Project 4
CS 303/503 Algorithms

Fall 2012

Due by: 8:00 a.m., Wednesday, December 5, 2012

This programming project is subject to the CSCI 303/503: Regulations for Programming Projects found at `www.cs.wm.edu/~va/CS303/project_regs.html`.

For this project you are to add one last public method to the class `GroupOfNumbers`

- `bool linear_select(long k, long & kth_smallest) const`

that will allow a user to select, in guaranteed worst-case linear time complexity for all legitimate inputs, the kth smallest number in a `GroupOfNumbers` object. You will need to add protected methods, of your own specification, to implement the linear selection algorithm; we simply specify the public interface that allows us to invoke your implementation from within our test program. Your full implementation of the linear selection algorithm *must* follow the general strategy outlined by Weiss in the first part of Section 10.2.3.

In addition, you are to submit timing results for *both* your suite of sorting methods and your suite selection methods. Be sure to allow yourself sufficient time for running the timing tests! I have a simple suggestion that should allow you ample time for running all of the timing tests you are required to submit: start running the tests for your existing implementations (if they are correct) while working on your new implementation (or while correcting implementations you know cannot be correct).

Further requirements and suggestions are detailed in the sections that follow.

Implementing the linear selection algorithm

Requirements for the implementation of the linear selection algorithm:

- Your implementation of the linear selection algorithm *must* follow the general strategy outlined by Weiss in the first part of Section 10.2.3, which means that your version *must* follow the same basic strategy employed in the quickselect algorithm given by Weiss in Section 7.7.6 *but* with median-of-three rule for choosing the pivot replaced by the median-of-median-of-five rule for choosing the pivot.

  Take care that your implementation of the linear selection algorithm does *not* invoke the quickselect methods you have already implemented. The fundamental difference between the linear selection algorithm, with its guaranteed worst-case time complexity of $O(n)$, and quickselect, with its expected time complexity of $O(n)$ but a worst-case time complexity of $O(n^2)$, is the rule used to select the pivot. The former uses median-of-median-of-five to choose the pivot, the latter uses median-of-three to choose the pivot.

  This particular requirement is critical.

  You must correctly implement and use the median-of-median-of-five strategy for choosing the pivot discussed by Weiss in the first part of Section 10.2.3 (also covered in Lecture 19 and on Homework 11) to receive *any* of the 54 (out of 100) points allocated to fully implementing the linear selection algorithm. Even more to the point, the median-of-median-of-five strategy must be used to choose the pivot at *every* level of recursion.

If you do not implement the strategy for linear select outline by Weiss in the first part of Section 10.2.3, you will receive no credit for either your implementation of the linear selection algorithm or the timing results you report for the linear selection algorithm.

- Note from the function prototype for `linear_select` that the argument `kth_smallest` is an output parameter. You must return the kth smallest number in the group in the argument `kth_smallest`.
- Further note from the function prototype for `linear_select` that the return value for the new public method is `bool`. You must return `true` if the value of `k` passed as an input parameter is valid (i.e., `1` $\leq$ `k` $\leq$ `total()`); otherwise, you must return `false`.
- Finally, note that the new public method includes `const` in its prototype. So, the object must not be modified (i.e., mutated) in any way by an invocation of this new method.

Some comments on `group.h`

The file `group.h` contains the `public` interface for the new method. I have outlined above the specifications you are required to meet when you implement the linear selection algorithm, but the remainder of the details regarding the specifics of the implementation are left up to you.

You will need to use at least one new `protected` method to implement the new `public` method. You may want to add additional `protected` methods to aid you in your implementation. Thus, you are allowed to *add* new methods to the `protected` section of the class `GroupOfNumbers`.

You may not in any way modify the prototypes of any of the methods of the class `GroupOfNumbers` that have already been defined in the copy of `group.h` that you download from the Web page for Project 4. In other words, you are allowed to *augment* the `protected` section of the class by adding new methods, but you are not allowed any other changes to the class definition.

The previous requirement is important. Read it again carefully to be sure you understand it.

In particular, if you modify the prototypes for any of the `public` methods, expect automatic point deductions, since modifying the public interface makes it impossible for us to use a test program for grading.

As a final note, since you will augment `group.h` during the course of your development, you will need to submit a copy of your augmented file `group.h` along with a copy of your implementation file `group.cc`.

Timing the sorting algorithms

Take your implementations of insertion sort, heapsort, mergesort, and quicksort (using four different pivot selection rules) and time how long it takes to sort a `GroupOfNumbers` object when the numbers in the group are randomly generated, sorted in order, sorted in reverse, and identical. As in Project 2, Part B, assume in all cases that the `GroupOfNumbers` objects contains $250,000$ numbers.

Once again, in effect, you are to fill out Table 1. This time, though, the goal is to ensure that the timing you report for *every* sorting algorithm is consistent with the timings expected, given the form of the input (e.g., randomly generated, sorted in order, sorted in reverse, and identical). For this project, be *sure*, as you are filling in the table, that the timings you are reporting are consistent with the the analytical time complexities for each algorithm applied to that particular type of input. If not, there are corrections that still need to be made to your implementation!

Those of you who submitted error- and warning-free implementations of the sorting algorithms for Project 2, Part B may simply reuse your timing results from that project. But if you had an error or warning in your implementation of a sorting algorithm, you must rerun your timing results for that sorting algorithm after making all the required corrections.

Recall that I gave some representative timing results on page two of Project 2, Part B so you have some idea of what to expect.

Sorting algorithm (including pivot rules for quicksort)	randomly generated	sorted in order	sorted in reverse	identical elements
insertion sort				
heapsort				
mergesort				
quicksort with pivot rule: choose median of 3				
quicksort with pivot rule: choose first element				
quicksort with pivot rule: choose middle element				
quicksort with pivot rule: choose last element				

Table 1: Timing results for the sorting algorithms with $n = 250000$.

Timing the selection algorithms

As warned in Project 3, for this project you are required to submit timing results for all six selection algorithms you implement. In effect, you are to fill out Table 2.

Take your implementations of sort select, partial sort select, minimum binary heap select, partial maximum binary heap select, quick select (with median-of-three rule for choosing the pivot), and linear select (with median-of-median-of-five rule for choosing the pivot) and time how long it takes, given a `GroupOfNumbers` object with 10,000,000 numbers, to select k for each of $k = 1$, $k = 5000000$ (i.e., k is the median) and $k = 10000000$ (i.e., $k = n$). Use randomly generated entries in the range [-10000000,10000000].

Again, as a sanity check, make sure that the timings you report are consistent with what the analysis of the time complexity for each algorithms says you should expect for the given choices of k. If not, that is an indication that your implementation needs to be reexamined.

In the specifications for Project 3 I gave you some idea of the timings you should expect for the five algorithms you implemented for that project.

One of the selection algorithms takes a excruciatingly painful amount of time to run for one particular choice of $k \in \left\{1, \lceil \frac{n}{2} \rceil, n\right\}$. You are not required to run this one particular test and give an actual time, but if you opt not to run this one test, you *must* explain why not and give a rough estimate of the time it would take to run (as justification for why you did not run it). So, for instance, in Figure 7.24, Weiss gives "NA" as his timing result for insertion sort applied to the largest problem instance. Recall that insertion sort has time complexity of $\Theta(n^2)$. As is clear from his table, as Weiss increases the size of the input by a factor of 10, the execution time goes up by a little more than a factor of 100. Since it took Weiss 110.492 seconds to execute insertion sort on a problem with $n = 100000$ (the third edition of your textbook was published in 2006), it is a safe bet that it would have taken him at least 11049.2 seconds – over 180 minutes (3 hours) – to execute insertion sort on a problem with $n = 1000000$. Now calculate an estimate of the time it would have required him to obtain the result for $n = 10000000$!

You may resort to using "NA" for one (and *only* one) of the entries in Table 2 but such an answer must be accompanied by a reasonably thorough (but succinct) explanation from you as to why the test would be too painful to run. (Honestly, I do not expect you to run this one test – for reasons that should become clear if you do a rough "back of the envelope" calculation of the time it would take!)

Selection algorithm	$k = 1$	$k = \lceil \frac{n}{2} \rceil$	$k = n$
sort select			
partial sort select			
minimum binary heap select			
partial maximum binary heap select			
quickselect (with median-of-three rule for choosing the pivot)			
linear select (with median-of-median-of-five rule for choosing the pivot)			

Table 2: Timing results for the selection algorithms with $n = 10000000$.

Some comments on running the timing tests

It is an interesting exercise to see how much your timings do – or do not – change if you compile with the optimization flag "`-O`". Frankly, it will take much less (total) time to run all your tests if you do run them after compiling with the optimization flag.

It is instructive to try pushing your implementations of the sorting and selection algorithms. Try pushing your implementations of the selection algorithms by giving them an input of 1,000,000,000 – but be sure to choose k judiciously! If your implementations are robust, they should be able to handle inputs of this size – and some of the timing results may surprise you. Similarly, try pushing your implementations of the sorting algorithms by sorting at least 500,000 numbers; but be sure that the stack size is set to `unlimited` so that all your recursive implementations can run successfully to completion.

Some comments on submitting your timing results

Since all projects submissions are handled electronically using `submit`, and since we do not want to have to deal with different electronic document formats, you are required to report your timing results in two text (`.txt`) files, `Table1.txt` and `Table2.txt`, which you are to submit along with your files `group.h` and `group.cc`. *We do not want – and will not accept – paper copies of your timing results.*

To keep the formatting consistent, I have posted two templates, `Table1.txt` and `Table2.txt`, on the Web page for the project. Download copies of these two files and report your timing results in the appropriate positions in these files. In addition, where further explanation/discussion is required, please do so in the positions indicated in these `.txt` files.

Final instructions

To submit your files for grading, go to the directory containing your files of `group.h`, `group.cc`, `Table1.txt`, and `Table2.txt` and at the prompt in that directory enter:

```
~gpeng/bin/submit cs303 project4
```

In spelling file names and identifiers that appear in these specifications (or in attached files) you must use the exact case and spelling.

Point Breakdown for the Project

To give you some idea of how to allocate your time and effort on this project, here is the point breakdown for the project:

54 points for implementing the linear selection algorithm, both the `public` method

```
bool linear_select(long k, long & kth_smallest) const
```

and the `protected` method(s) you add to ensure a correct implementation of linear select.

28 points for the timing results for the sorting methods (Table 1, reported in `Table1.txt`)

18 points for the timing results for the selection methods (Table 2, reported in `Table2.txt`)

Total: 100 points

Notice that there is no "past sins" category in the point allocation for this, the final, project of the semester. The reason is simple enough. If your implementation of a particular sorting or selection algorithm still is not correct, the timing results you report for that algorithm will not be accepted.