Assignment 2: Sorting

ADVIK BAHADUR

21365420

3rd Year Electronic and Computer Engineering

Task 2 & 3:

→The pivot point was chosen randomly. This was done to reduce the possibility of having the worst possible pivot point $O(n^2)$. While this does introduce the need to perform extra computations to get the random pivot point and additional swapping. But given the data below it can be seen that it greatly reduces the number of computations in the long run.

```
Arrays of size 10000:
Selection sort
                     TEST |      SORTED |       SWAPS |       COMPS |
                Ascending |         YES |       10000 |    49995000 |
               Descending |         YES |       10000 |    49995000 |
                  Uniform |         YES |       10000 |    49995000 |
      Random w duplicates |         YES |       10000 |    49995000 |
   Random w/o  duplicates |         YES |       10000 |    49995000 |


Insertion sort
                     TEST |      SORTED |       SWAPS |       COMPS |
                Ascending |         YES |           0 |       10000 |
               Descending |         YES |    49995000 |    50005000 |
                  Uniform |         YES |           0 |       10000 |
      Random w duplicates |         YES |    25096072 |    25106072 |
   Random w/o  duplicates |         YES |    23993371 |    24003371 |


Quick sort
                     TEST |      SORTED |       SWAPS |       COMPS |
                Ascending |         YES |       99215 |      171986 |
               Descending |         YES |       81043 |      168027 |
                  Uniform |         YES |       27591 |    50035001 |
      Random w duplicates |         YES |       84399 |      178784 |
   Random w/o  duplicates |         YES |       90921 |      181499 |
```

→ As can be seen, Quick sort greatly reduces the computations, while keeping the swaps low. Furthermore, it should be noted that the result clearly shows that each sort algorithm worked to accomplish the same result.

→It is interesting to note that the selection sort produced exactly 10000 swaps and 49995000 computations for each test as it does a comparison for each item in the array. Giving it a complexity of $O(n^2)$

→The insertion sort, while performing considerably fewer comparisons, had a lot more swaps

Task 4

→ For this task I decided to create a struct called ‘Games’ that had parameters defined by the CSV file headers:

i) Name
ii) Platform
iii) Score
iv) Year

→ Then I decided to create an array of ‘Games’ that will store the details for all the data.

→ A CSV parser was then run to get information from the CSV file:

1) I used a for loop to iter through the Games Array defining each Game one-by-one
2) I used the ‘next field’ function, to get each cell by cell.
3) Then I assigned each variable to the correct parameter of the Game

→ Once all the data was extracted, the next step was to sort the Game Reviews dependent on the Review score.

→ For this, I decided to make use of the Insertion Sort method, which we had implemented in the previous task.

→ To implement insertion sort on an array of Games structs based on their Score parameter, you would need to modify the comparison function to compare Score instead of the default comparison function. The algorithm would then work by comparing each Score value to those in the sorted part until it finds its correct position and inserts it into the sorted part.

→ The reason for this was due to the simpler implementation and running procedure compared to quicksort.

→ I soon realized how this wasn’t a great idea, since the CSV file contained close to 20000 reviews, mostly jumbled, and the cost of running/computation was quite high. An algorithm like quick sort would have been much better for such an implementation. This kind of sorting would be much more efficient for smaller arrays, and for arrays that are mostly sorted.

→ If I were to do this task again, for larger datasets similar to the one provided in this task, I would use the likes of Quick Sort which is much more efficient in terms of computation.

→ To verify the accuracy of the program:

1) I implemented a print function that would print the array iteratively.
2) To test it, I first printed the array unsorted and then printed the array sorted.
3) Looking at the scores, I was able to verify that my program worked correctly.