Practical Task 3.1

(Pass Task)

Submission deadline: 10:00am Monday, August 5
Discussion deadline: 10:00am Saturday, August 31

General Instructions

The purpose of this task is to study implementation of classical sorting algorithms such as *Bubble Sort*, *Insertion Sort*, and *Selection Sort*.

1. Explore the program code attached to this task. Create a new Microsoft Visual Studio project and import the Vector.cs file, or alternatively, extend the project inherited from Task 2.1 by copying the missing code from the enclosed template for the Vector<T> class. 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. Your first step in this task is to figure out how particular sorting algorithms represented via unique classes can be linked to the Vector<T> class you should have completed through Tasks 1.1 and 1.2. The Vector<T> class must have ability to flexibly switch between possible algorithms at any time when a user wants to change the sorting technique in use. In fact, a simple solution to implement this linkage is an interface that every class providing sorting functionality must implement. Such interface, named *ISorter*, is already prepared for you as part of the ISorter.cs file. Study the method's signature that the interface ensures. This method is generic and has the following purpose:
   - **Sort<K>(K[] sequence, IComparer<K> comparer)**
     Sorts the elements in the entire one-dimensional array of generic type K using the specified comparer IComparer<K>. When the comparer is not specified, i.e. is *null*, the default comparer Comparer<K>.Default is applied.

   Note that it imposes a constraint on the type K such that K must implement the IComparable<K> interface. This is done to guarantee that the actual data type substituting K in practice implements a default comparer, therefore the Sort<K> can sort elements of that type according to the default ordering rule determined by the IComparable<K>.
3. Now, switch to the attached template of the Vector<T> class. The given class has a public property, named *Sorter,* implementing the aforementioned *ISorter* interface. The purpose of the property is to refer to a particular instance of the sorting algorithm that is currently in use by the Vector<T> class. Note that it realizes so-called *Class Aggregation* as a particular form of class relationship in terms of object-oriented programming design. This makes the chosen sorting algorithm encapsulated as a class, e.g. *BubbleSort* or *InsertionSort*, so that it becomes "a part of" the Vector<T> class. (To review possible class relationships, explore Chapter 4 of the SIT232 Workbook, page 99). One may read and write to this property; that is, to get/set a reference to the object serving as a sorting algorithm. Obviously, by changing the value of this property, one may also switch the sorting approach in use.

   By default, this *Sorter* property of the Vector<T> class refers to the built-in internal class *DefaultSorter*, which implements the Sort<K> method prescribed by the imposed *ISorter* interface. The DefaultSorter class delegates sorting to the Array.Sort method, which you should be familiar with based on Task 2.1. Because of this, by default, the both Sort() and Sort(IComparer<T> comparer) methods guarantee exactly the same behaviour as supposed in Task 2.1.
4. Implement three sorting algorithms: Bubble Sort, Insertion Sort, and Selection Sort. For this purpose, create three new classes, i.e. *BubbleSort*, *InsertionSort*, and *SelectionSort*, 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. You should

rely on the code of *DefaultSorter* as an example of how your classes are to be implemented. Therefore, explore the code of the method and how it deals with the default comparer, i.e. the Comparer<K>.Default.

5. 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 object. For example,

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

should enable the Bubble Sort algorithm encoded via the *BubbleSort* class. Check whether you test the right algorithm.

Further Notes

- Explore Chapter 7 of SIT221 Workbook available in CloudDeakin in Resources → Additional Course Resources → Resources on Algorithms and Data Structures → SIT221 Workbook. It starts with general explanation of algorithm complexity and describes 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, Roberto Tamassia, and Michael H. Goldwasser (2014). You may access this 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.
- 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.

- 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.
- Please, note that you are free in writing your own code internal (private) to the required classes and methods. Therefore, we do not give any particular instructions about this. The only important

requirement is that you must fulfil the specified interface in terms of functionality (logic) and signatures of the requested methods.
- Submit your program code as an answer to the task via OnTrack submission system.
- Meet with your marking tutor to demonstrate and discuss your program in one of the dedicated practical sessions. Be on time with respect to the specified discussion deadline.
- Answer all additional (theoretical) questions that your tutor can ask you. Questions are likely to cover lecture notes, so attending (or watching) lectures should help you with this compulsory interview part. Please, come prepared so that the class time is used efficiently and fairly for all the students in it. You should start your interview as soon as possible as if your answers are wrong, you may have to pass another interview, still before the deadline. Use available attempts properly.

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.

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 marking you 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 quality of your solutions.

Expected Printout

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 Default Sort 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 Default Sort 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 Default Sort 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: [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 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 BubbleSort 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 BubbleSort 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: [724,596,958,752,120,122,966,772,722,100,780,312,236,213,995,263,175,299,511,333]
 :: SUCCESS

Test G: Sort integer numbers applying SelectionSort 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 SelectionSort 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 SelectionSort 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: [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 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 K: Sort integer numbers applying InsertionSort 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 L: Sort integer numbers applying InsertionSort 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: [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
```