CS271: Data Structures

Correctness Proof of Selection Sort

Consider the following code segment which adds the integers in an array.

```
ALGORITHM: sort array of integers
input: array A[1..n] of n unsorted integers
output: same integers in array A now in sorted order

1  for i = 1 to n-1
2     min = i
3     for j = i+1 to n
4        if A[j] < A[min]
5           min = j
6     swap A[i] with A[min]
```

First we prove the correctness of the inner loop: lines 3 to 5

Loop Invariant:
Before the start of each loop, `A[min]` is less than or equal to `A[i..j-1]`.
Initialization:
Prior to the first iteration of the loop, `j=i+1`. So the array segment `A[i..j-1]` is really just spot `A[i]`. Since line 2 of the code sets `min = i`, we have that `min` indexes the smallest element (the only element) in subarray `A[i..j-1]` and hence the loop invariant is true.
Maintenance:
Before pass `j`, we assume that `min` indexes the smallest element in the subarray `A[i..j-1]`. During iteration `j` we have two cases: either `A[j] < A[min]` or `A[j] ≥ A[min]`. In the second case, the if statement on line 4 is not true, so nothing is executed. But now `min` indexes the smallest element of `A[i..j]`. In the first case, line 5 switches `min` to index location `j` since it is the smallest. If `min` indexes an element less than or equal to subarray `A[i..j-1]` and now `A[j] < A[min]`, then it must be the case that `A[j]` is less than or equal to elements in subarray `A[i..j-1]`. Line 5 switches min to index this new location and hence after the loop iteration finishes, `min` indexes the smallest element in subarray `A[i..j]`.
Termination:
At termination of the inner loop, `min` indexes an element less than or equal to all elements in subarray `A[i..n]` since `j = n+1` upon termination. This finds the smallest element in this subarray and is useful to us in the outer loop because we can move that next smallest item into the correct location.