📘 **Definition:**
Selection Sort is a simple sorting algorithm that repeatedly finds the **smallest element** from the unsorted part of the array and **places it at the beginning**. It divides the array into a sorted and an unsorted region, and one by one moves the boundary forward by selecting the minimum element.

💡 **Real Life Analogy**

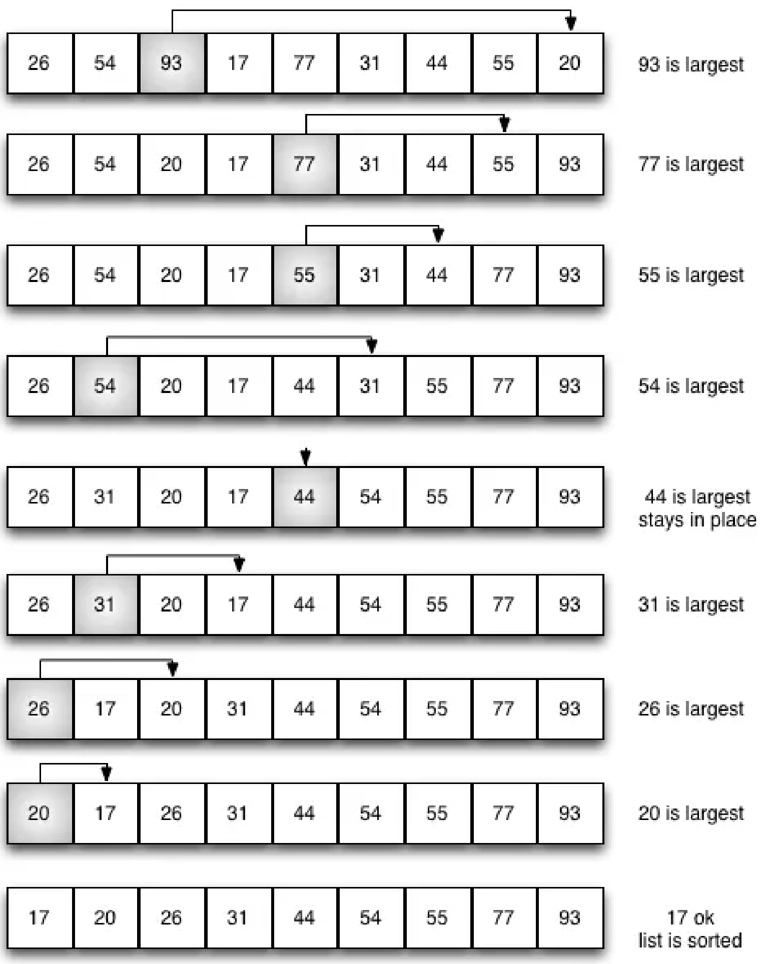
Socho tumhe sabse sasti sabzi leni hai ek market ke stall se.

Tum kya karoge?

1. Tum sabziyon ke rate dekho (unsorted prices)
2. Sabse sasti wali dhundo (minimum price)
3. Usko list ke beginning mein daal do (first position)
4. Fir next cheapest, and so on...

Yeh hi hai **Selection Sort**:

- Har round mein **minimum dhoondho**
- **Sahi jagah pe rakh do**
- Aage badho



✅ **C Code for Selection Sort**

```
#include <stdio.h>

int main() {
    int arr[] = {5, 1, 4, 2, 8};
    int size = sizeof(arr) / sizeof(arr[0]);

    for(int i = 0; i < size - 1; i++) {
        int min_index = i;

        for(int j = i + 1; j < size; j++) {
            if(arr[j] < arr[min_index]) {
                min_index = j;  // Find index of minimum element
            }
        }

        // Swap the found minimum with the first element
        int temp = arr[i];
        arr[i] = arr[min_index];
        arr[min_index] = temp;
    }

    // Print sorted array
    printf("Sorted array: ");
    for(int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

🧠 Step-by-Step Explanation

1. Assume the **first element is the smallest**
2. Compare it with the rest of the array
3. If you find a smaller element, update the minimum index
4. At the end of the inner loop, **swap** the smallest element with the current index
5. Repeat for all positions

📜 Pseudocode / Algorithm (For Notes or Exams)

```
1. Start
2. Input the array and its size
3. Repeat for i = 0 to size - 2:
   a. Set min_index = i
   b. Repeat for j = i+1 to size - 1:
      i. If arr[j] < arr[min_index], set min_index = j
   c. Swap arr[i] with arr[min_index]
4. Print sorted array
5. Stop
```

🧠 Time Complexity:

Case	**Comparisons**	**Time**
Best	$O(n^2)$	❌
Average	$O(n^2)$	❌
Worst	$O(n^2)$	❌