Lab03 Sorting and Selection

VE281 - Data Structures and Algorithms, Xiaofeng Gao, Autumn 2019

1 Motivation

1. To give you experience in implementing various sorting algorithms and two linear-time selection algorithms, i.e., random and deterministic selection algorithms.
2. To empirically study the run time efficiency of the sort algorithms and two selection algorithms and compare the run time efficiency of the selection algorithms with the sorting algorithms.

2 Programming Assignment

1. For the first part, you are required to implement five sorting algorithms: bubble sort, insertion sort, selection sort, merge sort, *in-place* partition quick sort. They should sort integers in **ascending order**.
2. For the second part, you are required to implement two linear-time selection algorithms: random and deterministic selection algorithms. The algorithm takes an array of n integers and a number $0 \leq i \leq n$ as its inputs. It will output the $i-$th smallest item in the array, which is called the order-i item. Note that i starts from 0. This means the order-0 item is the smallest one in the entire array and the order-$(n-1)$ item is the largest one.
3. All the above seven algorithms are supposed to be combined into one single program. Based on the user specification, a corresponding sorting algorithm will be called.

2.1 Input Format

You will read the data from **standard input**. (For the ease of testing, you can write each test case in a file and then use Linux file redirection function "<" to read from the file.)

The first line is an integer, which specifies the sorting algorithm you should call. The integer can take six values: 0 for bubble sort, 1 for insertion sort, 2 for selection sort, 3 for merge sort, 4 for in-place partition quick sort, 5 for random selection algorithm and 6 for deterministic selection algorithm. Other values are illegal, but you can assume that the user will not input illegal values.

2.1.1 Input for Sorting Algorithms

If the first integer you read is from 0 to 4 which means you need to sort an array, the second line specifies the number of integers you are asked to sort. Let that number be N. Then the following N lines list the N integers to be sorted.

An example of input for sorting is

```
3
5
-1
-3
2
0
4
```

This example calls merge sort to sort 5 integers -1, -3, 2, 0, and 4 in ascending order.

2.1.2 Input for Selection Algorithms

If the first integer you read is 5 or 6 which means you need to apply selection algorithms to an array, the second line specifies the number of integers in the input array. Let that number be n. The third line is an integer $0 \leq i \leq n-1$, which indicates the algorithm will output the order-i item in the array. You can assume that the user always inputs an order i in the valid range. The following n lines list the n integers in the array.

An example of input for sorting is

```
5
5
2
-1
-3
2
0
4
```

This example calls random selection algorithm to get the order-2 item in an array of 5 elements. That item should be 0.

2.2 Output Format

For sorting algorithms your program should write the result in **standard output**. Each line lists one number. For the above example in section 2.1.1, the output looks like,

```
-3
-1
0
2
4
```

For selection algorithms your program should write the result in the following output format,

```
The order-<i> item is <val>
```

For the above example in section 2.1.2, the output looks like,

```
The order-2 item is 0
```

3 Algorithms Runtime Comparison

In this Lab, we also ask you to study the runtime efficiency of these algorithms. To do this, you should generate different arrays of random integers with different sizes. Then you should apply your algorithms to these arrays. Note that for *selection algorithms* the runtime also depends on the specified order i. To eliminate the dependency on i, for a given array, you should choose multiple i's, run your algorithm over all these i's, and report the average runtime over all i's.

Finally for this part, you are supposed to hand in a report that contains **two** figures and necessary analysis. For the first figure, it should indicate the runtime of 5 different sorting algorithms versus the array size (five curves in one figure). In this lab, we also ask you to compare the runtimes of the two selection algorithms with the quick sort algorithm. So for the second figure, it should indicate the runtime of 2 different selection algorithms and quick sort versus the array size (three curves in one figure).

Hints:

1. You can use `mrand48()` to generate integers uniformly distributed in the range $[-2^{31}, 2^{31}-1]$.
2. You can use `clock()` function to get the runtime. See http://www.cplusplus.com/reference/ctime/clock/ for reference.
3. Although the major factor that affects the runtime is the size of the input array, however, the runtime for an algorithm may also weakly depend on the detailed input array. Thus, for each size, you should generate a number of arrays of that size and obtain the mean runtime on all these arrays. Also, for fair comparison, the same set of arrays should be applied to all the algorithms.
4. You should try at least 5 representative sizes.
5. You do not need to submit the source code for this comparison part.

4 Implementation Requirements and Restrictions

1. You must make sure that your code compiles successfully on a Linux operating system.
2. Only #`include <iostream>`, `<fstream>`, `<sstream>`, `<string>`, `<cstdlib>`, `<climits>`, `<ctime>`, and `<cassert>` are allowed in this Lab. No other system header files may be included, and you may not make any call to any function in any other library.

5 Compiling and Submitting

Write a `Makefile`. Put all your compiling commands in the `Makefile`. The output program should be named `main`.

You should submit all the source code files for the program described in Section 2, a `Makefile`, and a report of the runtime comparison of different algorithms. The report should be in pdf format. At the end of your report, please attach your source code for the program described in Section 2.

The source code files and `Makefile` should be submitted as a tar file via the online judgment system. As for your report, you should strictly follow the *Submission Requirements* for this course, and submit the package on the website.

6 Grading

First of all, please notice that the *Best Lab Works Bonus Policy* applies in this assignment. Your works will be graded in the following aspects, **Functional Correctness**, **Implementation Constraints**, **General Style**, **Performance** and **Report**.

So your assignment should fully realize the functions listed in Section 2 and meet all of the implementation requirements and restrictions. Meanwhile, you should also pay attention to your general coding style, which means your code should be understandable, clean and elegant. We will test your program with some large test cases. If your program is not able to finish within a reasonable amount of time, you will lose the performance score for those test cases. Finally, we will also read your report and grade it based on the quality of your performance study.