Programming Assignment Two: Linear-Time Selection

Out: Oct. 11, 2016; Due: Oct. 23, 2016.

I. Motivation

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

II. Programming Assignment

You are asked to implement two linear-time selection algorithms: random and deterministic selection algorithms. The algorithm takes an array of n integers and a number $0 \leq i \leq n-1$ as 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 in the largest one. Based on the user specification, a corresponding selection algorithm will be called.

1. Input format

You will read the data from the **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 selection algorithm you should call. The integer can take two values: 0 for random selection algorithm and 1 for deterministic selection algorithm. Other values are illegal, but you can assume that the user will not input illegal values.

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 is

```
0
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. Output Specification

Your program should output:

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

For the above example, the output is

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

III. Algorithm Runtime Comparison

We also ask you to study the runtime efficiency of these two selection algorithms. To do this, you should generate different arrays of random integers with different sizes. Then you should apply your selection algorithms to these arrays. Note the that 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. We also ask you to compare the runtimes of these two selection algorithms with the quick sort algorithm. Your final report should include a figure showing the runtimes of two selection algorithms and the quick sort algorithm versus the array size. For comparison purpose, you should plot these three curves in the same figure. (You do not need to upload the source codes for this comparison program.)

Hint:

1. You may want to write another program that calls the core selection procedures to do this study.
2. You can use mrand48() to generate integers uniformly distributed in the range $[-2^{31}, 2^{31}-1]$. See https://www.mkssoftware.com/docs/man3/drand48.3.asp
3. You can use clock() function to get the runtime. See http://www.cplusplus.com/reference/ctime/clock/
4. 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 average runtime over all these arrays. Also, for fair comparison, the same set of arrays should be applied to all the algorithms.
5. You should try at least 5 representative sizes.

IV. Implementation Requirements and Restrictions

- You must make sure that your code compiles successfully on a Linux operating system.
- Your compiled program (i.e., the program described in Section II, not in Section III) should be named as `p2` exactly.
- You may not leak memory in any way. To help you see if you are leaking memory, you may wish to call `valgrind`. Please refer to Programming Assignment One for the description on how to use the command `valgrind`.
- You may `#include <iostream>`, `<fstream>`, `<sstream>`, `<string>`, `<cstdlib>`, `<climits>`, `<ctime>`, and `<cassert>`. No other system header files may be included, and you may not make any call to any function in any other library.

V. Compiling

In order to let us test your code automatically, we ask you to provide a `Makefile` for us. Please refer to Programming Assignment One for the description on how to write a `Makefile`.

VI. Testing

To make sure that the program works, you should test each selection algorithm you write extensively. **A hint**: You can apply quick sort to verify whether the output of the selection algorithm is correct.

VII. Submitting and Due Date

You should submit all the source code files for the program described in Section II, a `Makefile`, and a report of the runtime comparison of the two selection algorithms and the quick sort algorithm. The `Makefile` compiles a program named `p2`. The report should be in doc, docx, or pdf format. These files should be submitted via the online judgment system. See announcement from the TAs for details about submission. The submission deadline is 11:59 pm on Oct. 23rd, 2016.

VIII. Grading

Your program will be graded along five criteria:

1. Functional Correctness
2. Implementation Constraints
3. General Style
4. Performance
5. Report on the performance study

Functional Correctness is determined by running a variety of test cases against your program, checking your solution using our automatic testing program. We will grade Implementation Constraints to see if you have met all of the implementation requirements and restrictions. In this project, we will also check whether your program has memory leak. For those programs that behave correctly but have memory leaks, we will deduct some points. General Style refers to the ease with which TAs can read and understand your program, and the cleanliness and elegance of your code. Part of your grade will also be determined by the performance of your algorithm. 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.