CMPT 125 Assignment 4 Solutions

Question 1 (15 marks)

In the class we have discussed how the binary search works and importance of using invariant or assertions to verify our program. In this problem we will be using assertions in our binary search program. Throughout this question, please refer to the Lecture 12 slide titled “Binary Search Code”, which shows a version of the binary search implemented using a while loop; the code excerpts in this question are taken from that slide.

a) What should be the first check or invariant when we implement binary search? Hint: this is a condition on the array required for using binary search (but not for linear search).

Write an assertion for your condition by filling in the blank inside the assert command with a function call, and implement the function that assert calls to check the invariant.

```c
int BinarySearch(int arr[], int len, int target) {
    assert( ________ );
    ...
}
```

**Solution:** Here, we just want to make sure that the input array is sorted. Note that checking whether an array is sorted takes $O(N)$ time, which is actually slower than the $O(\log N)$ time of binary search.

```c
int isSorted(int arr[], int len) {
    // go through every element of the array, and make sure
    //   that the next element is larger than the previous
    for (int i = 1; i < len; i++) {
        if (arr[i] < arr[i-1]) {
            // if any element is smaller than the previous one,
            //   the array is not sorted
            return 0;
        }
    }
    // if we made it all the way here without returning, then
    //   the array is sorted
    return 1;
}
```

```c
int BinarySearch(int arr[], int len, int target) {
    assert(isSorted(arr, len));
    ...
}
```

b) What must be true about target right after the while loop? Write a possible loop invariant for this step in the form of an assert statement. Simply write the logical condition inside the assert

```c
...
command. Hint: this can be done through a short, one-line expression, and no function is necessary.

```c
int BinarySearch(int arr[], int len, int target) {
    ...
    while (first <= last) {
        assert(_________);
        int mid = (first+last) / 2;
    }
}
```

**Solution:** There are many possible similar assertions here. Perhaps the cleanest one which accounts for both the cases of target being found and not being found is as follows: the value of target should be between arr[first] and arr[last].

```c
int BinarySearch(int arr[], int len, int target) {
    ...
    while (first <= last) {
        assert(arr[first] <= target && arr[last] >= target);
        int mid = (first+last) / 2;
    }
}
```

c) Make a short argument (4-5 sentences) to show that the Binary Search code (version with while loop) from class terminates.

**Hint:** Think about what happens to the values of “first” and/or “last” after every iteration in the while loop.

**Solution:** In every iteration in the while loop, either first increases, or last decreases. This means that the difference between first and last decreases by at least 1 every iteration. Therefore, after at most last-first (that's last minus first) iterations, the while loop will terminate.
Question 2 (12 marks)

The following function takes as input an integer array and its length, and sorts the elements in ascending order.

```c
void sort(int arr[], int len) {
    for (int i = 0; i < len; i++) {
        for (int j = i+1; j < len; j++) {
            if (arr[i] > arr[j]) {
                int tmp = arr[i];
                arr[i] = arr[j];
                arr[j] = tmp;
            }
        }
    }
}
```

Prove its correctness by answering the following.

a) If the “if” statement is entered, what happens to the values of arr[i] and arr[j]?

**Answer:** They are swapped.

b) In the first iteration in the outer loop, i is 0. What is true about arr[0] after the inner loop involving j finishes executing? Briefly explain in ~2-3 sentences.

**Answer:** arr[0] would contain the smallest element of the array. This is because i is 0, and j scans the entire array, from 1 to the end. Whenever a smaller element is found (i.e. arr[j] < arr[i], the condition in the if statement), it is swapped into position 0.

c) In the second iteration in the outer loop, i is 1. What is true about arr[1] after the inner loop finishes executing?

**Answer:** When i is 1, we can ignore arr[0], and make the same argument as in part b: arr[1] contains the smallest element in arr[1...len-1], since any element smaller than arr[1] is swapped into arr[1].

d) In the beginning of the outer loop involving i, what is true about the subarray arr[0...i-1] before the inner loop?

**Answer:** The subarray arr[0...i-1] is sorted, and contains the smallest i elements of the array.

e) In the outer loop involving i, what is true about the subarray arr[0...i] after the inner loop?

**Answer:** Since the remaining smallest element is swapped into arr[i] after the inner loop, now the arr[0...i] contains the smallest i+1 elements in sorted order.

f) To finish the proof, apply your argument in part e) to the largest value of i inside the outer loop (in ~2-3 sentences).
**Answer:** The largest value of $i$ that is active in the outer loop is $len - 1$. Therefore, after the outer loop is finished, $arr[0...len-1]$ contains the smallest $len$ elements in sorted order. In other words, the entire array is sorted.