Sorting Algorithms 2

BUBBLE SORT, SELECTION SORT, INSERTION SORT

Bubble Sort

Bubble sort, also known as exchange sort, is a very simple sorting method. It works by repeatedly moving the largest element to the highest position of the array. In bubble sort, we are comparing two elements at a time, and swapping is done if they are wrongly placed. If the element at a lower index or position is greater than the element at a higher index, then in that case both the elements are interchanged so that the smaller element is placed before the bigger one. This process is repeated until the list becomes sorted.

Bubble Sort Algorithm

Step-by-step Process:

1. Start from the first element.
2. Compare the current element with the next one.
3. Swap if the current element is greater than the next one.
4. Move to the next element and repeat.
5. After one full pass, the largest element is at the end.
6. Repeat for remaining elements until sorted.

Algorithm of a Bubble Sort

BUBBLE SORT(ARR, N)

Step 1: START

Step 2: Repeat Step 3 for I = 0 to N - 1

Step 3: Repeat for J = 0 to N - 1

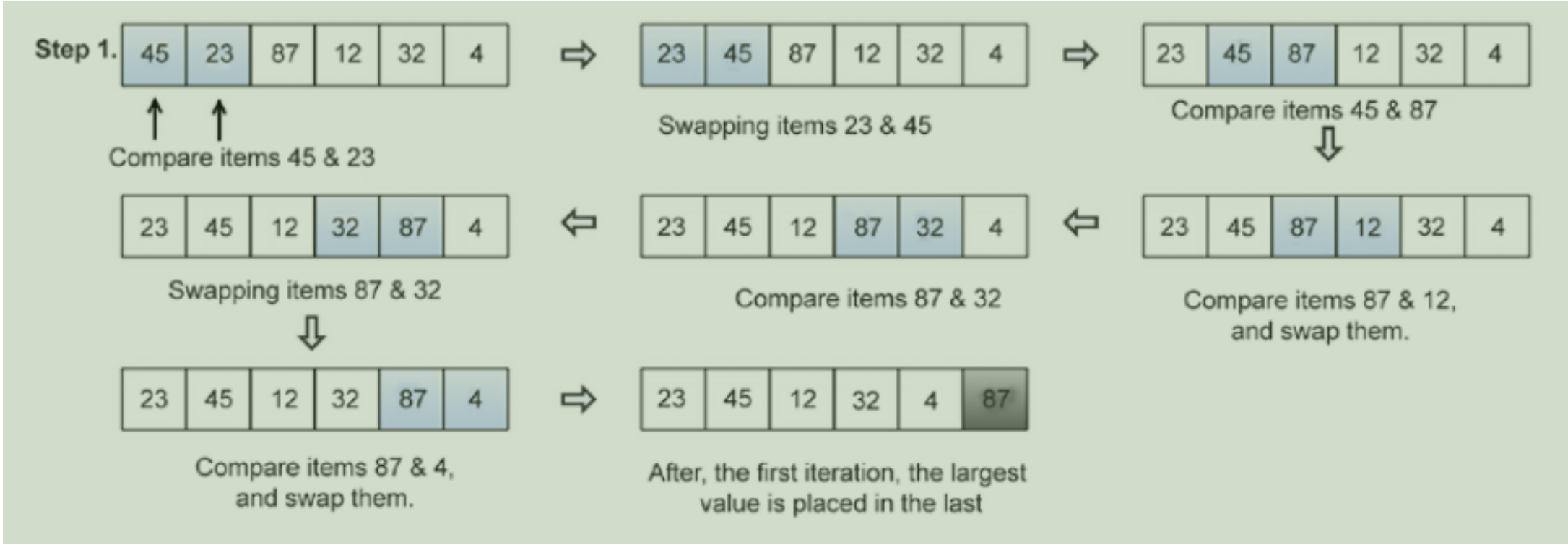
Step 4: IF (ARR[J] > ARR[J+1])

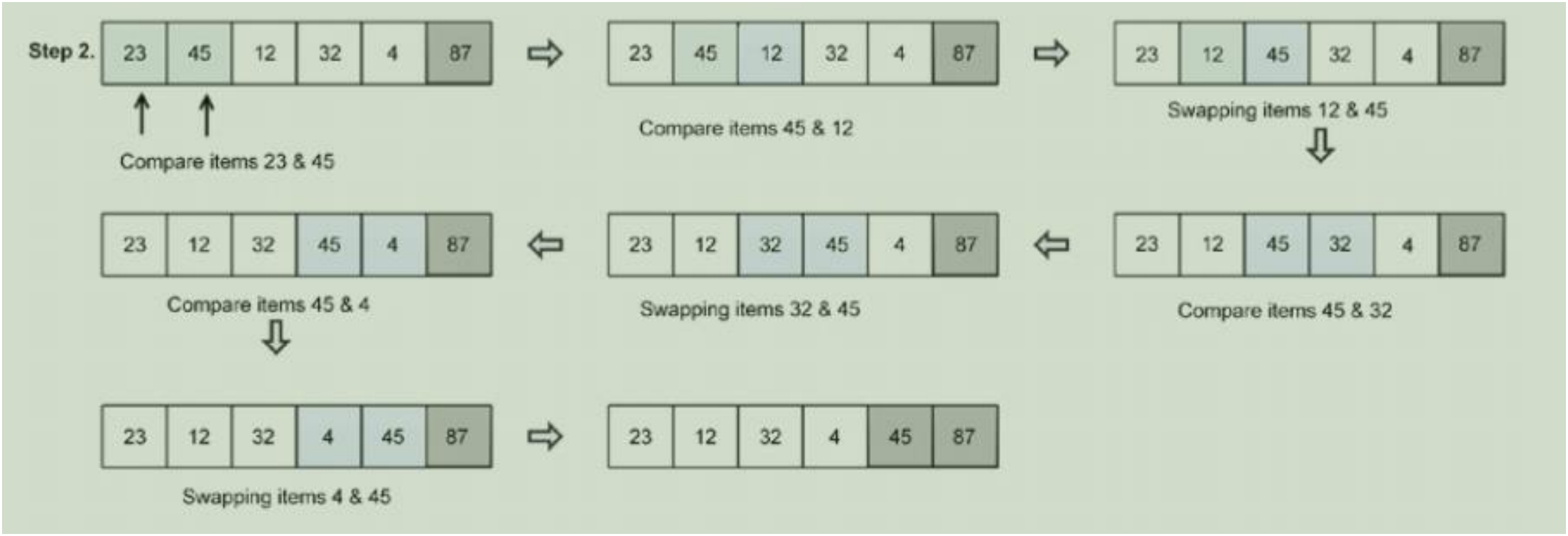
INTERCHANGE ARR[J] & ARR[J + 1]

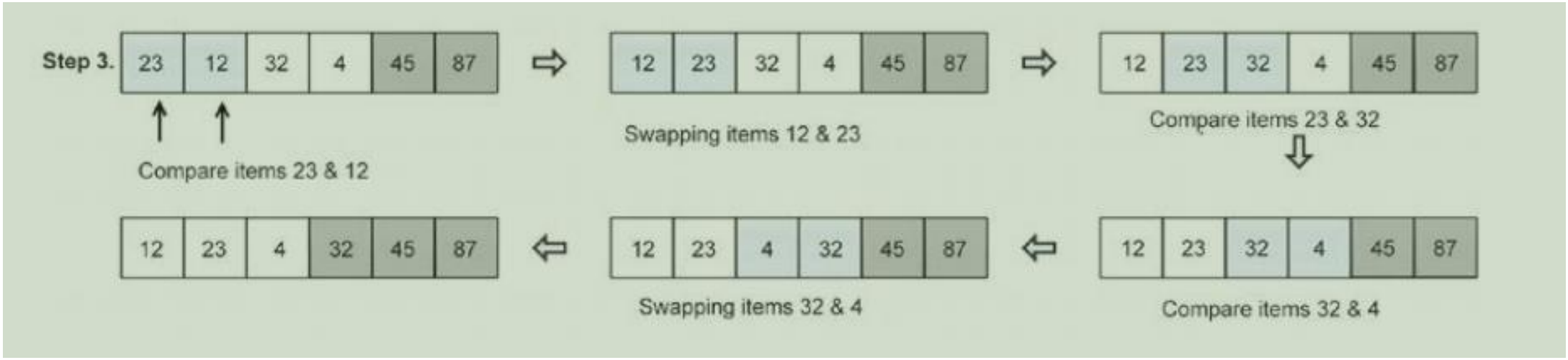
[End of Inner Loop]

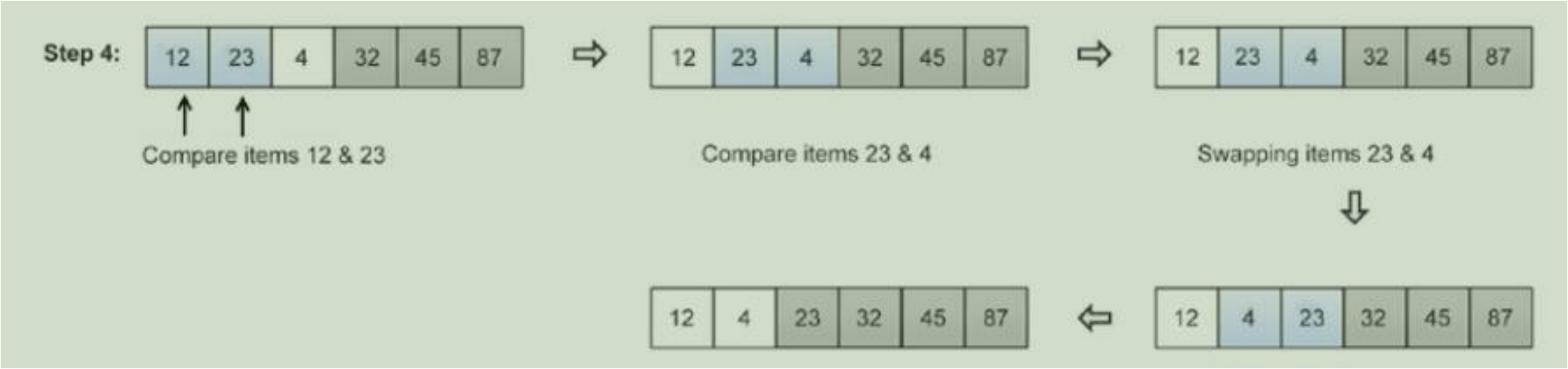
[End of Outer Loop]

Step 5: EXIT


Step 1.
45 23 87 12 32 4
Compare items 45 & 23
23 45 87 12 32 4
Swapping items 23 & 45
23 45 87 12 32 4
Compare items 45 & 87
23 45 87 12 32 4
Compare items 87 & 12, and swap them.
23 45 87 12 32 4
Compare items 87 & 32
23 45 12 87 32 4
Swapping items 87 & 32
23 45 12 32 87 4
Compare items 87 & 4, and swap them.
23 45 12 32 87 4
After, the first iteration, the largest value is placed in the last


Step 2.
23 45 12 32 4 87
Compare items 23 & 45
23 45 12 32 4 87
Compare items 45 & 12
23 12 45 32 4 87
Swapping items 12 & 45
23 12 45 32 4 87
Compare items 45 & 32
23 12 32 45 4 87
Swapping items 32 & 45
23 12 32 45 4 87
Compare items 45 & 4
23 12 32 4 45 87
Swapping items 4 & 45
23 12 32 4 45 87


Step 3.
23 12 32 4 45 87
Compare items 23 & 12
12 23 32 4 45 87
Swapping items 12 & 23
12 23 32 4 45 87
Compare items 23 & 32
12 23 4 32 45 87
12 23 4 32 45 87
Swapping items 32 & 4
12 23 32 4 45 87
Compare items 32 & 4


Step 4:
12 23 4 32 45 87
Compare items 12 & 23
12 23 4 32 45 87
Compare items 23 & 4
12 23 4 32 45 87
Swapping items 23 & 4
12 4 23 32 45 87
12 4 23 32 45 87


Step 5:
12
4
23
32
45
87
Compare items 12 & 4
4
12
23
32
45
87
Swapping items 12 & 4
4
12
23
32
45
87

Time & Space Complexity

- **Best Case (Already Sorted):** O(n)
- **Average Case:** $O(n^2)$
- **Worst Case (Reverse Sorted):** $O(n^2)$
- **Space Complexity:** O(1) (In-place Sorting)

Optimized : If no swaps occur in a pass, stop early.

Selection sort algorithm

Selection sort is a sorting technique that works by finding the smallest value in the array and placing it in the first position. After that, it then finds the second smallest value and places it in the second position. This process is repeated until the whole array is sorted. Thus, the selection sort works by finding the smallest unsorted element remaining in the entire array and then swapping it with the element in the next position to be filled. It is a very simple technique, and it is also easier to implement than other sorting techniques. Selection sort is used for sorting files with large records.

Selection Sort Algorithm

- Find the smallest element in the array.
- Swap it with the first element.
- Move to the next position and repeat.
- Continue until the entire array is sorted.

Step 1: START

Step 2: Repeat Steps 3 & 4 for I = 1 to N - 1

Step 3: Call MIN(ARR, I, N, POS)

Step 4: Swap ARR[I] with ARR[POS]

[End of Loop]

Step 5: EXIT

MIN(ARR, I, N, POS)

Step 1:Set SMALLEST = ARR[I]

Step 2:Set POS = I

Step 3: Repeat Step 4 for J = I + 1 to N – 1

Step 4: IF (ARR[J] < SMALLEST)

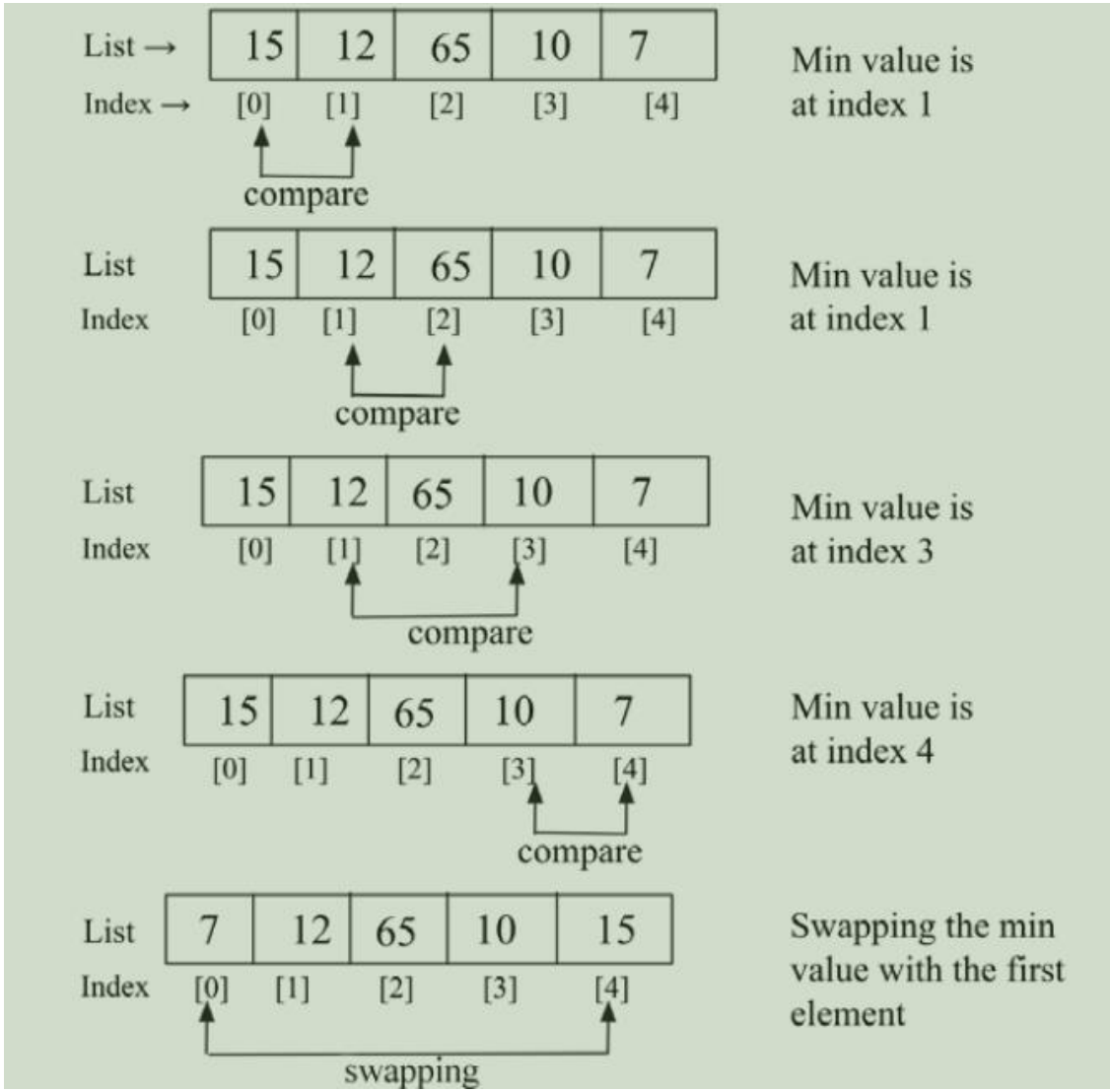
Set SMALLEST = ARR[J]

Set POS = J

[End of IF]

[End of Loop]

Step 5: Return POS


List →
15 12 65 10 7
Index →
[0] [1] [2] [3] [4]
compare
Min value is at index 1
List
15 12 65 10 7
Index
[0] [1] [2] [3] [4]
compare
Min value is at index 1
List
15 12 65 10 7
Index
[0] [1] [2] [3] [4]
compare
Min value is at index 3
List
15 12 65 10 7
Index
[0] [1] [2] [3] [4]
compare
Min value is at index 4
List
7 12 65 10 15
Index
[0] [1] [2] [3] [4]
swapping
Swapping the min value with the first element

Time & Space Complexity

- **Best Case:** $O(n^2)$
- **Average Case:** $O(n^2)$
- **Worst Case:** $O(n^2)$
- **Space Complexity:** O(1) (In-place Sorting)

Use Cases & Limitations

Advantages:

- Fewer swaps than Bubble Sort
- Good for small arrays

Disadvantages:

- Inefficient for large datasets
- Slower than Quick/Merge Sort

Insertion Sort

Insertion sort is another very simple sorting algorithm, it inserts each element into its proper position in the concluding list. To limit the wastage of memory or, we can say, to save memory, most implementations of an insertion sort work by moving the current element past the already sorted elements and repeatedly swapping or interchanging it with the preceding element until it is placed in its correct position.

Algorithm for an Insertion Sort

INSERTION SORT(ARR, N)

Step 1: START

Step 2: Repeat Steps 3 to 6 for I = 1 to N – 1

Step 3: Set POS = ARR[I]

Step 4: Set J = I - 1

Step 5: Repeat while POS <= ARR[J]

Set ARR[J + 1] = ARR[J]

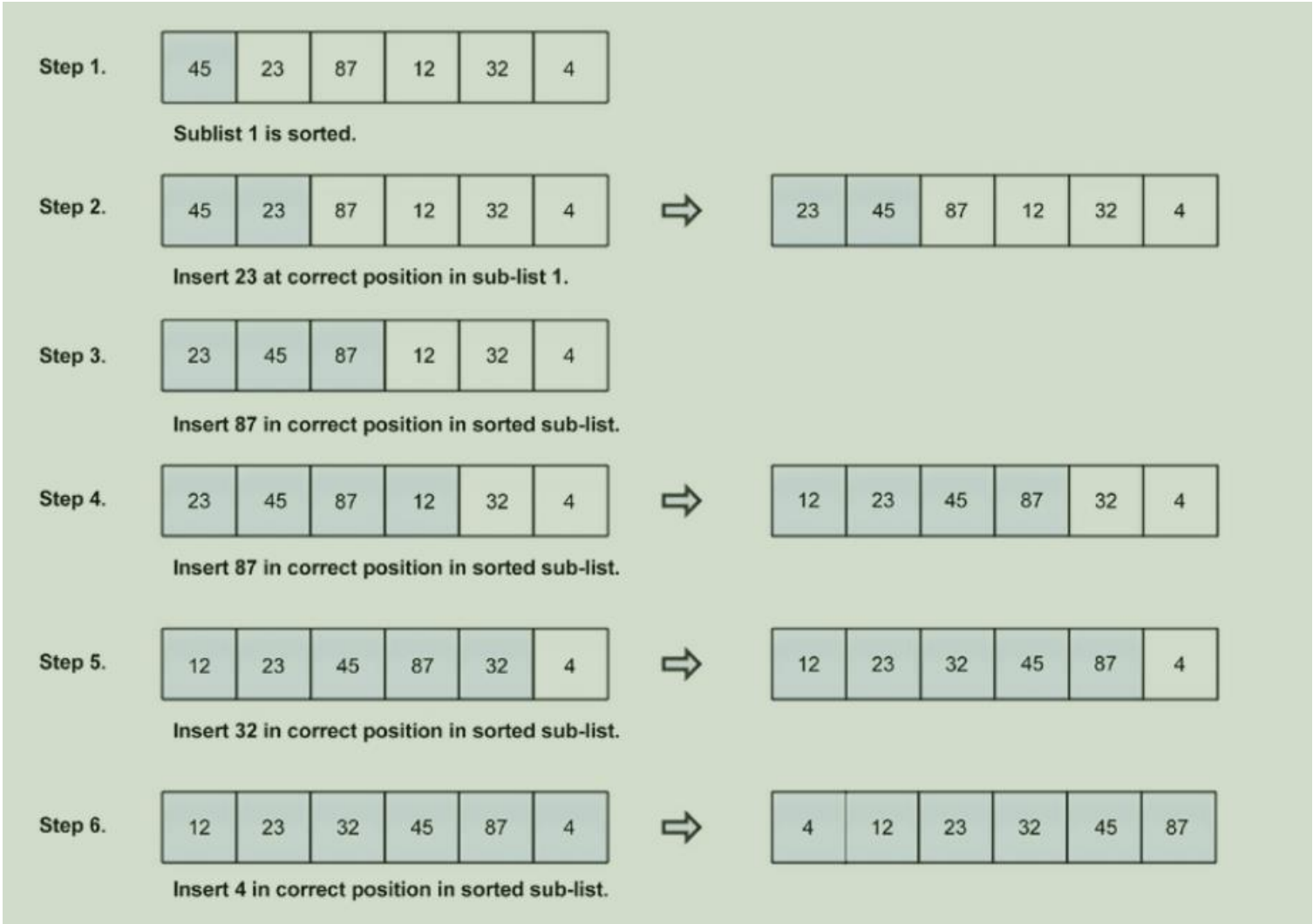
Set J = J – 1

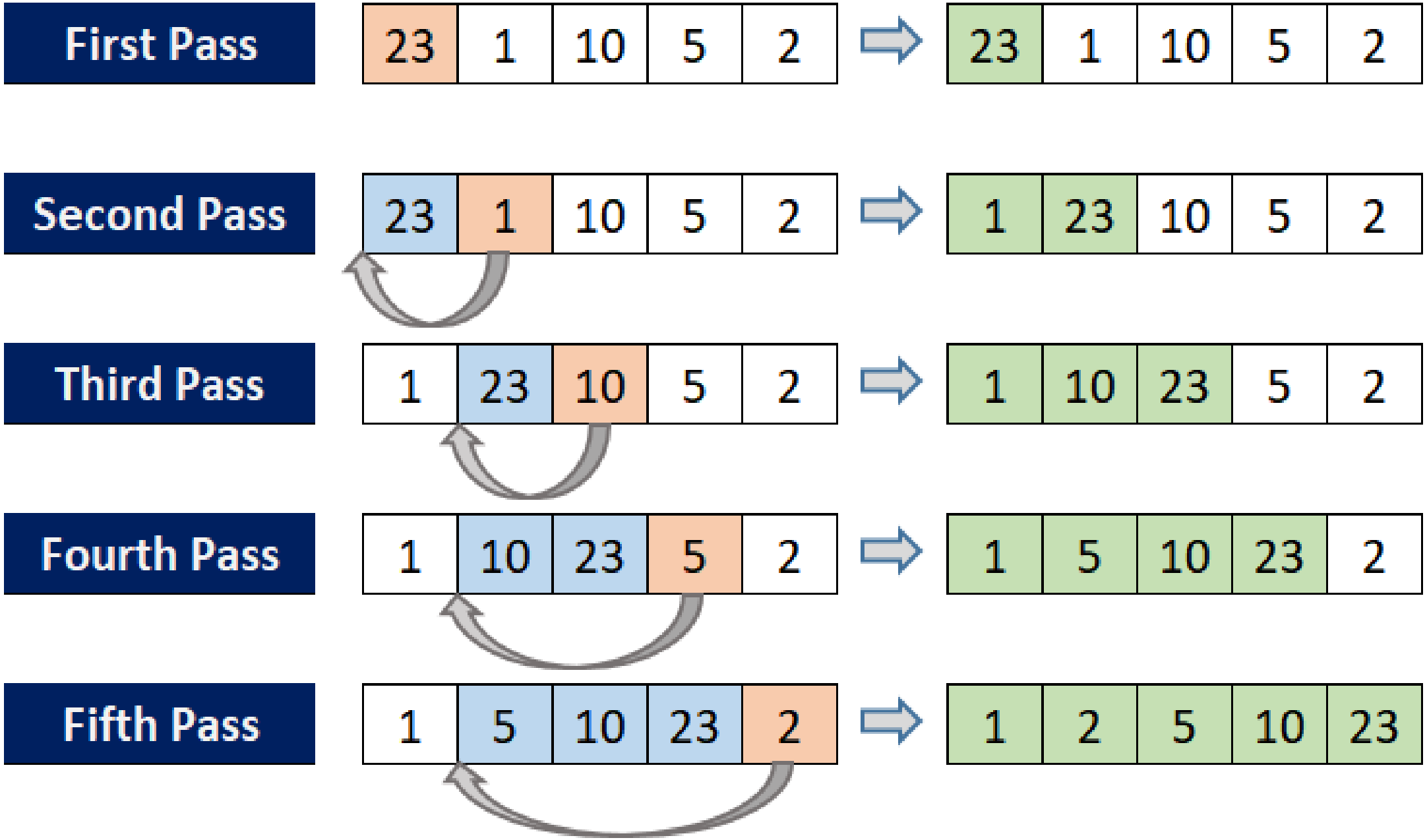
[End of Inner while loop]

Step 6: Set ARR[J + 1] = POS

[End of Loop]

Step 7: EXIT


Step 1.
45 23 87 12 32 4
Sublist 1 is sorted.
Step 2.
45 23 87 12 32 4
23 45 87 12 32 4
Insert 23 at correct position in sub-list 1.
Step 3.
23 45 87 12 32 4
Insert 87 in correct position in sorted sub-list.
Step 4.
23 45 87 12 32 4
12 23 45 87 32 4
Insert 87 in correct position in sorted sub-list.
Step 5.
12 23 45 87 32 4
12 23 32 45 87 4
Insert 32 in correct position in sorted sub-list.
Step 6.
12 23 32 45 87 4
4 12 23 32 45 87
Insert 4 in correct position in sorted sub-list.


First Pass
23 1 10 5 2
23 1 10 5 2
Second Pass
23 1 10 5 2
1 23 10 5 2
Third Pass
1 23 10 5 2
1 10 23 5 2
Fourth Pass
1 10 23 5 2
1 5 10 23 2
Fifth Pass
1 5 10 23 2
1 2 5 10 23

Time & Space Complexity

- **Best Case (Already Sorted):** $O(n)$
- **Average Case:** $O(n^2)$
- **Worst Case (Reverse Sorted):** $O(n^2)$
- **Space Complexity:** $O(1)$ (In-place Sorting)

End.