Lab 1, Part 2: Experimental Evaluation of Algorithms

Due date: April 8, at the beginning of lab period.

Lab Assignment

Assignment Preparation

This is a pair programming lab continuing Lab 1-1. If you worked on your Lab 1-1 assignment on your own, you may continue working Lab 1-2 part alone. You may also join forces with another student who worked on his/her own on Lab 1-1. If you worked as a pair and both members continue with the class, you stay with your partner.

Please note, that (a) only one submission per pair is required, and (b) the names of BOTH students must be found at the top of every file submitted.

The Task

The purpose of this assignment is to let you develop tools and techniques that assist in experimental evaluation of algorithms. The assignment has two parts:

- Experimental evaluation of your BST implementation.
- Implementation and evaluation of performance of efficient algorithms for finding the largest and the second largest element in an array.

Note: The purpose of the lab is to introduce you to the process of implementing and evaluating performance of algorithms. We do this using two rather simple algorithms: finding the largest and the second largest element in an array.

Task 1: Evaluation of Binary Search Trees

You will modify your implementation of the BST ADT from Lab 1-1 to add the following features:

Comparison count. Each BST ADT operation now **must** keep track of the number **comparisons** computed during its execution. Thus, you shall modify your implementation of the BST ADT and the `insert()`, `delete()`, and `find()` methods to

1. keep a count of the number of comparisons **involving BST keys and prospective BST keys** used in each method.
2. expose this information outside of the individual methods. Your implementation should be able to find how many comparisons a specific method call yielded, and it should be able to compute the total number of comparisons needed to perform a complex tast that requires multiple calls to the ADT methods.

Additionally, you shall modify your implementations of the `sort()`, `findMax()` and `findSecondMax()` methods to do the same. Please note, that since these methods *first build* a BST, the total number of comparisons computed in those methods *should include the comparisons made while building the BST*[1]

Evaluation. The main goal of your updates is to establish a framework for evaluating the running time of your algorithms *expressed in terms of the number of comparison operations.* Once you modify your code to compute and pass along this information, you will conduct a study of the running time of your implementations of the `sort()`, `findMax()` and `findSecondMax()` algorithms.

The study will be conducted as follows.

- **Dependent variable.** Your study will measure the number of comparisons used up by your implementation and the *average number of comparisons* over a set of runs.
- **Independent variables.** The main independent variable in your study is the size of input measured as the *number of elements* of in the input array. You will also conduct a separate study to see if the order of elements has any effect on the number of comparison necessary to complete the computations.

You will conduct two studies.

[1] In fact, if you implement your methods correctly, these will be the only comparisons made.

Evaluation Framework.Study 1. In the first framework, you will evaluate the dependency of the average number of comparisons for the `sort()`, `findMax()` and `findSecondMax()` operations on the size of the input array. You will perform the study, by creating an evaluation framework as follows:

- Your evaluation framework will be run as a Java program `BSTStudy.java`.
- You will test how your `sort()`, `findMax()` and `findSecondMax()` methods work on arrays of the following sizes:

  10, 20, 50, 75, 100, 200, 500, 750, 1000, 2000, 5000, 7500, 10000

- Your evaluation framework will work as follows. For each array size from the list above, you generate 1000 test arrays of that size. Each array shall be generated using Java's pseudorandom number generation mechanism, with the random number generator seed set at the beginning of the `BSTStudy01.java` program in a way that ensures that different arrays get generated on subsequent runs of the program (i.e., DO NOT set the seed to a constant).

  (Note, Java has a `java.util.Random` class for generating random numbers.)

- On each array generated in this way, you run your `sort()`, `findMax()` and `findSecondMax()` methods and collect the information about the number of comparisons used.
- For each array size, you compute the **average** among the 1000 arrays of that size, number of comparisons used by each of the three methods.
- Your framework shall output three sets of 13 numbers: the average number of comparisons for each size and each of the three methods being tests. The output shall be human-readable, and in a format, that is easily importable in a spreadsheet application (e.g., a CSV, i.e., a comma-separated values format will work great). This is because:
- You will use the output of your program to construct a plot indicating, for each of the three methods being tested, the dependency of the average number of comparisons on the input size. (I recommend using a spreadsheet program like `MS Excel` or `Open Office calc`. Other possibilities include using `MatLab` or `gnuplot` available in the CSL, but you are repsonsible for knowing how to use the software to construct the plots).

Evaluation Framework. Study 2. This study allows you to see how your implementation works in the best-, worst- and average cases. For this study, you need to do the following.

- You will work with arrays of only one size: 20.

- Analyze the features of the BST ADT and your implementation, and figure out what constitutes the best- and, worst- and the average-case scenarios for the fixed-size input. (Hint, the average-case scenario is essentially a randomly generated array.)

- Create a number of arrays of size 20 (you can use code to crate some of them, and you can create some of the manually to match your perceptions of the best- and worst-case scenarios).

- Run the `sort()`, `findMax()` and `findSecondMax()` on each of the arrays you created. Collect the running time (number of comparisons) information.

- Create a plot/chart/graph showing your discoveries. Write up your observations and conclusions.

Task 2: Implementation and Evaluation of Efficienct findMax() and findSecondMax() Algorithms

Your second task is to implement and evaluate using techniques similar to those of **Task 1**, the efficient algorithms for finding the largest and second largest elements of an array.

findMax() . You will implement an efficient algorithm for finding the largest element of an array as a method called `findMax()`. Among the parameters for this method should be `int N`, the size of the array and `int[] A`, the array itself.

You method will count the number of comparisons between the elements of the array made during its execution. You will develop a mechanism for reporting this number back to the calling method.

findSecondMax() . You will implement an efficient algorithm for finding the second largest element of an array as a method called `findSecondMax()`. Among the parameters for this method should be `int N`, the size of the array and `int[] A`, the array itself. (We will be discussing this algorithm in class either on Thursday, April 1, or on Tuesday, April 6).

You method will count the number of comparisons between the elements of the array made during its execution. You will develop a mechanism for reporting this number back to the calling method.

Validation wrapper. You will create a "validation wrapper" for these two methods called `MaxValidate.java`. This wrapper will take as input a name of a file containing a comma-separated list of numbers, read in and parse the contents of the input file, create an array contiaining the parsed numbers, run both `findMax()` and `findSecondMax()` on the array, and report back (print) both the computed largest and second largest numbers in the array (please mark them appropriately), and the number of comparisons it took to compute each.

Evaluation study. As with **Task 1**, you will perform a simple evaluation study designed to determine the dependency of the running time of your implementations of `findMax()` and `findSecondMax()` (expressed as the number of comparisons made) on the size of the input.

The **dependent variable** in your study is the average number of comparisons over a sequence of related runs of a method.

The **independent variable** in your study is the size of the input array. You will test arrays of the following sizes:

> 8, 16, 32, 64, 100, 128, 256, 512, 1024, 2048, 4096, 5000, 8192, 10000, 16384

You will create a Java program `MaxStudy.java` responsible for running the study. This program will automaticall generate **100** arrays for each size specified above using the same random-number generation techniques as in the study for **Task 2**. For each array, the program will run `findMax()` and `findSecondMax()` and obtain the number of comparisons it took.

For each array size, your program will compute the average number of comparisons it took to find the largest, and the second largest element of the array. These numbers shall be printed out in readable format by the program.

Reporting. Similarly to **Task 1**, you will use the output of `MaxStudy.java` program to create plots documenting the relationship between the array size and the average number of comparisons for each of the methods.

Create a simple report document that incorporates the plot you obtained, the table of the results, and your thoughts/observations about it.

Deliverables

This part of the lab has both electronic and hardcopy deliverables. All electronic deliverables shall be submitted by the assignment deadline using the following `handin` command:

Use `handin` to submit:

```
$ handin dekhtyar-grader lab01-2 <files>
```

The hardcopy deliverables shall be submitted to the instructor during the lab period on Thursday, April 8.

Please note, that we will start a new lab on April 8, so your work shall be completed by then (you are allowed to submit your materials, both electornic and hardcopy during that lab, but you are not allowed to work on Lab 1-2 assignment then).

Electronic Deliverables. For **Task 1** the electronic deliverables are:

- Your Java file(s) of the new BST ADT implementation.
- `BSTStudy.java`.
- Java programs and data you used for the second study.
- The softcopies of your reports for both studies (feel free to submit as a single document). The softcopies should contain the tables you used to create your plots/graphs/charts, the plots/graphs/charts for both studies and your observations about them. The report for the second study should also mention what you consider to be the best- and worst-case scenarios for your methods and provide examples that you used.

  You can prepare your reports in any format, but please submit your reports in PDF. (Convert your MS Word documents to PDF if necessary).

For **Task 2** the electronic deliverables are:

- Java program(s) containing your implementations of `findMax()` and `findSecondMax()`.
- `MaxValidate.java`
- `MaxStudy.java`
- Softcopy of your evaluation study (in PDF).

In addition, please submit a `README` file which

- Contains the list of all students on your team.
- Contains the list of all the files you submitted and describes each file.
- Contains any necessary compilation instructions.

Note that we expect that all your programs will compile from the command line. Make sure you test for that.

Also, **please include the names of all students on the team** in the header comment for each submitted file.

Hardcopy Deliverables. Submit in hardcopy:

- Report of Study 1 for **Task 1**.
- Report of Study 2 for **Task 1**.
- Report of your study for **Task 2**.

Good Luck!