# SIT221 –DATA STRUCTURES AND ALGORITHMS

## LAB2: LET'S EXTEND OUR COLLECTION CLASS TO SORT AND SEARCH

### LAB OBJECTIVE:

Extend Vector class to support sorting & searching

### SUBMISSION INSTRUCTIONS

1. You should zip your solution into **one file only,** and
2. Submit this file to Week 2 assignment folder by 5pm Monday, 24 July

### PREPARATION

1. If you feel still not confident with collection classes, please read this: https://msdn.microsoft.com/en-us/library/7y3x785f(v=vs.110).aspx

2. In this unit we will be switching between building our own collection classes so we get a better understanding of how these classes work; and using existing collection classes in .NET framework.

3. Download the template project available on the weekly resources folder. The solution has two projects: **DataStructures_Algorithms & Runner** projects.

4. In the Runner project, there is a Data Folder for Week01 – Note that the dataset has three files (1H.txt, 1T.txt, 1M.txt) [1H = 100 points, 1T = 1000 points, 10T = 10,000 points]

## LAB TASKS

### 1. EXTEND VECTOR CLASS TO SUPPORT DEFAULT SORTING

In this task, we want to extend the **Vector** class (from week 01) to support sorting and searching of object elements. Please add the following capabilities:

a. **public void Sort():** This method should call the default Sort method in the Array class on vector data array. Check Array.Sort here https://msdn.microsoft.com/en-us/library/6tf1f0bc(v=vs.110).aspx
b. **public void Sort(IComparer<T> comparer):** This method should call the Sort method in the Array class on vector data array that expects a comparer object. Check Array.Sort here https://msdn.microsoft.com/en-us/library/6tf1f0bc(v=vs.110).aspx
c. Adjust **Runner02_Task1** class to load (already implemented in the given sample), sort the data loaded from 1H.txt (a file of 100 elements) to store the sorted data to S_1H.txt.
d. Now, can you add the sorted file (e.g. S_1H.txt) as another input argument to your solution? Add it in the properties window, and use it as the file name to store your sorted data to.

### 2. LET'S SORT PARTS – ICOMPARABLE

In this task, we want to modify our solution from task 1 to sort array of Part objects. In the template solution - Week02 folder, you are given a Part class. Please check Part class definition. Then do the following:

a. Check **Runner02_Task2** class, create a vector of Part objects, then sort it, and finally store it to a file.
b. Try to compile **Runner02_Task2** (make sure to adjust Main function to call **Runner02_Task2** instead of **Runner02_Task1**). What did you get? Exception? Can you tell why?
c. Now modify Part class to implement IComparable interface. This means you need to implement CompareTo method.

   CompareTo returns int. It should return < 0 if part id of object A is less than part id of object B, and so on. Please see this: https://msdn.microsoft.com/en-us/library/43hc6wht(v=vs.110).aspx
d. Now, go back and try to run your project, is it working?

## 3. SORT PARTS DESCENDING – ICOMPARER

In this task, we need to modify our solution from task 2 to sort arrays of Part objects in descending order. Then do the following:

a. Add a new class under week02 called PartsComparer that implements IComparer interface: https://msdn.microsoft.com/en-us/library/8ehhxeaf(v=vs.110).aspx
b. Now, modify the **Runner02_Task2** to run the sort method using your new Comparer class to get descending order.