WinForms and .NET
Cpt S 321 Homework Assignment
Washington State University

Submission Instructions:

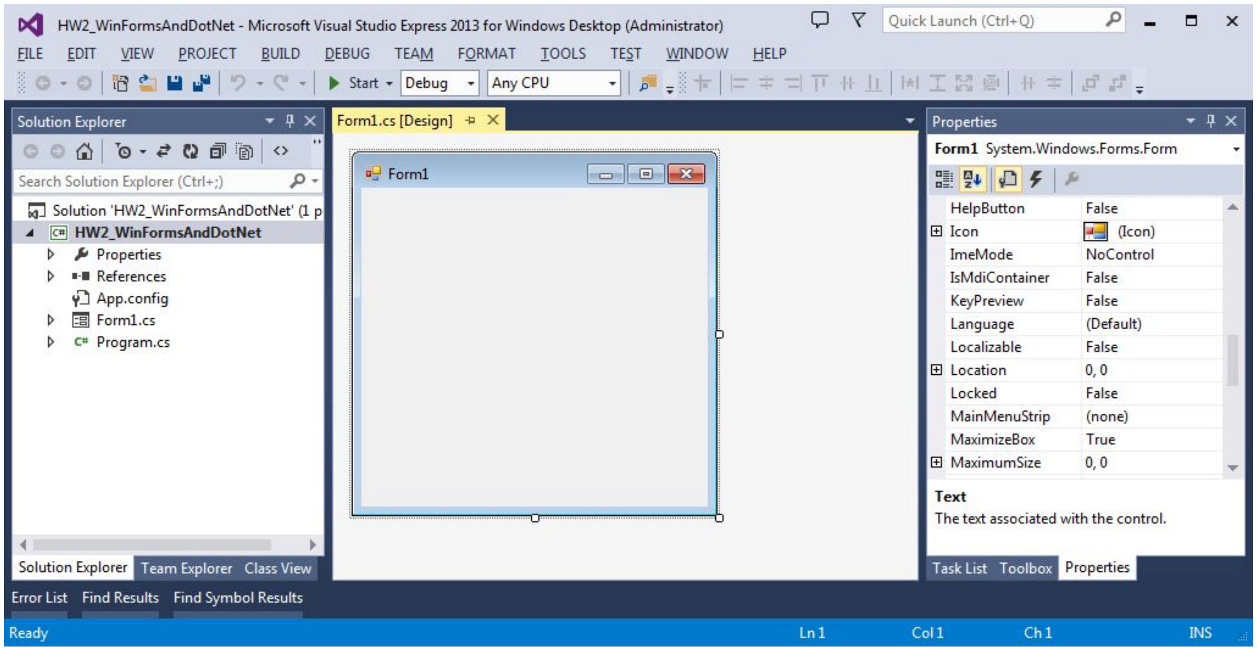
Check your project into your git repo.
Project / folder name: WinForms
Tag for final commit: WinForms-v1.0

Assignment Instructions:

Read all the instructions carefully before you write any code.

The starting materials will only include this PDF, a branch called WinForms, and a directory called WinForms. All code is for you to start with a new .NET WinForms project.

Using Visual Studio or SharpDevelop (http://www.icsharpcode.net/), create a new WinForms application in the WinForms directory. You will see the WinForms designer appear and it will look something like the following:

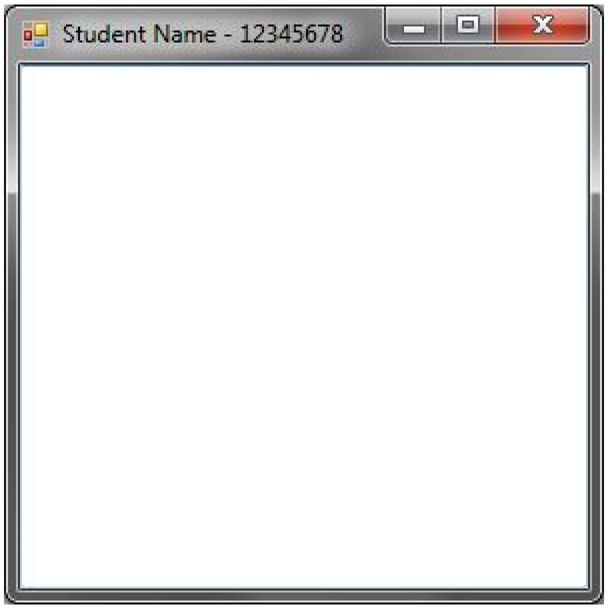


HelpButton	False
Icon	(Icon)
ImeMode	NoControl
IsMdiContainer	False
KeyPreview	False
Language	(Default)
Localizable	False
Location	0, 0
Locked	False
MainMenuStrip	(none)
MaximizeBox	True
MaximumSize	0, 0

Text
The text associated with the control.

In this application you will just be doing computations with simple text output for the results, but instead of displaying such results in the console/terminal window, you'll be displaying them in a text box within your interface. Add a TextBox control from the toolbox into your interface. Set the Multiline property to true and the Dock property to "Fill". Also set the Text property of your form to your name and ID #, so that it appears in the title bar.

After constructing the interface but before writing code, you can run the app to see what it looks like. Your interface should look something like the following at this point:



Next, create a function for the Load event of the form. This function will be executed after the form loads and it's where you'll implement the processing tasks described below. Visual Studio will flip over to your code at this point and your form class will look something like the following:

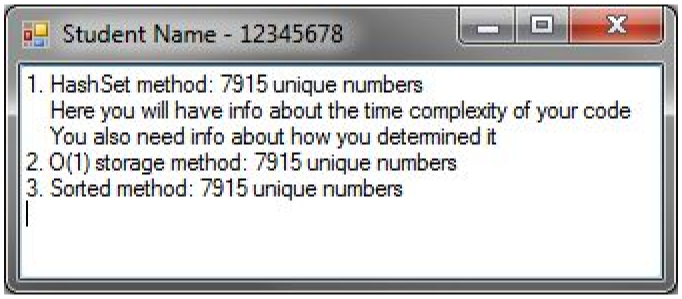
```
public partial class Form1 : Form
{
        public Form1()
        {
                InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {

        }
}
```

Use the Random class to create a list (System.Collections.Generic.List) with 10,000 random integers in the range [0, 20,000] (give or take a few hundred in that range is fine). Then determine how many distinct integers are in the list with 3 different approaches. Also, have them run in the order listed below. Do not change the order.
A note on what you're doing: you're taking an array of numbers and conceptually just removing duplicates and counting how many are left. If the input array was {1,1,3,5,6,6,7,7,7,9} then the distinct number set is {1,3,5,6,7,9}, implying 6 distinct numbers.

1. Do not alter the list in any way and use a hash table (.NET Dictionary aka C++ STL unordered_map) to determine the number of distinct integers in the list. The result will be included in the output, which is discussed more below. Also, include in the output information about the time complexity of this method. Be careful with this and describe how you determined it. Use a good amount of detail, as too sparse an explanation might not be worth full points. Use the MSDN documentation to assist you.
2. Next, do not alter the list in any way and determine the number of distinct items it contains while keeping the storage complexity (auxiliary) at O(1). Do not dynamically allocate any memory either. This means you cannot allocate additional lists, arrays, or containers of any sort. If you have an algorithm that would require more storage if the list had more items, then it is not O(1) storage complexity. There is some leniency on the time complexity requirements for this part. Due to the strict memory requirements your method may end up being quite slow next to the other 2, but it still should execute in a matter of a few seconds on any computer made within the last 5 years.
3. Then lastly sort the list (use built-in sorting functionality) and then use a new algorithm to determine the number of distinct items with O(1) storage, no dynamic memory allocation, and O(n) time complexity. Do not alter the list further after sorting it. Determine the number of distinct items in O(n) time (not including the sorting time, which you can ignore), where n is the number of items in the list.

Output:

Use either a string or StringBuilder object to compile text results from all of your methods. Put this string in the text box after completion, so that it shows up when the app runs. Examine your results before submitting your code and make sure they make sense. Obviously all 3 methods should be giving you the same answer. It should look something like the following (but of course with more details filled in where I have neglected to give you the answer) when you're finished:


Student Name - 12345678

```
1. HashSet method: 7915 unique numbers
   Here you will have info about the time complexity of your code
   You also need info about how you determined it
2. O(1) storage method: 7915 unique numbers
3. Sorted method: 7915 unique numbers
```

Point breakdown:
This assignment is worth 15 points

- 4 points for correct implementation of part 1
- 4 points for correct implementation of part 2
- 5 points for correct implementation of part 3
- 2 points is for the remainder of items, such as correct UI design, adequate comments in the code, proper coding style, and so on