Arguing the Correctness of Selection Sort

Selection Sort Overview:

Selection sort works by dividing the array into two parts: the sorted portion and the unsorted portion. Initially, the sorted portion is empty, and the unsorted portion contains all the elements. The algorithm repeatedly selects the smallest element from the unsorted portion and swaps it with the leftmost unsorted element, thereby growing the sorted portion by one element with each iteration.

Formal Argument for Correctness:

To argue the correctness of the selection sort algorithm, we can rely on the following concepts:

1. **Loop Invariant:**
   - A loop invariant is a condition that holds true before and after each iteration of a loop. Proving that the loop invariant holds throughout the execution of the algorithm is a common technique for proving correctness.

2. **Inductive Argument:**
   - We use mathematical induction to prove that, after each iteration, the array is sorted up to the current index.

Loop Invariant

Invariant: At the start of each iteration i (0-based index) of the outer loop, the subarray $arr[0:i]$ is sorted, and each element in $arr[0:i]$ is smaller than or equal to every element in the subarray $arr[i:n]$, where n is the length of the array.

Proof:

1. **Initialization (Base Case):**
   - Before the first iteration (i.e., when $i = 0$), the sorted subarray is empty. Trivially, an empty subarray is sorted, so the invariant holds.

2. **Maintenance (Inductive Step):**
   - Assume the invariant holds at the start of the ith iteration. This means the subarray $arr[0:i]$ is sorted and all elements in $arr[0:i]$ are smaller than or equal to all elements in $arr[i:n]$.
   - During the ith iteration, the algorithm selects the minimum element from the subarray $arr[i:n]$ and swaps it with $arr[i]$.
   - After the swap, the subarray $arr[0:i+1]$ is sorted, and all elements in $arr[0:i+1]$ are smaller than or equal to all elements in $arr[i+1:n]$.
   - Therefore, the invariant holds after the ith iteration.

3. **Termination:**
   - The algorithm terminates after n iterations, where n is the length of the array.

- At this point, $i = n$, so the entire array $arr[0:n]$ is sorted, and the invariant ensures that the entire array is correctly sorted.

*Conclusion

Since the loop invariant holds for the base case, is maintained after every iteration, and ensures the array is sorted at termination, we can conclude that the selection sort algorithm correctly sorts the array.

Why This Argument Works:

- **Correctness Through Invariance:** The loop invariant approach effectively breaks down the problem into manageable parts, ensuring that the algorithm maintains correctness as it progresses. By ensuring that the array is partially sorted correctly after each iteration, we guarantee that the entire array will be sorted once the algorithm finishes.

- **Termination:** The argument also ensures that the algorithm terminates, which is an important aspect of proving correctness. Since selection sort always progresses towards completion by reducing the unsorted portion by one element each iteration, it will eventually finish sorting the entire array.

In summary, the selection sort algorithm is correct because it maintains a loop invariant that ensures partial correctness at each step and guarantees that the array is fully sorted upon termination.