CS300 – Spring 2019-2020 - Sabancı University

Homework #5 – Sorting

Due: 20/05/2020, Wednesday, 23:55

Brief Description

In this homework, you have a PhoneBook that you kept updating without caring about sorting it **alphabetically**, and since you are a CS student, you care about the time taken to sort the PhoneBook so you want the **fastest algorithm** to perform this operation and the **fastest search structure**. Therefore, you must compare **4** Sorting algorithms and conclude which one is faster then perform search operation as you did in HW2.

You will be implementing the PhoneBook as you did already in the 2nd homework, in addition to using **vectors** to store the list of contacts in it loaded from the input file. You should load the PhoneBook in 4 different vectors then perform **comparison** between them (in terms of running times).

The sorting algorithms:

- Insertion Sort
- Quick Sort (pivot = [median])
- Merge Sort (in place; without using auxiliary memory storage)
- Heap Sort

Please note that after sorting the vectors, you should conduct a **binary search** about a contact or list of contacts on the sorted vector and compare the running times with those of the BST and AVL tree.

Program Flow

First, you have to read the input file and load the PhoneBook into vectors where you are going to apply a sorting algorithm on each one. In order to insert a contact, you need to insert its information into a data structure of your definition. There are 2 types of search that should be valid in your implementation, Partial and Full name search (similar to HW2).

After loading the PhoneBook into each vector copy, your program should ask for a search query (Partial or Full), then all 4 vectors should be sorted according to the sorting algorithms listed above.
Concerning the search operation, it should be performed on each PhoneBook copy (BST, AVL tree and sorted vector) so we can make comparisons between the search running times on each data structure.

For the comparisons, you should measure the running time without printing anything to the screen then display it for each structure.

You are also required to calculate the speedups between algorithms, the equation for calculating the speedup is given below.

$$Speedup_i = \frac{Slower\ SA\ exec\ time}{Faster\ SA\ exec\ time}$$

SA: Sorting Algorithm

The same formula applies when comparing between AVL tree, BST and the vector search times:

$$Speedup_i = \frac{Slower\ Search\ exec\ time}{Faster\ Search\ exec\ time}$$

Input

First, your program should ask for an input file (PhoneBook file) then it asks for a query entry where you should enter a full/partial contact name in one line.

Output

After inserting the inputs, you should first display the sorting times, second, the search times for AVL, BST and the vector, and last, the speedups list (check the sample runs to have a more clear image).
After getting the execution times, you will be able to compute the speedups and also display them.

Important notes

- ➔ Even though you are not asked to print out the results to the screen, your search functions should return correct results, this will be tested during the grading process.
- ➔ Your classes should be **template-based**.
- ➔ **Note:** The previous sample runs were taken from a machine with the following specifications:
  - CPU: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz (4 cores - 8 threads)
  - RAM: 4GiB

  Therefore, we understand that you have different machines, it can be powerful or weak in terms of specifications, so, we don't expect the speedup ratios to be exactly the same as in the sample runs, however, the results should be reasonable and conform with the results correctness, for example:
  - In **sample run 6**, the speedup ratio (Insertion Sort / Quick Sort) is so high (`42.7911`) so we will accept lower ratios like 30 or 20 or even 10. However, ratios beneath 2 would be suspicious and ratios like 0.9 or less (less than 1) are not acceptable and it shows that the code is problematic somehow.
- ➔ **In order to find the exact time, you can run the search function N times then you take the measured time and divide it by N, you will have much sharper timings (OPTIONAL).**

Example

```
int N = 1000; // if it takes too much time then N = 100
int timeStart  // start measuring
for(int i = 0; i < N; i++)
     search_function()
int timeEnd  // end measuring
 executionTime = (endTime - startTime) / N
```

Sample runs

Sample Run 1

```
Please enter the contact file name:
PhoneBook-sample2.txt
Please enter the word to be queried :
Shane
```

```
Sorting the vector copies
=====================================
Quick Sort Time: 267392 Nanoseconds
Insertion Sort Time: 1485445 Nanoseconds
Merge Sort Time: 1375031 Nanoseconds
Heap Sort Time: 518291 Nanoseconds

Searching for Shane
=====================================
The search in BST took 20098 Nanoseconds
The search in AVL took 9647 Nanoseconds
Binary Search Time: 16210 Nanoseconds

SpeedUps in search
=====================================
(BST / AVL) SpeedUp  = 2.08334
(Binary Search / AVL) SpeedUp  = 1.68032
(Binary Search / BST) SpeedUp  = 0.806548

SpeedUps between Sorting Algorithms
=====================================
(Insertion Sort/ Quick Sort) SpeedUp = 5.55531
(Merge Sort / Quick Sort) SpeedUp = 5.14238
(Heap Sort / Quick Sort) SpeedUp = 1.93832
```

Sample Run 2

```
Please enter the contact file name:
PhoneBook-sample1.txt
Please enter the word to be queried :
Char

Sorting the vector copies
=====================================
Quick Sort Time: 26376 Nanoseconds
```

```
Insertion Sort Time: 44829 Nanoseconds
Merge Sort Time: 42555 Nanoseconds
Heap Sort Time: 41510 Nanoseconds

Searching for Char
====================
The search in BST took 4635 Nanoseconds
The search in AVL took 5161 Nanoseconds
Binary Search Time: 10613 Nanoseconds

SpeedUps in search
=====================================
(BST / AVL) SpeedUp  = 0.898082
(Binary Search / AVL) SpeedUp  = 2.05638
(Binary Search / BST) SpeedUp  = 2.28975

SpeedUps between Sorting Algorithms
=====================================
(Insertion Sort/ Quick Sort) SpeedUp = 1.69961
(Merge Sort / Quick Sort) SpeedUp = 1.6134
(Heap Sort / Quick Sort) SpeedUp = 1.57378
```

Sample Run 3

```
Please enter the contact file name:
PhoneBook-sample2.txt
Please enter the word to be queried :
Tina Gulko

Sorting the vector copies
=====================================
Quick Sort Time: 93656 Nanoseconds
Insertion Sort Time: 512859 Nanoseconds
```

```
Merge Sort Time: 489748 Nanoseconds
Heap Sort Time: 202962 Nanoseconds

Searching for Tina
====================
The search in BST took 1260 Nanoseconds
The search in AVL took 1193 Nanoseconds
Binary Search Time: 3889 Nanoseconds

SpeedUps in search
=====================================
(BST / AVL) SpeedUp  = 1.05616
(Binary Search / AVL) SpeedUp  = 3.25985
(Binary Search / BST) SpeedUp  = 3.08651

SpeedUps between Sorting Algorithms
=====================================
(Insertion Sort/ Quick Sort) SpeedUp = 5.47599
(Merge Sort / Quick Sort) SpeedUp = 5.22922
(Heap Sort / Quick Sort) SpeedUp = 2.1671
```

Sample Run 4

```
Please enter the contact file name:
PhoneBook-sample3.txt
Please enter the word to be queried :
Gulsen Demiroz

Sorting the vector copies
=====================================
Quick Sort Time: 135189 Nano secs
Insertion Sort Time: 435134 Nano secs
Merge Sort Time: 449873 Nano secs
```

```
Heap Sort Time: 235754 Nano secs

Searching for Gulsen
====================
The search in BST took 2665 Nanoseconds
The search in AVL took 2730 Nanoseconds
Binary Search Time: 9550 Nanoseconds

SpeedUps in search
======================================
(BST / AVL) SpeedUp  = 0.97619
(Binary Search / AVL) SpeedUp  = 3.49817
(Binary Search / BST) SpeedUp  = 3.58349

SpeedUps between Sorting Algorithms
====================================
(Insertion Sort/ Quick Sort) SpeedUp = 3.21871
(Merge Sort / Quick Sort) SpeedUp = 3.32773
(Heap Sort / Quick Sort) SpeedUp = 1.74388
```

Sample Run 5

```
Please enter the contact file name:
PhoneBook-shuffled.txt
Please enter the word to be queried :
Virg

Sorting the vector copies
======================================
Quick Sort Time: 1666571 Nanoseconds
Insertion Sort Time: 71955853 Nanoseconds
Merge Sort Time: 58151770 Nanoseconds
Heap Sort Time: 2802353 Nanoseconds
```

```
Searching for Virg
=====================================
The search in BST took 14232 Nanoseconds
The search in AVL took 6898 Nanoseconds
Binary Search Time: 6175 Nanoseconds

SpeedUps in search
=====================================
(BST / AVL) SpeedUp  = 2.06321
(Binary Search / AVL) SpeedUp  = 0.895187
(Binary Search / BST) SpeedUp  = 0.433881

SpeedUps between Sorting Algorithms
=====================================
(Insertion Sort/ Quick Sort) SpeedUp = 43.176
(Merge Sort / Quick Sort) SpeedUp = 34.8931
(Heap Sort / Quick Sort) SpeedUp = 1.68151
```

Sample run 6

```
Please enter the contact file name:
PhoneBook-shuffled.txt
Please enter the word to be queried :
James

Sorting the vector copies
=====================================
Quick Sort Time: 1678394 Nanoseconds
Insertion Sort Time: 71820330 Nanoseconds
Merge Sort Time: 57941282 Nanoseconds
Heap Sort Time: 2779508 Nanoseconds
```

```
Searching for James
====================================
The search in BST took 26988 Nanoseconds
The search in AVL took 21170 Nanoseconds
Binary Search Time: 5809 Nanoseconds

SpeedUps in search
====================================
(BST / AVL) SpeedUp  = 1.27482
(Binary Search / AVL) SpeedUp  = 0.274398
(Binary Search / BST) SpeedUp  = 0.215244

SpeedUps between Sorting Algorithms
====================================
(Insertion Sort/ Quick Sort) SpeedUp = 42.7911
(Merge Sort / Quick Sort) SpeedUp = 34.5219
(Heap Sort / Quick Sort) SpeedUp = 1.65605
```

General Rules and Guidelines about Homeworks

The following rules and guidelines will be applicable to all homeworks, unless otherwise noted.

How to get help?

You may ask questions to TAs (Teaching Assistants) of CS300. Office hours of TAs can be found here. Recitations will partially be dedicated to clarify the issues related to homework, so it is to your benefit to attend recitations.

What and Where to Submit

Please see the detailed instructions below/in the next page. The submission steps will get natural/easy for later homeworks.

Grading

Careful about the semi-automatic grading: Your programs will be graded using a semi-automated system. Therefore, you should follow the guidelines about input and output order; moreover, you should also use

the exact same prompts as given in the Sample Runs. Otherwise semi-automated grading process will fail for your homework, and you may get a zero, or in the best scenario you will lose points.

Grading:

- **We will hold a demo session for your homework grading. Each one of you must show that your code is running as expected and explain several parts of your code if necessary. Wait for an announcement for the demo scheduling.**
- Late penalty is 10% off the full grade and only one late day is allowed.
- **Having a correct program is necessary, but not sufficient to get the full grade. Comments, indentation, meaningful and understandable identifier names, informative introduction and prompts, and especially proper use of required functions, unnecessarily long program (which is bad) and unnecessary code duplications will also affect your grade.**
- Please submit your own work only (even if it is not working). It is really easy to find out "similar" programs!
- For detailed rules and course policy on plagiarism, please check out http://myweb.sabanciuniv.edu/gulsend/courses/cs201/plagiarism/

Plagiarism will not be tolerated!

Grade announcements: Grades will be posted in SUCourse, and you will get an Announcement at the same time. You will find the grading policy and test cases in that announcement.

Grade objections: Since we will grade your homeworks with a demo session, there will be very likely no further objection to your grade once determined during the demo.

What and where to submit (IMPORTANT)

Submission guidelines are below. Most parts of the grading process are automatic. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.

Add your name to the program: It is a good practice to write your name and last name somewhere in the beginning program (as a comment line of course).

Name your submission file:

- Use only English alphabet letters, digits or underscore in the file names. Do not use blank, Turkish characters or any other special symbols or characters.
- Name your cpp file that contains your program as follows.

  "SUCourseUserName_yourLastname_yourName_HWnumber.cpp"
- Your SUCourse user name is actually your SUNet username which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be:

  cago_ozbugsizkodyazaroglu_caglayan_hw5.cpp
- Do not add any other character or phrase to the file name.
- Make sure that this file is the latest version of your homework program.
- You need to submit ALL .cpp and .h files including the data structure files in addition to your main.cpp in your VS solution.
- The name of the main cpp file should be as follows.

"SUCourseUserName_yourLastname_yourName_HWnumber.cpp"

For example zubosman_Osmanoglu_Zubeyir_hw5.cpp is a valid name, but hw5_hoz_HasanOz.cpp, HasanOzHoz.cpp are NOT valid names.

Submission:

Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!
Anes Abdennebi and Gülşen Demiröz