Fundamental Structures Lab 8 March 5, 2020

Submission Instructions: This lab is due *March 12*. Upon completion you *must upload* .py file (or .cpp if done in C++) to Canvas using the following convention Username_Lab#.py by 11:59 PM of the due date. You must also show your completed lab to your instructor during the lab to receive credit.

"The simplest answer is often correct." - Occam's Razor

Problems

In this lab, we will implement a **sort class** in either C++ or Python with three sorting algorithms (methods) and compare their running time. Note that you must implement the pseudocode presented in the lab and are not allowed to use another "flavor" of the algorithm. Translate the given pseudocode into the language of your choice. Ensure your program is well commented.

1. Generate random integers 10, 100, 1000, 10,000, and 100,000 starting from 1. (10 points)
2. Implement a test case to sort these numbers in increasing order using the following algorithms: Selection Sort, Quick Sort, and Merge Sort. (10 points)
3. Plot the running time against the length of the different randomly generated vectors. In your report explain your conclusions about the efficiency of various sorting algorithms based on their running time. (10 points)
4. Bonus (10 points): Create a txt file with 20 rows of unsorted numbers. Read the numbers from the file and store in an array or vector. Using your MergeSort function, sort the numbers in the array and then write the sorted numbers to another txt file. Also, display the sorted numbers to the standard output.

Note: The name of the class should be called Sort. The names of the three sorting methods should be called SelectionSort, QuickSort, and MergeSort. Test case will be of this form:

Sort.SelectionSort([Vector of Random Integer]).

You can only pass one parameter which is the vector of random numbers to the sorting methods. The sort class implements only the default constructor with no input parameter.

Selection Sort (20 points):

1. Set the marker (U) for the unsorted section at the end of the list.
2. Repeat steps 3 - 10 until the unsorted section has just one element.
    3. Set the marker (L) for the largest so far at the beginning of the list.
    4. Set the current marker (C) on the second element of the list.
    5. Repeat steps 6-8 until C is to the right of U.
        6. If the element at C is larger than the element at L then
            7. Set L to the current position C.
        8. Move C to the right one position.
    9. Exchange the element at L with the element at U.
    10. Move U to the left one position.

11. Stop.

Quick Sort (20 points):

1. If the list to sort has more than one element then
    2. If the list has exactly two elements then
        3. If the two element are out of order then
            4. Exchange them.
    5. Otherwise
        6. Exchange the median of the first three elements with the first.
        7. Set the pivot marker (P) on the first of the elements.
        8. Set the lower marker (L) on the second element.
        9. Set the upper marker (U) on the last element.
        10. Repeat steps 11-16 until L is to the right of U.
            11. Repeat step 12 until the element at L is larger than the element at P.
                12. Move L to the right one position.
            13. Repeat step 14 until the element at U is less than the element at P.
                14. Move U to the left one position.
            15. If L is to the left of U then
                16. Exchange the element at L with the element at U.
        17. Exchange the element at P with the element at U.
        18. Apply quick sort to the sublist of elements to the left of U.
        19. Apply quick sort to the sublist of elements to the right of U.
20. Otherwise
    21. Stop.

Merge Sort (20 points): As shown in the class.

Sort Class (10 points): All the functions (methods) implemented in the class (abstract data type). This class has no properties but implements the three given sorting algorithms.

No attendance = Zero Credit
Have fun!