Homework 4

Question 1 (5 pt.)

A palindrome is a string that reads the same backward as forward (e.g., “madam”, “a”, “aba”, “abba”, or even the empty string). The definition of a palindrome can be expressed recursively as follows: a string is a palindrome if *i)* its length is equal to 0 or 1, or *ii*) its first and last character are the same and the rest of the string between these two characters is itself a palindrome.

Write a Java class named `Palindrome` containing the following static functions:

- Function `IsPalindrome()` is a recursive function that takes an input string as an argument and returns a Boolean value indicating whether the string is a palindrome. This function should be implemented using recursion based on the recursive palindrome definition above.

  You can use functions `length()`, `charAt()`, and `substring()` on top of a `String` object to obtain its length, a character at a specific index, and a substring, respectively. Access online Java documentation to understand the exact behavior of these functions.

- Function `main()` is a main program asking the user to enter an input string. The program invokes function `IsPalindrome()` to determine whether the entered string is a palindrome, and prints a message for the user indicating the answer.

These are two examples of the program execution:

```
$ java Palindrome
Enter string: madam
The string is a palindrome

$ java Palindrome
Enter string: abc
The string is not a palindrome
```

Create a directory named `q1` and place file `Palindrome.java` in it. Run your main program by entering various strings and verify its correct execution. Create a package named `q1.zip` containing directory `q1`, and submit it on Canvas.

Question 2 (5 pt.)

Create a directory named `q2` and create the following files in it:

- Copy file `SelectionSort.java` from the code presented in class. Remove functions `main()` and `Print()` from this file, and make function `Sort()` public. You can now invoke this function from other classes by calling `SelectionSort.Sort()`.

- Copy file `MergeSort.java` from the code presented in class. Remove functions `main()` and `Print()` from this file, and make function `Sort()` public. You can now invoke this function from other classes by calling `MergeSort.Sort()`.

- Create a new file called `Test.java` containing a `main()` function that expects two arguments from the command line, called `algorithm` and `size`. Argument `algorithm` is a string set to "`selection`" or "`merge`", and argument `size` is an integer value. These arguments are passed when the program is executed, and are available in array `args` in the header of `main()`. This example shows how to run your program:

  ```
  $ java Test merge 20
  ```

The `main()` function performs the following actions:

- Read argument `algorithm` from `args[0]`.

- Read argument `size` from `args[1]`, and convert it to an integer. Use function `Integer.parseInt()` for that purpose.

- Create an array of `size` elements and initialize them with random values. You can use built-in class `Random` for this purpose (see online documentation).

- If variable `algorithm` is set to "`selection`", sort it with selection sort. If it is set to "`merge`", sort it with merge sort.

Run your program for both algorithms and different array sizes, and prepend your shell command with `time`. This will make the shell add information about how long your program took to execute. You can use the time component labeled *user*, which only includes the time that the processor devoted to your

program, and not other users' programs. Example:

```
$ time java Test selection 10000

real  0m3.029s
user  0m3.019s
sys   0m0.048s
```

Show your results in a written answer and write a short paragraph commenting them. Explain at what array size each sorting time begins to be noticeably high, and why.

Upload your written answers on Canvas in a PDF document named `hw4.pdf`. Create a package named `q2.zip` containing directory `q2`, and upload it on Canvas, too.