Worksheet 9: Searching and Sorting

Updated: 17th May, 2020

- Search through an array using a Linear Search
- Sort an array with Bubble Sort
- Sort an array with Insertion Sort
- Sort an array with Selection Sort
- Search through an array using a Linear Search

Note: You must submit this practical worksheet by:

Sunday, 24th May 2020, 5pm (local time)

Submissions must be done **electronically** through Blackboard. You are only required to submit once for this practical (your **entire** P09 directory in **.tar.gz** format). You must submit everything you have completed in this practical, both in class and at home.

You must also not modify the files in ~/Documents/PDI/P09 until after your submission has been marked. During your practical your tutor will check the modified dates of the files using `ls -l` (el) before marking your work. To create a gzipped tarball use the following command from your **PDI** folder:

```
[user@pc]$ tar -cvzf <studentID>_P09.tar.gz P09
```

For more information on what each argument of the above command does use:

```
[user@pc]$ man tar
```

To get your **.bash_history** into a submittable format, first close all terminals down, using <ctrl>-d. Then open a new terminal, and type this command from anywhere:

```
[user@pc]$ history >~/Documents/PDI/P09/BashHistoryP09.txt
```

1. Student Class: Reminder

You will be required to use your **Student** Class from last week. For your reference:

The university has asked you to design a "**Student**" Class that will be able to store a students name, as well as 1 of their Test Marks. You are required to store:

- **String name** – It is considered valid if not blank or `null`
- **Integer studentID** – It is considered valid if it is between `10000000` and `99999999`
- **Real mark** – It is considered valid if it is between `0.0` and `100.0` (inclusive)

While writing this class, as well as having all of the required methods from the lecture slides (3 Constructors, Accessors for each Classfield, Mutators for each Classfield, **toString**, **equals** and **clone**) you are required to have the following:

- **String getGrade()** – Returns the "grade" equivalent of the **mark**
- **String toFileString()** – Returns the object reconstructed into its CSV equivalent. e.g., `"Mark Upston,10000001,100.0"`

Once written in pseudocode, convert it to Java and test it fully.

You are also required to use the File IO created last week for this weeks implementation. You will be building upon it whilst using the source file: **RandomNames7000.csv**

2. Linear Search

You are required implement a Linear Search from the lecture slides. You will need to import the **RandomNames7000.csv** and search for a Student Name, that the user will give.

Note: Be careful about case-ing, if the user enters `"mArK UpStoN"` that should still match with `"Mark Upston"`

You are required to also implement an effective timing mechanism to check the speed in which it takes to search your array.

Note: These may be useful to you:

```java
long startTime = System.nanoTime();
  // Do sorting or anything that you want to test the time for
long endTime = System.nanoTime();
int total = (int)((double)(endTime - startTime) / 1000.0);
```

3. Bubble Sort

You are required implement Bubble Sort from the lecture slides. You will need to import the `RandomNames7000.csv` and sort the file via Bubble Sort. You are required to also implement an effective timing mechanism to check the speed in which it takes to sort your array.

4. Insertion Sort

You are required implement Insertion Sort from the lecture slides. You will need to import the `RandomNames7000.csv` and sort the file via Insertion Sort. You are required to also implement an effective timing mechanism to check the speed in which it takes to sort your array.

5. Selection Sort

You are required implement Selection Sort from the lecture slides. You will need to import the `RandomNames7000.csv` and sort the file via Selection Sort. You are required to also implement an effective timing mechanism to check the speed in which it takes to sort your array.

6. Binary Search

Now that we have a sorted file, we need to implement and test our Binary Search algorithm. You will need to import the `RandomNames7000.csv` and search for a Student Name, that the user will give. You are required to also implement an effective timing mechanism to check the speed in which it takes to search your array.

7. Findings

In a text document note your findings for each Searching and each Sorting Algorithm. Also note why some might be higher/lower/same.

End of Worksheet