Name : Umamah Hussain

Roll No. : 21L-1858

Section : BSCS – 4H

**Design and Analysis of Algorithms**

**Assignment 1**

**Question 1)**

**Loop invariants**

1. **Find Min**

**Proof:**

**Initialization:**

Before the first iteration of the loop, we set min to be the first element of A. Therefore, the loop invariant holds true, since the minimum value in the array so far is just the first element of the array.

**Maintenance:**

Suppose the loop invariant holds true at the start of the ith iteration of the loop. Then, the minimum value found in the array so far from the elements A[1] to A[i-1] is stored in the variable min.

Now, we consider the (i+1)th iteration of the loop. We compare the (i+1)th element of A to min. If the (i+1)th element of A is less than min, we set min to be the (i+1)th element of A. Otherwise, we leave min unchanged. In either case, the loop invariant still holds true at the start of the (i+1)th iteration of the loop, since min still contains the minimum value found in the array so far from the elements A[1] to A[i].

**Termination:**

When the loop terminates, we have examined all the elements of the array. By the loop invariant, min contains the minimum value found in the array so far. Therefore, min must be the minimum value in the entire array. The algorithm returns min, which is the correct output.

1. **Bubble Sort**

**Proof:**

**Initialization:**

Before the first iteration of the outer loop, i = 1, and the subarray A[1...i] is empty, so it is sorted in non-decreasing order. Therefore, the loop invariant holds true.

**Maintenance:**

Suppose the loop invariant holds true at the start of the ith iteration of the outer loop. That is, the subarray A[1...i-1] is sorted in non-decreasing order. Now, we consider the (i+1)th iteration of the outer loop.

In this iteration, we run the inner loop from j = 0 to n-i-2. By the loop invariant, the subarray A[0...i-1] is sorted in non-decreasing order, so we know that A[1], A[2], ..., A[i-1] are all less than or equal to any element in A[i...n-1].

We compare adjacent elements A[j] and A[j+1] in the inner loop. If A[j] > A[j+1], we swap them. After we have gone through the inner loop, the largest element in the subarray A[0...n-i-1] is at the end of the subarray. Therefore, the subarray A[1...i] is now sorted in non-decreasing order. Thus, at the start of the (i+1)th iteration of the outer loop, the subarray A[1...i] is sorted in non-decreasing order. Therefore, the loop invariant holds true for the (i+1)th iteration.

**Termination:**

When the outer loop terminates, i = n-1, and the subarray A[0...i-1] is A[0...n-2]. By the loop invariant, A[0...n-2] is sorted in non-decreasing order, which means that A is sorted in non-decreasing order. The algorithm returns A, which is the correct output.

Therefore, we have shown that the "Bubble Sort" algorithm is correct using loop invariants

1. **Selection Sort**

**Proof:**

**Initialization:**

Before the first iteration of the outer loop, i = 0, and the subarray A[0...i-1] is empty, so it is sorted in non-decreasing order. Therefore, the loop invariant holds true.

**Maintenance:**

Suppose the loop invariant holds true at the start of the ith iteration of the outer loop. That is, the subarray A[0...i-1] is sorted in non-decreasing order. Now, we consider the (i+1)th iteration of the outer loop.

In this iteration, we find the index of the smallest element in the subarray A[i...n-1] and store it in smallest\_Index. By the loop invariant, we know that A[0], A[1], ..., A[i-1] are all less than or equal to any element in A[i...n-1]. Therefore, we know that A[smallest\_Index] is the smallest element in the subarray A[i...n-1].

If smallest\_Index is not equal to i, we swap A[i] and A[smallest\_Index]. This puts the smallest element in the subarray A[i...n-1] in the correct position in the sorted subarray A[0...i], which maintains the loop invariant.

Thus, at the start of the (i+1)th iteration of the outer loop, the subarray A[1...i] is sorted in non-decreasing order. Therefore, the loop invariant holds true for the (i+1)th iteration.

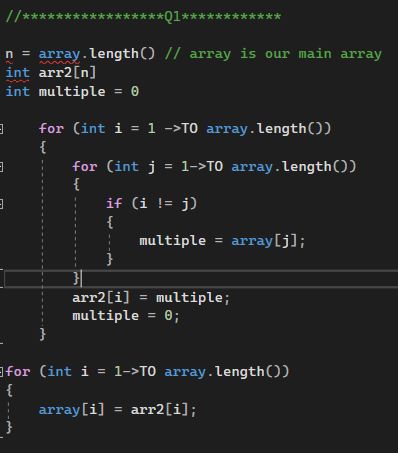
**Termination:**

When the outer loop terminates, i = n-1, and the subarray A[0...i-1] is A[0...n-2]. By the loop invariant, A[0...n-2] is sorted in non-decreasing order, which means that A is sorted in non-decreasing order. The algorithm returns A, which is the correct output.

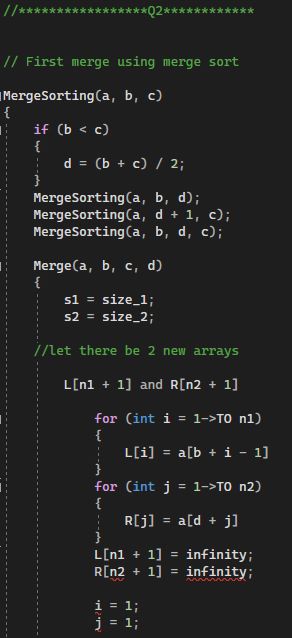
Therefore, we have shown that the "Selection Sort" algorithm is correct using loop invariants.

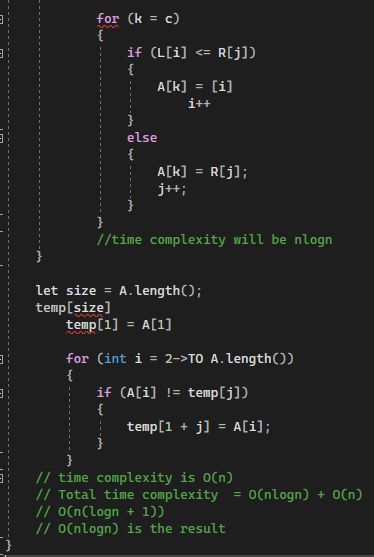
**Question 2)**

**1)**

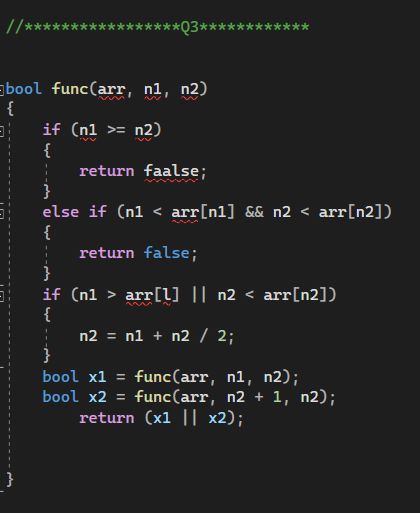


**2)**

****

****

**3)**

****

4)

def largest(array1, array2, n) :

if length(array1) == 0 :

return array2[n]

elif length(array2) == 0 :

return array1[n]

mida1 = length(array1) // length is 3 # integer division

mida2 = length(array2) // length is 3

if middle1 + middle2 < n :

if arr1[middle1] > arr2[middle2]:

return largest(array1, array2[middle2 + 1:], n - middle2 - 1)

else :

return largest(arr1[middle1 + 1:], arr2, n - middle1 - 1)

else:

if array1[middle1] > array2[middle2]:

return largest(array1[:middle1], array2, n)

else :

return largest(array1, array2[:middle2], n)