Selection Sort

What is Selection Sort?

Selection Sort is a simple comparison-based sorting algorithm that repeatedly selects the **smallest element** from an unsorted portion of the array and swaps it with the first unsorted element. It continues this process until the entire array is sorted.

Algorithm Explanation

1. Start with the first element of the array.
2. Find the **minimum** element in the unsorted part of the array.
3. Swap this minimum element with the first unsorted element.
4. Move the boundary of the sorted portion by one.
5. Repeat steps 2-4 until the entire array is sorted.

Dry Run Example

Input: {64, 25, 12, 22, 11}

Pass 1: (Find minimum & swap)

- Find the smallest element (11).
- Swap 11 with 64.
- **Array after 1st pass:** {11, 25, 12, 22, 64}

Pass 2:

- Find the smallest in the remaining elements (12).
- Swap 12 with 25.
- **Array after 2nd pass:** {11, 12, 25, 22, 64}

Pass 3:

- Find the smallest in the remaining (22).
- Swap 22 with 25.
- **Array after 3rd pass:** {11, 12, 22, 25, 64}

Pass 4:

- 25 and 64 are already in place, so no swap needed.

Final Sorted Array: {11, 12, 22, 25, 64}

Time & Space Complexity

- **Worst-case time complexity:** $O(n^2)$ (when elements are in reverse order)
- **Best-case time complexity:** $O(n^2)$ (even if already sorted, it still finds the minimum)
- **Average-case time complexity:** $O(n^2)$
- **Space Complexity:** O(1) (in-place sorting, no extra space used)

Common Interview Questions on Selection Sort

1. How does Selection Sort work?

Answer: Selection Sort works by selecting the smallest element from the unsorted part of the array and swapping it with the first unsorted element. This process continues until the entire array is sorted.

2. What is the time complexity of Selection Sort?

Answer:

- Worst-case: $O(n^2)$
- Best-case: $O(n^2)$
- Average-case: $O(n^2)$

3. Is Selection Sort a stable sorting algorithm?

Answer:
No, **Selection Sort is not stable** because swapping might change the relative order of equal elements. For example:

Input: {3A, 3B, 2, 1}
After Sorting: {1, 2, 3B, 3A} (Relative order of 3A and 3B is changed)

4. Is Selection Sort an in-place sorting algorithm?

Answer:
Yes, Selection Sort is **in-place** because it does not use extra memory.

5. When should you use Selection Sort?

Answer:

- When memory space is limited (O(1) space complexity).
- When the number of swaps must be minimized (compared to Bubble Sort).
- For small datasets.

6. What are the advantages of Selection Sort?

Answer:

1. **Simple to implement** and easy to understand.
2. **Performs well on small datasets**.
3. **Less swap operations** compared to Bubble Sort.

7. What are the disadvantages of Selection Sort?

Answer:

1. **Time complexity is $O(n^2)$**, making it inefficient for large datasets.
2. **Not stable**, meaning the order of duplicate elements might change.
3. **Performs the same number of comparisons** regardless of input order.

Conclusion

Selection Sort is **better than Bubble Sort** in terms of swaps but still **not efficient** for large datasets due to its $O(n^2)$ time complexity.

For your **job interview**, be ready to: Explain **how Selection Sort works**
- Discuss **time & space complexity**
- Compare **Selection Sort vs. other sorting algorithms**
- Explain **why it is not used in real-world scenarios**
- Write **optimized Java code**