Practical Task 6.2

(Distinction Task)

Submission deadline: 11:59 pm Sunday, September 13th
Discussion deadline: 11:59 pm Sunday, September 27th

Task Objective

In this task you will design and implement an algorithms that solve the problem faced by Alex in the following scenario.

Background

Alex and Cindy, two SIT221 students whom you may be familiar with, recently spent some time on treasure hunting. Apart from scrap metal, they found a number of boxes full of old coins. Boxes are of different value and now are lined up in a row. Cindy proposes an idea to divide the treasure into two parts. She thinks that a fair way is that she and Alex take turns, and each of them chooses one box from either left or right side of the line. Cindy is a very generous person and lets Alex to go first.

Alex wants to check whether this idea is actually good for him. He asks you to write a program to calculate the total value that he will get compared to how much Cindy will get if he chooses a box first. You can be sure that they both have a beautiful mind, and always select the next box in such way that it leads to the best overall individual solution for them.

Task Details

Your task is to implement an algorithm to solve the problem

1. Download the source code attached to this task. Create a new Microsoft Visual Studio project and import the three files. Your newly built project should compile without errors. Note that the file Tester.cs already provides you with a Main method as a starting point of your program and is important for the purpose of debugging and testing. Another file, Generator.cs, serves the Tester class in Tester.cs with hard-coded test instances. Finally, BoxOfCoins.cs contains a template for the algorithm that you need to develop and implement as part of the BoxOfCoins class.
2. Explore the existing BoxOfCoins class to find the static method named Solve. This method must implement your algorithm. It accepts just a single argument: an array of the values of all the boxes of coins in the order in which they are lined up. The method must return how much more reward Alex will get than Cindy if they both behave in the optimal way. Alex fears that the answer might be a negative number since Cindy proposed the plan. You may assume that they found between 1 and 50 boxes of coins inclusive. Each box has value between 1 and 100,000 inclusive.
3. Consider the following examples:
   - For a given array of coin boxes [7, 2] the algorithm must return 5. Alex will choose the 7, and then Cindy gets the 2. So the result is 7 - 2 = 5.
   - For a given array of coin boxes [2, 7, 3] the algorithm must return -2. It doesn't matter whether Alex chooses the 2 or the 3. Cindy will choose the 7 and Alex will get the remaining box. (2+3) - 7 = -2.

- For a given array of coin boxes [1000, 1000, 1000, 1000, 1000] the algorithm must return 1000. Since Alex chooses first, he will get 3 boxes and Cindy will get only 2 boxes. They all have the same value so (1000+1000+1000) - (1000+1000) = 1000.
- For a given array of coin boxes [823, 912, 345, 100000, 867, 222, 991, 3, 40000] the algorithm must return -58111.

4. Remember that you are free to write your program code within the BoxOfCoins.cs as you wish. The only requirement is to meet the expected signature of the Solve method. Therefore, you may add any extra private methods and attributes if necessary.

   Hint: for your algorithmic solution, remember that computer memory is another important resource that you have access to, and its use can significantly reduce computational time, especially when runtime is dramatically long.

5. As you progress with the implementation of your algorithmic solution, you should start using the Tester class in order to test your code for potential logical issues and runtime errors. This (testing) part of the task is as important as coding.

6. Finally, note that cheating is not allowed in this task; that is, it is unacceptable to simply return the expected answers of a test instance you are given. Such illegal solutions will be marked as failed.

Expected Printout

Appendix A provides an example printout for your program. If you are getting a different printout then your implementation is not ready for submission. Please ensure your program prints out Success for all tests before submission.

Further Notes

Reading sections 5.1, 5.3 and 5.4 of the course book "Data Structures and Algorithms in Java" by Michael T. Goodrich, Irvine Roberto Tamassia, and Michael H. Goldwasser (2014) should help you with the solution to this task. You may access the book on-line for free from the reading list application in CloudDeakin available in Resources → Additional Course Resources → Resources on Algorithms and Data Structures → Course Book: Data structures and algorithms in Java.

Marking Process and Discussion

To get your task completed, you must finish the following steps strictly on time.

1. Work on your task either during your allocated lab time or during your own study time.
2. Once the task is complete you should make sure that your program implements all the required functionality, is compliable, and has no runtime errors. Programs causing compilation or runtime errors will not be accepted as a solution. You need to test your program thoroughly before submission. Think about potential errors where your program might fail. Note we can sometime use test cases that are different to those provided so verify you have checked it more thoroughly than just using the test program provided.
3. Submit your solution as an answer to the task via the OnTrack submission system. This first submission must be prior to the submission "S" deadline indicated in the unit guide and in OnTrack.
4. If your task has been assessed as requiring a "Redo" or "Resubmit" then you should prepare a new submission. You will have 1 (7 day) calendar week from the day you receive the assessment from the tutor. This usually will mean you should revise the lecture, the readings indicated, and read the unit discussion list for suggestions. After your submission has been corrected and providing it is still before the due deadline you can resubmit.
5. If your task has been assessed as correct, either after step 3 or 4, you can "discuss" with your tutor. This first discussion must occur prior to the discussion "D".
6. Meet with your tutor or answer question via the intelligent discussion facility to demonstrate/discuss your submission. Be on time with respect to the specified discussion deadline.
7. The tutor will ask you both theoretical and practical questions. Questions are likely to cover lecture notes, so attending (or watching) lectures should help you with this compulsory interview part. The tutor will tick off the task as complete, only if you provide a satisfactory answer to these questions.
8. If you cannot answer the questions satisfactorily your task will remain on discussion and you will need to study the topic during the week and have a second discussion the following week.
9. Please note, due to the number of students and time constraints tutors will only be expected to mark and/or discuss your task twice. After this it will be marked as a "Exceeded Feedback".
10. If your task has been marked as "Exceeded Feedback" you are eligible to do the redemption quiz for this task. Go to this unit's site on Deakin Sync and find the redemption quiz associated with this task. You get three tries at this quiz. Ensure you record your attempt.
    I. Login to Zoom and join a meeting by yourself.
    II. Ensure you have both a camera and microphone working
    III. Start a recording.
    IV. Select Share screen then select "Screen". This will share your whole desktop. Ensure Zoom is including your camera view of you in the corner.
    V. Bring your browser up and do the quiz.
    VI. Once finished select stop recording.
    VII. After five to ten minutes you should get an email from Zoom providing you with a link to your video. Using the share link, copy this and paste in your chat for this task in OnTrack for your tutor to verify the recording.
11. Note that we will not check your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work through the unit.
12. Final note, A "Fail" or "Exceeded Feedback" grade on a task does not mean you have failed the unit. It simply means that you have not demonstrated your understanding of that task through OnTrack. Similarly failing the redemption quiz also does not mean you have failed the unit. You can replace a task with a task from a higher grade.

Appendix A

This section displays the printout produced by the attached Tester class, specifically by its *Main* method. It is based on our solution. The printout is provided here to help with testing your code for potential logical errors. It demonstrates the correct logic rather than an expected printout in terms of text and alignment.

```
Attempting test instance 0 with [7, 2] as the argument and 5 as the expected answer
 :: SUCCESS


Attempting test instance 1 with [2, 7, 3] as the argument and -2 as the expected answer
 :: SUCCESS


Attempting test instance 2 with [1000, 1000, 1000, 1000, 1000] as the argument and 1000 as the expected answer
 :: SUCCESS


Attempting test instance 3 with [823, 912, 345, 100000, 867, 222, 991, 3, 40000] as the argument and -58111 as the
expected answer
 :: SUCCESS


Attempting test instance 4 with [23, 35, 12, 100000, 99234, 86123, 3245] as the argument and -83644 as the expected
answer
 :: SUCCESS


Attempting test instance 5 with [23, 35, 12, 100000, 99234, 86123, 3245, 1] as the argument and 83645 as the expected
answer
 :: SUCCESS


Attempting test instance 6 with [7, 7, 7, 7, 7, 7, 80, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7] as the argument and -66 as the expected answer
 :: SUCCESS


Attempting test instance 7 with [7, 7, 7, 7, 7, 7, 7, 80, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7] as the argument and 73 as the expected answer
 :: SUCCESS


Attempting test instance 8 with [91, 56, 23, 45, 87, 65, 45, 45, 78, 23, 20, 41, 17, 54, 51, 51, 94, 62, 74, 42, 76, 76] as the
argument and 96 as the expected answer
 :: SUCCESS


Attempting test instance 9 with [92834, 95461, 15911, 56189, 6369, 80545, 31811, 51263, 30076, 68867, 36905, 32499,
59799, 334, 82991, 46636, 98741, 66601] as the argument and 42958 as the expected answer
 :: SUCCESS


Attempting test instance 10 with [129] as the argument and 129 as the expected answer
 :: SUCCESS


Attempting test instance 11 with [35463, 88121, 4362, 94457, 86235, 83680, 72686, 6003, 93069, 2015, 10436, 2139,
93162, 30380, 19067, 76335, 78941, 48620, 55887, 15679] as the argument and 101879 as the expected answer
```

```
 :: SUCCESS


Attempting test instance 12 with [19335, 97643, 11468, 86267, 79718, 59584, 12129, 52642, 86575, 62307, 11545, 52658,
72377, 39986, 74850, 1992, 86928] as the argument and 1846 as the expected answer
 :: SUCCESS


Attempting test instance 13 with [91883, 97793, 54567, 64714, 98624] as the argument and 82567 as the expected answer
 :: SUCCESS


Attempting test instance 14 with [98473, 41866, 71129, 65936, 42626, 9194, 46718, 96921, 45613, 47677, 8763, 54634,
47259, 71448, 9918, 22666, 32711, 21692, 40207, 2017, 23040, 86083, 77809, 15472, 30718, 39085, 87911, 54827,
41686, 28354, 37203, 6548, 74184, 3043, 43961, 95189, 1238, 22002, 93507, 63546, 32527, 42778, 31614] as the
argument and -14953 as the expected answer
 :: SUCCESS


Attempting test instance 15 with [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50] as the argument and 25 as the
expected answer
 :: SUCCESS


Attempting test instance 16 with [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] as the argument and 0 as the expected answer
 :: SUCCESS


Attempting test instance 17 with [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 11, 11, 11, 11, 111, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
11, 11, 11, 11, 112, 312, 312, 123, 123, 123, 123, 123, 123, 123, 123, 123, 123, 123, 123, 123, 123, 231, 31, 312] as the
argument and 316 as the expected answer
 :: SUCCESS


Attempting test instance 18 with [1234, 1233, 12, 312, 32, 23, 434, 12, 312, 45, 1234, 1233, 12, 312, 32, 23, 434, 12, 312,
45, 1234, 1233, 12, 312, 32, 23, 434, 12, 312, 45, 1234, 1233, 12, 312, 32, 23, 434, 12, 312, 45, 1234, 1233, 12, 312, 32,
23, 434, 12, 312, 45] as the argument and 1995 as the expected answer
 :: SUCCESS


Attempting test instance 19 with [1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3] as
the argument and 1 as the expected answer
 :: SUCCESS


Attempting test instance 20 with [9, 100, 1, 8] as the argument and 98 as the expected answer
 :: SUCCESS


Attempting test instance 21 with [1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 66, 5, 4, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 6, 5, 4, 5, 6, 3, 4, 4, 5, 6, 3,
4, 5, 6, 3, 4, 5, 6, 3, 4, 5, 6, 3, 4, 5, 6] as the argument and 68 as the expected answer
 :: SUCCESS


Attempting test instance 22 with [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 1, 2, 3, 4, 65, 67, 2, 3, 4, 7, 2, 3, 4, 6, 6, 7, 2, 3, 4,
7, 78, 8, 82, 2, 3, 4, 7, 2, 2, 34, 4, 6, 7, 3, 2] as the argument and 128 as the expected answer
 :: SUCCESS
```

```
Attempting test instance 23 with [100, 10, 10] as the argument and 100 as the expected answer
 :: SUCCESS


Attempting test instance 24 with [1] as the argument and 1 as the expected answer
 :: SUCCESS


Attempting test instance 25 with [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 4, 3, 5,
4, 6, 7, 5, 6, 10, 2, 5, 4, 3, 4, 5, 6, 7, 9, 10] as the argument and 28 as the expected answer
 :: SUCCESS


Attempting test instance 26 with [6, 4, 3, 5, 8, 8] as the argument and 2 as the expected answer
 :: SUCCESS


Attempting test instance 27 with [1, 5, 20, 2, 1] as the argument and -13 as the expected answer
 :: SUCCESS


Attempting test instance 28 with [1, 2, 3, 4, 5, 6, 6, 7, 8, 767, 765, 111, 76576, 5, 64, 654, 64, 7, 7657, 76575, 64, 65, 6454,
64, 654, 65464, 7, 5435, 65, 746, 7, 546, 7, 654, 7, 5435, 547, 6, 6, 7, 6547, 7654, 6, 754, 54353, 65, 7, 8] as the argument
and 118231 as the expected answer
 :: SUCCESS


Summary: 29 tests out of 29 passed
Tests passed (0 to 29): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28
Tests failed (0 to 29): none
```