CS330 - HW #1

Chandini Toleti
U29391556

(1)

A.

I. The algorithm sorts integers in the array from least to greatest (left to right).

II. The **smallest** value swaps can take on for an array of length *n* is zero. This occurs when the array is already sorted from least to greatest. Therefore, no values will be swapped after the algorithm runs. On the other hand, when an array is sorted in the opposite direction (greatest to least), swaps takes on its **largest value**. For an array of size *n*, this value is the summation of *1 to n-1* or $\frac{n(n-1)}{2}$. This means the algorithm has a worst-case time efficiency of O(n^2).

iii.

[A] False. Consider the counter-example: A={4,3,2,1}. After the first three iterations of the inner loop, j=3 and A = {3,1,2,4}. At this point, A[1…j+1] is {3,1,2,4} which is **not** sorted in increasing order. Therefore, the assertion is false.

[B] True. Consider the inductive proof.

Say the base case is j=k=1. Consider arbitrary array A= {x,y,z}, let x be A[1], y be A[2], and z is A[3]. After the first iteration of the inner loop j=1 and by Observation 1: x<y. Since j=1, A[j+1] = y and A[1…j+1] = {x, y} it is true that A[j+1] is the largest in the subarray when k=1.

Now consider k+1. Consider A = {x,y,z} with the same principles. By observation 1, after the second iteration of the inner loop, x<y and y<z. Since j=2, A[j+1] = z and A[1…j+1] = {x,y,z} it is true that A[j+1] is the largest in the subarray when k=2.

By induction, we've considered all elements from 1 to n-1, and we've proved A[n-1] will be the greatest in said subarray. By induction, we know this implies the assertion is true for all natural numbers n as we continue to compare A[n-1] to A[n].

Observation 1: The code within the "if" statement swaps the element considered with the following element if it's greater. For example: Say A[1] = 4 and A[2]=3, then at the end of the inner loop, A[1] = 3 and A[2]=4.

```
A[1] = 4+3; //A[1] = 7
A[2] = 7-3;  //A[2] = 4
A[1] = 7-4; //A[1] = 3
```

[C] False. Consider the counter-example: A={4,3,2,1}. After the first three iterations of the inner loop, j=3 and A = {3,1,2,4}. At this point, A[1…j+1] is {3,1,2,4} and A[1] = 3. 3 is **not** the smallest element in the subarray. Therefore, the assertion is false.

[D] False. Consider the counter-example: A={4,3,2,1}. After the first iteration of the inner loop, j=1 and A={3,4,2,1}. At this point, A[j]= A[1]=3 and A[j…n]= {3,4,2,1}. At this point, A[1] is **not** the smallest value in said subarray. Therefore, the assertion is false.

B. (i) The final values of swaps and dec will always be equal after both algorithms finish running. (ii) Consider the direct proof: Since Both algorithms receive the same input, If P pairs of elements are out-of-order for one algorithm, P pairs of elements are also out-of-order for the other. In an array with P out-of-order pairs, both algorithms will perform $\frac{P(P-1)}{2}$ swaps. Therefore, at the end of the algorithm, the same number of swaps will occur, although possibly in a different order. By observation of the algorithms, we see this implies that swaps and dec will be equivalent.

2.

Inputs: 2-D Array H[][] (hospital preferences), 2-D Array R[][] (resident preferences), M[] Array (representing matches)
Return: String “yes” or “no”

```
n → length(M)
Initialize Inverse:
    InvH[n][n]//initialize with zeroes
    InvR[n][n]//initialize with zeroes
    For i=1 to n do:
        For j=1 to n do:
            InvH[i][H[i][j]]=j
            InvR[i][R[i][j]]=j
Compare:
    For i=1 to n do:
        If H[i][M[i]] != 1 AND R[M[i]][i]!= 1 do:
            Score → H[i][M[i]] + R[M[i]][i]
            For j=1 to n do:
                If H[i][j] + R[j][i]<score AND j!=I do:
                    Return “no”
    Return “yes”
```