CS214: Data Structures 2018
Assignment (2)
[140 points]

Deadline & Submission

1. At least one team member should submit the compressed group solution as zip file containing the programs under acadox –> tasks (name your assignment file **“A2_ID1_ID2_G#_G#.zip”**). eg. A2_20168383_201638838_G1_G1.zip
2. The deadline for submitting the electronic solution of this assignment is
25/3/2018

About this assignment

1. This assignment will be solved in teams of 2 except those who got exceptions from the same lab.
2. The weight of the assignment is 140 points
3. All team members should understand all assignment problems.
4. All code must be standard ANSI C++.
5. Assume any missing details and search for information yourself.
6. Any cheating in any part of the assignment is the responsibility of the whole team and whole team will be punished.

Problems

Problem 1: (20 Point)

Write a program that processes student information (objects of class student). Each student has name, age and grade.
- Your program should have a menu of three options:

- Add student
- Search student by name
- List students in alphabetical order
- Update student grade (user enters student name and the new grade)
- Find the highest grade student and prints student information (name, age,
grade).

Use the **most suitable STL containers** (you may use more than one container) **and STL algorithms.** Organize your code into classes and functions. Use class header files and class cpp files.

Name a folder “**A2_P1_ID1_ID2**” and put your files inside it (even if it’s only one file)

Problem 2: (40 Point)

In this problem, you will measure the average of binary search algorithm in terms of the number of comparisons it does to find a word. Your task is as follows:

1- Implement a generic class called searcher that approximately has the following interface:

- **loadData (....)** // Loads required num of words from file
- **int binarySearch (....)** // Looks for a given item in the data & return its index or -1
- **int testPerformance (..)** // Gets the time & num of comparisons

  *** Add any missing functions**

2- Preferably, implement binary search algorithm to work on vectors not arrays. It should (1) calculate the time taken to search for a given word and (2) the number of comparisons it did.

3- We want to measure the (1) time and (2) number of comparisons in two cases using **testPerformance** and the given English list of words:

- First when the word is found. For this case, pick a random word (use random function C++ to pick an index between 0 and last index) and then search for it in the data.
- Do this 100 times and calculate the average time and average number of comparison.
- Second, makeup a random non-existing word and search for it and calculate the time and number of comparisons done until algorithm returns that word is not found.
- Do this 100 times and calculate the average time and average number of comparison.

4- Repeat the previous step using a file of 10000, 20000, 30000, ..., 80000 words and draw the results on a plot using excel or any drawing tool.

* There is a list of English words to use at
http://www-01.sil.org/linguistics/wordlists/english/

Name a folder "**A2_P2_ID1_ID2**" and put your files inside it (even if it's only one file)

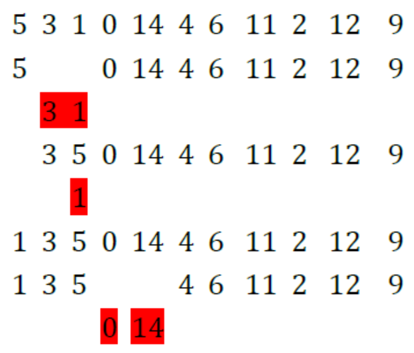
Problem 3: (20 Point)

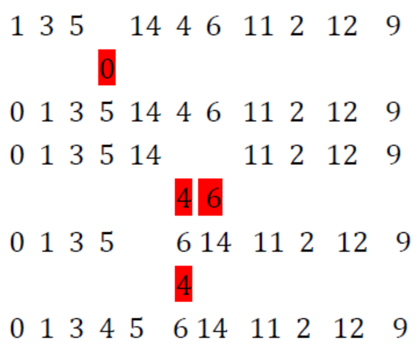
Insertion sort inserts one item at a time. Will it enhance performance to insert two items together? Your task is to find out the effect of this improvement.

In this modification, we pick two items at a time. Then we look for the right place of the largest item, shifting items by two places at a time. After putting the largest item in its place, we keep shifting one item at a time till finding the right place for the smallest item.

Plot the performance of the algorithm **against the original** insertion sort.

Name a folder “**A2_P3_ID1_ID2**” and put your files inside it (even if it’s only one file)





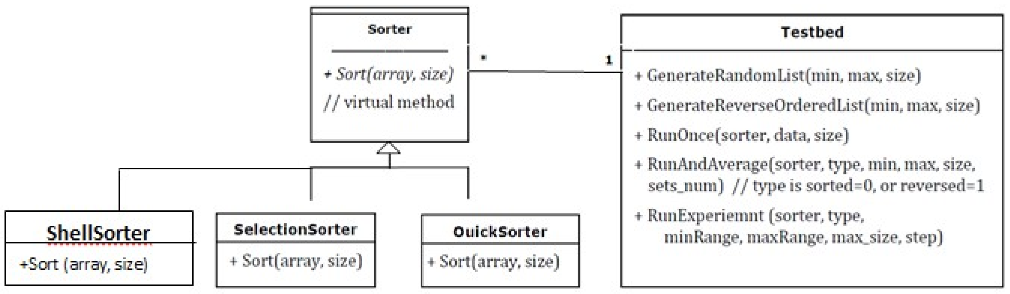
Problem 4: (20 Point)

Insertion sort uses linear search to find the right place for the next item to insert. Would it be faster to find the place using binary search (reduce number of comparisons)? We still have to shift 1 item at a time from the largest till the right place.
Use binary search on the already sorted items to find the place where the new element should go and then shift the exact number of items that need to be shifted and placing the new item in its place. The algorithm works the same, except that instead comparing and shifting item by item, it will compare quickly using binary search but it will still shift one by one till the right place (without comparison).

1) **Plot the performance** of the algorithm **against the original** insertion sort.
2) Name a folder “**A2_P4_ID1_ID2**” and put your files inside it (even if it’s only one file)

Problem 5: (40 Point)

In this problem, we will develop classes to use for testing three sorting algorithms (Selection, Quick and Shell sort). Sorter Class should sort any data type (Generic). The class will have methods to support experimenting and analyzing sorting algorithms performance. Below is a high level UML diagram for your classes. Add any missing details.


Sorter
+ Sort(array, size)
// virtual method
*
1
Testbed
+ GenerateRandomList(min, max, size)
+ GenerateReverseOrderedList(min, max, size)
+ RunOnce(sorter, data, size)
+ RunAndAverage(sorter, type, min, max, size, sets_num) // type is sorted=0, or reversed=1
+ RunExperiemnt (sorter, type, minRange, maxRange, max_size, step)
ShellSorter
+Sort (array, size)
SelectionSorter
+ Sort(array, size)
QuickSorter
+ Sort(array, size)

Testbed class has the following functions that you should complete:

2. **GenerateRandomList(min, max, size)** Generate a given number of random integer data from a certain range. For example, one can generate a vector/array of 10000 integer numbers that fall in the range from 1 to 100000, e.g., [5554, 32300, 98000, 342, …]
3. **GenerateReverseOrderedList(min, max, size)** Generate a given number of reverse ordered integer data from a certain range. You can first generate random data and then sort them reversed
4. **RunOnce(sorter, data, size)** Run a given sorting algorithm on a given set of data and calculate the time taken to sort the data
5. **RunAndAverage(sorter, type, min, max, size, sets_num)** Run a given sorting algorithm on several sets of data of the same length and same attributes (from the same range and equally sorted; e.g., random or reversed) and calculate the average time
6. **RunExperient (sorter, type, min, max, min_val, max_val, sets_num, step)** Develop an experiment to run a given sorting algorithm and calculate its performance on sets of different sizes (e.g., data of size 10000, 20000, etc.) as follows:
   i) All sets are generated with values between min and max
   ii) First, generate **sets_num** sets with size **min_val**. Use **RunAndAverage** () and record average time to process the sets
   iii) Next, repeat step ii but with sets whose size increases by step till reaching **max_val**. Each time record average time to process the sets
   iv) For example I should be able to design an experiment to run Quick sort algorithm on randomly sorted integers data taken from the range (1 to 1,100,000) and with input value (data size) from 0 to 100000, with step 5000. This means we will run the algorithms on data sets of 5000, 10000, 15000, …, 100000 randomly sorted integers. Note that with each step you will generate **sets_num** different sets and take the average of their runs

v) The output of the experiment goes to screen as a table with two columns; first column indicates set size, and second column indicates average time

Write a **main ()**demo to show that the function works correctly and to measure the performance of Quick sort, Selection sort and Shell sort in cases of random data and reverse ordered data using **Testbed** class. **Draw plots of your results**.

Name a folder “**A2_P5_ID1_ID2**” and put your files inside it (even if it’s only one file)

Notes: In problems 3 ,4 and 5 requires you to plot your results which means you provide chart in excel document with a table of the numbers you obtained from the experiment.
The plot will have running time (in sec or msec) which is represented on y-axis and size of the data on x-axis. Array size rages from 100 to 1000000 or more.

Rules:

1. Cheating will be punished by giving -2 * assignment mark.
2. Cheating is submitting code or report taken from any source that you did not fully write yourself (from the net, from a book, from a colleague, etc.)
3. Giving your code to others is also considered cheating both the giver and the taker.
4. People are encouraged to help others fix their code but cannot give them their own code.
5. Do not say we solved it together and we understand it. You can write the algorithm on paper together but each group should implement it alone.
6. If you do not follow the delivery style (time and files names), your assignment will be rejected.