.parseInt(args[0]);    // assume n >= 4
        // print out special cases whose ordinal doesn't end in "th"
        System.out.println("1st Hello");
        System.out.println("2nd Hello");
        System.out.println("3rd Hello");

        int i = 4;
        while (i <= N)
        {
                System.out.println(i + "th Hello");
                i = i + 1;
        }

   }
}
```

```java
public class Harmonic
{
        public static void main(String[] args)
        {

                // command-line argument
                int N = Integer.parseInt(args[0]);

                // compute 1/1 + 1/2 + 1/3 + ... + 1/N
                double sum = 0.0;
                for (int i = 1; i <= N; i++)
                {
                        sum += 1.0 / i;
                }

                // print out Nth harmonic number
                System.out.println(sum);
        }

}
```

```java
public class DivisorPattern
```

```
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);

        for (int i = 1; i <= N; i++)
        {
            for (int j = 1; j <= N; j++)
            {
                if (i % j == 0 || j % i == 0)
                {
                    System.out.print("* ");
                }
                else
                {
                    System.out.print("  ");
                }
            }
            System.out.println(i);
        }
    }
}
```

```
class BubbleSort
```

```java
{
  public static void main(String []args)
  {
    int n, c, d, swap;
    Scanner in = new Scanner(System.in);

    System.out.println("Input number of integers to sort");
    n = in.nextInt();

    int array[] = new int[n];

    System.out.println("Enter " + n + " integers");

    for (c = 0; c < n; c++)
      array[c] = in.nextInt();

    for (c = 0; c < ( n - 1 ); c++)
    {
      for (d = 0; d < n - c - 1; d++)
      {
        if (array[d] > array[d+1]) /* For descending order use < */
        {
          swap       = array[d];
          array[d]   = array[d+1];
          array[d+1] = swap;
        }
      }
    }

    System.out.println("Sorted list of numbers");

    for (c = 0; c < n; c++)
      System.out.println(array[c]);
  }
}
```

2. Given the following number of operations for some unspecified algorithms, give the Big-Oh notation of their growth rates:

a. $T(n) = n\log(n) + 10n + 1000$

b. $T(n) = 500\,n2 + 2n$

c. $T(n) = 500\log(n) + n3 + 300n + 7$

d. $T(n) = 1{,}000{,}000 + 10\,log\,n + 5n$

e. $T(n) = 15n + 60n^4 + n!$

f. $T(n) = n\sqrt{n} + 100n\log(n) + 10$

g. $T(n) = 50\log(n) + 5\sqrt{n} + 500$