**Report**

**Assignment 2**

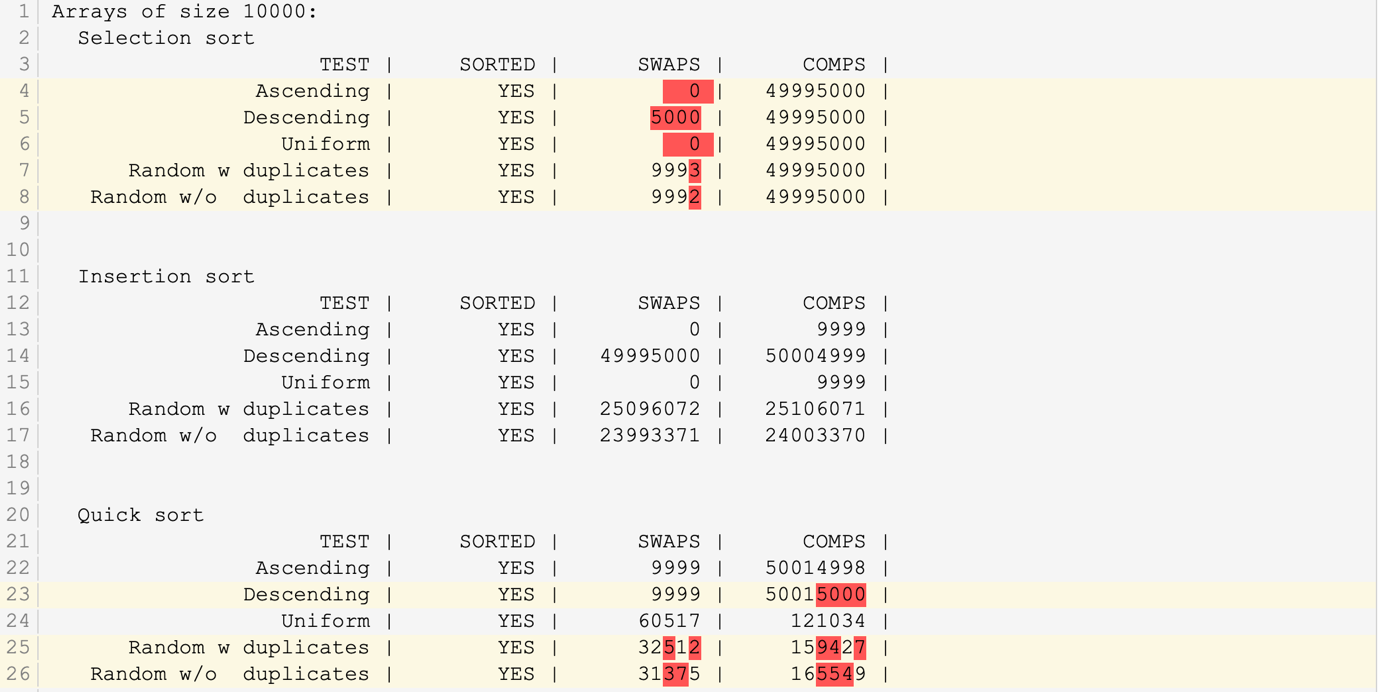
**Task2: (Justification of design choices in Quicksort Implementation)**

Since we didn’t have to edit the header file of t2\_skeleton.c , I implemented another function called quickSort\_recursive. This function implemented QuickSort traditionally and had nothing different or special. It took the array, low index and high index as its arguments.

In the quicksort function, I simply called the quicksort\_recursive function.

For the partition function, I took the pivot as the **first** element, as opposed to the last element, that was taught in class. Talking specifically about task2, it doesn’t really make a difference whether the first or the last element is taken as the pivot, since we had to implement quicksort function on all types of arrays.

**Task3: (Results of Task 3 and comment on them)**



1. Selection Sort:
   1. I got different number of Swaps than Submitty. That’s because in my code, when two values are the same, I don’t count it as swapping. And technically, for a uniform array and an ascending array, no matter which sorting we apply, we shouldn’t have to swap any values. Hence, the zero swaps.
   2. Since Selection Sort has both Best Case and Worst Case as O(n^2), I expected the number of comparisons to be the same for all cases. And that’s exactly the output that I got.
2. Insertion Sort:
   1. I have got the same number of swaps and comparisons as Submitty. I implemented a very traditional way of Insertion Sorting, where I considered a single element to be sorted and gradually increased the size of my ‘Sorted’ array by inserting elements into it in a sorted fashion.
   2. Since Insertion Sort has a best case time complexity of O(n), I expected the number of comparisons to be the least for the ascending array and the uniform array. And that’s exactly the output that I got. 9999 comparisons for Ascending and Uniform array of size 10,000 .
3. Quick Sort:
   1. Since I took the first element of the array to be the **pivot**, I’ve got different number of swaps and comparisons w.r.t Submitty.
   2. While I had only 2 more comparisons on a descending array than submitty, I got a whopping 3,625 less comparisons for a random array with duplicates.
   3. However, I discovered that taking the first or the last element as the pivot for descending arrays is not a good approach, since the time complexity of implementing Quick Sort would reach O(n^2).

**Task4: (How would you get the top 5 games for each of the last 20 years?)**

I am assuming that I already have an array of structs that has loaded all the values in the CSV file. I will sort this array in two parts in my main function.

**Part 1:** I will implement Quick Sort on this array, sorting the array on the basis of release years only. I chose Quick Sort because this is a random with duplicates type of array, and as per my Submitty results, Quick Sort performs the best in this type of array with time complexity O(nlogn), as compared to Insertion Sort and Selection Sort with time complexity O(n^2)

**Part 2:** Having sorted the array on the basis of Release Years, I will again implement Quick Sort to sort the array on the basis of scores now. However, there will be a small change in the Partition Function. I will insert an If statement, checking if the release years are the same.

I will swap two Structs only if their release years are same, else I will move on to the next one.