Joseph Martinez
PID: 3816842

Problem #1: (45 pts = 7 pts + 8 pts + 20 pts + 10 pts)

(a) Given the following list of numbers:

90 8 7 56 123 235 9 1 653
trace the execution for:

(a.1) QuickSort (pivot = the first element).

90 8 7 56 123 235 9 1 653

{ **8**, 7, 56, 9, 1 } 90 { **123**, 235, 653 }

{ **1**, 7 } 8 { **56**, 9 } 123 { **235**, 653 }

1 { 7 } { 9 } 56 235 { 653 }

1 7 8 9 56 90 123 235 653

(a.2) MergeSort.

90 8 7 56 123 235 9 1 653

Elements = 9
Partition = Element / 2 => 4.5 => 4

[90, 8, 7, 56, | 123, 235, 9, 1, 653]

[90, 8, 7, 56] [123, 235, 9, 1, 653]

S1 Partition = 4/2 => 2

S2 Partition = 5 / 2 = 2.5 => 2

[90, 8 | 7, 56] [123, 235 | 9, 1, 653]

[90, 8] [7, 56] [123, 235] [9, 1, 653]

S1 Partition = 2 / 2 => 1
S2 Partition = 2 / 2 => 1

S1 Partition = 2 / 2 => 1
S2 Partition = 3 / 2 = 1.5 => 1

[90 | 8] [7 | 56]

[90] [8] [7] [56]

[123 | 235] [9 | 1, 653]

[123] [235] [9] [1, 653]

S3 Partition = 2 / 2 => 1

[90] [8] [7] [56]

[90] [8] [7] [56]

[123] [235] [9] [1 | 653]

[123] [235] [9] [1] [653]

Joseph Martinez
PID: 3816842

MERGE

[8 , 90] [7, 56]

[7, 8, 56, 90]

[123, 235] [1 , 653]

[123, 235] [1, 9, 653]

[1, 9, 123, 235, 653]

Result: [1, 7, 8, 9, 56, 90, 123, 235, 655]

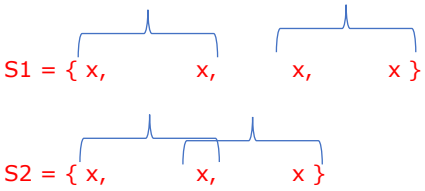
(b) How would you implement (in Java) the Mergesort algorithm without using recursion?

PLEASE SEE JAVA IMPLEMENTATION

(c) Given an array of distinct integers. You need to find the maximum product of two integers in an array. For example, consider the array {-10, -3, 5, 6, -2}. The maximum product is formed by the (-10, -3) or (5, 6) pair.
Suggest a sub-quadratic running time complexity algorithm (only pseudo code) to solve this problem. What is the overall complexity of your solution?

/**
To accomplish finding all pairs of integers that yield the largest product we first need to consider organizing/sorting the elements first. We can use Merge Sort which will guarantee the worst time in O(n log n). Other algorithms like Radix and Bucket sort would have provided a possible O (N), however, considering the input is random and has negative numbers, we would run into potential problems with space complexity.

After sorting, we can check if the list is greater or equal to three. Anything less can be returned. If we know the list has at least three elements, we can check the right and left side indexes. Like so…



When the list is sorted we will always have the largest numbers on the left (negative) and right (positive) half of the list. Last, we compare the product of the left and right half and return the values base on the three conditions: left > right, left < right, or, left == right.

**/

```
function findMaxProducts ( int[] list )
        mergeSort (list)
        if ( list.size >= 3)
                if ( ( list[0] * list[1] ) > ( list[size -1] * list[size -2] ) )
                        return list = { list[0], list[1] }
                else if ( ( list[0] * list[1] ) < ( list[size -1] * list[size -2] ) )
                        return list = { list[size - 1], list[size - 2] }
        else if (list.size <= 2)
                return list
```

Joseph Martinez
PID: 3816842

Problem #2: (20 pts)

Given a binary array, implement in **Java** an algorithm to sort it in *linear running time complexity and constant space complexity*.

Example:
Input: [1,0,1,0,0,0,1,0,1,0,0,1]
Output: [0,0,0,0,0,0,0,1,1,1,1,1]

PLEASE SEE JAVA IMPLEMENTATION

Problem #3: (35 pts = 5 pts + 8 pts + 8 pts + 14 pts)

(a) Either draw a graph with the following specified properties, or explain why no such graph exists:
A simple graph with five vertices with degrees 2, 3, 3, 3, and 5.

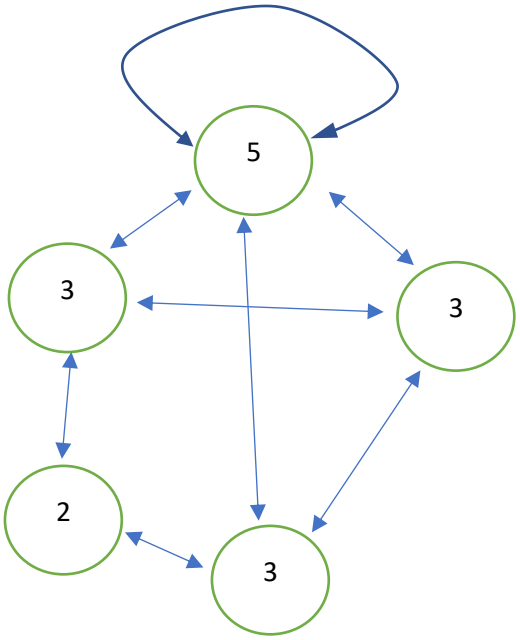
Vertex = 5;

Degree = { 2, 3, 3, 3, 5 }

$2\,|E| = \sum \deg(v)$

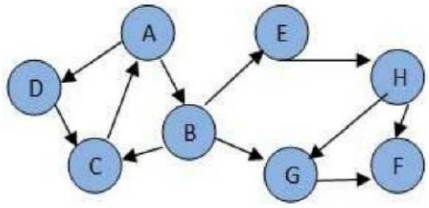
$2\,|E| = \sum 16$

$|E| = 8$ Edges



Joseph Martinez
PID: 3816842

(b) Consider the following graph. If there is ever a decision between multiple neighbour nodes in the BFS (Breadth First Search) or DFS (Depth First Search) algorithms, **assume we always choose the letter closest to the beginning of the alphabet first**.



(b.1) In what order will the nodes be visited using a Breadth First Search (Queue)?

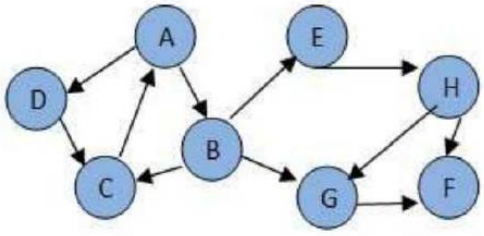
Out Put: **A, B, C, D, E, H, F, G**

Queue									
~~A~~	~~B~~	~~D~~	~~C~~	~~E~~	~~G~~	~~E~~	~~G~~	H	F

(b.2) In what order will the nodes be visited using a Depth First Search (Stack)?

Out Put: A, B, C, G, F, E, H, D

Stack
D
~~H~~
~~E~~
~~F~~
~~G~~
~~E~~
~~C~~
~~B~~
A



Joseph Martinez
PID: 3816842

(c) Show the ordering of vertices produced by the topological sort algorithm given in class starting from vertex V_1 when it is run on the following DAG (represented by its adjacency list, in-degree form).

V0, V2, V4, V6, V7
V1, V5, V6, V7,
V0, V4, V6, V7
V0, V3
V1, V3
V0, V2, V6, V7

V_0	---
V_1	---
V_2	V_0, V_1
V_3	V_0, V_1
V_4	V_0, V_2
V_5	V_1
V_6	V_2, V_4, V_5
V_7	V_6