CS214: Assignment #2 - 2019

Data Structure Course

(140 marks)

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"). e.g. A2_20168383_201638838_G2_G1.zip

2. The deadline for submitting the electronic solution of this assignment is 21/3/2019

About this Assignment

1. This assignment will be solved in teams of 2.
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 FCI library books information (objects of class book).
Each book has name, author, publish year and number of available versions.
- Your program should have a menu of these options:

1. Add book
2. Search book by name
3. Search book by author
4. List books in alphabetical order (Ascending)
5. Update available number of versions of a book (user enters book name and the new number)
6. Find the books which have the highest number of versions and print books information.

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: (20 point)

Use the **map<string, int>** STL container to count occurrences of words.
Use the word as the key and the count as the value.

- User should be able to enter words as desired until the word "done" is written.
- After entering all words, your program should print each word with its count.
- **Then,** erase all words that start with letter 'a' and print again.

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

Problem 3: (40 point)

In this problem, you will measure the complexity of binary search algorithm practically And see if it matches the theoretical result of $O(\log n)$. Complexity will be measured by 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 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
   - List of words to use is at http://www-01.sil.org/linguistics/wordlists/english/

5- **Draw the results on a plot using excel or any drawing tool**.

Name a folder "**A2_P3_ID1_ID2**" and put your files inside it (even if it's only one file).

Problem 4: (20 point)

Given a vector of integers that represent sides of triangles, we need to find is it possible to construct at least one non-degenerate triangle using this array?
Sides of non-degenerate triangle should follow these constraints:
S1 + S2 > S3 and
S2 + S3 > S1 and
S3 + S1 > S2
Where S1, S2, S3 are the length sides of triangle. Use any STL containers or algorithm needed.

For example:
4, 1, 2 cannot construct a non-degenerate triangle.
But 5, 4, 3, 1, 2 can generate with sides 2, 3, 4 a non-degenerate triangle.

Name a folder "**A2_P4_ID1_ID2**" and put your files inside it (even if it's only one file)

Problem 5: (20 point)

We know that both Merge and Quick sorts are O (n log n) in average case. But Quick sort is faster. We like to compare their performance and study how faster Quick sort is. Is it two times faster? Or three times faster? Or what? You task will be:

1. Implement quick and merge sort algorithms given in the lecture or book. Understand them very well. Choose a good technique for choosing the pivot of Quick sort.
2. Implement a program that generates arrays (or better vector) of random integer data of size 5,000 items, 10,000 items, etc till 100,000 items.
3. For each data array, run both algorithms and record the time it takes in sec or msec.
4. Plot the performance of the algorithms against each other on the same graph.

Name a folder "**A2_P5_ID1_ID2**" and put your files inside it (even if it's only one file)

Problem 6: (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).
Plot the performance of the algorithm **against the original** insertion sort.

Name a folder "**A2_P6_ID1_ID2**" and put your files inside it (even if it's only one file)

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 reject