# Exercise: Arrays, Lists, Stacks, Queues

This document defines in-class exercise problems from the [“Advanced C#“ Course @ Software University](http://softuni.bg/courses/advanced-csharp/). You are presented with some problems and certain steps you need to take in order to accomplish the tasks.

## Problem 1. Sort Array of Numbers Using Bubble Sort

Write a program to sort an **array of integer numbers** and then print them back on the console. The numbers should be entered from the console **on a single line, separated by a space**. Print the sorted array in the following format: **“[element1, element2… elementN]”.**

**Condition: Do not use the built-in sorting method, you should write the logic yourself. Use the** [**bubble sort algorithm**](https://en.wikipedia.org/wiki/Bubble_sort)**.**

*Hint: To understand the sorting process better you may use a visual aid, e.g.* [*Visualgo*](http://visualgo.net/sorting.html)*.*

### Step 1. Read and Process the Input

Since the array elements are entered on a single line, you’ll need to read a line from the console and **split it** by space. This will produce an **array of strings**. Now you need to create an array of integers and fill it by **parsing the numbers** in the string array. You may write your own logic or use an already available method to do so (e.g. a static method in the Array class or a LINQ extension method). **Test** to see if you’re processing the input correctly.

### Step 2. Search the Algorithm and Understand How It Works

Bubble sort is a very simple sorting algorithm. At each step, two consecutive elements in the array are compared and if the second one is smaller than the first the elements are **swapped** (they exchange places). The process continues until all elements are sorted – an entire iteration has passed without a swap occurring.

### Step 3. Implement the Algorithm

It is recommended that you extract the sorting logic into a **method**, so you may re-use it many times in the same program. Methods will be covered in more detail in a future lecture, so this is not a requirement.

You may use [pseudocode](https://en.wikipedia.org/wiki/Pseudocode) and re-write it in C# syntax or **write the logic yourself (highly recommended)**. In order to accomplish the task you’ll need to **iterate the array repeatedly**, at each iteration you’ll need to take two elements and compare them and in some cases you may need to **swap** the elements. Swapping elements in an array is the same as swapping variable values which you’ve already done in the Programming Basics Course at SoftUni. While writing your program you will practice **iterating an array, accessing and setting its elements by index.**

### Step 4. Print the Result

When you have taken all the necessary steps and the array is sorted, you need to **print** the array in the specified **format**. There are shortcuts to accomplish this, but you may use a simple loop.

**Test your program** with different inputs to make sure it works as intended.

## Problem 2. Join Lists

Write a program that takes as input two lists of integers and **joins them**. The result should hold all numbers from the first list, and all numbers from the second list, **without repeating numbers**, and arranged in **increasing order**. The input and output lists are given as integers, separated by a space, each list at a separate line. **Do not use LINQ! This problems aims to help you exercise your algorithmic thinking and not how well you are familiar with built-in .NET functionalities. Use only arrays and lists.** Examples:

|  |  |
| --- | --- |
| **Input** | **Output** |
| 20 40 10 10 30 80  25 20 40 30 10 | 10 20 25 30 40 80 |
| 5 4 3 2 1  6 3 2 | 1 2 3 4 5 6 |
| 1  1 | 1 |

### Step 1. Read and Process the Input

As with the first problem, the first step is to take the input numbers and store them in two separate sets (arrays or lists).

### Step 2. Create a New List to Hold the Result

You can’t know for sure how many numbers will be found in the two sets. Instead of creating an array with a fixed size (e.g. the combined size of both sets), a much better approach is to use a **List<int>**. There are better choices for this particular problem (SortedSet<T> will be covered in the next lecture), but then it would be too easy, wouldn’t it?

### Step 3. Fill the List by Iterating the Sets

Iterate each set and if the current element is not already in the resulting list – add it. If the number is already present, there is no need to add it (you would need to filter out repeating numbers afterwards which is unnecessary).

### Step 4. Sort the List and Print It

After you’ve made sure all numbers were added to the list, sort it. You may use the .Sort() method this time. Finally, print the result on the console.

Remember to **test your program**! It is recommended that you **perform manual tests at each step** of the process and not just at the very end. This is how you solve much more complex problems.