Name	Prakhar Gupta
UID No.	2021300040
Class & Division	S.E. COMPS A (BATCH C)
Experiment No.	1b

Aim: Experiment on finding the running time of an algorithm.

Theory:

Insertion sort– It works similar to the sorting of playing cards in hands. It is assumed that the first card is already sorted in the card game, and then we select an unsorted card. If the selected unsorted card is greater than the first card, it will be placed at the right side; otherwise, it will be placed at the left side. Similarly, all unsorted cards are taken and put in their exact place.

Selection sort– It first finds the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array. In this algorithm, the array is divided into two parts, first is sorted part, and another one is the unsorted part. Initially, the sorted part of the array is empty, and unsorted part is the given array. Sorted part is placed at the left, while the unsorted part is placed at the right. In selection sort, the first smallest element is selected from the unsorted array and placed at the first position. After that second smallest element is selected and placed in the second position. The process continues until the array is entirely sorted.

Algorithm:

1. Initialize a for loop to iterate over the elements of the array "a". Let's call the variable used for iteration "i".
2. Within the for loop, initialize a variable "t" and assign it the value of "a[i]".
3. Initialize another variable "j" with the value of "i - 1".
4. Within the for loop, create a nested for loop. The nested for loop will continue as long as "j>=0" and "a[j]>t".
5. Within the nested for loop, swap the values of "a[j]" and "a[j + 1]".
6. After swapping the values, decrement the value of "j" by 1.
7. End the nested for loop.
8. After the nested for loop, set the value of "a[j + 1]" equal to "t".
9. Initialize a for loop to iterate over the elements of the array "b". Let's call the variable used for iteration "i".
10. Within the for loop, initialize a variable "minIndex" to the value of "i".
11. Initialize another for loop to iterate over the remaining elements of the array "b", starting from "i + 1". Let's call the variable used for iteration "j".
12. Within the second for loop, compare the value of "b[j]" with the value of "b[minIndex]". If "b[j]" is less than "b[minIndex]", update the value of "minIndex" to "j".
13. End the second for loop.
14. After the second for loop, check if "minIndex" is not equal to "i". If it is not equal, swap the values of "b[i]" and "b[minIndex]".

Code:

```
#include <stdio.h>
#include<stdlib.h>
#include<time.h>
void main()
```

```
{
	int n=0;
	for(int k=0; k<(100000/100); k++)
	{
		n=n+100;
		int num[n];
		int insert[n];
		int select[n];
		int j, min;
		clock_t start_t, end_t;
	   double total_t;
		printf("%d\t",n);
		for(int i=0; i<n; i++)
		{
			num[i]=rand() % 10;
			insert[i]=num[i];
			select[i]=num[i];
		}
		start_t = clock();
	   for (int i = 1; i < n; i++)
	   {
	      int a = insert[i];
	      j = i - 1;
	      while (j >= 0 && insert[j] > a)
	      {
	         insert[j + 1] = insert[j];
	         j = j - 1;
	      }
	      insert[j + 1] = a;
	   }
	   end_t = clock();
		total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC;
		printf("%f\t", total_t  );
		start_t = clock();
	   for (int i = 0; i < n; i++)
	   {
	      min = i;
	      for (j = i+1; j < n; j++)
	      {
	        if (select[j] < select[min])
	         {
	         min = j;
	         }
			}
	      if(min != i)
	      {
	         int temp=select[i];
	         select[i]=select[min];
	         select[min]=temp;
	      }
	   }
	   end_t = clock();
	   total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC;
		printf("%f\n", total_t  );
	}
}
```

Observation:

n	Insertion sort	Selection sort
100	0	0
200	0	0
300	0	0
400	0	0
500	0	0
600	0	0
700	0.000998	0.000988
800	0.001017	0.001002
900	0.001002	0.001993
1000	0.001998	0.001
1100	0.001001	0.001006
1200	0.001011	0.001993
1300	0.000998	0.001
1400	0.000998	0.002027
1500	0.001	0.001995
1600	0.001996	0.001996
1700	0.001996	0.001996
1800	0.001997	0.002991
1900	0.002994	0.002993
2000	0.004989	0.002994
2100	0.002994	0.003993
2200	0.002993	0.003992
2300	0.003991	0.004988
2400	0.003991	0.005985
2500	0.004989	0.012992
2600	0.008979	0.007031
2700	0.006015	0.006983
2800	0.00701	0.007015
2900	0.00701	0.008017
3000	0.007968	0.008951
3100	0.00798	0.009033

3200	0.008032	0.015994
3300	0.010001	0.010011
3400	0.01097	0.011019
3500	0.01002	0.018992
3600	0.011971	0.012016
3700	0.011994	0.012966
3800	0.01197	0.01401
3900	0.012968	0.014014
4000	0.013965	0.016017
4100	0.014953	0.016999
4200	0.014981	0.017001
4300	0.015958	0.016986
4400	0.015987	0.017998
4500	0.017001	0.019985
4600	0.017946	0.019915
4700	0.017983	0.020973
4800	0.01898	0.020979
4900	0.019951	0.021971
5000	0.020944	0.023947

Conclusion: Successfully wrote a program to implement insertion and selection sort. Made relevant observations and found the running time of both sorting methods. We can conclude that insertion sort is faster.