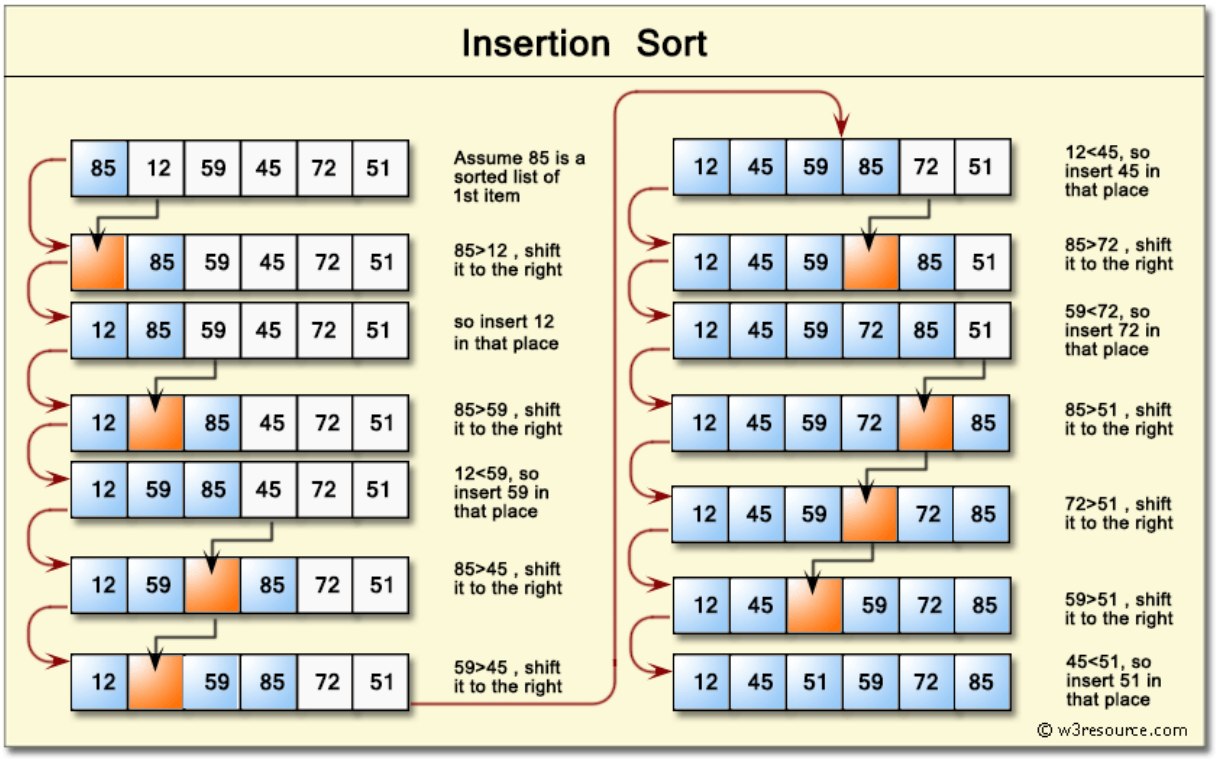
1. Sorting

Create an array of 10 random numbers (-1000, 1000) and try insertion sort. Do it step-by-step and yourself and try to understand. Add short comments after every line of code explaining what is the purpose and what is done there.

Insertion sort is a comparison-based algorithm that builds a final sorted array one element at a time. It iterates through an input array and removes one element per iteration, finds the place the element belongs in the array, and then places it there.



Example: https://www.w3resource.com/w3r_images/Insertion-sort-animation.gif

Tips: use smaller list in the beginning; print out the results after every step.

2. String manipulation

We have user accounts in 3 different possible formats:

a) "domain\username"
b) username@domain.com
c) i:0#.w|domain\username

Create methods for decoding and encoding user names and domains.

Decoding: takes the user account as parameter and prints name and domain (in upper case letters)
Examples:
DecodeAccount("ttu\mari.maasikas") -> Name: Mari Maasikas Domain: TTU
DecodeAccount("juhan.kannu.juurikas@ttu.ee") -> Name: Juhan Kannu Juurikas Domain: TTU
DecodeAccount("i:0#.w|microsoft\liina.miina") -> Name: Liina Miina Domain: MICROSOFT

Encoding: takes the string which contains the domain and name as parameter and prints account in 2 different formats (using '.' as a delimiter). If there are multiple forenames (*eesnimi*) then join them:
Examples:
EncodeAccount("ttu.ee mari maasikas") -> mari.maasikas@ttu.ee, ttu\mari.maasikas
EncodeAccount("ttu.ee juhan kannu juurikas") -> JuhanKannu.Juurikas@ttu.ee, ttu\juhankannu.juurikas

Reminder: use "@" sign to escape characters; example: string a= @"te/re"

3. Small accounting system

Create a program that can store up to 100 costs in order to create a small domestic accounting system. Users can enter their expenses and incomes (*sissetulekud/väljaminekud*). For each expense (or income) there should be following information:

- Date (8 characters: DDMMYYYY format)
- Description of expense
- Category – can be chosen by the user
- Amount (positive or negative)

The program should allow the user to perform the following operations:

1 - Add a new expense (the date should "look right": day 01 to 31 months from 01 to 12 years between 1000 and 3000). The description must not be empty. Needless to validate the other data.

2 - Show all expenses of a certain category (eg, "studies") between two certain dates (eg between "01012011" and "31122011"). Number is displayed, date (format DD / MM / YYYY), description, category in parentheses, and amount to two decimal places, all in the same line, separated by hyphens. At the end of all data show the total amount of data displayed.

3 - Search costs containing a certain text (in the description or category without distinguishing case sensitive). Number is displayed, the date and description (the description is displayed in the sixth truncated blank, if any spaces six or more).

4 - Modify an entry. User can choose the item to change by index. Before changing the text, display the current values. (And keep in mind that user might want to change only a few of the fields.)

5 - Delete some data, from the number that you enter (change value to "*null*")

6 –BONUS TASK; NOT MANDATORY: Sort data based on amount descending

7 - Normalize descriptions: convert description to lower case.
Example: DOSMTH->dosmth

8. View all expenses as summary – sums up all expenses in the array (both incomes and outgoings) and displays the total amount (the result can be either positive or negative).

Example: if there are 3 expenses in the array with sums 10, -50 and 13 then sum is -27.

Tips:

- Add some data for testing in the main method
- If you add test data and use the approach with counter (for keeping track of the amount of items added to the array), then don't forget to update it also.
- By default an array of 100 *null* costs are created (an array of 100 empty objects).
- Console.WriteLine() is your friend
- This is how you can check if the object has value:
  ```
  if(arrayOfExpenses[i]!=null){
      .. do smth
  }
  ```

Reading:
http://www.functionx.com/csharp/arrays/Lesson03.htm

Example usage:

```
Household accounts

1.- Add data.
2.- View all data.
3.- Search data.
4.- Modify data.
5.- Delete data.
6.- Sort alphabetically
7.- Fix spaces
8.- View filtered data
Q,T.-Quit.
Option: 1
Enter date (DDMMYYYY): 21122017
Enter Description: Bought some food
Enter category: Food
Enter the amount: 50

Household accounts

1.- Add data.
2.- View all data.
3.- Search data.
4.- Modify data.
5.- Delete data.
6.- Sort alphabetically
7.- Fix spaces
8.- View filtered data
Q,T.-Quit.
Option: 2
1 - 21/12/2017 - Bought some food -(Food) - 50.00

Household accounts

1.- Add data.
2.- View all data.
3.- Search data.
4.- Modify data.
5.- Delete data.
6.- Sort alphabetically
7.- Fix spaces
8.- View filtered data
Q,T.-Quit.
Option: 1
Enter date (DDMMYYYY): 03032018
Enter Description: School stuff
Enter category: university
Enter the amount: -69

Household accounts

1.- Add data.
2.- View all data.
3.- Search data.
4.- Modify data.
5.- Delete data.
6.- Sort alphabetically
7.- Fix spaces
8.- View filtered data
Q,T.-Quit.
Option: 4
Enter the record number: 1
Date (was 21122017; hit ENTER to leave as is):
Description (was Bought some food; hit ENTER to leave as is):
Category (was Food; hit ENTER to leave as is):
Amount (was 50; hit ENTER to leave as is): -90

Household accounts

1.- Add data.
2.- View all data.
3.- Search data.
4.- Modify data.
5.- Delete data.
6.- Sort alphabetically
7.- Fix spaces
8.- View filtered data
Q,T.-Quit.
Option: 2
1 - 21/12/2017 - Bought some food -(Food) - -90.00
2 - 03/03/2018 - School stuff -(university) - -69.00
```