NAME:	Eshan Bhuse
UID:	2021300013
SUBJECT	Design and analysis of algorithm
EXPERIMENT NO :	1B
AIM:	Experiment on finding the running time of an insertion sort and selection sort.
ALGORITHM	**Main function:** step 1: start Step2: call generate_numbers() function Step 2: call operation()function Step 3: end generate_numbers() function: step 1: start step 2: crate the file pointer step 3: open the file in writing mode step 3: starts the loop from 0 to 100000 step 4: insert the 100000 random numbers in the file step 5: close the file handle step 6: end operation function(): step 1: start step 2: open the file in reading mode step 3: start the loop from 0 to 100000 and increment it with 100 step 4: create two arrays step 5: start the loop from 0 to j and scan the data from file

	step 6: before sorting store the time step 7: perform selection sort step 8: check the time after the sorting step 9: calculate the time taken by the algorithm step 10: before sorting store the time step 11: perform selection sort step 12: check the time after the sorting step 13: calculate the time taken by the algorithm **Selection sort:** step 1: start step 2: start the loop step 3: initialize the min element step 4: start the loop from i+1 to n step 5: check the condition: if jth element less than min element then minimum element will be j. step 6: if minimum element not equal to i, then initialize variable t with array(i) perform ith element = array of min array(min) = t step 7: end. **Insertion sort:** Step 1: start Step 2: start the loop from 1 to n Step 3: initialize j with i-1 Step 4: current element is array(i) Step 5: if array(key)>0 and j>=0 Repeat below steps 6,7 Step 6: j+1th element will jth element Step 7: decrement j Step 8: array(j+1) = current. Step 9: end.

THEORY:

Insertion Sort Algorithm: In this article, we will discuss the Insertion sort Algorithm. The working procedure of insertion sort is also simple. This article will be very helpful and interesting to students as they might face insertion sort as a question in their examinations. So, it is important to discuss the topic.

Insertion sort 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.

The same approach is applied in insertion sort. The idea behind the insertion sort is that first take one element, iterate it through the sorted array. Although it is simple to use, it is not appropriate for large data sets as the time complexity of insertion sort in the average case and worst case is **$O(n^2)$**, where n is the number of items. Insertion sort is less efficient than the other sorting algorithms like heap sort, quick sort, merge sort, etc.

Time Complexity:

Case	Time Complexity
Best Case	$O(n)$
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

- **Best Case Complexity -** It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of insertion sort is **O(n)**.
- **Average Case Complexity -** It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of insertion sort is **$O(n^2)$**.
- **Worst Case Complexity -** It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of insertion sort is **$O(n^2)$**.

Selection Sort Algorithm:

In this article, we will discuss the Selection sort Algorithm. The working procedure of selection sort is also simple. This article will be very helpful and interesting to students as they might face selection sort as a question in their examinations. So, it is important to discuss the topic.

In selection sort, the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array. It is also the simplest algorithm. It is an in-place comparison sorting algorithm. 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.

The average and worst-case complexity of selection sort is **$O(n^2)$**, where **n** is the number of items. Due to this, it is not suitable for large data sets.

Time Complexity:

Case	Time Complexity
Best Case	$O(n^2)$
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

- **Best Case Complexity -** It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of selection sort is **$O(n^2)$**.
- **Average Case Complexity -** It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of selection sort is **$O(n^2)$**.
- **Worst Case Complexity -** It occurs when the array

elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of selection sort is **$O(n^2)$**

Tabular Difference between Insertion Sort and Selection Sort:

Sr.No	Insertion Sort	Selection Sort
1.	Inserts the value in the pre-sorted array to sort the set of values in the array.	Finds the minimum / maximum number from the list and sort it in ascending / descending order.
2.	It is a stable sorting algorithm.	It is an unstable sorting algorithm.
3.	The best-case time complexity is $\Omega(N)$ when the array is already in ascending order. It have $\Theta(N^2)$ in worst case and average case.	For best case, worst case and average selection sort have complexity $\Theta(N^2)$.
4.	The number of comparison operations performed in this sorting algorithm is less than the swapping performed.	The number of comparison operations performed in this sorting algorithm is more than the swapping performed.
5.	It is more efficient than the Selection sort.	It is less efficient than the Insertion sort.
6.	Here the element is known beforehand, and we search for the correct position to place	The location where to put the element is previously known

		them.	we search for the element to insert at that position.
	7.	The insertion sort is used when: • The array is has a small number of elements • There are only a few elements left to be sorted	The selection sort is used when • A small list is to be sorted • The cost of swapping does not matter • Checking of all the elements is compulsory • Cost of writing to memory matters like in flash memory (number of Swaps is O(n) as compared to O(n2) of bubble sort)
	8.	The insertion sort is Adaptive, i.e., efficient for data sets that are already substantially sorted: the time complexity is **O(kn)** when each element in the input is no more than **k** places away from its sorted position	Selection sort is an in-place comparison sorting algorithm

PROGRAM:

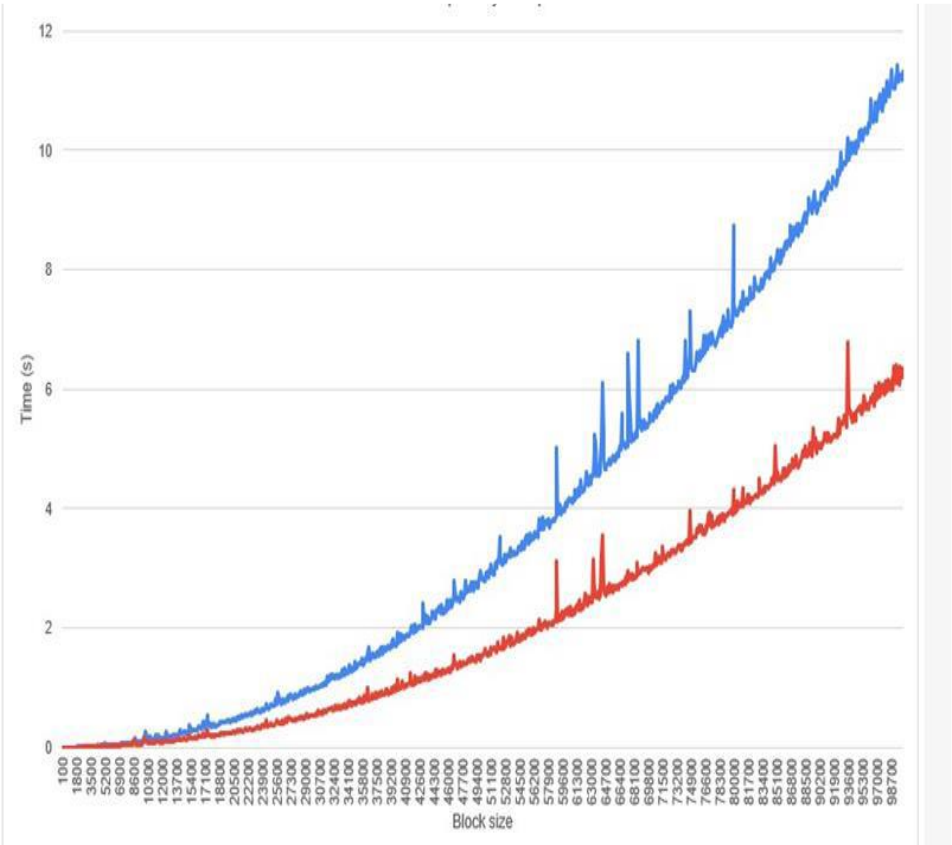
```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<time.h>
void selectionsort(int arr[],int n)
{
for(int i=0;i<n;i++)
{
int min_ind=i;
for(int j=i+1;j<n;j++)
{
if(arr[j]<arr[min_ind]) min_ind=j;
}
if(min_ind!=i)
{
int t=arr[i];
arr[i]=arr[min_ind];
arr[min_ind]=t;
}
}
}
void insertionsort(int arr[],int n)
{
for(int i=1;i<n;i++)
{
int j=i-1;
int key=arr[i];
while(j>=0 && arr[j]>key)
{
arr[j + 1] = arr[j]; j = j - 1;
}
arr[j + 1] = key;
}
}

void generate_numbers()
{

FILE *ptr;
ptr=fopen("number.txt","w");
for(int i=0;i<100000;i++)
```

```c
FILE *ptr;
ptr=fopen("number.txt","w");
for(int i=0;i<100000;i++)
{
    fprintf(ptr,"%d\n",rand() % 100000);
}
fclose(ptr);
}
void operation()
{
FILE *ptr;
ptr=fopen("number.txt","r");
for(int j=0;j<100000;j+=100)
{
int arr1[j]; int arr2[j];
for(int i=0;i<j;i++)
{
fscanf(ptr,"%d\n",&arr1[i]);
}
for(int i=0;i<j;i++)
{
arr2[i]=arr1[i];
}
clock_t start_selection=clock(); selectionsort(arr1,j);
clock_t end_selection = clock(); double currs=(double)(end_selection-
start_selection)/CLOCKS_PER_SEC;


clock_t start_insertion=clock(); insertionsort(arr2,j);
clock_t end_insertion=clock(); double curri=(double)(end_insertion-
start_insertion)/CLOCKS_PER_SEC; printf("\n%d\t%f\t%f",j,currs,curri);
}
}
int main()
{
generate_numbers(); operation();
return 0;
}
```

OBSERVATION:	**Graph of Selection sort and Insertion sort:**  Here, the graph of insertion sort and selection sort is a curve which is exponential and increases with time but the time of insertion sort is less than selection sort and therefore insertion sort is more efficient than selection sort.

CONCLUSION:	Successfully performed the experiment of Selection sort and insertion sort and found the running time for each sorting algorithm in C Language. Concluded that insertion sort is efficient than selection sort.

PROGRAM:	