3D5A: Data Structures
Assignment 2:
Rahul Seth
17302557

Task 1: Quick Sort

The objective for Task 1 was to implement quicksort algorithm on an array with 10 values and 10000 values. There were 5 test cases for each of the array. I made use of majorly 3 functions to implement sorting algorithm. There was the swap function, the partition function and the quicksort function. For my partition function, I picked the last element of the array as the pivot and compared and swapped all elements smaller than the pivot on the left of the pivot and all elements higher than the pivot on the right.

```
int partition(int array[], int low, int high){
    int pivot = array[high];
       int i = (low - 1);

       for (int k = low; k <= high - 1; k++)
       {
           no_of_comparasions++;
           if (array[k] < pivot)
           {
               i++;
               swap(&array[i], &array[k]);
               no_of_swaps++;
           }
       }
       swap(&array[i + 1], &array[high]);
       return (i + 1);
}
```

Listing 1: Partition Code

The following is the quicksort function. Here the variable pivot index makes use of the partition function and calls quicksort recursively to sort the array. I have taken a flag as swapped which returns true if sorting has been successful.

```
int quicksort(int array[], int low, int high){

    int swapped = 0;

    if(low>high)
        return 0;

    if(low<high){
        int pivot_index = partition(array, low, high);
        quicksort(array, low, pivot_index - 1);
        quicksort(array, pivot_index + 1, high);
        swapped=1;
    }
    return swapped;
}
```

Listing 2: Quick Sort Code

The following tables are observations of the performance of quicksort for this this task.

Test Cases (10 elements)	**Number of swaps**	**Number of comparisons**
Random Array	8	23
Random Array no duplicates	16	26
Ascending Sorted	45	45
Descending Sorted	20	45
Uniform Array	0	45

Table 1: Task 1 observations (Array of 10 elements)

Test Cases (10000 elements)	**Number of swaps**	**Number of comparisons**
Random Array	84987	156852
Ascending Sorted	49995000	49995000
Descending Sorted	74985001	99990000
Uniform Array	74985001	149985000

Table 2: Task 1 observations (Array of 10000 elements)

Looking at the number of swaps and comparisons from the above tables, it can be seen that the performance of quick sort method degraded with bigger datasets.

Task 2: Insertion Sort

The objective for this task to implement another sorting algorithm. I chose to implement insertion sort algorithm. I implemented this algorithm from moving over from the second element to the last comparing each element to the previous one as the index iterates.

```
void insertionsort(int array[], int n){

   int index, key, j;

   for(index=1; index < n; index++){

      no_of_comparasions++;
      key = array[index];
      j = index - 1;

      while(j >= 0 && array[j] > key){
         no_of_comparasions++;
         array[j+1] = array[j];
         no_of_swaps++;
         j = j - 1;
      }
      array[j+1] = key;
   }
}
```

Listing 2: Insertion Sort Algorithm

The following table is the observation on the performance of insertion sort for this this task.

Test Cases (10 elements)	Number of swaps	Number of comparisons
Random Array	11	20
Random Array no duplicates	10	19
Ascending Sorted	0	9
Descending Sorted	36	45
Uniform Array	0	9

Table 3: Task 2 observations (Array of 10 elements)

Test Cases (10 elements)	Number of swaps	Number of comparisons
Random Array	24863994	24873993
Ascending Sorted	0	9999
Descending Sorted	49975003	49985002
Uniform Array	0	9999

Table 4: Task 2 observations (Array of 10000 elements)

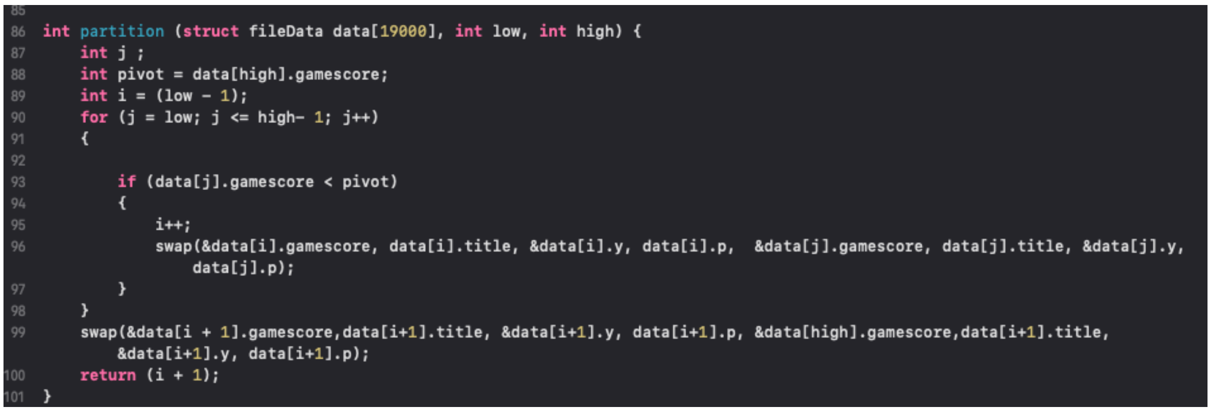
From the above tables, it can be seen that the performance of Insertion sort was drastically better than quick sort. The number of swaps and comparisons reduce drastically when we use insertion sort for bigger data sets.
Another observation that was made during the debugging of the program was that insertion sort was much faster in execution as compared to the quicksort algorithm.

Task 3 – Sorting csv file

The objective for Task 3 was to sort a csv file provided to us. We were given game reviews for the past 20 years and we had to output the 10 most popular games. Here I made use of the quicksort algorithm from Task 1 with some modifications to it.

```
void swap(int* a,char t1[MAX_BUFFER], int *yr1, char p1[MAX_BUFFER], int* b, char
t2[MAX_BUFFER], int *yr2, char p2[MAX_BUFFER]) {
  int tmp1 = *a;
   *a = *b;
   *b = tmp1;
   char tmp2[MAX_BUFFER];
   strcpy(tmp2,t1);
   strcpy(t1,t2);
   strcpy(t2,tmp2);
   int tmp3 = *yr1;
   *yr1 = *yr2;
   *yr2 = tmp3;
   char tmp4[MAX_BUFFER];
   strcpy(tmp4,p1);
   strcpy(p1,p2);
   strcpy(p2,tmp4);
}
```

Listing 3: Modified Swap function



```c
int partition (struct fileData data[19000], int low, int high) {
    int j ;
    int pivot = data[high].gamescore;
    int i = (low - 1);
    for (j = low; j <= high- 1; j++)
    {

        if (data[j].gamescore < pivot)
        {
            i++;
            swap(&data[i].gamescore, data[i].title, &data[i].y, data[i].p,  &data[j].gamescore, data[j].title, &data[j].y,
                data[j].p);
        }
    }
    swap(&data[i + 1].gamescore,data[i+1].title, &data[i+1].y, data[i+1].p, &data[high].gamescore,data[i+1].title,
        &data[i+1].y, data[i+1].p);
    return (i + 1);
}
```

Listing 4: Partition function

The partition and the quicksort function were almost identical as the one used in task 1. The idea behind the sorting for the ign list was to implement quicksort on the score columns and chose a pivot from there. After pivot selection, the program compares score with the respective platforms, titles and the release years and swaps the values when required.

I made use of various other functions to fetch the data from the csv and another function to store the data in a list created with the same parameters as the ign csv.

The next aim of the task was to design a strategy to get the top 5 games for each of the years. The way I would approach this problem is to divide the csv into each year and implement quicksort for each year in the csv.

****

References

1. https://www.geeksforgeeks.org/quick-sort/
2. https://www.geeksforgeeks.org/linked-list-set-1-introduction/
3. https://hackr.io/blog/quick-sort-in-c
4. https://www.tutorialspoint.com/data_structures_algorithms/insertion_sort_algorithm.htm
5. 3D5A: Data Structures & Algorithms Lectures