COP 3503C Programming Assignment # 1

Read all the pages before starting to write your code

As we learned from the sorted list matching problem that a problem can be solved in various ways and being clever can improve the performance of an algorithm a lot. In this problem you also need to come up with a solution that can run within a given run-time.

What should you submit?

Write all the code in a single Main.java file. **Submit your** Main.java and **analysis.txt file to Codegrade platform. The content of the analysis.txt file will be discussed at the last part of the assignment description.**

Please include the following commented lines in the beginning of your code to declare your authorship of the code:

```
/* COP 3503C Assignment 1
This program is written by: Your Full Name */
```

Compliance with Rules: UCF Golden rules apply towards this assignment and submission. Assignment rules mentioned in syllabus, are also applied in this submission. The TA and Instructor can call any students for explaining any part of the code in order to better assess your authorship and for further clarification if needed.

Caution!!!

Sharing this assignment description (fully or partly) as well as your code (fully or partly) to anyone/anywhere is a violation of the policy. I may report to office of student conduct and an investigation can easily trace the student who shared/posted it. Also, getting a part of code from anywhere will be considered as cheating.

Deadline:

See the deadline in Webcourses. The assignment submission will be locked after the deadline. An assignment submitted by email will not be graded and such emails will not be replied according to the course policy.

What to do if you need clarification on the problem?

I will create a discussion thread in webcourses and I highly encourage you to ask your question in the discussion board. Maybe many students might have same question as you. Also, other students can reply, and you might get your answer faster. Also, you can write an email to the TAs and put the course teacher in the cc for clarification on the requirements.

How to get help if you are stuck?

According to the course policy, all the helps should be taken during office hours. Occasionally, we might reply in email.

Problem: Help Gustavo to choose the games!

Gustavo got a Knights arcade game card with T points in it. Knights arcade has n unique games where each game requires a unique number of points to play. Gustavo is allowed to play only two games and he wants to spend the full T points to play two games so that no points will remain in the card. So, he needs to decide whether is it possible or not to play two games that will cost exactly T points. Given an array of n distinct none zero values representing the points needed to play the games. Also, given the target points T available on the card. Determine in *O(n) time* whether or not there exist two distinct points in the array that sum to T. The given array maybe sorted or may not be already sorted. It will be specified in the input whether it is sorted or not and you have to fulfill specific requirement based on the sorted status. (For example, if the array contained 3, 5, 6, 7, and 9 and T = 14, then the method you are to write should return points pair (5,9), since 5+9 = 14. It should return points pair (0,0) for the same array of points if T = 17.)

Input Format (Your code must read from standard input (no file i/o is allowed))

The first line of the input will have a single positive integer k, representing the number of test cases in the inputs. The next 2*k lines will contain the test cases, with two lines being used for each test case. The first value on the first line of each test case will be the sorted status (0 means unsorted, 1 means sorted). The next number in the line is n, the size of the array (the number of games in the Knights arcade). The rest of the line will contain n distinct none zero integers representing the points needed to play, each separated by spaces. The second line of each test case will contain a single integer, T, the target for the problem (The number of points in the game card). Here's an example input contents:

```
4
1 5 3 5 6 7 9
14
0 3 1 6 3
11
0 6 4 1 -8 6 45 10
16
1 6 -8 1 4 6 10 45
16
```

Output Format

For each test case, output a line with one of the following two formats:

```
Test case #m: Spend X points by playing the games with n1 points and n2 points.

Test case #m: No way you can spend exactly X points.
```

where m (1 ≤ m ≤ k), represents the appropriate test case. In the output n1 also must be less than n2

Example output for the above inputs:

```
Test case#1: Spend 14 points by playing the games with 5 points and 9 points.

Test case#2: No way you can spend exactly 11 points.

Test case#3: Spend 16 points by playing the games with 6 points and 10 points.

Test case#4: Spend 16 points by playing the games with 6 points and 10 points.
```

Specific Restrictions:

Your code must incorporate the following restrictions to receive full credit:

1. **If you see that a particular test case is sorted (sorted status = 1), your code must process it with O(n) operation to receive more than 50% credit on this part. For this part of the implementation, you are not allowed to use Hashset.**
2. **For finding the pair in the sorted array, you must implement the following function:**
   a. `_________ getCandidatePair(int A[], int target)`: This function receives an int array and the target and returns the pair of ints from the array that can addup to target. For this purpose, you may consider having another class and return an object of that class. If a pair does not exist, the function should return the pair as (0, 0). **Note that you are not allowed to print any information in this function as part of the output.**
3. **If you see that a particular test case is not sorted (sorted status = 0), your code must process it with O(n) operation to receive more than 75% credit on this part. If your code works with O(n log n), you can get up to 75% on this part. Note that Arrays.sort() method sometimes can result in O(n^2). The Collections.sort() method could be a good idea to guarantee O(n log n) sorting. If you would like to get 100% credit for this part and get O(n) run-time, then Hashset would be the best option (Note that HashSet insert and lookup O(1)). If you are unable to find a solution within even O(n log n), then at least solve it with O(n^2) to receive some partial credit.**

Submission:

Submit the following files in codegrade. Make sure your code work in codegrade platform.

1. **Main.java file that process your file**
2. **analysis.txt file: In this file you will explain the run-time of your algorithm for the situation of sorted array and unsorted array. Please have two paragraphs, one for each situation. Mainly explain why do you think it is O(n) or O(n log n) or O(n^2). Don't try to argue that your code process this in O(n), if it is not the case!**

Rubric (subject to change):

According to the Syllabus, the code will be compiled and tested in Codegrade Platform for grading. If your code does not compile in Codegrade, we conclude that your code is not compiling and it will be graded accordingly. We will apply a set of test cases to check whether your code can produce the expected output or not. Failing each test case will reduce some grade based on the rubric given bellow. If you hardcode the output, you will get -200% for the assignment.

1. If a code does not compile, the code may get 0. However, some partial credit maybe awarded. A code having compiler error cannot get more than 35% even most of the codes are correct.
2. Passing test cases (exact output format): 50%
3. Fulfilling the O(n) requirements for sorted cases (if passed test cases): 21%
4. Fulfilling requirements for unsorted cases (if passed test cases): 21%
   a. O(n) can get full credit 21/21
   b. O(n log n) can get 16 out of 21
   c. O(n^2) gets points only for passing test cases (rubric#2 above)
5. Analysis file with correct content: 8 pts

Penalty:

6. Note that there can be penalty if you don't fulfill the function requirement: -8%
7. Not putting header comment: -10%
8. Poor indentation in code and spacing in code: -10%

9. Not putting proper comments on important block of code: -5% (don't start commenting every line. Only important block of code)
10. If you use file i/o (if you do not read from standard input): -100%

Some hints:
Remember we have used two trackers for SLMP?

Good Luck!