Practical Task 3.1

(Pass Task)

Submission deadline: Monday, March 31
Discussion deadline: Friday, April 18

Instructions

In this task, you will need to implement three classical sorting algorithms: the *Bubble Sort*, the *Insertion Sort*, and the *Selection Sort*. These algorithms then can be used to sort data elements stored in an instance of the Vector<T> collection. We strongly recommend you start your work with watching Lecture 2, which provides time-efficient implementations and explains the sorting process performed by each of these algorithms.

The following steps indicate what you need to do.

1. Create a copy of your solution for Task 2.1 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 your 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 will output when working correctly.

2. Import and explore the attached ISorter.cs file. It contains the ISorter interface, which consists of only a single generic method with the following signature and purpose:

   – **void Sort<K>(K[ ] array, int index, int num, IComparer<K> comparer)**

     Sorts the data elements stored in the specified one-dimensional **array** in the range defined by the starting position **index** and the number of subsequent elements **num**. The order of sorted elements is determined by the specified **comparer**, or, if the **comparer** is null, by the default comparer for type K obtained through the Comparer<K>.Default property. This method must throw the following exceptions:

     - ArgumentNullException if the **array** is null,
     - ArgumentOutOfRangeException if the given **index** or **num** or both are negative,
     - ArgumentException if the values of **index** and **num** do not specify a valid range within the given **array**.

   In order to make the type **K** compatible with the Sort method, there is a constraint applied to this generic type that expects the implementation of the IComparable<K> interface. That is, any concrete type to replace the placeholder **K** during the runtime must implement the IComparable<K> and its mandatory CompareTo(K *another*) method. This requirement ensures that the call to the Comparer<K>.Default property is possible in case when the specified **comparer** equals null.

3. Extend your Vector<T> class by adding the following method:

   – **void Sort(ISorter algorithm, IComparer<T> comparer)**

     Sorts the elements in the entire Vector<T> collection using the specified sorting **algorithm** and **comparer**. If the **algorithm** is null, the sorting operation should be delegated to the Array.Sort as the default sorting method. If the **comparer** is null, the order of sorted elements is determined by the default comparer for type T; that is, by Comparer<T>.Default.

4. Your task now is to create a new program class, **BubbleSort**, that implements the ISorter interface, and thus its Sort method. An object of this class must apply the Bubble Sort algorithm to the range of elements of the array given as the input of the Sort method. Make sure that your implementation of the Bubble Sort matches its expected best- and worst-case time complexities. Keep in mind that you may add any private methods into the BubbleSort class if this helps you with its implementation. Once your Bubble Sort algorithm is ready, you should be able to sort integers enabling various comparison classes available in the Tester.cs file (i.e., the AscendingIntComparer, the DescendingIntComparer, and the EvenNumberFirstComparer) and succeed with tests D, E, and F. To enable these tests, do not forget to uncomment the respective code lines in the Main method of the Tester class.

5. Continue your work and implement the two remaining sorting algorithms, i.e., the Insertion Sort and the Selection Sort. These algorithms must be presented by their respective program classes: **InsertionSort** and **SelectionSort**. As you complete each of these algorithms, activate the corresponding tests provided in the Tester class.
6. This final step is crucial. You must test your sorting algorithms independently of the Vector<T> class. To do this, extend the existing set of tests by adding new cases with focus on sorting an array of data elements (e.g., integers or strings) by a direct call of the Sort methods of the BubbleSort, InsertionSort, and SelectionSort classes. Be creative and think about the tests to be most effective in testing the algorithms for logical errors and runtime exceptions. Later, in a face-to-face discussion with your tutor, you will need to clearly communicate your testing strategies and explain the purpose of proposed tests.

Further Notes

- Explore Chapter 7 of SIT221 Workbook available in CloudDeakin in Content → Learning Resources → SIT221 Workbook. It starts with a general explanation of algorithm complexity and describes the Bubble Sort, Insertion Sort, and Selection Sort algorithms. Study the provided examples and follow the pseudocodes as you progress with coding of the algorithms.
- The implementation of the Insertion Sort algorithm is also detailed in Chapter 3.1.2 of the 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 for free 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. In other chapters of the workbook, you may find more important topics required to complete the task, for example, the material on exceptions handling.
- 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, have no runtime errors, and pass all test cases of the task. 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.
- Once your solution is accepted by your tutor, you will be invited to **continue its discussion and answer relevant theoretical questions through a face-to-face interview**. Specifically, you will need to meet with the tutor to demonstrate and discuss the solution in one of the dedicated practical sessions (run online via MS Teams for online students and on-campus for students who selected to join classes at Burwood\Geelong). Please, come prepared so that the class time is used efficiently and fairly for all students in it. Be on time with respect to the specified discussion deadline.

  You will also need to **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: Sort integer numbers applying Default Sort with the AscendingIntComparer:
Initial 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 Default Sort with the DescendingIntComparer:
Initial 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 Default Sort with the EvenNumberFirstComparer:
Initial data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [724,596,958,752,120,122,966,772,722,100,780,312,236,213,995,263,175,299,511,333]
 :: SUCCESS

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

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

Test J: Sort integer numbers applying InsertionSort with the AscendingIntComparer:
Initial 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 K: Sort integer numbers applying InsertionSort with the DescendingIntComparer:
Initial 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 L: Sort integer numbers applying InsertionSort with the EvenNumberFirstComparer:
Initial data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [236,312,780,100,722,966,724,122,120,752,958,596,772,333,511,213,263,175,299,995]
 :: SUCCESS

------------------- SUMMARY -------------------
Tests passed: ABCDEFGHIJKL
```