*Owen Lindsey*

*Professor Demland, David*

*CST-201 Exercise 2*

*09/14/2024*

**Exercise 1.3 - 1**

*Algorithm Explanation*

1. *Define the Sort method, which takes an array A[ ] as input and returns the sorted array S[ ]. It uses an additional array Count[ ]to keep track of how many elements are smaller than each element in A[ ].*
2. *In the first nested loop, initialize the Count[ ] array by comparing each element of A[ ] with every other element. For each pair of elements, if A[ i ] < A[ j ], the count for A[ j ] is incremented; otherwise, the count for A[ i ] is incremented. This count indicates how many elements are smaller than each element in A[ ].*
3. *After counting, the Count[ ] array provides the positions where each element should be placed in the sorted array S[ ]. We iterate over the array A[ ], placing each element in its correct position in S[ ] according to the value in Count[ ].*
4. *The Sort method returns the sorted array S[ ].*
5. *In the main method, we initialize an example array, call the Sort method, and print both the original and sorted arrays.*

**Exercise 1.3 - 1**

1. *Applying [ 60, 35, 81, 98, 14, 47] to the list.*
2. *Start with an array A[] of n elements. For example, A[] = [60, 35, 81, 98, 14, 47].*
3. *Create an array Count[] of the same size n and initialize all values to 0. This array will store the number of smaller elements for each element in A[].*
4. *Iterate over the array A[] using two nested loops:*

*The outer loop runs from i = 0 to n-2.*

*The inner loop runs from j = i + 1 to n-1.*

*For each pair (i, j), we compare A[i] and A[j]:*

*If A[i] < A[j], increment Count[j] because A[i] is smaller.*

*If A[i] >= A[j], increment Count[i] because A[j] is smaller.*

*Example:*

*For A[0] = 60 and A[1] = 35: since 60 > 35, we increment Count[0] (for*

*60).*

*This process continues until all comparisons are done.*

*Count[] = [3, 1, 4, 5, 0, 2], where each value represents how many elements are smaller than the corresponding element in A[].*

1. *Now that we know how many elements are smaller than each element, we use this information to place each element in its correct position in a new sorted array S[].*

*For each i, the element A[i] goes into position Count[i] in S[].*

*Example:*

*A[0] = 60 has Count[0] = 3, so we place 60 at position 3 in S[].*

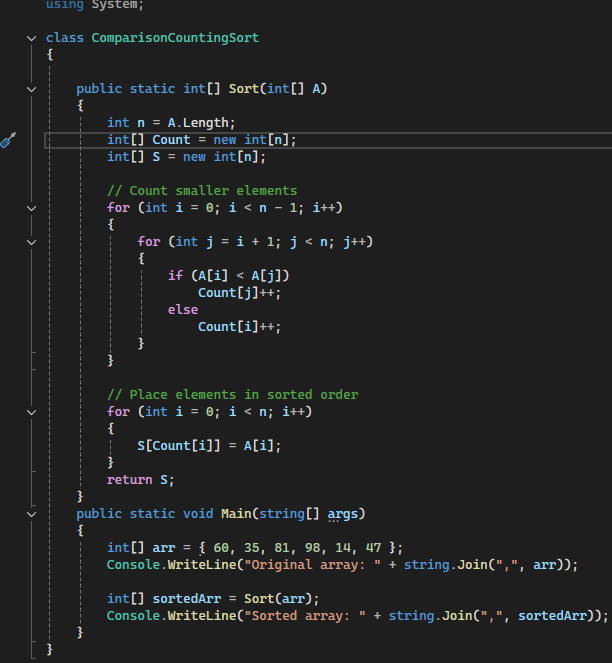
*After processing all elements, the sorted array is S[] = [14, 35, 47, 60, 81, 98].*

1. *The final sorted array S[] is returned as the output.*

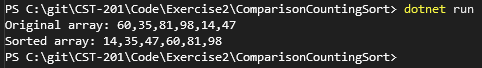
**Exercise 1.3 - 1**

*b. Is the algorithm stable?*  
 *No, the algorithm is not stable. In the counting phase, when two*   *elements are equal, the order of the original elements are not kept.*

*c. Is the algorithm in-place?*  
 *No, it is not an in-place algorithm because it requires an additional array*   *S[] to store the sorted elements.*

**Program Screenshots:**   
  
*ComparisonCountingSort code:*  


*ComparisonCountingSort Output:*

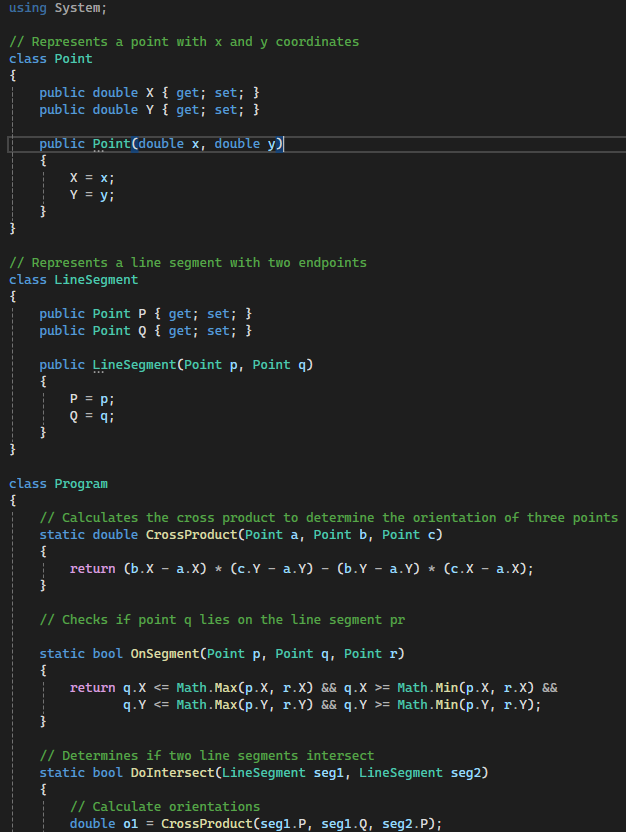


**Exercise 1.3 - 10**

*Algorithm Explanation*

1. *Define a Point class to represent a point x and y coordinates, and a LineSegement class that holds two Point objects, P and Q, representing the endpoints of a line segment.*
2. *Create a CrossProduct method to compute the orientation of three points. This helps in determining the relative position of the points to each other, which is essential for checking whether two line segments intersect.*
3. *Define a OnSegment method to check if a given point lies directly on a line segment. This is important for handling special cases where the segments are collinear.*
4. *Define a DoIntersect method to check if a given point lies directly on a line segment. It first calculates the cross products (orientations) for four possible combinations of the two-line segments` endpoints.*
5. *Check the general case of intersection by evaluating if the orientations differ. If the orientations are different, the line segments intersect.*
6. *Special cases are also checked in the DoIntersect method to account for situations where the endpoints of one segment lie direcly on the other segment.*
7. *In the Main method, we read the coordinates for two line segments from the user and create LineSegment objects based on those inputs.*

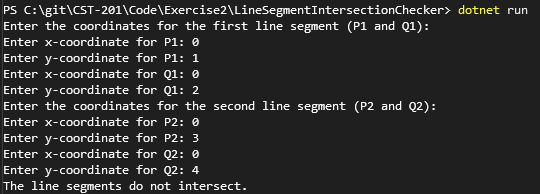
**Program Screenshots:**   
  
*LineSegmentChecker code 1/2:*



*LineSegmentChecker code 2/2:*



*LineSegmentChecker Output No Intersect:*

  
  
*LineSegmentChecker Output Valid Intersects:*  
