CS 340 PROJECT I: COMPARISON OF 2 SORTING ALGORITHMS

This project will compare the time complexities of InsertionSort and HeapSort.

The program:
To begin, you need to write an algorithm to fill an array with random numbers. It should take as input the size of the array and generate numbers based on the size of the array. For example, if the array has 50,000 cells, it should fill with numbers between 1 and 50,000 (in random order). You'll also need a second method to fill an array with sorted, consecutive numbers. It is also ok to use a built-in function to fill your arrays.

You will need to program both insertion sort and heap sort. It would be best to write them as separate classes.

In your main method, fill arrays with 100000, 200000, 300000, 400000 and 500000 numbers. For each array size, make one that is filled with random numbers, and one that is filled with sequential (or already sorted) numbers. This gives 10 arrays that must be sorted by two different sorting algorithms. Do the sorting and record how long each array took to sort. Use a timing function to get an exact time.

Implementing the Algorithms:
All algorithms (insertion sort and heapsort) should be implemented in Java, C++ or C#. We are going to use the heapsort code in a future project, so be very careful with it and make sure that it works properly. Also, you may not need all the heap operations to do this project, but you (or in a real-life scenario, someone else who uses your code) might need them later. So you must write all the methods contained in heapsort as shown in your book whether you actually use them in your sort or not.

Write A Report:
You must submit your working code, as well as a report that describes your results. The report should be 3-5 pages, double spaced, and submitted as a pdf file. In your report, you will need to include a table and plot that compares running times for each sorting method, each number of items sorted, and each random or already-sorted status. You will also want to create a plot (for example, a line graph) that shows how the times compare. Your plots should have input size on the x axis, and sorting time on the y axis. Obviously, the expectation is that each line will be at least a little bit upward sloping (showing that sorting time increases as input size increases). The plot should have 4 lines: InsertionSort with sorted and unsorted input, and HeapSort with sorted and unsorted input. Make this plot in a way that looks good and effectively conveys information.

You should discuss the following in your report:

1. What is the time complexity of each of the 4 sorting situations?
2. Do the plots reflect the expectation set by the time complexity?

What to Turn In:
Submit the source code as a zipped file. Submitting a Visual Studio Code project is ok.
Submit your report **as a separate PDF** file.
Grading will be according to this rubric:

Random number array generator	0 points: no implementation	5 points: implementation with errors	10 points: Correct implementation	
Insertion Sort	0 points: no implementation	5 points: implementation with errors	10 points: Correct implementation	
Max Heapify	0 points: no implementation	5 points: Implementation with serious errors	10 points: Implementation with minor errors	15 points: Correct implementation
Build Max Heap	0 points: no implementation	5 points: Implementation with serious errors	10 points: Implementation with minor errors	15 points: Correct implementation
Heapsort	0 points: no implementation	5 points: Implementation with serious errors	10 points: Implementation with minor errors	15 points: Correct implementation
Report: Plot	0 points: no plot	5 points: Plot shows minor errors	10 points: Plot implemented correctly	
Report: Discussion	0 points: no discussion	5 points: Discussion with major errors or omissions	10 points: Discussion with minor errors or omissions	15 points: Complete and correct discussion.
Quality of Report	0 points: Many writing errors.	5 points: Some writing errors or sloppiness.	10 points: High quality writing.	