Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058, India
(Autonomous College Affiliated to University of Mumbai)

Experiment No.	1-b
Aim	Experiment on finding the running time of an algorithm.
Name	Harsh Mukesh Jain
UID No.	2021300048
Class & Division	SE Comps A (C – Batch)
Date of Performance	6-02-2023
Date of Submission	12-02-2023

Theory/Experiment:

1.INSERTION SORT

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

PSEUDO CODE:-

```
insertionSort(arr)
for i = 0 to arr.length() – 2
        for j = (i + 1) to j > 0
                if arr[j - 1] > arr[j]
                        swap (arr[j - 1] , arr[j])
        end for
end for
```

ALGORITHM:-

This algorithm sorts an array of items by repeatedly taking an element from the unsorted portion of the array and inserting it into its correct position in the sorted portion of the array.

1. The procedure takes a single argument, **‘arr’**, which is a list of sortable items.
2. The outer for loop starts at index **‘0’** and runs for **‘arr.length()-2’** iterations.
3. The inner while loop starts at the current index j of the outer for loop and compares each element to its left neighbor. If an element is smaller than its left neighbor, the elements are swapped.
4. The inner while loop continues to move an element to the left as long as it is smaller than the element to its left.
5. Once the inner while loop is finished, the element at the current index is in its correct position in the sorted portion of the array.
6. The outer for loop continues iterating through the array until all elements are in their correct positions and the array is fully sorted.

CODE :-

```
void insertionSort(int *arr , int size)
{
  //Storing current time in begin
   time_t begin = time(NULL);

   for(int i = 0 ; i < size - 1 ; i++)
   {
      for(int j = i + 1 ; j > 0 ; j--)
      {
        //Comparing rightmost element of sorted element and leftmost element of
unsorted element.
         if(arr[j - 1] > arr[j])
         {
                int temp = arr[j - 1];
                arr[j - 1] = arr[j];
                arr[j] = temp;
         }
      }
   }
//Storing end time in end.
   time_t end = time(NULL);
//Printing (end - begin)
   printf("%d\n",(end - begin) );
}
```

2.SELECTION SORT

Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted portion. This process is repeated for the remaining unsorted portion of the list until the entire list is sorted.

PSEUDO CODE:-

```
selectionSort(arr)
for i = 0 to arr.length() – 1
        for j = (i + 1) to j < arr.length() - 1
                if (arr[j] < arr[min]) min = j;
        end for
        if(min != i)  swap (arr[i] , arr[min])
end for
```

ALGORITHM:-

This algorithm sorts an array of items by repeatedly finding minimum element in the array and replacing at it's correct index.

1. The procedure takes a single argument, **'arr'**, which is a list of sortable items.
2. The outer for loop starts at index **'0'** and runs for **'arr.length()-1'** iterations.
3. The inner while loop starts at the current index j from i + 1 for finding the minimum element in the arr upto size-1 . If the minimum element is found , then it's index is stored in min variable.
4. Once the inner loop is completed , check if the min is equal to i or not.If it's not equal then swap arr[i] and arr[min].
5. Repeat the outer loop until it reaches the last index.

CODE :-

```
void selectionSort(int *arr , int size)
{
	time_t begin = time(NULL);

	// One by one move boundary of unsorted subarray
	for (int i = 0; i < size; i++)
	{
      // Find the minimum element in unsorted array
		int min = i;
		for (int j = i+1; j < size; j++)
		{
			if (arr[j] < arr[min]) min = j;
		}

      // Swap the found minimum element with the first element
		if(min != i)
		{
			int temp = arr[i];
			arr[i] = arr[min];
			arr[min] = temp;
		}
	}
	time_t end = time(NULL);
	printf("%d\n", (end - begin));
}
```

CODE FOR GIVEN QUESTION:-

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void insertionSort(int *arr , int size)
{
  //Storing current time in begin
   time_t begin = time(NULL);

   for(int i = 0 ; i < size - 1 ; i++)
   {
      for(int j = i + 1 ; j > 0 ; j--)
      {
         //Comparing rightmost element of sorted element and leftmost element of unsorted
element.
          if(arr[j - 1] > arr[j])
          {
                int temp = arr[j - 1];
                arr[j - 1] = arr[j];
                arr[j] = temp;
          }
      }
   }
//Storing end time in end.
   time_t end = time(NULL);
//Printing (end - begin)
   printf("%d\n",(end - begin) );
}

void selectionSort(int *arr , int size)
{
          time_t begin = time(NULL);

          // One by one move boundary of unsorted subarray
          for (int i = 0; i < size; i++)
          {
      // Find the minimum element in unsorted array
                int min = i;
                for (int j = i+1; j < size; j++)
                {
                      if (arr[j] < arr[min]) min = j;
```

```
                }

        // Swap the found minimum element with the first element
                if(min != i)
                {
                        int temp = arr[i];
                        arr[i] = arr[min];
                        arr[min] = temp;
                }
        }
        time_t end = time(NULL);
    printf("%d\n", (end - begin));
}
int main()
{
        //Generating random numbers in "Random_No.txt"
        FILE *fptr = fopen("Random_No.txt" , "w");
        int size = 100000 , r;
        for(int i = 0 ; i < size ; i++) {
                r = rand()%100000 + 1;
                //Printing random numbers in Random_No.txt file
                fprintf(fptr , "%d\n" , r);
        }
        //Closing Random_No.txt file
        fclose(fptr);

        int block = 100;
        for(int i = block ; i <= size ; i = i + 100)
        {
                //Reading random numbers from "Random_No.txt"
                fptr = fopen("Random_No.txt" , "r");
                //Copying block by block elements into arr and arr_copy arrays
                int *arr = (int *)malloc(sizeof(int) * i);
                int *arr_copy = (int *)malloc(sizeof(int) * i);
                for(int j = 0 ; j < i ; j++)
                {
                        fscanf(fptr , "%d" , &r);
                        //printf("%d\t",r);
                        arr[j] = r;
                        arr_copy[j] = arr[j];
```

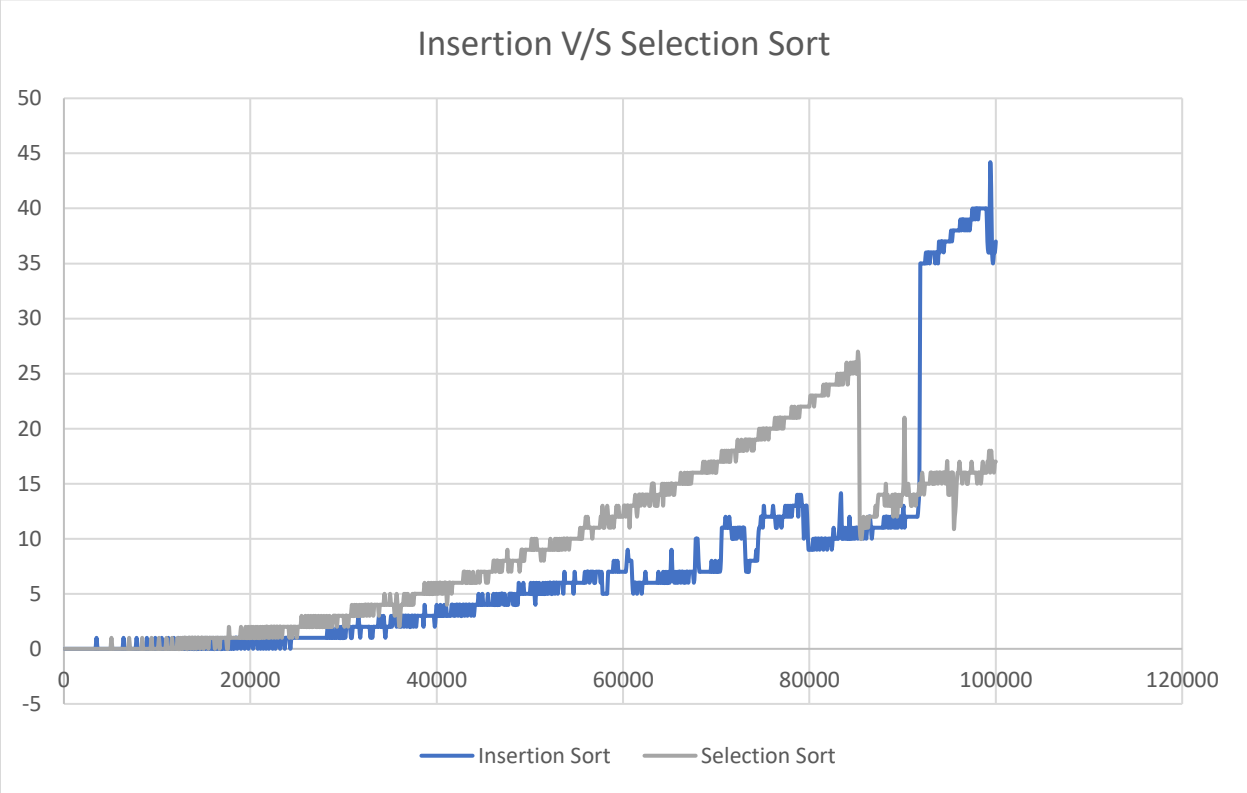
```
            }

            //Calling insertionSort and selectionSort function for calculating time
            insertionSort(arr , i);
            selectionSort(arr_copy , i);

            printf("\n\nAfter Sorting  %d elements i.e (%d block) :-\n",i,(i / 100 ));
            for(int j = 0 ; j < i ; j++)
            {
                  printf("%d\t",arr[j]);
            }
            //printf("%d\n",i);
            free(arr);
            free(arr_copy);
            fclose(fptr);
      }

}
```

INPUT:-

Generated random 1,00,000 numbers in Random_No.txt file using rand() function and used this input as 1000 blocks of 100 integer numbers to Insertion and Selection sorting algorithms.

OUTPUT:-

1) Stored the randomly generated 100000 integer numbers to a text file named Random_No.txt.
2) Below represent the graph of 2 function i.e Insertion Sort and Selection Sort plot on y axis versus the number of blocks marked on x axis.



Initially the time taken by insertion sort was less (i.e for sorting 80,000 elements or upto block 800) as compare to time taken by selection sort.
But after Block No. 850 time taken by selection sort is more than time taken by insertion sort.

3)OBSERVATION:-

The space complexity of both the algorithm is O(1). This means that the memory usage of both the algorithms remains constant which is independent from the size of input elements.
Both Insertion Sort and Selection Sort requires a single extra memory space to temporarily store the value being inserted at the correct position.
Thus it makes algorithm more efficient in terms of spcae complexity.

However, the difference between the two algorithms lies in their time complexity. Insertion Sort has a time complexity of $O(N^2)$, which means its performance decreases as the size of the input data increases. On the other hand, Selection Sort has a time complexity of $O(N^2)$ as well, which means it has a similar performance to insertion sort for larger input data.

CONCLUSION:-

By performing this experiment , I understood the concept of two sorting algorithms i.e Insertion Sort and Selection Sort.
Using time.h header file , I calculated time taken by both algorithms to sort 1,000 blocks i.e 1,00,000 elements and plotted a graph in Excel.
It was clear from the graph that the performance of insertion sort is best for small data inputs in comparison to the performance of selection sort.