ASSIGNMENT-2
DATA STRUCTURES
(CSU33D05)

SUBMITTED TO:
Dr. Mélanie Bouroche

SUBMITTED BY:
PRACHI SINGHROHA
(Student ID: 21355131)

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Declaration concerning plagiarism

I have read and I understand the plagiarism provisions in the General Regulations of the *University Calendar* for the current year, found at http://www.tcd.ie/calendar

I have completed the Online Tutorial in avoiding plagiarism 'Ready, Steady, Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write

STUDENT NUMBER: 21355131

SIGNED:

DATE: 30.10.2021

Task 1:

Generated arrays according to types given in the assignment.

Task 2:

Implementation of following algorithms has been done.
1. Insertion Sort
2. Selection Sort
3. Quick Sort

Quick Sort I used works with partition function which takes last element as pivot and place it at such a position that all elements smaller than the pivot are on its left and the greater ones on the right. The partition function is as follows:

```
int partition(int array[], int beg, int end){
   int pivotIndex = array[end];
   int pIndex= beg-1;
   for(int i =beg; i<end; i++){
      if(array[i] < pivotIndex){
         number_comparisons++;
         pIndex++;
         swap(&array[pIndex], &array[i]);
         number_swaps++;
      }
   }
   swap(&array[pIndex+1], &array[end]);
   number_swaps++;
   return pIndex+1;
}
```

Task 3:

My results as as follows for an array without duplicates.

Number of swaps:

	Size = 10	Size = 1000	Size = 10000
Selection Sort	10	1000	10000
Insertion Sort	35	240009	24090469
Quick Sort	23	5871	91958

Number of comparisons:

	Size = 10	Size = 1000	Size = 10000
Selection Sort	45	499500	49995000
Insertion Sort	25	239009	24080469
Quick Sort	33	10357	159782

Insertion sort works well when the size of the array is small because it is stable and requires less memory but as the number increases it becomes slow because the number of comparisons increases. *Selection sort* on works very similar to insertion sort but number of comparisons are more than compared to other sorting algorithms because it sorts by choosing the smallest element from the array and exchange it until it is at its correct position. *Quick Sort* on the other hand is the works the most efficiently out the sorting methods used specifically when the size of array is large because it linearly scans he input and linearly partitions it i.e. it is cache friendly. Also asymptotic average runtime of Quicksort is better than the sorting algorithms used here. As we can see from the table above the output is as expected!

Task 4:

My approach:

i. To get top 10 most popular games of last 20 years - I created a linked list which inserts values using *insertValue* function and *insertion sorting*. The other two functions used to perform this task, *fetch_field* and *next_field*, have been taken from assignment 0. My main reason for choosing insertion sort here is that it is stable algorithm.
ii. To get top 5 games for each of last 20 years - We will basically fetch the games from one particular year by sorting them year-wise then we'll apply the same method was used to get the top 10 games, the only difference being that instead of entire 20 years we'll be applying the sorting on the games of a particular year.