**CMPSC 412 – Lab-3** (25 points)

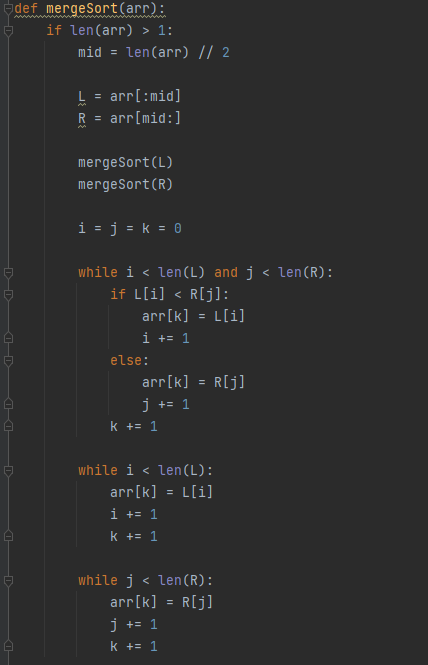
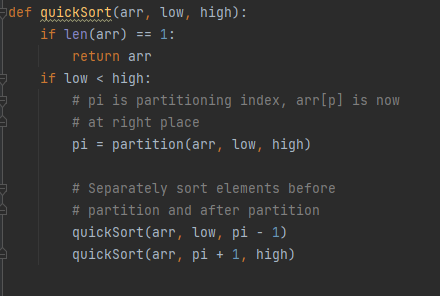
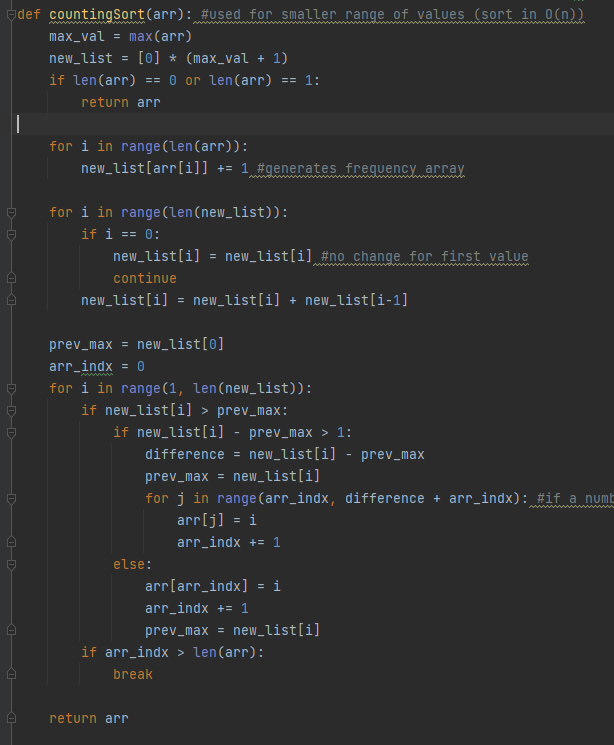
**Sorting Algorithms**

**Due date: 2/10/2022**

**Note:** attach screenshots of your program and results under each programming exercises. Please make sure that the screenshot is readable. Don’t attach a very small screenshot image.

**Lab Exercises:**

1. Write the algorithm for the following sorting algorithms and perform the time complexity analysis:

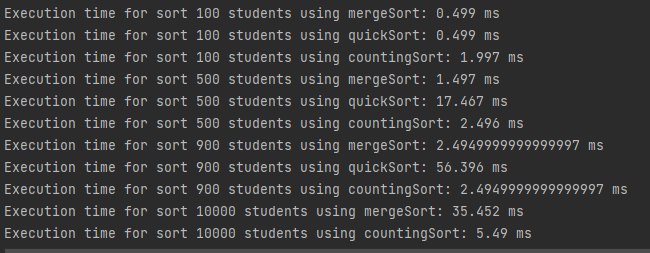
* Merge Sort
  + 
  + Merge sort first recursively paritions elements a total of log n times. Once they are partitioned into arrays of size 1, they are merged back together by going over the entire array and is therefore O(nlogn)
* Quick Sort
  + 
  + The worst case of quicksort occurs when the array is already sorted and the pivot is either the smallest or largest value in the array. In this case we will start with n operations, then n-1 and so on, resulting in O(n^2)
* Counting Sort
  + 
  + Counting sort uses 2 consecutive for loops (not nested) that each run in O(N). After that is a nested for loop. The outer loops will run n times, but the number of executions for the inner loop, is dependent on the difference between the largest and smallest value in the list. Therefore the time complexity is O(N + K) where K is the range b/w the smallest and largest value.

1. Generate a list of random N integers with values ranging from 1 to 10000. Write a function which takes the entire list and sorts the entire list. Implement the sorting using merge sort and quick sort. Print out how much CPU time it took to sort the data for each sorting algorithms. You can import a library to calculate the CPU time. Tabulate your readings for 50,000 and 100,000 random integers for 3 trails (different set of random numbers for each trail).

|  |  |  |  |
| --- | --- | --- | --- |
|  | Merge Sort | Quick Sort | Counting Sort |
| 100 students | .499 ms | .499 ms | 2.495 ms |
| 500 students | .998 ms | 17.468 ms | 2.495 ms |
| 900 students | 2.496 ms | 58.891 ms | 2.495 ms |

From the above table, we can see that in the first trial, both merge sort and quick sort executed in under 1 ms, while counting sort took 2.5 seconds. However, as we increased the input size, the times for quick sort grew exponentially. Merge sort also increased its execution time but not nearly as fast as quick sort. Meanwhile, regardless of the input size, counting sort was able to execute in 2.5ms. At the maximum input size of 900, merge sort and counting sort executed at almost the same time. Since quicksort has a maximum recursion depth, it cannot be implemented for inputs greater than 900 in python. In order to get a better understanding of merge sort vs counting sort, I ran these 2 algorithms on a dataset of 10,000 students.

|  |  |  |
| --- | --- | --- |
|  | Merge Sort | Counting Sort |
| 10,000 students | 35.452 ms | 5.49 ms |



Here we can see that the time for merge sort on 10000 elements had a drastic jump, while counting sorts increase was much smaller. However, if the dataset contained one extreme outlier, we would likely see a major jump in time for counting sort, while merge sort would remain the same.