2. Selection Sort Correctness

A basic sorting algorithm based on comparison is called selection sort. Its concept is to separate the input list into two sections. A region that is sorted and another that is not. The method continually chooses the initial element of the unsorted region and swaps it with the smallest (or largest depending on the sorting order) element from the unsorted region. Until the entire array is sorted, this process is to be repeated.

Pseudo code for Selection Sort:

```
SelectionSort(Array)
for i from 0 to length(Array) - 1
minimumIndex = i
for j from i + 1 to length(Array)
if Array[j] < Array[minimumIndex]
minimumIndex = j
swap Array[i] with Array[minimumIndex]
```

To argue selection sort correctness, below are the points:

1. Initialization:

The array's items are all present in the right half of the array before the first iteration of the outer loop, while the left half of the array is empty. The requirement that the left part be sorted and the right part be left unsorted is satisfied by this.

2. Maintenance:

The method chooses the smallest entry from the unsorted list for each iteration of the outer loop. This preserves the feature that the right portion is unsorted and the left section is sorted.

3. Termination:

When the entire array has been sorted, the process comes to an end. When the outer loop has finished its iterations, this happens. By now, all of the elements in the array have been sorted, meeting the requirements.

The smallest element from the unsorted zone is relocated to the appropriate location in the sorted region at the conclusion of each iteration of the outer loop, which is the selection sort's invariant and the basis for its accuracy. Until the entire array is sorted, these steps are repeated.

In conclusion, selection sort is an appropriate sorting algorithm that effectively arranges an array in either ascending or descending order according to the comparison operator that is applied. Selection sort is less effective than certain other sorting algorithms for large datasets, nevertheless, with a time complexity of O(n^2), where n is the number of elements in the array.

To demonstrate that the selection sort algorithm is accurate, loop invariant is to be used:

Loop Invariant: The subarray arr[0...i-1] is sorted at the beginning of each outer loop iteration.

Demonstrating the three points:

Initialization: I is 0 prior to the outer loop's first iteration. As a result, the subarray arr[0...i-1] is empty and so satisfies the requirement of being sorted without any problems.

Maintenance: Assume that the subarray arr[0...i-1] has been sorted prior to an outer loop iteration. The algorithm now locates the smallest element in the array's remaining unsorted section (arr[i...n-1]) and replaces it with arr[i] during this iteration. The smallest element in arr[i...n-1] is now at index i as a result of this exchange. The subarray arr[0...i] is sorted as a result.The subarray (arr[i+1...n-1]) does not change in the remaining portions. As a result, the loop invariant is preserved.

Termination: When i reaches the length of the array n-1, the loop comes to an end. Since the loop invariant ensures that the subarray arr[0...n-2] is sorted, the full array is now sorted. Since the final member in the array is the greatest element, it is already in the proper location.

The selection sort method is therefore accurate since, as a result of the loop invariant, we may conclude that the entire array is sorted at the conclusion of the procedure.