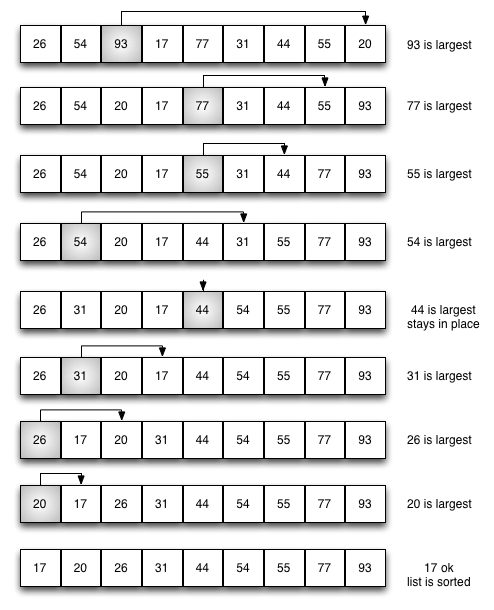
Vihas Gowreddy

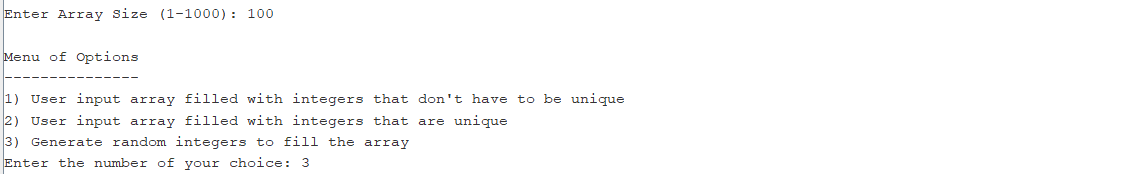
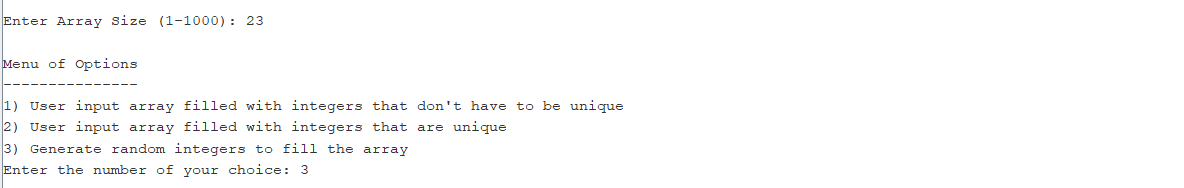
Professor Mazidi

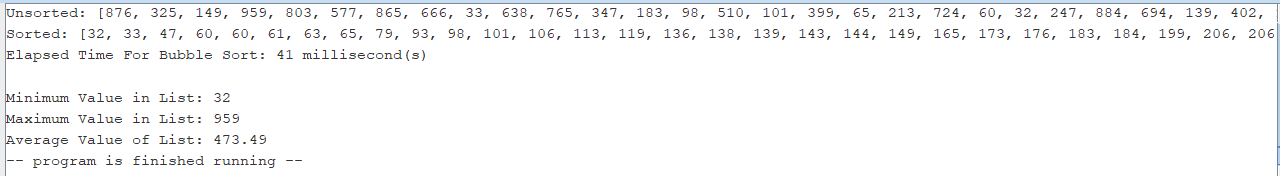
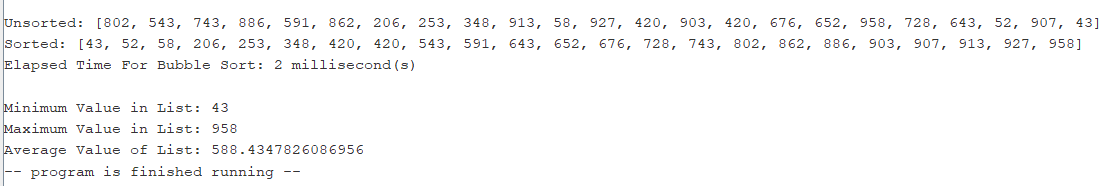
CS 3340.005

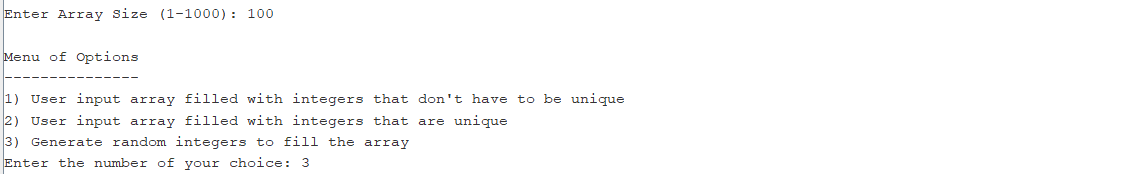
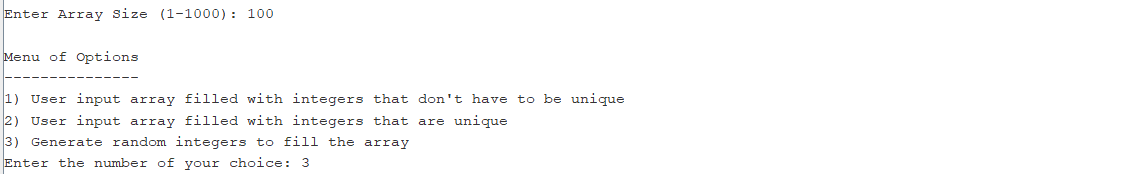
2 December 2018

Project External Documentation

 In general, this program looks at the various functions that can be applied to arrays of integers and the information that can be gleamed from these arrays. A huge focus of my project are the bubble sort and selection sort and the effect that different sizes of arrays has on the time complexity of each of these sorts. Looking at selection and bubble sort in java shows us that although both have the same worst-case time complexity, they actually have different best-case time complexities. The figures below show how bubble sort and selection sort normally work.

The worst-case time complexity is when the array is in opposite order so bubble sort and selection sort will both have n\*n(n-1)/2 comparisons (or O(n^2) time complexity) where n is the number of elements. The best case time complexity is when the array is already in order and bubble sort is more efficient in the case as it will only make one pass through the entire array with only n-1 comparisons while selection sort will still have n passes with n\*(n-1)/2 key comparisons therefore making bubble sort faster with its O(n) time complexity compared to selection sorts consistent O(n^2) time complexity. My code uses the syscall 30, or system timer, to calculate the amount of time in milliseconds it takes to sort an array of integers and allows the user some freedom to explore how changing the size of the array and inputting values into the unsorted array has a significant effect on how long the sort takes. My program starts off by asking the user for the size of the array they want to input within a range of 2-1000. If a value outside this range is inputted, a prompt will show telling the user that the input was not within the range and will force you to reenter a number until you enter one in the range. Once this occurs, a menu with three options will appear asking the user to choose one. The first option allows the user to input any nonzero values they want into the array of size x defined by the user in the previous step and a prompt will appear telling the user to reenter a value if 0 is entered. The second option allows the user to input unique nonzero values into the array with the program checking whether the input is unique or not. If an input is not unique, a prompt will appear telling the user so and allowing them to reenter a value. Because the possibility remains that the user may enter 0, input validation still occurs, and the prompt will appear if you try to enter 0. The last option fills the array with random values from the range of 0 – 1000. This option is your best friend for testing cases with large array sizes because of how tedious it is to enter many values. The next prompt will ask the user if they want to use bubble sort or selection sort to order the values. Based on which menu number you enter, the corresponding function will take place. Finally, the user has the option to choose whether to sort the array in ascending or descending order. All the menus have input validation and if a value not on the menu is inputted, a prompt will appear saying the value was not on the menu and make the user enter a valid number. All inputs crash when a value that is not an integer is typed as input type validation was not required for this project. After the user inputs all this information, the unsorted array and sorted array will be printed with proper formatting and underneath, the execution time, minimum value, maximum value, and average value of the array are given. The average is outputted as a decimal.



Testing the difference in runtime between bubble sort and selection sort will require the user to input larger array sizes as system time is not given as a decimal and noticeable difference in milliseconds is not usually present with small array sizes.

