CS4040/5040

Homework # 4

due Oct. 22, 2018

(50 pts)

Write two functions (`inbetweenGPA` and `inbetweenAlpha`). `inbetweenGPA(Students, j, k)` must return a vector of students whose GPA's lie between the `j`th biggest and `k`th biggest, of Student Records. `inbetweenAlpha(Students, j, k)` must return a vector of students whose names (see below for correct sort order) lie between the `j`th biggest and `k`th biggest, of Student Records (see below for details). The returned vector need not be sorted. You must write the functions to have the minimal asymptotic complexity to solve this problem.

The class definition for the records is:

```
class Student {
  public:
  string SSN;
  string first_name;
  string last_name;
  string major;
  float  gpa;
};
```

The functions must be modular in that three files, `student.h`, `inbetween.cc` and `inbetween.h` along with your `Makefile` are sufficient to compile your function into an object file `inbetween.o`. You must also write a `main.cc` program to test your inbetween function, and submit test cases. I will run your algorithm on different inputs of my own choosing, and time how fast it runs. Whoever has the fastest correct routine will get a bonus of 10 pts. Whoever uses the fewest comparisons with a correct algorithm will also get a bonus of 10 pts. You are free to use any method you wish for this algorithm. You must include a text write-up describing in detail the algorithm you chose, why you chose this algorithm, and analyze its best, worst, and average case running times.

The exact data structure used to hold the data will be: `vector<Student>`

The prototypes for your `inbetween` functions must be:

```
vector <Student> inbetweenGPA(    vector <Student>      list,
                                  const size_t j,
                                  const size_t k,
                                  size_t &num_compares);

vector <Student> inbetweenAlpha(  vector <Student>      list,
                                  const size_t j,
                                  const size_t k,
                                  size_t &num_compares);
```

The sorting order for the Alpha version is last name, followed by first name, followed by SSN. That is, if two students have the same last name, break the tie by using their first

names, and if they have the same first names, use SSN to break that tie. You may use the string class's less than operator to compare strings.

The functions must return a vector of all the students that are between the `j`th largest and `k`th largest inclusive in vector `list`. `j` and `k` start at 0, and run to the `size of list - 1`. Your code must not modify the vector passed in. In addition, you are to count the number of comparisons your algorithm performs between entries in the student vector passed in. Modify the `num_compares` variable so that when your function returns, it has correctly counted the number of comparisons your code needed to compute the answer.

For example, if I called:

```
inbetweenGPA(students, 0, 0, num_compares);
```

It must return a vector with one element, the student with the lowest GPA. Similarly if I called it as follows with students containing 100 student records, it must return a vector with one element, the student with the highest GPA.

```
inbetweenGPA(students, 99, 99, num_compares);
```

If I called it with:

```
inbetweenGPA(students, 0, 9, num_compares);
```

It must return a vector of the ten students with the lowest GPAs. Note: the returned vector need not be sorted.

If I called it with:

```
inbetweenGPA(students, 10, 19, num_compares);
```

It must return a vector of ten students whose GPAs are next lowest relative to the previous call to inbetweenGPA.

Similarly, the following call:

```
inbetweenAlpha(students, 0, 9, num_compares);
```

must return a vector of the ten students whose names come alphabetically first (as per the sort order listed above). Note: the individual elements of the returned vector need not be sorted.

Your program will be graded on correctness, efficiency, as well as style. You must analyze the efficiency of all the algorithms used in your program and include this analysis in comments in your code. In particular, you must document the efficiency of each function in your code with an asymptotic analysis for both time and space. This is also true for any code you include from other sources. In addition to the in-line comments, an overall analysis of the entire program, detailing the top-level tasks, with justifications and explanations of any

design decisions you make (e.g.,choice of algorithm or data structures used in your code) must be placed in the file `analysis.txt` and submitted with the rest of your program.

If you have any questions about how to implement this program, or on how to interpret any arguments to the function, be sure to bring these issues up in class.

You will submit an electronic version of your program through the submit program on the prime machines. Simply type:

```
% ~cs404/bin/submit prog1 student.h inbetween.h inbetween.cc main.cc readme.txt Makefile
```

Will submit the files `student.h inbetween.h inbetween.cc main.cc readme.txt` and `Makefile` for the assignment called `prog1`. There is also a menu version of the command, type:

```
% ~cs404/bin/submit
```

To be prompted for what to do, and to be able to see what you've already submitted, etc.

Your programs must conform to the style guidelines as given on the course home page.

A file with the class definition is available on prime in: `~cs404/examples/home4/student.h`