**AIM :** Evaluation of bubble Sort, insertion sort, selection sort, merge sort efficiency.

**EXPERIMENT:** Implement insertion sort, selection sort bubble sort and merge sort. The number of inputs elements has to be passed from command line arguments. The elements have to be generated randomly within the code. Compute:

a. Check the performance of program by varying the number of elements.

b. Compute the time taken by each case (for particular number of inputs) n = 10, 100, 1000, 10000, 1000000.

c. Plot a graph with number of inputs Vs time taken in seconds.

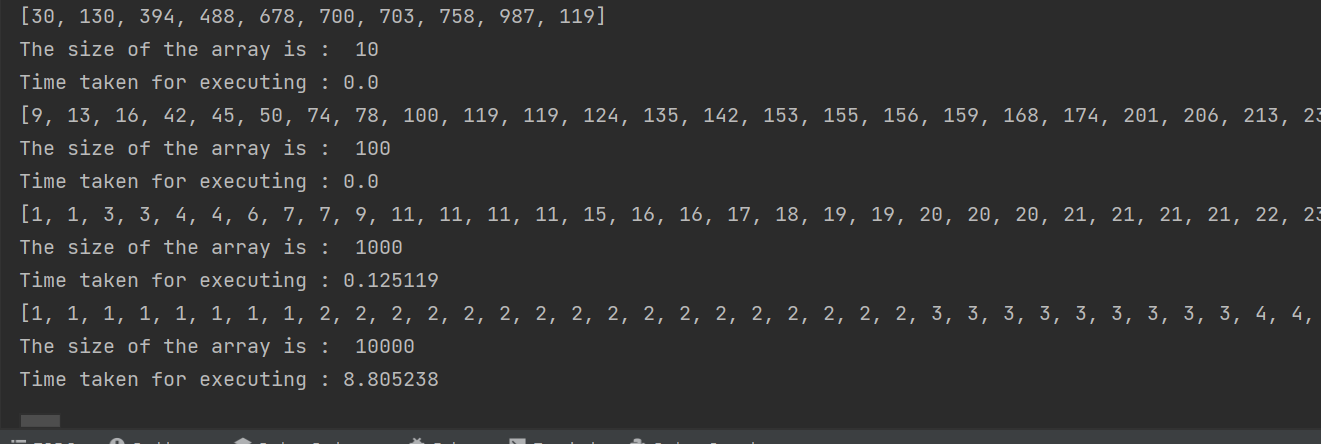
d. Compare the graphical plots for each sorting algorithms with its theoretical time complexity.

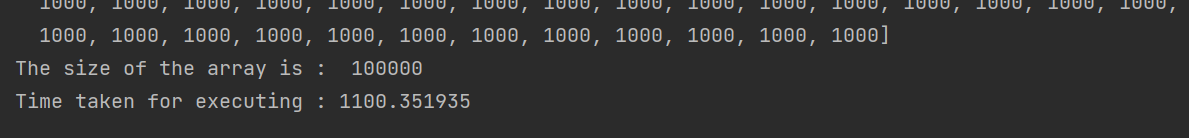
e. Also compute the time taken for sorted array (worst case) and compare with different number of elements. A sample example of sorting has been given below.

**BUBBLE SORT**

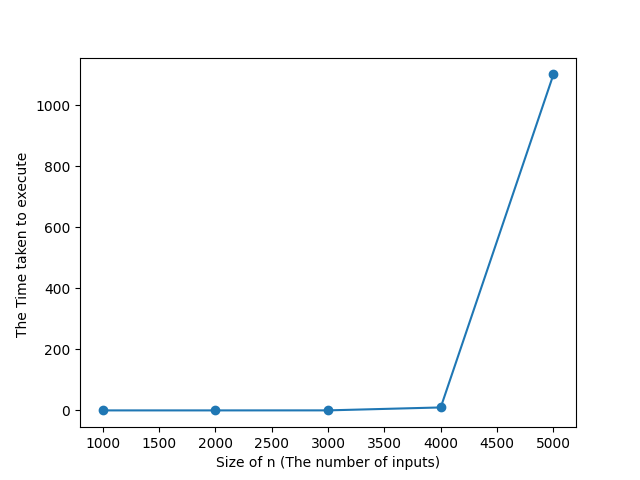
Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

**OUTPUT :( TIME OF EXECUTION WITH RESPECT TO THE SIZE OF INPUT)**





**GRAPH: (\_TIME OF EXECUTION WITH RESPECT TO THE SIZE OF INPUT )**

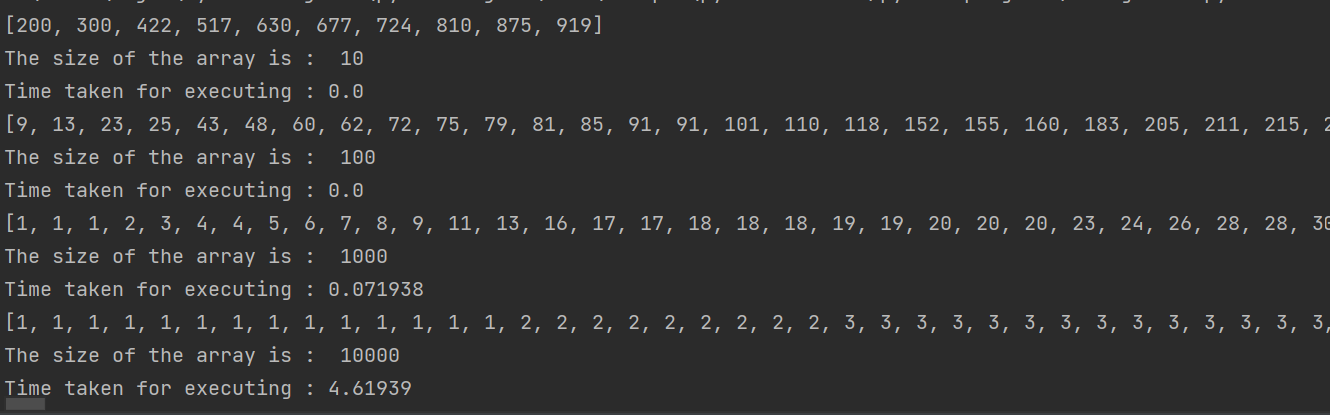


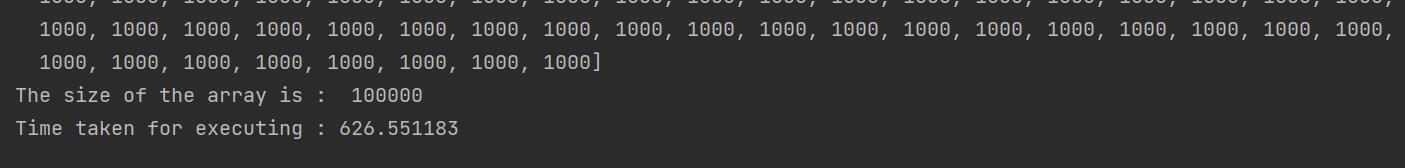
**INSERTION SORT**

Insertion sort is a simple sorting algorithm in which 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.

**OUTPUT :**

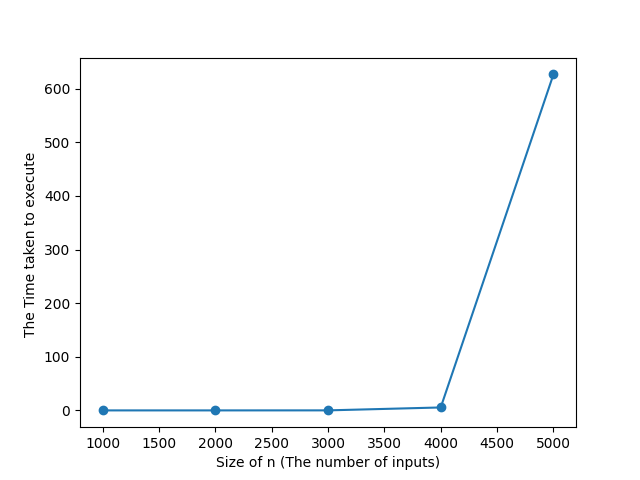
**(SHOWING TIME OF EXECUTION WITH RESPECT TO THE SIZE OF INPUT)**





**GRAPH:**

**(\_TIME OF EXECUTION WITH RESPECT TO THE SIZE OF INPUT )**

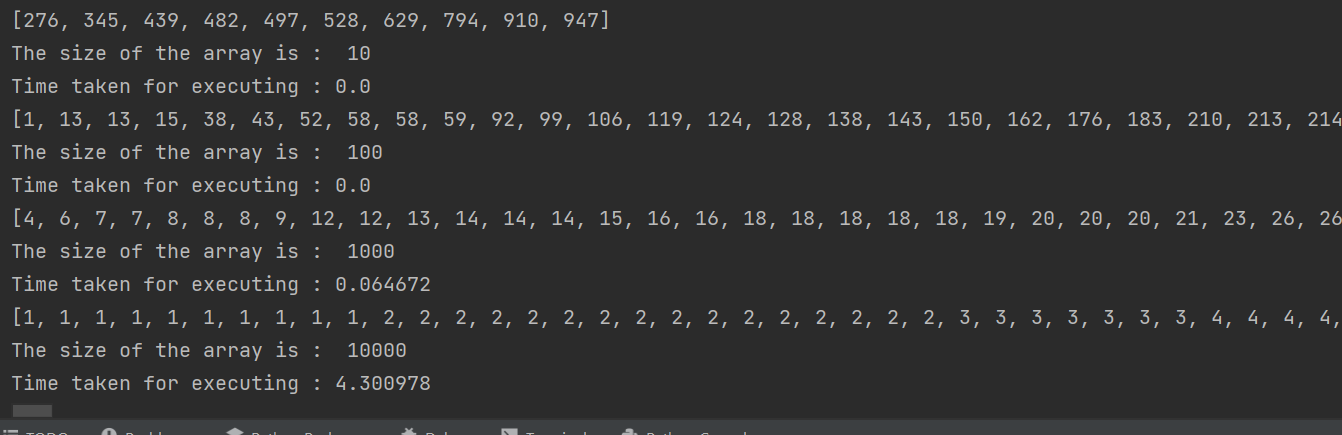


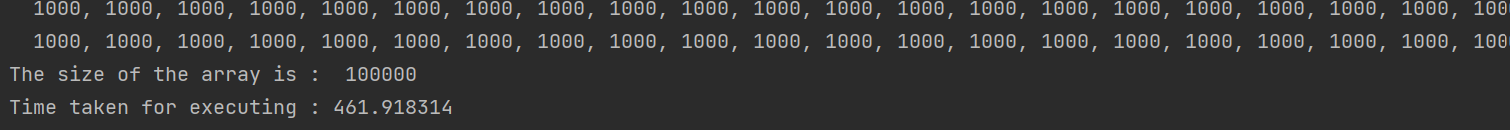
**SELECTION SORT**

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.

**OUTPUT :**

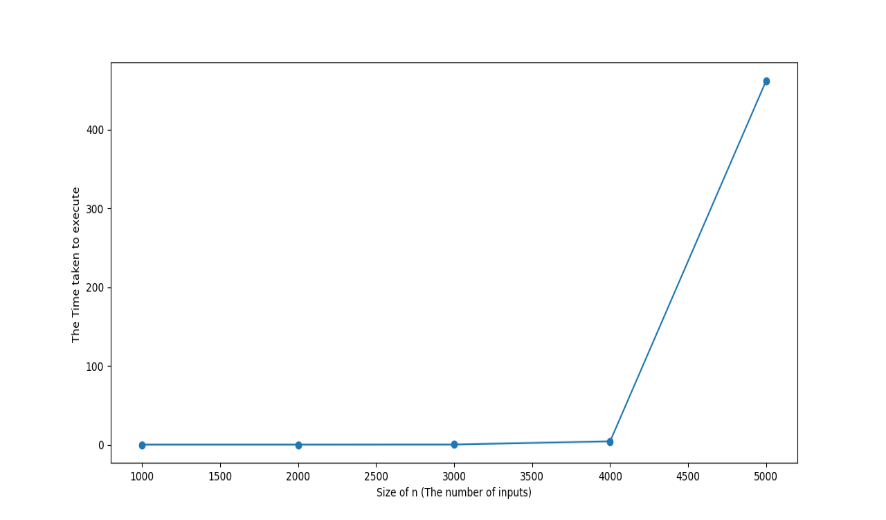
**(SHOWING TIME OF EXECUTION WITH RESPECT TO THE SIZE OF INPUT)**





**GRAPH:**

**(\_TIME OF EXECUTION WITH RESPECT TO THE SIZE OF INPUT )**

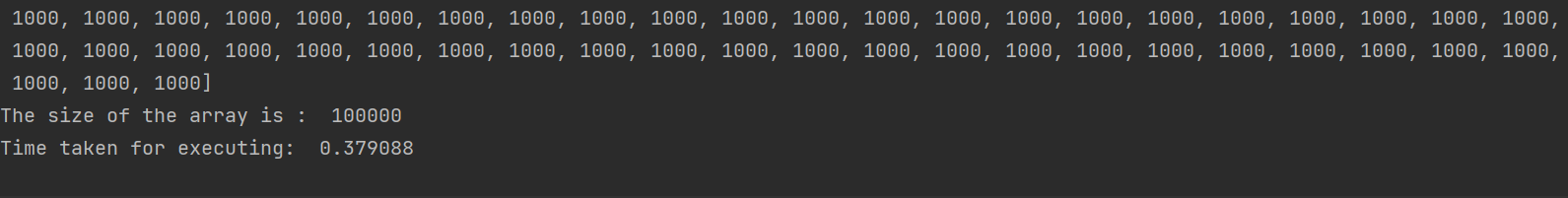
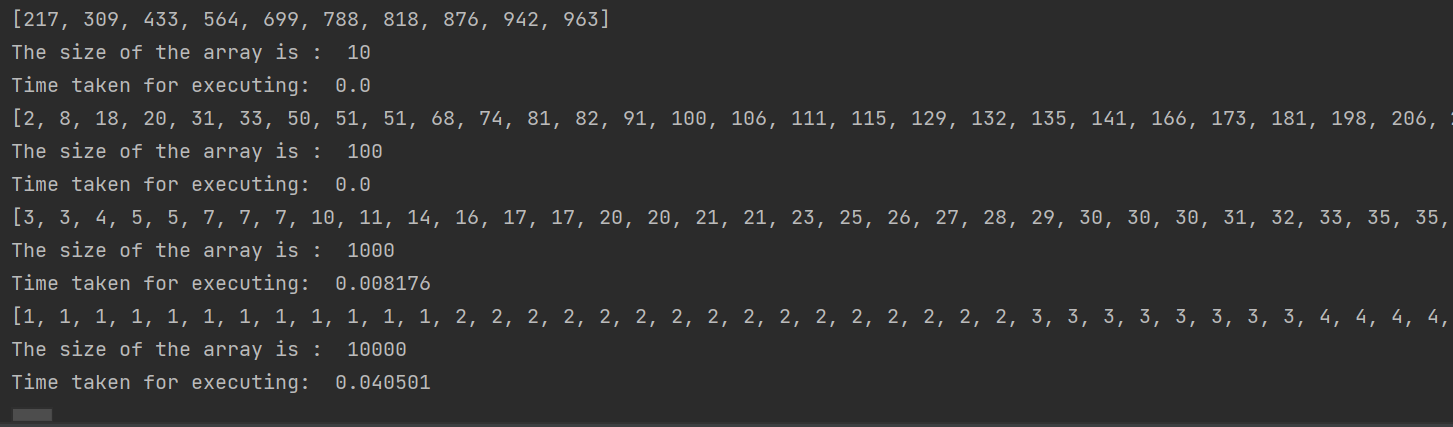


**MERGE SORT**

 Merge Sort divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves.

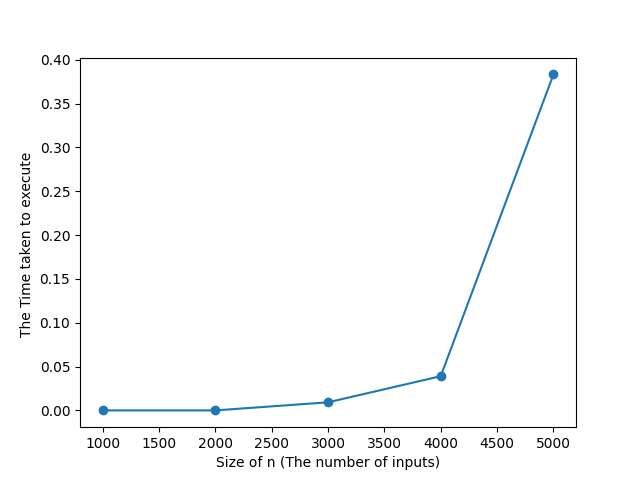
**OUTPUT :**

**(SHOWING TIME OF EXECUTION WITH RESPECT TO THE SIZE OF INPUT)**



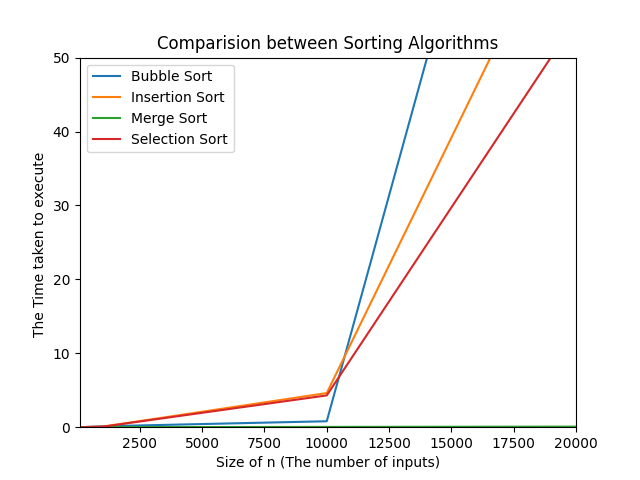
**GRAPH:**

**(\_TIME OF EXECUTION WITH RESPECT TO THE SIZE OF INPUT )**



**RESULT :**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SORTING ALGORITHM | TIME COMPLEXITY | | | SPACE  COMPLEXITY |
| BEST | AVERAGE | WORST |
| BUBBLE | O(n) | O(n2) | O(n2) | O(1) |
| INSERTION | O(n) | O(n2) | O(n2) | O(1) |
| SELECTION | O(n2) | O(n2) | O(n2) | O(1) |
| MERGE | O(nlogn) | O(nlogn) | O(nlogn) | O(n) |



**CONCLUSION:** The efficiency of an algorithm is the base parameter to define the best possible sorting technique in the present use. The sorting efficiency is defined by the comparison over execution time with respect to the varying size of inputs to sort.