DSA Experiment 1

AIM: - Implement iterative and full recursive versions of the following sorting techniques and repeat the experiment for larger values of n with different input patterns [10K, 50K, 100K], and plot the comparison graph of a number of elements versus execution time taken. The elements can be read from a file or can be generated using a random number generator.

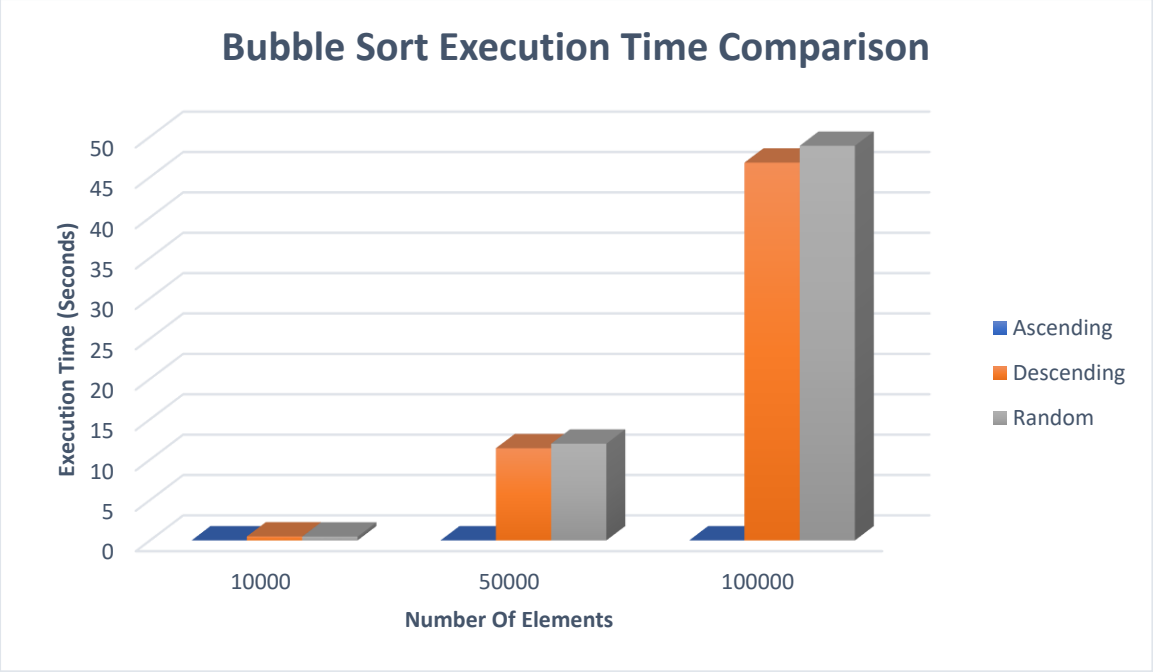
1) Bubble Sort
2) Insertion Sort
3) Selection Sort
4) Quick Sort

1. Bubble Sort

- Comparison Table

Elements	Ascending	Descending	Random
10000	0 Second	0.49 Second	0.473 Seconds
50000	0 Seconds	11.414 seconds	11.977 seconds
100000	0 seconds	46.786 seconds	48.87 seconds

- Comparison Graph



Conclusion

- From the recorded execution times and the plotted graph, it's clear that Bubble Sort performs extremely poorly on large datasets, especially when the input is in descending or random order.
- The ascending case finished instantly because the algorithm stops early if no swaps are needed — an optimization that works perfectly for already sorted data.
- However, for descending and random inputs, execution time increases dramatically as the number of elements grows. For 100,000 elements the time jumped beyond 48 seconds, which shows Bubble Sort's $O(n^2)$ time complexity in action.

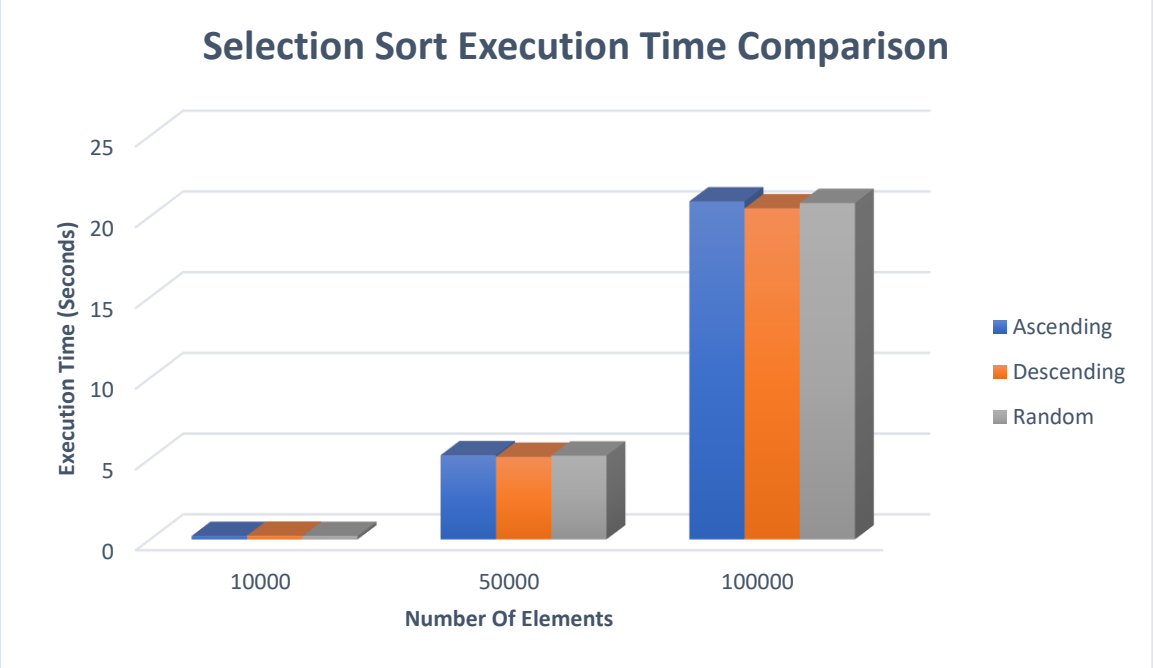
In short: Bubble Sort is simple and fine for tiny or nearly sorted datasets, but it is not practical for large or unordered data due to its quadratic growth in execution time.

2. Selection Sort

- Comparison Table

Elements	Ascending	Descending	Random
10000	0.212 Second	0.23 Second	0.216 Seconds
50000	5.215 Seconds	5.113 seconds	5.188 seconds
100000	20.93 seconds	20.499 seconds	20.832 seconds

- Comparison Graph



Conclusion

- From the given results, Selection Sort shows consistent performance regardless of input order (ascending, descending, or random).
- This happens because Selection Sort always goes through the entire list, making the same number of comparisons and swaps no matter how the data is arranged.
- For small datasets like 10,000 elements, the times are very short (around 0.21 seconds). However, as the number of elements grows, the execution time increases significantly, reaching about 21 seconds for 100,000 elements.
- This behaviour matches its $O(n^2)$ time complexity. The bar graph would show three bars of almost equal height for each dataset size, reflecting the algorithm's lack of sensitivity to initial ordering.

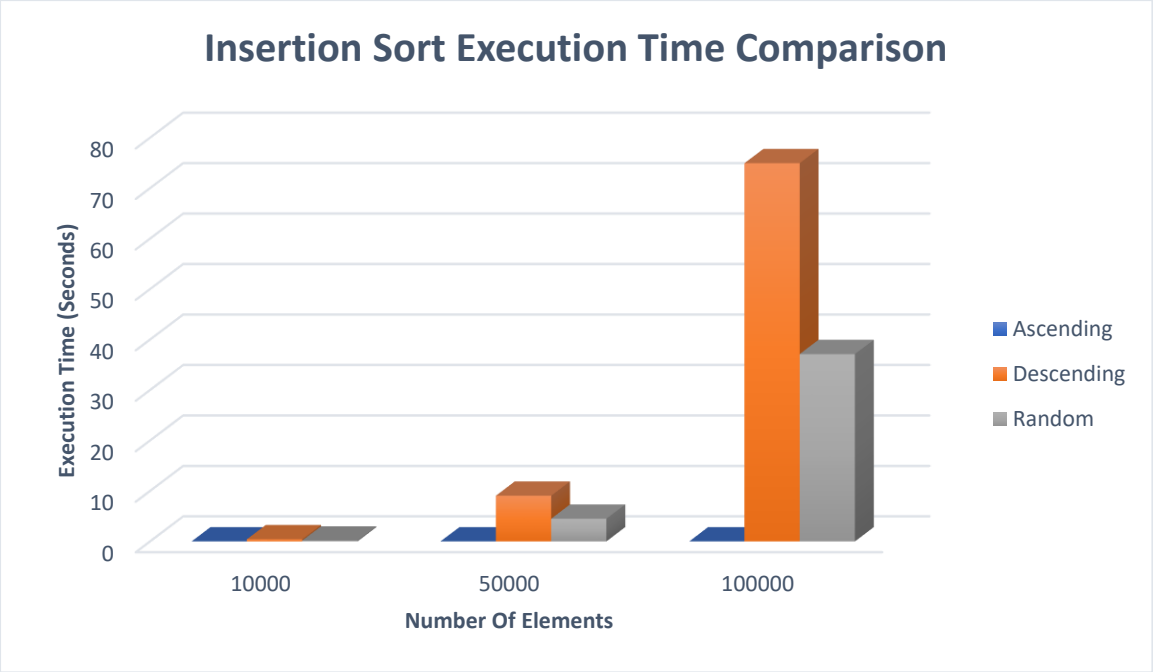
In short: Selection Sort is simple and predictable, but still not suitable for large datasets due to quadratic growth in time.

3. Insertion Sort

- Comparison Table

Elements	Ascending	Descending	Random
10000	0 Second	0.427 Second	0.162 Seconds
50000	0.001 Seconds	9.05 seconds	4.486 seconds
100000	0 seconds	74.945 seconds	37.116 seconds

- Comparison Graph



Conclusion

- The results show that Insertion Sort can be extremely efficient for already sorted data — in the ascending case, it completed instantly for all tested sizes because it only needs one pass with no shifting.
- However, the performance drops significantly for descending order inputs, where each new element must be shifted to the beginning of the list. For 100,000 descending elements, the time skyrocketed to about 75 seconds.
- Random inputs perform better than descending ones but still take considerable time — roughly 37 seconds for 100,000 elements.
- This difference in performance is due to Insertion Sort's nature:
- Best case (O(n)) for nearly sorted data
- Worst case ($O(n^2)$) for reverse-ordered data
- The graph would clearly show:
    - A flat line near zero for ascending
    - Very tall bars for descending
    - Medium-height bars for random inputs

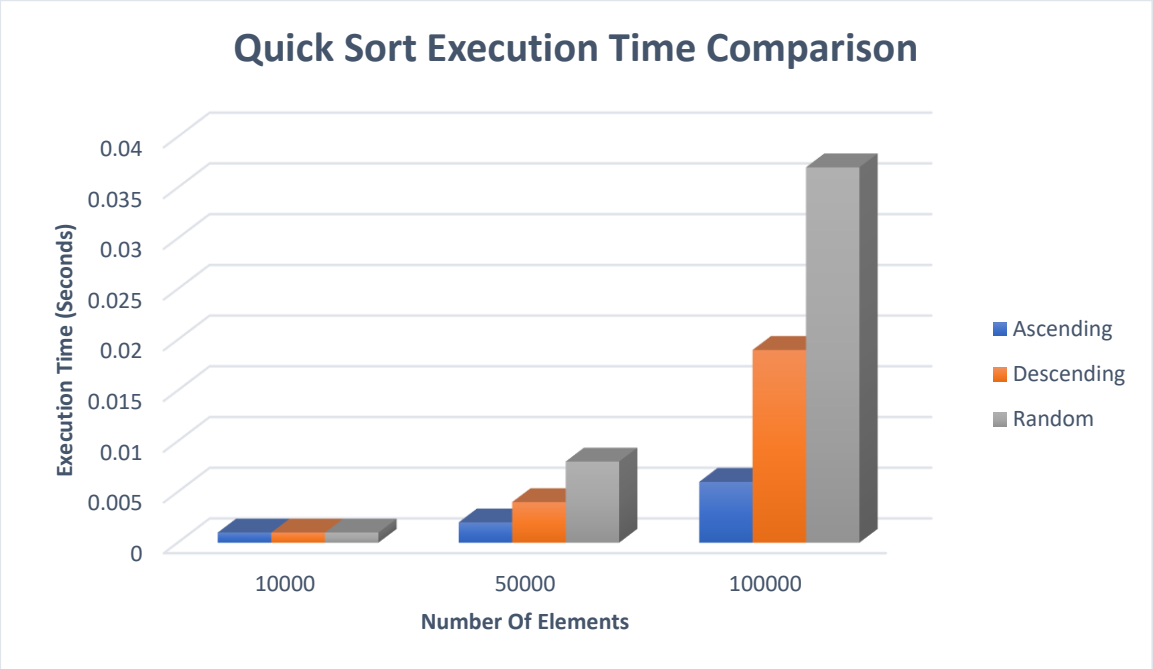
In short: Insertion Sort is excellent for small or nearly sorted datasets, but is inefficient for large, unordered data.

4. Quick Sort

- Comparison Table

Elements	Ascending	Descending	Random
10000	0.001 Second	0.001 Second	0.001 Seconds
50000	0.002 Seconds	0.004 seconds	0.008 seconds
100000	0.006 seconds	0.019 seconds	0.037 seconds

- Comparison Graph



Conclusion

- The results demonstrate why Quick Sort is considered one of the fastest sorting algorithms in practice:
- For 10,000 elements, all three input types (ascending, descending, random) finish almost instantly (~0.001 seconds).
- At 50,000 elements, performance remains excellent: even the worst case (random) takes only 0.008 seconds, while ascending is as low as 0.002 seconds.
- With 100,000 elements, times are still under 0.04 seconds for all cases — an outstanding result compared to quadratic algorithms.
- The graph clearly shows that execution time grows linearly with input size, matching the expected O (n log n) complexity.

Unlike Bubble, Selection, or Insertion Sort, Quick Sort handles large inputs with ease, making it highly practical for real-world applications.