**Performance of Different Algorithms**

**Table:-**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Algorithm | Array Size | Array Type | Number of runs | Average Time | |  |
| **Selection** | 100 | **Random** | 10k | 121444 Micro Secs | |  |
| **Selection** | 100 | **sorted** | 10k | 94541.9 Micro Secs | |  |
| **Bubble** | 100 | **Random** | 10k | 224723 Micro Secs | |  |
| **Bubble** | 100 | **sorted** | 10k | 18.5 Micro Secs | |  |
| **Insertion** | 100 | **Random** | 10k | 81833.1 Micro Secs | |  |
| **Insertion** | 100 | **sorted** | 10k | 31.85 Micro Secs | |  |
|  |  |  |  |  |  |  |

**NO OF RUNS: 10**

**Array size for this Report = 10k**

**Insertion Sort**: It performs the best in both random and best-case scenarios, making it the top choice in this context.

Insertion Sort performs the better than the both Selection Sort and Bubble Sort in both random and best-case scenarios. It's relatively faster, especially for the sorted array, making it a better choice in these conditions.

**Bubble Sort**: It performs reasonably well in the best-case scenario but poorly in the random array case.

**Selection Sort**: It performs the worst among the three algorithms for both random and best-case scenarios.

It has high time complexity in both the cases, so its not recommended for larger set of values

**Insertion Sort Performs the best**

**Selection Sort Performs the least due to very high time complexity i.e., O(n2)**

Note: - the results are considered by taking the average of 10 runs