CMPUT 175 - Lab 9: Sorting

Submit by: Nov 23, 2020 *Demo in your lab starting: Nov 24, 2020*

Goal: Gain an in-depth understanding of the Selection Sort and Merge Sort algorithms, and practice using recursion.

Exercise 0:

Download and complete the two sorting practice worksheets on eClass to become more familiar with various sorting algorithms.

- Practice – Bubble Sort, Selection Sort, Insertion Sort
- Practice – Merge Sort, Quicksort

Be prepared to talk about your work with your TA for any questions related to the Selection Sort and the Merge Sort.

Exercise 1:

In this exercise, you will implement two sorting algorithms: selection sort and merge sort. Implement both of the algorithms using recursion, and compute their time to sort different lists of data.

Task 1: Selection Sort

a. Implement a Selection Sort algorithm using recursion to sort a list of numbers **in descending order**. Start with the file, **`exercise_1.py`**, downloaded from eClass. DO NOT RENAME THIS FILE.

b. Complete the function **`recursive_selection_sort()`** in this file. You may want to define your own additional functions to complete the task, but they must be called inside of **`recursive_selection_sort()`**. Your function should sort a list in-place, so it will not need to return the sorted list.

c. Test your solution with the tests provided in **`test_selection_sort.py`** file. (i.e. just run it.) Make sure your sorting function passes all the tests.

Task 2: Merge Sort

a. Implement Merge Sort using recursion to sort a list of numbers **in descending order**. Do this by completing the function **`recursive_merge_sort()`** in **`exercise_1.py`**. You may want to define your own additional functions to complete the task, but these functions must be called inside **`recursive_merge_sort()`**. This function does NOT sort the list in-place, so don't forget to return the sorted list from the function.

b. Test your solution with the tests provided in **`test_merge_sort.py`** file. Make sure your sorting function passes all the tests.

Once you have successfully completed Task 1 and 2, run the **`exercise_1.py`** file. Its **`__main__`** portion compares how long each sort function takes to sort lists of (a) randomly generated integers, (b) ascending integers, and (c) descending integers. This part is already written for you. **Which sorting algorithm takes the least amount of time to sort each list? Why?**

Exercise 2:

In this exercise, you will sort a list of complex objects. Each object corresponds to a student, with private attributes: id, name, and mark (see file **`exercise_2.py`**). A number of Student objects will be created according to data stored in the text file, **`student_list.txt`** (provided), where each line of the input file corresponds to information for one student. The newly created **`Student`** objects will be stored in a list. This list should be sorted according to student marks (ascending order). To accomplish this, you are asked to complete the following tasks:

Tasks: Sorting Objects

a. Complete the method **`__lt__(self, anotherStudent)`** to compare the **receiver object** with **anotherStudent** object. Specifically, you will determine if the mark of the **receiver object** is less than that of **anotherStudent**. Return either **True** or **False**. Note that this is a special method in Python, and will be called when you compare 2 **`Student`** objects using the < operator.

b. Complete the function, **`recursive_merge_sort()`**, which will use recursive Merge Sort to sort a list of **`Students`** in ascending order by their mark. You can adapt your implementation of Merge Sort from Exercise 1. Use the < operator to compare two students' marks for sorting. You may define your own functions to complete the task. However, they may only be called inside of the **`recursive_merge_sort()`** function. Remember that this method must return a new list containing the sorted **`Students`** because Merge Sort does not sort in-place.

Once you have successfully completed these tasks, run **`exercise_2.py`**. Its output should look like the following (formatted as one long column, not 2 columns):

```
Original data:                    Sorted data:
 - 129246, George Daniel, 89       - 140775, Mary Simon, 35
 - 139897, Amir Jahani, 76         - 149122, Brian Johnson, 42
 - 139256, Sarah Kylo, 90          - 139887, Samuel Picard, 55
 - 136898, Breanne Lipton, 82      - 139897, Amir Jahani, 76
 - 140991, Robert George, 95       - 145887, Ali Gendi, 77
 - 126775, Jeff Anderson, 84       - 136898, Breanne Lipton, 82
 - 136781, Xialing Liu, 86         - 126775, Jeff Anderson, 84
 - 145887, Ali Gendi, 77           - 136781, Xialing Liu, 86
 - 140775, Mary Simon, 35          - 142886, Georgina Moore, 88
 - 142886, Georgina Moore, 88      - 129246, George Daniel, 89
 - 149122, Brian Johnson, 42       - 139256, Sarah Kylo, 90
 - 139887, Samuel Picard, 55       - 140991, Robert George, 95
```