Name: _____ Andrew Id: _____

15-121 Assessment 6

60 minutes. No calculators, no notes, no books, and no computers.
Place answers on the answer sheet.

1. **Short Answer**

   Answer the following in 1-2 sentences; longer answers will be marked incorrect.

   (a) (2 points) Is selection sort a stable sort? Explain why or why not.

   (b) (1 point) Why doesn't quick sort require a step to combine the sorted left and right parts like merge sort does?

   (c) (1 point) Why does the choice of pivot matter with regards to the efficiency of quick sort?

2. **Searching and Sorting**

Consider the following list of integers:

63, 36, 81, 74, 19, 42, 91, 25, 58

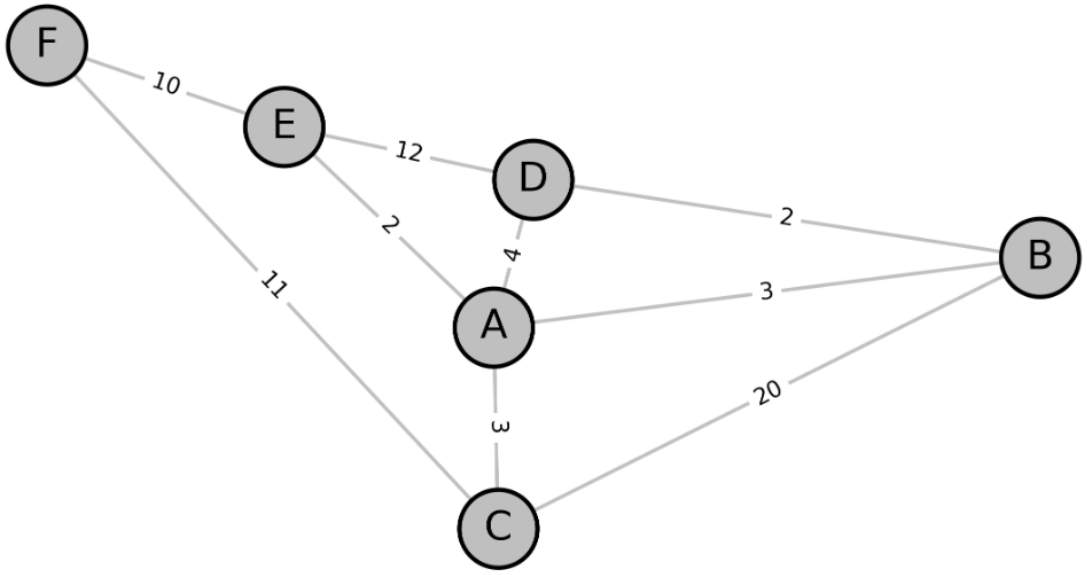
(a) (4 points) Assuming that the array provided at the beginning of this problem receives *two passes* from the loop in the Selection Sort algorithm (assume it is the first and second pass, with `i = 0` and then `i = 1`), what will be the new state of the array? Write your answer in the provided boxes. Show your work in the blank space below the boxes. Answers without appropriate work may receive no credit.

(b) (5 points) Assuming that the array provided at the beginning of this problem receives a single pass from the loop in the Bubble Sort algorithm, what will be the new state of the array? Write your answer in the provided boxes. Show your work in the blank space below the boxes. Answers without appropriate work may receive no credit.

(c) (5 points) Assuming that the array provided at the beginning of this problem is partitioned using the Quick Sort partition algorithm (use 58 as your pivot) as presented in class, what will be the new state of the array? Write your answer in the provided boxes. Show your work in the blank space below the boxes. Answers without appropriate work may receive no credit.

3. **Minimum Spanning Trees**

Consider the following weighted graph:



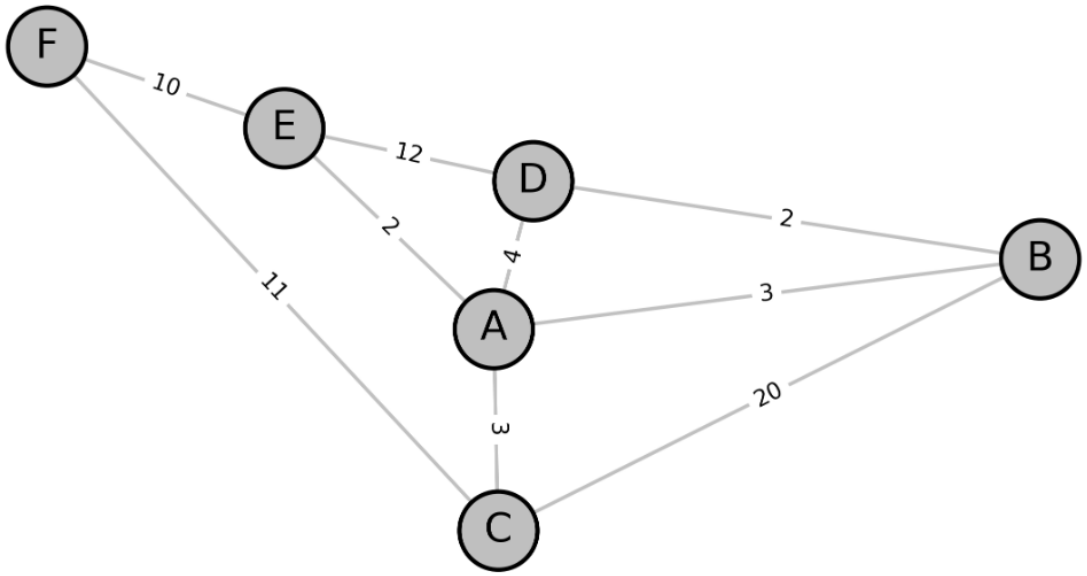
For the two problems below, **if the algorithm requires a starting node then start at node E**.

(a) (5 points) Find a minimum spanning tree for this graph using Kruskal's algorithm. Be sure to show your work in such a way that it is clear you understand how the algorithm works (for example, by explicitly stating the order in which edges are added to the MST). Answers without appropriate work will receive no credit. Draw your final MST onto the provided graph below.

(b) (5 points) Find a minimum spanning tree for this graph using Prims's algorithm. Be sure to show your work in such a way that it is clear you understand how the algorithm works (for example, by explicitly stating the order in which edges are added to the MST). Answers without appropriate work will receive no credit. Draw your final MST onto the provided graph below.

4. (10 points) **Dijkstra**

Using Dijkstra's algorithm, find the shortest paths from F to every node in the following graph. Be sure to show your work as you run the algorithm. Answers without appropriate work will not receive credit.



5. (12 points) Graph Code

Recall the following code, slightly simplified from what we wrote in class, for implementing a graph:

```
public class Edge {
    private Vertex src;
    private Vertex dst;
    private int weight;

    public Edge(Vertex src, Vertex dst, int weight) {
        this.src = src;
        this.dst = dst;
        this.weight = weight;
    }

    public Vertex getSrc() {
        return this.src;
    }

    public Vertex getDst() {
        return this.dst;
    }

    public int getWeight() {
        return this.weight;
    }
}

public class Vertex {
    private String name;
    private ArrayList<Edge> edges = new ArrayList<Edge>();

    public Vertex(String name) {
        this.name = name;
    }

    public void addEdge(Vertex dst, int weight) {
        Edge tmp = new Edge(this, dst, weight);
        edges.add(tmp);
    }

    public String getName() {
        return this.name;
    }

    public ArrayList<Edge> getEdges() {
        return edges;
    }

    @Override
    public int hashCode() {
        return Objects.hash(edges, name);
    }
```

```
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Vertex other = (Vertex) obj;
        return Objects.equals(edges, other.edges) && Objects.equals(name, other.name);
    }
}

public class Graph {
    private HashMap<String, Vertex> vertices = new HashMap<String, Vertex>();

    public Graph(String filename) {
        FileReader fr;
        try {
            fr = new FileReader(filename);
        } catch (FileNotFoundException fne) {
            System.err.println("File not found!");
            return;
        }

        Scanner inp = new Scanner(fr);
        while (inp.hasNextLine()) {
            String line = inp.nextLine();
            String[] vals = line.split(",");
            if (vals.length != 3) {
                System.err.println("Invalid file format");
                System.err.print(line);
                return;
            }
            String src = vals[0];
            String dst = vals[1];
            int w = Integer.parseInt(vals[2]);

            addVertex(src);
            addVertex(dst);
            addEdge(src, dst, w);
            addEdge(dst, src, w);
        }
        inp.close();
    }

    public void addVertex(String vertexName) {
        if (!vertices.containsKey(vertexName)) {
            Vertex tmp = new Vertex(vertexName);
            vertices.put(vertexName, tmp);
        }
    }
```

```
    public void addEdge(String src, String dst, int weight) {
        Vertex srcV = vertices.get(src);
        Vertex dstV = vertices.get(dst);

        srcV.addEdge(dstV, weight);
    }

     /**
     * Determine if there exists a path between two given nodes in the graph.
     * This method checks whether dst is reachable from src by following
     * the edges of the graph.
     *
     * @param src the starting node for the search
     * @param dst the destination node we want to reach
     * @return true if a path exists from src to dst, false otherwise
     */
    public boolean hasPath(String src, String dst) {
        // You will write this code
    }


    public static void main(String[] args) {
        Graph g = new Graph("graph.txt");
        System.out.println(g.hasPath("A", "B"));
    }
}
```

Write the method hasPath(String src, String dst) in the Graph class. For full credit, your solution should be better than $O(N^2)$ where N is the number of vertices in the graph.