NAME:	Rahil Ankhad
UID:	2021300003
SUBJECT:	DAA
EXPERIMENT NO:	1b
DATE OF PERFORMANCE:	02/2/23
DATE OF SUBMISSION: **AIM:**	09/2/23 Experiment on finding the running time of an algorithm.
Theory:	**Details**: The understanding of running time of algorithms is explored by implementing two basic sorting algorithms namely Insertion and Selection sorts. These algorithms work as follows. **Insertion sort:** It works like the sorting of playing cards in hands. It is assumedthat 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 putin 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 the sorted part, and another one is the unsorted part. Initially, the sorted part of the array is empty, and the 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.

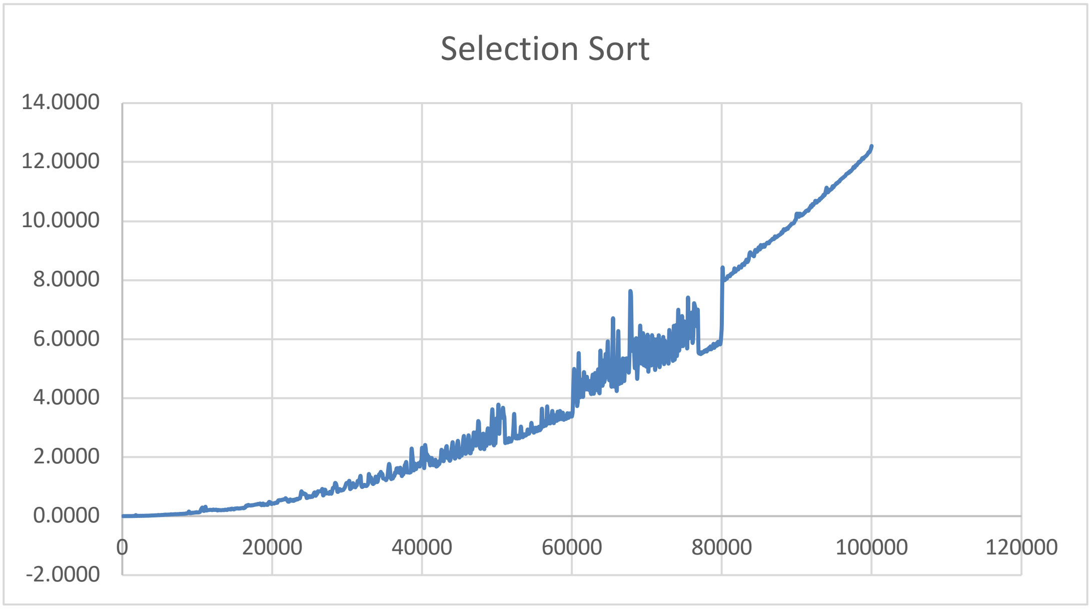
Problem Definition & Assumptions:

For this experiment, you need to implement two sorting algorithmsnamely Insertion and Selection sort methods. Compare these algorithms based on time and space complexity. Time required sorting algorithms can be performed using high_resolution_clock:: now() under namespace std::chrono.You have to generate 1,00,000 integer numbers using the C/C++Rand function and save them in a text file. Both the sorting algorithm uses these 1,00,000 integer numbers as input as follows. Each sortingalgorithm sorts a block of 100 integer numbers with array indexes numbers A[0..99], A[0..199], A[0..299],..., A[0..99999]. You need to use the high_resolution_clock::now() function to find the time required for100, 200, 300.... 100000 integer numbers.

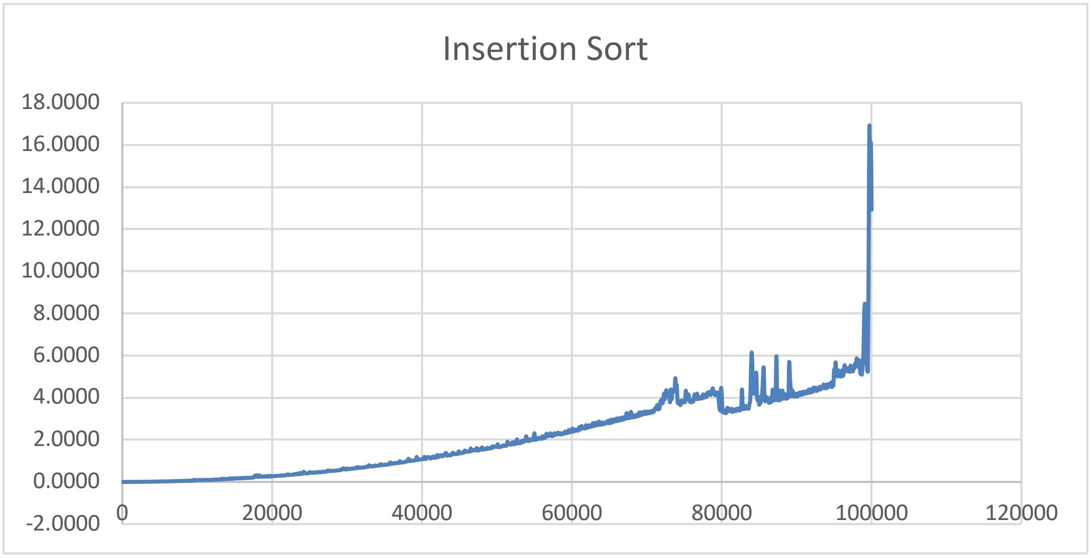
Note: You must use C/C++ file processing functions for reading andwriting randomly generated 100000 integer numbers.

Algorithm:	Insertion Sort Function: • Iterate from arr[1] to arr[N] over the array. • Compare the current element (key) to its predecessor. • If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element. Selection Sort Function: • Initialize minimum value(min_idx) to location 0. • Traverse the array to find the minimum element in the array. • While traversing if any element smaller than min_idx is found then swap both the values. • Then, increment min_idx to point to the next element. • Repeat until the array is sorted. GetInput Function: • Function to make 100000 random numbers to sort and put into a text file Readfile Function: • Function to read numbers from the text file Main Function: 1. Make a menu-driven function and ask the user his choice of sorting technique2. If insertion sort, call the function and calculate the time interval at every 100 numbers getting sorted up to 1000 times/block. 3. If Selection sort, call the function and calculate the time interval at every 100 numbers getting sorted up to 1000 times/block. 4. Else if, invalid input. **Code link:** https://github.com/Rahank262003/DAA_EXP

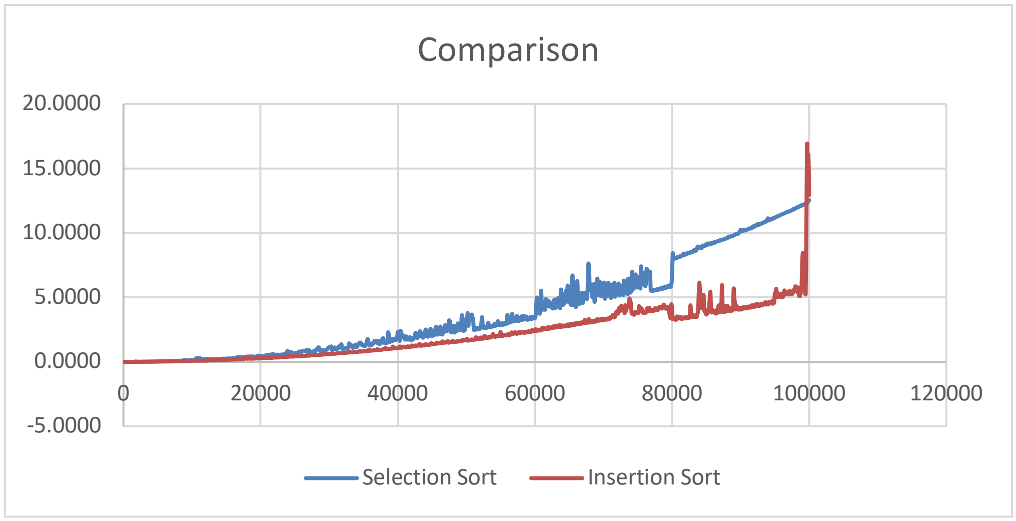
Graph:



Selection Sort
14.0000
12.0000
10.0000
8.0000
6.0000
4.0000
2.0000
0.0000
-2.0000
0
20000
40000
60000
80000
100000
120000



Insertion Sort
18.0000
16.0000
14.0000
12.0000
10.0000
8.0000
6.0000
4.0000
2.0000
0.0000
-2.0000
0
20000
40000
60000
80000
100000
120000

	
Conclusion:	Performing this experiment, we understood the implementation of insertion sort and selection sort with their time complexity using the graph and we can identify the better algorithm for given numbers of values.

Observation:	From this experiment we observed that for value of n ranging from 0-100000 insertion sort is better in terms of time complexity than selection sort, but when n starts approaching 100000 insertion sort sees a very sharp increase in time complexity.