COP3530 – Assignment 3

Objective

Students will be able to conduct a comparison study of the performance of searching algorithms. An experiment will be implemented to analyze the running times of well-known searching algorithms in a given scenario.

Assignment Problem

Consider the following problem. A large set of integers, stored in an array named **dataset**, must be searched continuously to determine the presence of values coming from a certain source **dataSource**. It is understood that the set of values in **dataset** will not change ever. Suppose that after careful analysis, you are left with two final candidates for algorithms:

Algorithm A1: For each value of **dataSource**, search for its presence in **dataset** using linear search.
Algorithm A2: Sort **dataset** with **quicksort** once, and then search each value of **dataSource** in **dataset**, using binary search for each.

Task: design and implement an experiment to empirically estimate the number of elements **k** in **dataSource**, so that the running time of Algorithm 2 outperforms Algorithm 1 for any number of values in **dataSource** greater than or equal to **k**.

In the example below (the numbers are not taken from a real example; they are offered to illustrate the problem), such a **k** would be 145. Note that the values under the second and third columns represent seconds.

k	Algorithm A1	Algorithm A2
1	0.012	0.606
2	0.02	0.648
3	0.718	0.632
4	0.03	0.668
5	0.019	0.628
6 – 144	A1 takes less time	A2 takes more time
145	0.801	0.798
>146	A1 takes more time	A2 takes less time

Requirements

- Type of the elements both in **dataset** and **dataSource**: int
- Size of **dataset**: 2×10^7
- Range of values in **dataset**: random integers in $[0, \text{size of dataset}]$, generated with the **nextInt** method of the **Random** class
- The data source, **dataSource**, will be an array

- Range of values in **dataSource**: random integers in $[0, 2 \times \text{size of dataset}]$, generated with the **nextInt** method of the **Random** class
- Units of time: seconds (note that Java offers time measuring utilities in milliseconds and nanoseconds; conversion to seconds must be performed by your program)
- Sorting algorithm: **quicksort**; implementation in separate file (you are required to use the provided implementation)
- Your program will save the results to a **.csv** file. Use Excel to depict graphically the results of your experiment (Excel can open .csv files).

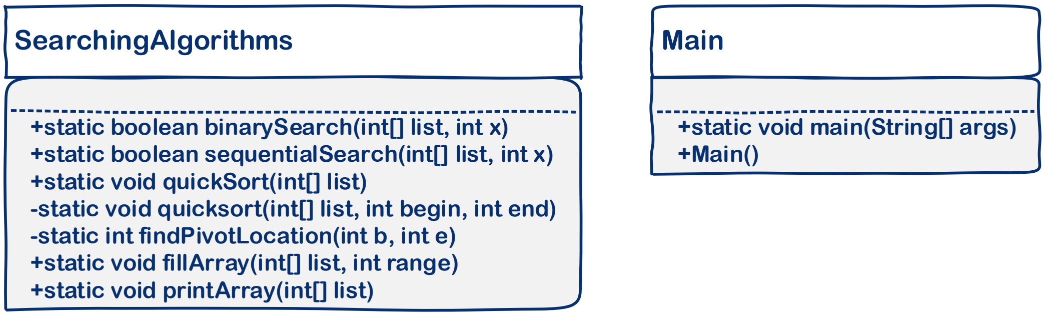
  Example of the content of .csv file:

  ```
  1,  0.012, 0.606
  2, 0.02, 0.648
  3, 0.018, 0.632
  4, 0.03, 0.768
  5, 0.019,0.828
  . . .
  ```

- A **Conclusions** document will be submitted, which will include:

  a) What was the value of **k** obtained?
  b) Explanation on the results obtained.
  c) The Excel picture(s), chart(s), or diagram(s).
  d) What did you learn?

- Students are required to structure the code as indicated in the UML class diagram:


SearchingAlgorithms
+static boolean binarySearch(int[] list, int x)
+static boolean sequentialSearch(int[] list, int x)
+static void quickSort(int[] list)
-static void quicksort(int[] list, int begin, int end)
-static int findPivotLocation(int b, int e)
+static void fillArray(int[] list, int range)
+static void printArray(int[] list)
Main
+static void main(String[] args)
+Main()

Guidelines

- The assignment is to be completed individually or in teams of two students. *Only one member* of a team will submit the assignment.
- The given problem is based on the content studied in class on searching algorithms.
- You are allowed to use all of the code discussed in the lectures. In those cases, make sure you properly credit its source.

Deliverables:

- A compressed folder, *PID Assignment 3* (e.g. *1234567 Assignment 3*), containing

    - all of the source code of the exercise
    - Conclusions (Word or PDF file) document with the content and structure indicated above
    - the .csv file
    - the Excel file

- Include **only** the .java files mentioned above; do not include other files or folders generated by the IDE.

- Make sure you write name(s) and Panther ID(s) in the class comment section of each Java file.

- In teams of two students, make sure the member who submits the assignment writes names and Panther IDs of both students in the comment section of the submission window.

Grading Rubric

The assignment is worth 115 points (out of 1000 total course points). Grade components:

<table>
<tr><th>Component</th><th>Points</th><th>Description</th></tr>
<tr><td>Submission</td><td>5</td><td>The student has submitted the project solution using the requirements for deliverables specified in the <i>Deliverables</i> section.</td></tr>
<tr><td>Organization</td><td>5</td><td>Code is expected to be neat, organized, and readable.</td></tr>
<tr><td>Content</td><td>105</td><td><table>
<tr><th>Deliverable</th><th>Points</th></tr>
<tr><td>source code</td><td>65 pts</td></tr>
<tr><td>conclusions</td><td>20 pts</td></tr>
<tr><td>.csv file</td><td>10 pts</td></tr>
<tr><td>Excel file</td><td>10 pts</td></tr>
</table></td></tr>
</table>