Write a WPF program that animates a sorting algorithm. The program should start by creating a random array of 20 values of type `double`. Then, the program should create a visual representation of the array as discussed in class.

Use a constant in place of 20 so that the program can be made to work with different array sizes with minimal effort.

Here is more on the visual representation of the array. Specifically, each array item $a[k]$ will be represented by a rectangle added to a `Canvas` panel. The rectangles representing all the array elements will be arranged in order, with the rectangle corresponding to $a[k]$ being the $k$th element from the left edge of the panel. Each rectangle will have width $W/n$, where $W$ is the width of the canvas and $n$ is the length of the array. All rectangles will have their bottom hard against the bottom edge of the canvas, and the height of the $k$th rectangle will be $a[k] \times H$, where $H$ is the height of the canvas. Thus the rectangles will be arranged side by side in the form of a bar graph.

Fill each rectangle with a brush of a random color. You may use `LinearGradient` brushes to fill the rectangles: if you do, the gradient brushes should use random colors.

Select two algorithms: one from group 1, and one from group 2 below

**Group 1 is**

(1) Insertion Sort  
(2) Bubblesort

**Group 2 is**

(1) Mergesort  
(2) Quicksort

The goal of your program is to create an easy way to animate the two algorithms you selected, with minimal changes to your code. To achieve this flexibility, use either delegates or polymorphism. Your program should be able to switch from animating one algorithm to the other merely by changing one or two lines of code in one place in your program.

In class we talked about the two approaches to animation: on-line and off-line. We have not had a chance to talk about on-line, to use off-line for this assignment. To do an off-line animation, start with the input array $a[0..n-1]$ and create a visual representation of it. Then run your sorting algorithm off-line on an array $b[0..n-1]$ that is a copy of the input array. Save all swaps performed on by the sorting algorithm in a list. Finally, create and use a timer object to walk through the list of swaps, applying the swaps successively to the entries in the original array. As you do this, swap the fill properties in the rectangles corresponding to the array.
entries being swapped, and then set the height of the two rectangles to be canvas height proportion of the corresponding array entry.

You will need to make the time interval of the swap timer flexible, so we can experiment by running the timer at different speeds. Again, use a constant for the timer \texttt{Interval} property.

So, you have three things you want to be able to change with a minimum of fuss: The size of the array, the sorting algorithm, and the parameter for the time interval. You should consider making these three values application-wide data by storing them in the \texttt{Application} object’s resource dictionary.

You need to have a pretty good grasp of all concepts needed to do this project by Wednesday, the day of the test. You do not need to have the project \textit{all} done by then (It is good if you do), but you should have made sufficient progress on it so you know you understand all underlying concepts.

This project is due next Friday.