Practical Task 4.1

(Pass Task)

Submission deadline: Monday, April 7
Discussion deadline: Friday, May 2

Instructions

In this task, you will need to further extend your Vector<T> class by adding a search method that can be applied to a sorted sequence of data elements stored in an instance of this class in order to find the position of a requested element. Specifically, your task is to learn and implement the Binary Search algorithm.

The following steps indicate what you need to do.

1. Create a copy of the latest version of your Vector<T> class and replace the Tester.cs file by that attached to this task. The Main method of the new Tester class is to be compatible with the Vector<T> class. Again, it contains a series of tests that will verify if the functionality that you need to develop in this task works correctly. See Appendix A for an example of what the program should output.
2. Extend your Vector<T> class by adding the following method:
   - **int BinarySearch(T item, IComparer<T> comparer)**
     Searches for the specified **item** within the entire sorted sequence of elements of the Vector<T> collection. The comparison of elements with the **item** is made with the use of the specified **comparer**, or Comparer<T>.Default as the default comparer for type T, if the provided **comparer** is null. This method returns the zero-based index of the **item**, when the **item** is found; otherwise, a negative integer number (e.g., -1).

   Note that you are not allowed to delegate the binary search operation to the Array.BinarySearch method or a similar existing method of the .NET Framework; you must provide your own implementation of the Binary Search.
3. As you progress with the implementation of the algorithm, you should start using the Tester class in order to test your solution 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 solution is checked against other potential mistakes. To enable the tests, remember to uncomment the corresponding lines of provided code.

Further Notes

- To find out how to program the Binary Search algorithm, explore Chapter 8.1.2 of SIT221 Workbook available in CloudDeakin in Content → Learning Resources → SIT221 Workbook.

  For further details, read Chapter 5.1.3 of the SIT221 Course Book "Data Structures and Algorithms in Java" by Michael T. Goodrich, Irvine Roberto Tamassia, and Michael H. Goldwasser (2014). You may access the book on-line from the reading list application in CloudDeakin available in Content → Reading List → Course Book: Data structures and algorithms in Java.
- If you still struggle with such OOP concept as Generics, you may wish to read Chapter 11 of SIT232 Workbook available in Content → Learning Resources → SIT232 Workbook. You may also have to read Chapter 6 of SIT232 Workbook about Polymorphism and Interfaces as you will need an excellent understanding of these topics in order 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 subsequent tasks.

- We will test your code using the .NET Core 6.0. You are free to use any IDE (for example, Visual Studio Code), though we recommend you work from Microsoft Visual Studio 2022 due to its simple installation process and good support of debugging. You can find the necessary instructions and installation links by navigating in CloudDeakin to Content → Learning Resources → Software.

Submission Instructions and Marking Process

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

- Make sure your programs implement the required functionality. They must compile and have no runtime errors. Programs causing compilation or runtime errors will not be accepted as a solution. You need to test your programs thoroughly before submission. Think about potential errors where your programs might fail.
- **Submit** the expected code files as a solution to the task via OnTrack submission system. You must **record a short video explaining your solution** to the task. Remember to explain the implementation issues and communicate about the iterative and recursive versions of the Binary Search Algorithm. Upload the video to one of accessible resources and refer to it for the purpose of marking. You must provide a private working link to the video to your marking tutor in OnTrack. Note that the video recording must be made in the **camera on mode**; that is, the tutor must see both the presenter and the shared screen.
- Once your solution is accepted by the tutor, you will be invited to **continue its discussion and answer relevant theoretical questions** through the Intelligent Discussion Service in OnTrack. Your tutor will record several audio questions. When you click on these, OnTrack will record your response live. You must answer straight away in your own words. As this is a live response, you should ensure you understand the solution to the task you submitted.
  Answer all additional questions that your tutor may ask you. Questions will cover the lecture notes; so, attending (or watching) the lectures should help you with this **compulsory discussion part**. You should start the discussion as soon as possible as if your answers are wrong, you may have to pass another round, still before the deadline. Use available attempts properly.

Note that we will not accept 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 throughout the unit.

Remember that this is your responsibility to keep track of your progress in the unit that includes checking which tasks have been marked as completed in the OnTrack system by your marking tutor, and which are still to be finalised. When grading your achievements at the end of the unit, we will solely rely on the records of the OnTrack system and feedback provided by your tutor about your overall progress and the quality of your solutions.

Appendix A: Expected Printout

The following provides an example of the output generated from the testing module (Tester.cs) once you have correctly implemented all methods of the Vector<T> class.

```
Test A: Search for key 333 in the array of integer numbers sorted via the AscendingIntComparer:
Elements in the Vector: [100,120,122,175,213,236,263,299,312,333,511,596,722,724,752,772,780,958,966,995]
 :: SUCCESS


Test B: Search for key 99 in the array of integer numbers sorted via the AscendingIntComparer:
Elements in the Vector: [100,120,122,175,213,236,263,299,312,333,511,596,722,724,752,772,780,958,966,995]
 :: SUCCESS


Test C: Search for key 996 in the array of integer numbers sorted via the AscendingIntComparer:
Elements in the Vector: [100,120,122,175,213,236,263,299,312,333,511,596,722,724,752,772,780,958,966,995]
 :: SUCCESS


Test D: Search for key 333 in the array of integer numbers sorted via the DescendingIntComparer:
```

```
Elements in the Vector: [995,966,958,780,772,752,724,722,596,511,333,312,299,263,236,213,175,122,120,100]
 :: SUCCESS

Test E: Search for key 994 in the array of integer numbers sorted via the DescendingIntComparer:
Elements in the Vector: [995,966,958,780,772,752,724,722,596,511,333,312,299,263,236,213,175,122,120,100]
 :: SUCCESS

Test F: Search for key 101 in the array of integer numbers sorted via the DescendingIntComparer:
Elements in the Vector: [995,966,958,780,772,752,724,722,596,511,333,312,299,263,236,213,175,122,120,100]
 :: SUCCESS



 ------------------- SUMMARY -------------------
Tests passed: ABCDEF
```