CS340 Homework 1 25 points Name Stuart Cech

SAVE AS A PDF FILE BEFORE SUBMITTING ON MOODLE.

Consider the following algorithm to sort n numbers stored in array A:

1. Find the smallest element of A and exchange it with the element in A[1].
2. Find the second smallest element of A, and exchange it with A[2].
3. Continue in this manner for the first n-1 elements of A.

1. (4 points) What loop invariant does this algorithm maintain?

loop invariants that after the i-th iteration the first i elements of the array 'A' are sorted and they are the smallest i elements in the array

2. (2 points) Prove the loop invariant initialization phase.

Before the loop starts ($i = 0$), the subarray containing the first 0 elements is sorted since it has no elements, and it contains the smallest 0 elements of the array. So loop invariant holds true prior to first iteration.

3. (2 points) Prove the loop invariant maintenance phase.

Assume the loop invariant holds true before the i-th iteration, During the i-th iteration, the algorithm finds the i-th smallest element and swaps it with A[i]. Now the smallest i elements are at the front of the array and they are sorted. Therefore, the loop invariant holds for the next iteration.

4. (2 points) Prove the loop invariant termination phase.

The loop ends after the $n-1$-th iteration. According to the loop invariant, the first $n-1$ elements are sorted and are the smallest $n-1$ elements. Since the array has n elements and the smallest $n-1$ elements are already sorted, it must be that the remaining element is the largest and it is in its correct place. The array is sorted when loop terminates.

5. (2 points) What are the best and worst case time complexities for the above sorting algorithm (in theta notation)?

Both the best and worst case time complexities are $\Theta(n^2)$

Following is code for an algorithm for finding if two numbers in a *sorted* array, A, sum to a given number, X.

```
public static boolean twoNumbersEqualX(int[] A, int X) {
        int start = 0;
        int end = A.length - 1;
        while (start < end) {
            int result = A[start] + A[end];
            if (result == X) return true;
            else if (result > X) end--;
            else start++;
        }
        return false;
}
```

1. (4 points) Consider the while loop. What is the loop invariant?

The start of each iteration of the while loop, no pair of numbers A[i] and A[j] such that $0 \leq i < start \leq j < end \leq A.length - 1$ will sum to x.

2. (2 points) Prove that the loop invariant is true in the initialization stage:

start is 0 and end is A.length - 1. This means that we have not ruled out any elements, and the entire array is within start and end range.

3. (2 points) Prove that the loop invariant is true in the maintenance stage:

Assume that the loop invariant holds at the beginning of an iteration. If result > x we decrement end. A is sorted and no pair that includes A[end] will satisfy the condition. The loop invariant holds because we have now ruled out the numbers that would pair with A[end] to sum to x.

If result < x we decrement start for similar reasons. Any pair that includes A[start] would be to small so they can be ruled out.

4. (2points) Prove that the loop invariant is true upon termination:

start will be equal to or greater than end. At this point we have checked all possible pairs and found none that sum to x, so the invariant still holds, ruling out any pair in A that could sum to x.

5. (3 points) What is the time complexity of this algorithm?

2n iterations $\longrightarrow O(n)$