Project 2 Eric Sabelhaus - Design Documentation

1. Design Decisions
   1. HandleSorting
      1. Constructor - assigns array as argument to private array which is then processed using public void methods
      2. quickSort(int left, int right) - Accepts integers of left and right most points to sort over. Implements findPivot(left, right) as it recursively processes the data of the provided array.
      3. insertionSort() - Implemented as a public void. It now processes the private array initialized within the constructor of HandleSort.
      4. selectionSort() - The same changes to insertSort were applied here.
      5. printMyArray() - This was implemented to validate that data is sorted actually being sorted. It iterates over an array, printing the element at each index.
      6. jumbleSome() - This method is intended for demonstration of the best case for the insert sort method.
      7. clearCounters() - This method sets the value of both private counting integers to zero. It is necessary for testing customized data to exercise different cases for each algorithm.
      8. toS() - This method provides a final output of the metrics counted during sorting in string format. It is formatted in an easily consumable string for creating data analysis documentation.
   2. RandomData
      1. Constructor - creates an array of random integers of the size provided
      2. getRandomData() - Implemented to get the array back once it is initialized in constructor.
   3. Added statements
      1. Selection Sort
         1. I added a comparison count statement for each iteration of the inner loop, as an if statement is processed every iteration
         2. I added three assignment counts for each swap
      2. Insertion Sort
         1. I added two assignment counters inside the main loop for the assignment of "temp" and "myArray[j +1]"
         2. I added a comparison counter to the main loop, as it does a comparison with the while statement
         3. I added an assignment counter within the inner loop, as it creates an assignment of "myArray[j + 1]"
      3. Quick Sort
         1. I added two comparison counters inside of the main loop findPivot(), as each while statement compares two data points
         2. I added three assignment counts for each swap
2. Data Choices
   1. SelectionSort
      1. I simply chose one set of data random data, incremented by 100, in size multiples of 100 from 100 to 1000. They are all the same randomized data, as the performance of Selection sort will always be O(n^2)
   2. InsertionSort
      1. Best Case
         1. I chose to presort the data, and shuffle 3 data points. The best case of insertion sorting is seen from presorted data where a few random elements need to be sorted.
      2. Average/Worst Case
         1. I used the same data as in selection sort. Average and worst cases are seen when using completely randomized data
   3. QuickSort
      1. Best Case
         1. The best case for quick sort is its base case, this is determined by the number of elements to be sorted. I used the same data as in selection sort to display this case.
      2. Average/Worst Case
         1. The worst case is seen when sorting a few elements into an already sorted array. I presorted an array using quicksort, but I left the first 17 elements unsorted. When sorting through a second time to get the remaining elements, it uses excessively more cycles to sort based upon the amount of elements being sorted.
3. Interpretation of Data Analysis
   1. Data Diagram Layout
      1. I chose line graphs to display my data.
      2. The X axis represents the length of the array times 100
      3. The line graphs gave a valid depiction of how the data operations between the algorithms differ.
   2. Data Table information
      1. The first column provides the Length of the array which was sorted
      2. The second column provides the number Comparisons during the sort
      3. The third column provides the number of Assignments during the sort
      4. The fourth column provides the total of Comparisons and Assignments done by the sorting algorithm
4. Lessons Learned
   1. The primary lesson I learned was that each sorting algorithm has a specific case were it performs the best, and that used correctly, they can truly improve the efficiency of an application.
   2. I also learned, that certain problems can benefit from the use of different sorting algorithms.
      1. Specifically, if I were to have data that is brand new, I may use a quick sort on it initially, to sort and store that data.
      2. As new data is added to the comparable object, I would simply run an insertion sort to quickly sort the new data into its place.