Practical Task 3.2

(Distinction Task)

Submission deadline: 11:59 pm Sunday, August 9th
Discussion deadline: 11:59 pm Sunday, August 30th

Task Objective

In this task you will extend the implementation done in the 3.1P task and implement three more advanced sorting algorithms based around recursion (*RandomizedQuickSort, MergeSortTopDown*, and *MergeSortBottomUp*,).

Task Details

Using the Vector class you implemented as part of Task 1.1P work through the following steps to complete the task:

1. Explore the program code attached to this task. Extend the project inherited from Task 3.1. Import the Tester.cs file to the project to access the prepared Main method important for the purpose of debugging and testing the required algorithmic solutions.
2. Proceed with the implementation of the three sorting algorithms: Randomized Quick Sort, the top-down version and the bottom-up version of the Merge Sort. For this purpose, create three new classes, i.e. *RandomizedQuickSort*, *MergeSortTopDown*, and *MergeSortBottomUp*, respectively. Ensure that the new classes implement the *ISorter* interface along with its prescribed method *Sort<K>*. Each class must have a default constructor. You may add any extra private methods and attributes if necessary.
3. First, start with the Randomized Quick Sort algorithm. The corresponding method must select the pivot element uniformly at random from a range of elements available within a recursive call. Furthermore, it must perform sorting operations "*in-place*".

   To get more insights about these particular requirements, and the Randomized Quick Sort in general, explore the material of Chapter 7 of SIT221 Workbook. Though it explains the algorithm, the pivot point selection used there follows the "median three" policy. As your aim is the random choice policy, peruse Chapter 12.2 of the SIT221 course book entitled "Data structures and algorithms in Java". You should focus on Section 12.2.2 covering the *in-place* version of the algorithm and Pseudocode 12.6 as this is one that you must encode. Selecting pivot points at random is explained in 12.1.1. This technique must replace line 7 of Pseudocode 12.6 with a randomized selection function. As you need to generate random numbers, use the standard *Random* class available within the .NET Framework.
4. Second, proceed with the top-down version of the Merge Sort algorithm. You can read about the algorithm in Section 12.1 of the SIT221 course book "Data structures and algorithms in Java". Make sure that you grasp its recursive nature. Follow the general description of the Merge Sort given in Section 12.1.1 and the code fragments 12.1 and 12.2 to implement its array-based top-down version.
5. Finally, complete the task by encoding the bottom-up version of the Merge Sort. Section 12.1.5 provide in the SIT221 course book details its implementation. Make sure that the issue related to temporary memory usage, which makes two versions different not just from the implementation perspective, but also with respect to efficiency, is clear for you
6. As you progress with the implementation of the algorithms, you should start using the Tester class in order to test them for potential logical issues and runtime errors. This (testing) part of the task is as important as coding. You may wish to extend it with extra test cases to be sure that your solutions

are checked against other potential mistakes. To enable the tests, remember to uncomment the corresponding code lines. Remember that you may change the sorting approach for an instance of the Vector<T> class by referring its *Sorter* property to a new instance of that algorithm. For example

```
vector.Sorter = new MergeSortTopDown ();
```

should enable the top-down Merge Sort algorithm encoded in the MergeSortTopDown class. Check whether you test the right algorithm.

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

- The SIT221 Workbook is available in CloudDeakin in Resources → Additional Course Resources → Resources on Algorithms and Data Structures → SIT221 Workbook.
- The course book "Data Structures and Algorithms in Java" by Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser (2014) can be accessed 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.
- If you still struggle with such OOP concepts as Generics and their application, you may wish to read Chapter 11 of SIT232 Workbook available in Resources → Additional Course Resources → Resources on Object-Oriented Programming. You may also have to read Chapter 6 of SIT232 Workbook about Polymorphism and Interfaces as you need excellent understanding of these topics to progress well through the practical tasks of the unit. Make sure that you are proficient with them as they form a basis to design and develop programming modules in this and all the subsequent tasks. You may find other important topics required to complete the task, like exceptions handling, in other chapters of the workbook.
- We will test your code in Microsoft Visual Studio 2017. Find the instructions to install the community version of Microsoft Visual Studio 2017 available on the SIT221 unit web-page in CloudDeakin at Resources → Additional Course Resources → Software → Visual Studio Community 2017. You are free to use another IDE if you prefer that, e.g. Visual Studio Code. But we recommend you to take a chance to learn this environment.

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.

```
Test A: Sort integer numbers applying RandomizedQuickSort with AscendingIntComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [100,120,122,175,213,236,263,299,312,333,511,596,722,724,752,772,780,958,966,995]
 :: SUCCESS

Test B: Sort integer numbers applying RandomizedQuickSort with DescendingIntComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [995,966,958,780,772,752,724,722,596,511,333,312,299,263,236,213,175,122,120,100]
 :: SUCCESS

Test C: Sort integer numbers applying RandomizedQuickSort with EvenNumberFirstComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [772,958,724,596,312,100,236,120,722,966,752,122,780,175,299,213,995,333,263,511]
 :: SUCCESS

Test D: Sort integer numbers applying MergeSortTopDown with AscendingIntComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [100,120,122,175,213,236,263,299,312,333,511,596,722,724,752,772,780,958,966,995]
 :: SUCCESS

Test E: Sort integer numbers applying MergeSortTopDown with DescendingIntComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [995,966,958,780,772,752,724,722,596,511,333,312,299,263,236,213,175,122,120,100]
 :: SUCCESS

Test F: Sort integer numbers applying MergeSortTopDown with EvenNumberFirstComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [772,596,958,752,120,122,724,966,722,100,780,312,236,995,299,175,263,213,511,333]
 :: SUCCESS

Test G: Sort integer numbers applying MergeSortBottomUp with AscendingIntComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [100,120,122,175,213,236,263,299,312,333,511,596,722,724,752,772,780,958,966,995]
 :: SUCCESS

Test H: Sort integer numbers applying MergeSortBottomUp with DescendingIntComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [995,966,958,780,772,752,724,722,596,511,333,312,299,263,236,213,175,122,120,100]
 :: SUCCESS
```

```
Test I: Sort integer numbers applying MergeSortBottomUp with EvenNumberFirstComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [772,596,958,752,120,122,724,966,722,100,780,312,236,995,299,175,263,213,511,333]
 :: SUCCESS



 ------------------- SUMMARY -------------------
Tests passed: ABCDEFGHI
```