Practical 8: Sorting and Searching

Due date: 10:00am, 9 October 2017

1 General Instructions

All submissions for the practical assignments should be under version control. Submission procedure remains the same with the first practical assignment.

The directory under version control for this assignment should be named as

```
https://version-control.adelaide.edu.au/svn/aXXXXXXX/2017/s1/adds/assignment8/
```

where aXXXXXXX is your student ID.

If you get stuck on one of the hidden test cases and really cannot resolve it yourself, please feel free to ask the practical tutors for hints.

We encourage you to finish your work before the practical session and take the session as consulting time.

2 Problem Description

2.1 Objective

This practical will test your knowledge on sorting and searching algorithms. In this practical, you will be implementing a number of algorithms. Each of these algorithms will be constructed in a different class. You should make sure that you know the complexity of the algorithms you implement.

2.2 Design

In a file named `design.pdf`, describe how you are going to solve the problem and test your implementation with the test cases you designed based on the stages below.

Please make sure you submit your design document in your first submission. Failing to do so will result in 0 mark for the design phase.

Testing Hint: it's easier if you test things as a small piece of code, rather than building a giant lump of code that doesn't compile or run correctly. As part of your design, you should also sketch out how you are going to build and test the code.

2.3 Sorting

Implement a base class called `Sort`. All other sorting algorithms must be derived from this class. You should decide how you are going to store the data that is going to be sorted or worked with as part of your sorting process. It is up to you to decide the storage mechanism.

Step 1: Implement bubble sort in a class called `BubbleSort` that extends `Sort`.

`http://en.wikipedia.org/wiki/Bubble_sort`

Step 2: Implement quick sort in a class called `QuickSort` that extends `Sort`.

`http://en.wikipedia.org/wiki/Quicksort`

As a requirement, for the pivot selection, when the list is of length at least 2, please always chose the second value in the list (e.g., if the (sub)list is formed by 4 elements 2, 4, 9, 1, then the pivot is 4).

Optional challenge: Can you implement quick sort without using recursion?

2.3.1 Searching

Implement **recursive** binary search in a class called `RecursiveBinarySearch`. Binary search was discussed in Lecture 8-1, but the demonstration presented an iterative algorithm. In recursive binary search, the parameter list should include a list of elements to be sorted, a starting point and an ending point. When we search, we examine the middle element of our array. If we find the element we are looking for, return the index of the element. If the element we are looking for has value less than the middle element, call binary search with the same starting point and an ending point one position less than the current middle index. If the element we are looking for has a value greater than the middle element, call binary search with the same ending point and a starting point one greater than the current middle index.

2.4 Main function

The test script will compile your code using `g++ -o main.out -std=c++11 -O2 -Wall *.cpp`. It is your responsibility to ensure that your code compiles on the university system. g++ has too many versions, so being able to compile on your laptop does not guarantee that it compiles on the university system. You are encouraged to debug your code on a lab computer (or use SSH).

Create a main function that takes in a sorting algorithm indicator and a list of integers (one line, separated by space) int1 int2 int3 ... intn. The sorting algorithm indicator is a character. The value 'B' indicates using Bubble sort and 'Q' indicates using Quick sort.

In the main function, the list should be sorted in ascending order first using certain searching algorithm, then binary search is used to determine whether 6 belongs to the list or not. Your output should start with either "true" (6 belongs to the list) or "false", and then followed by the sorted list. Please separate the results using space.

Sample input: B 1 3 5 4 -5 100 7777 2014

Sample output: false -5 1 3 4 5 100 2014 7777

Sample input: Q 0 3 6 4 -5 100 7777 2014

Sample output: true -5 0 3 4 6 100 2014 7777

3 Marking Scheme

No late submissions will be accepted.

Your submission should contain at least `design.pdf`, `main.cpp` and other cpp files and header files. The design document should be submitted in the first submission to the websubmission system.

This practical is worth 3% of your final mark.

Every practical assignment is marked out of 6.

- Design (2 marks):
  - UML diagram of central classes and explanation of core functions (1 mark)
  - Details of your own test cases/schemes (1 mark)
- Style (2 marks):
  - Proper usage of C++ language elements and paradigm (1 mark)
  - Comments on non-trivial code blocks and functions (1 mark)
- Functionality (2 marks):
  - Passing public test cases (1 mark)
  - Passing hidden test cases (1 mark)

Your final mark for this assignment will be based on your latest submission in the web-submission system before the deadline.

The functionality marks are awarded based on the marks given by the web submission system and a manual check on specific requirement. Please take care of the warnings from the compilers. Missing required functionality will cause deduction of marks.