1.) (8 points) Consider the following graph:

a.) (2 points) Create an adjacency matrix for this graph where $M[i][j]$ denotes the weight of an edge from node $i$ to node $j$.

b.) (3 points) Perform a depth-first traversal on this graph starting from node $a$. Show your results as a table shown below. Also what data structure would be best suited for a DFS and why?

<table>
<thead>
<tr>
<th>Node traversed</th>
<th>Current elements in data structure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

c.) (3 points) Now perform a breadth-first traversal on this graph starting from node $a$. Show your results in the same table format as in part b.). What data structure would be best suited for a BFS and why?
2.) (7 points) Referring to the same graph as in Question 1

a.) (5 points) Run a Dijkstra’s algorithm to determine the shortest path from **a** to **g**. Complete the table below

<table>
<thead>
<tr>
<th>Step</th>
<th>Vertex</th>
<th>Nodes traversed</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>a</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>a,b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

b.) (2 points) Using an example from part a.) explain how ‘relaxation’ is observed in Dijkstra’s algorithm.
3.) (7 points) Consider the following C++ code

```cpp
void myQuicksort(int* a) {

    int N = a.length; // # elements
    int hi = 0; // # high elements
    int lo = 0; // # low elements

    int* H = new int[N]; // a list of elements that are higher
    int* L = new int[N]; // a list of elements that are lower

    for (int i = 1; i < N; i++) { // partition on first element
        if (a[i] > a[0])
            H[hi++] = a[i]; // high piece
        else
            L[lo++] = a[i]; // low piece
    }

    if (lo > 0) {
        int* temp = new int[lo-1]; // resize
        for (int i = 0; i < lo-1; i++)
            temp[i] = L[i]; // copy low piece
        myQuicksort(temp); // sort low piece
        a[lo] = a[0]; // partition element
        for (int i = 0; i < lo-1; i++)
            a[i] = temp[i]; // copy back
    }

    if (hi > 0) { // Similar commenting as above
        int* temp = new int[hi-1];
        for (int i = 0; i < hi-1; i++)
            temp[i] = H[i];
        myQuicksort(temp);
        for (int i = 0; i < hi-1; i++)
            a[lo+1+i] = temp[i];
    }
}
```

a.) (2 points) Analyze the following code for accuracy? Will this code work? Explain your answer

b.) (3 points) Will the time complexity for the above algorithm be the same as that of the original quicksort algorithm? Explain any differences there might be.
c.) (2 points) Comment on the memory usage of the above code. Is it efficient? How can we make it better?

4.) (8 points) You've been working late at night in the CSE labs, trying to finish PA2. It's due tomorrow afternoon, and you've already used up all of your late days. Needless to say, it's going to be a long night. Since you can't spare the time to leave the labs, you decide to round up a few of your friends, and see if they'll help you out. You have four tasks that you want them to help you with:

i.) Go get some food for you (food)
ii.) Retrieve your class notes on the LZW algorithm (notes)
iii.) Return a book to the library that's due today, which you forgot about until now (library)
iv.) Fill in for you at your part-time job at McDonald's (job)

You've also got four friends: Mike, Sarah, Joe, Bill.

You would like to think that your friends would be altruistic, and understand your situation (i.e. Help you out for free). Unfortunately, conniving as they are, your friends agree to help you, only under the condition that you pay them for their assistance.

Due to their different preferences, they each have a different price for a particular task. These prices are listed in the following matrix:

<table>
<thead>
<tr>
<th></th>
<th>food</th>
<th>notes</th>
<th>library</th>
<th>job</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mike</td>
<td>$11</td>
<td>$12</td>
<td>$18</td>
<td>$40</td>
</tr>
<tr>
<td>Sarah</td>
<td>$14</td>
<td>$15</td>
<td>$13</td>
<td>$22</td>
</tr>
<tr>
<td>Joe</td>
<td>$11</td>
<td>$17</td>
<td>$19</td>
<td>$23</td>
</tr>
<tr>
<td>Bill</td>
<td>$17</td>
<td>$14</td>
<td>$20</td>
<td>$28</td>
</tr>
</tbody>
</table>

The values in the matrix correspond to how much you would have to pay a particular person to do a certain task. For example, to get Joe to go to the library, it will cost you $19.

a.) (4 points) Use branch-and-bound to find the optimal assignment of friends to tasks (with one task per person). The optimal assignment is the one that costs you the least. An example assignment is: <Mike->food, Sarah->notes, Joe->library, Bill->job>, having a total cost of $11 + $15 + $19 + $28 = $73. You must show all intermediate steps to receive credit for this problem.

b.) (4 points) Write a pseudo-code function that takes in an NxN array, such as the one you used in part a, and outputs the optimal cost of getting your
friends to help you. You must use branch-and-bound to determine the optimal cost.

5.) (8 points) An undirected graph with an edge between every pair of nodes is sometimes referred to as a complete graph. Consider a complete graph having \( N \) nodes, with labeled nodes 1, 2, ..., \( n \), in which the edge joining nodes \( i \) and \( j \) has the weight \( c_{i,j} = i+j \) for all \( i \neq j \) (eg. The weight of the edge connecting nodes 4 and 7 will have weight \( 4+7=11 \)).

Use Prim's algorithm to determine a minimum weight spanning tree for

a. (2 points) The complete graph with \( N=4 \) nodes.
b. (2 points) The complete graph with \( N=6 \) nodes.
c. (2 points) The complete graph with \( N=8 \) nodes.
d. (2 points) The complete graph with \( N=10 \) nodes.

You must show all work for each part to receive credit.

6.) (7 points) Topological Sorting.

a. (3 points) Give a topological sort (a topological ordering of the nodes) of the graph given in problem 1, earlier in this homework:

b. (2 points) Construct a graph of 10 nodes, having no nodes in common with the graph from part a, which has a unique topological sort. What is the unique topological sort?

c. (2 points) Construct a graph with \( N \) nodes (you choose what \( N \) is) which has EXACTLY 24 unique topological sorts. What are these topological
sorts? The nodes of your graph should be labeled numerically using each of the numbers from 1 to N exactly once (but it doesn't matter which node has which number).