Programming Hand in 3

R. Brooks

For this and the remaining handins you must work in your semester project group.

The objective of this hand in is to make you acquainted with sorting algorithms in Java. This topic was covered primarily in Lecture 10.

You do not need to implement your own sorting algorithm in any of the exercises but are allowed to use Java's standard library, i.e. you are allowed to use `Collections.sort();` to sort the input list (or any list you may create as part of your solution).

For each of the below exercises there is an associated template. You can find all templates in the zip file 'dmaProg3.zip'. Create a new project in IntelliJ for all your DMA programming handins. It is up to you how you want to structure your hand ins but it may be a good idea to have a module for each of the four hand ins. Unpack the zip file to the newly created project folder.

Do not change any of the existing code, e.g. do not rename methods, etc., although you may need to change the reference to the package (first line in templates). You will only need to add the body of your methods in the relevant template. For each of the programs, replace the comment with your code. Note, you do not need to create methods for user inputs. All you really need to do is to write the logic of the methods, i.e. the algorithms.

Each exercise is accompanied by a test class. You can use this to test whether your method works correctly. The scoring is stated explicitly in each exercise. If an exercise awards extra points for complexity analysis, these are made in the code as per the example shown in class and which is similar to what you must do in your SEP1 project. Naturally, you only need to analyse the code you have written so you do not need to comment on existing code in the templates. You can at most obtain 11 points in this hand in which means you need 6 points to get it approved.

You upload your hand in as a zip-file where you simply zip the handin3 module with all your code/files.

If you get stuck doing any of the exercises, it is recommended seeking out the student instructor at the study café.

Program 1: `findClosest`

In this problem, the input to your program is an arraylist of N integers, and your program must output the smallest difference between two numbers in the input. For instance, if the input array contains these ten elements

```
<< 31, -41, 57, 26, -53, 59, 97, -93, -23, 84 >>
```

then your program should return the difference 2, i.e. $59 - 57 = 2$.

Concretely, you must implement a public method named `findClosest` that takes an `ArrayList<Integer> input` as argument and returns an `int`. Use the template **closest.java**.

Scoring:

- 1 point for correct algorithm
- 1 extra point for correct and fast algorithm
- 1 point for correct algorithm analysis

An $\mathcal{O}(N \log N)$ solution is fast enough to get the extra point.

Hint:
You are allowed to use `Collections.sort();` to sort the input list (or any list you may create as part of your solution). That is, after running `Collections.sort(input);` in `findClosest()`, the input list is sorted into ascending order. This might be useful to you when solving the exercise. Note, in your analysis you can assume that `Collections.sort(input);` takes $\mathcal{O}(N \log N)$.

Remember to have `import java.util.*;` in the beginning of your solution in order to be able to use `Collections.sort()` (which is defined in `java.util`). It is imported as default in the template.

Program 2: `closestBall`

This problem is based on a similar problem from Aarhus University.

Dodgeball Madness is a game in which two teams of N players play against each other using M balls. It is your team's turn to play, and you must quickly determine which of the N players on your team is closest to one of the M balls you can throw, since the faster your team gets the first throw in, the more likely it is that you will hit one of the players on the other team!

Since all players and all balls are currently on a straight line, the distance between a player at position x and a ball at position y is simply $|x - y|$, that is, the absolute value of the difference between the two numbers x and y.

Your task is to write a program that takes the positions of the N players and the M balls and outputs the shortest distance between a player and a ball.

For instance, if the players are located at positions

```
<< 95, 66, 82, 63, 17 >>
```

and the balls at positions

```
<< 75, 38, 25, 77 >>
```

then your program should output 5, i.e. $|82 - 77| = 5$.

Concretely, you must implement a public method named `computeClosest` that takes an `ArrayList<Integer> players` and an `ArrayList<Integer> balls` as arguments and returns an `int`. Use the template **closestBall.java**.

Scoring:

- 1 point for correct algorithm
- 1 extra point for correct and fast algorithm
- 1 point for correct algorithm analysis

An $\mathcal{O}(N \log N + M \log M)$ solution is fast enough to get the extra point.

Hint: For the fast solution, you're allowed to use `Collections.sort()` just like in the `findClosest` program.

Program 3: `maxSubSum`

This problem is based on Bentley Chp. 8.

In this problem, the input to your program is an array of N integers, and your program must output the maximum sum found in any *contiguous* subarray of the input. For instance, if the input array contains these ten elements

```
<<31, -41, 57, 26, -53, 59, 97, -93, -23, 84>>
```

then your program should return the sum $186 = 57 + 26 + (-53) + 59 + 97$. (For convenience, the maximum sum is 0 if all the integers are negative.)

Concretely, you must implement a public method named `findMaxSubSum` that takes an `ArrayList<Integer> input` as argument and returns an `int`. Use the template **maxSubSum.java**.

Scoring:

- 1 point for correct algorithm
- 1 extra point for correct and fast algorithm
- 1 extra point for correct algorithm analysis

A linear solution is fast enough to get the extra point.

Program 4: `Inversions`

Note: This exercises is a bit of challenge exercise. If you find it too troublesome just skip it.

Let $A[1..N]$ be an array of N integers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an inversion of A. In this task you must implement an algorithm to count the inversions in an array of N integers from 1 to N.

For instance, the list

```
1, 4, 2, 3
```

contains two inversions, namely (2, 3) and (2, 4), since $A[2] > A[3]$ and $A[2] > A[4]$.

The list

```
5, 4, 3, 2, 1
```

has 10 inversions, namely (1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (3, 5), (4, 5).

Concretely, you should use the template `Inversions.java` and implement a public method named `countInversions` that takes an `ArrayList<Integer> input` as argument and returns an `int`.

Input constraints:

- $1 \leq N \leq 65536$
- Each element is between 1 and N
- Each number between 1 and N occurs exactly once in the list, i.e. there are no duplicate elements
- The number of inversions fits in a Java `int` without overflowing

Scoring:

- 1 point for correct algorithm
- 1 extra point for correct and fast algorithm

An $O(N \log N)$ algorithm is fast enough for the extra point.