CSCI 2114: Tashfeen's Data Structures

**Homework 1**

**Question 1.** Briefly define the following terms in context of computer hardware.

1) Register
2) Memory
3) Disk

**Question 2.** What is the smallest addressable unit of memory in most modern computers?

**Question 3.** Give the number of bytes in the following memory units as either 2 or 10 raised to an appropriate power. For example, a Kibibyte (KiB) is $2^{10}$ bytes while a Kilobyte (KB) is $10^3$ bytes.

1) Mebibyte (MiB)
2) Megabyte (MB)
3) Gibibyte (GiB)
4) Gigabyte (GB)
5) Tebibyte (TiB)
6) Terabyte (TB)
7) Pebibyte (TiB)
8) Petabyte (TB)

Why is there a need for two byte-prefixed unit systems?

**Question 4.** Java class in listing 1 prints the maximum numerical value each Java primitive type can store. Explain how these are calculated, i. e., their connection to register size and numerical sign.

```java
public class Constants {
  public static void main(String[] args) {
    System.out.printf("byte   %25d%n", Byte.MAX_VALUE);
    System.out.printf("short  %25d%n", Short.MAX_VALUE);
    System.out.printf("char   %25d%n", (int)Character.MAX_VALUE);
    System.out.printf("int    %25d%n", Integer.MAX_VALUE);
    System.out.printf("long   %25d%n", Long.MAX_VALUE);
    System.out.printf("float  %25f%n", Float.MAX_VALUE);
    System.out.printf("double %25f%n", Double.MAX_VALUE);
  }
}
```

Listing 1. Java class to print maximum values of primitive types.

**Question 5.** Write a Java program that prints out one line of text to the console. It can be anything but "Hello World!" What did you print? Store the code in a file called `Anything.java`.

**Question 6.** Write a Java program that populates an array of size $n$ with the first $n$ Fibonacci numbers. The program should print out the array as shown in figure 1. Here $n$ should be the first command line argument. You may do it anyway you like but one and arguably the most elegant way to do it is recursively as shown in the listing 2. What is the name of the implicit call structure that is used in listing 2? Hint: Stack Overflow. Store the code in a file called `Fibonacci.java`.

```java
  public static int fibonacci(int i, int[] arr) {
    if (i < 2)
      return arr[i] = i;
    return arr[i] = fibonacci(i - 1, arr) + fibonacci(i - 2, arr);
  }
```

Listing 2. Java function to generate the Fibonacci sequence.

**Question 7.** Using the Sieve of Eratosthenes, populate a boolean array of size $n$ (Java booleans initialise to false) marking all the indices that are Composite numbers to true. Here $n$ should be the first command line argument. The indices remaining false at the end should be Prime numbers. Store the code in a file called `Eratosthenes.java`.

1) For debugging, have your program print all the prime numbers less than a 100. You should get the following: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.
2) The program should print out at most the five largest prime numbers it computed and the time (seconds) it took to compute all the primes less than $n$. Here is a way to compute seconds taken by a function call `eratosthenes(toSieve)`.

```
double startTime = System.nanoTime();
eratosthenes(toSieve);
double duration = System.nanoTime() - startTime;
duration = duration / Math.pow(10, 9);
```

3) With your program, calculate how long does it take (in seconds) to compute all the 30 bit prime numbers. These are all primes less than $n = 2^{30} = 1073741824$.
4) Can your implementation of the Sieve of Eratosthenes compute all the 32 bit prime numbers? If yes, give the time it takes or if it can not, then why not?

**Question 8.** Read all bytes in the file `half_gaps.bin`. You may use the function in code listing 3.

```
public static byte[] getFileBytes(String path) {
  byte[] bytes = null;
  try {
    bytes = Files.readAllBytes(Paths.get(path));
  } catch (IOException e) {
    e.printStackTrace();
  }
  return bytes;
}
```

Listing 3. Java function to read in a file's (signed) bytes.

The function in code listing 3 reads in signed bytes. While this maybe suitable for some binary arrangements, we want the bytes to be unsigned. One way to achieve this is to just loop and use `Byte.toUnsignedInt(byte x)` as seen in listing 4.

```
  byte[] bytes = getFileBytes("../media/half_gaps.bin");
  int[] gaps = new int[bytes.length];
  for (int i = 0; i < bytes.length; i++)
    gaps[i] = Byte.toUnsignedInt(bytes[i]);
```

Listing 4. Converting Java signed bytes to unsigned longs.

Compute the array of integers' cumulative sum, i. e.,

$$\left\{ x_i \in \mathsf{cumsum}(x) : x_i = \sum_{k=1}^{i} x_k \right\}$$

Now multiply each of the sums with 2 and then add a 3.

$$\left\{ x_i \in \mathsf{cumsum}(x) : y_i = 2x_i + 3 = 2\left( \sum_{k=1}^{i} x_k \right) + 3 \right\}$$

1) Print out the first fifteen and the last five elements of this final array.
2) Time this program (the reading of bytes, the cumulative sum computation and the doubling with adding a three) and print the result in seconds.

3) Do you recognise the printed numbers? What will these be if we further prepended a 2 and a 3 to them?

Store the code in a file called `Primes.java`.

**Question 9.** Accumulate the approximate probability that an integer $2 \leq x \leq 2^{31} - 1$ is prime. You can do this by generating random numbers between 2 and $2^{31} - 1$ within a big enough loop and check if the number is prime (this is known as a primality test) by searching for it in the array of prime numbers we constructed in question 8. The main loop is shown in listing 5. You need to implement the linear search and the binary search and uncomment each, one at a time to report the times in seconds taken by each type of search.

```
    double start = System.nanoTime();
    double favourable = 0;
    for (int i = 1; i <= 1000000; i++) {
      long x = getRandomInt(2, Integer.MAX_VALUE);
      favourable += linearSearch(x, primes) ? 1.0 : 0.0;
      // favourable += binarySearch(x, primes, 0, primes.length - 1) ? 1.0 : 0.0;
      System.out.println(favourable / i);
    }
    double duration = (System.nanoTime() - start) / Math.pow(10, 9);
    System.out.printf("Time taken: %.2f seconds%n", duration);
```

Listing 5. Linear and binary search for a prime number.

Does the printed number converge? Store the code in a file called `Search.java`.

**Question 10.** Break the Affine cipher. Your professor encrypted a plain text file called `plain.txt` using the program given in listing 6. Store the code in a file called `Decipher.java`.

He then redirected the output to a cipher file called `cipher.txt`.

1) Use the cipher text file and the code in listing 6 to recover the plain text. *Hint*: $7^{-1} = 55 \mod 2^7$.
2) What should the $2^7$ tell you about the text encoding of the original plain text file?

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class Decipher {
  public static void main(String[] args) {
    int m = (int)Math.pow(2, 7);
    char[] cipher = new char[m];
    for (int i = 0; i < m; i++)
      cipher[i] = (char)mod(7 * i + 3, m);
    String plain = read("../media/plain.txt");
    for (int i = 0; i < plain.length(); i++)
      System.out.print(cipher[plain.charAt(i)]);
  }
  public static String read(String path) {
    String ret = null;
    try {
      ret = new String(Files.readAllBytes(Paths.get(path)));
    } catch (IOException e) {
      e.printStackTrace();
    }
    return ret;
  }
  public static int mod(int x, int n) {
    return ((x % n) + n) % n;
  }
}
```

Listing 6. An Affine cipher in Java.

### 10.1. Example Executions

Figure 1 shows how the output of the code for the files `Fibonacci.java` and `Eratosthenes.java` should look like on the standard out. All your programs must compile/run from the command line using `javac` and `java` commands, e. g.,

```
javac Program.java
java Program
```

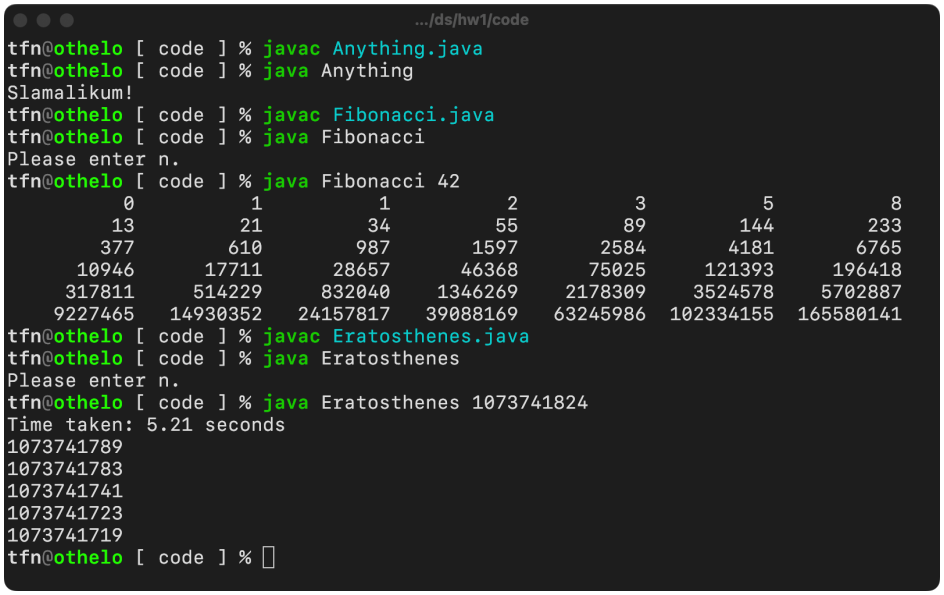


Figure 1. Example execution of the code for the first three questions.

### 10.2. Submission Instructions

- Submit `Anything.java`, `Fibonacci.java`, `Eratosthenes.java`, `Primes.java`, `Search.java`, `Decipher.java` and `sol.pdf` at the online classroom.
- The files `Anything.java`, `Fibonacci.java`, `Eratosthenes.java`, `Primes.java`, `Search.java`, `Decipher.java` should contain the Java source code for the relevant questions. Do not turn in the dot class files.
- The PDF file `sol.pdf` should contain written answers to questions as well as a screenshot similar to the one in figure 1 that demonstrates your code being compiled and ran.