Problem 1. [10 points]
One of the basic motivations behind the Minimum Spanning Tree Problem is the goal of designing a spanning network for a set of nodes with minimum total cost. Here we explore another type of objective: designing a spanning network for which the most expensive edge is as cheap as possible.

Specifically, let $G = (V, E)$ be a connected graph with $n$ vertices, $m$ edges, and positive edge costs that you may assume are all distinct. Let $T = (V, E')$ be a spanning tree of $G$; we define the bottleneck edge of $T$ to be the edge of $T$ with the greatest cost.

A spanning tree $T$ of $G$ is a minimum-bottleneck spanning tree if there is no spanning tree $T'$ of $G$ with a cheaper bottleneck edge.

(a) Is every minimum-bottleneck tree of $G$ a minimum spanning tree of $G$? Prove or give a counterexample.

(b) Is every minimum spanning tree of $G$ a minimum-bottleneck tree of $G$? Prove or give a counterexample.

Problem 2. [10 points]
Consider an algorithm whose running time $T(n)$ on an input of size $n$ satisfies the following recurrence:

$$T(n) = aT\left(\frac{n}{b}\right) + cn$$

where we assume the recurrence holds when $n > 1$, and that $T(1) = c$.

(a) How many nodes are there at level $i$ of the recursion tree?

(b) What is the input size for a problem at level $i$ of the recursion tree?

(c) How much work is done in a single function call at level $i$ of the recursion tree? (Just as in class, count only the work done in the function itself, excluding recursive calls.)

(d) What is the total work done at level $i$ of the recursion tree?

(e) How many levels are in the recursion tree?

Problem 3. [10 points]
Let $G = (V, E)$ be an undirected graph with $n$ nodes. A subset of the nodes is called an independent set if no two of them are joined by an edge. Finding maximal independent sets is difficult in general; but here we'll see that it can be done efficiently if the graph is "simple" enough.

Call a graph $G = (V, E)$ a path if its nodes can be written as $v_1, v_2, \ldots, v_n$, with an edge between $v_i$ and $v_j$ if and only if the numbers $i$ and $j$ differ by exactly 1. With each node $v_i$, we associate a positive integer weight $w_i$.

Consider, for example, the five-node path drawn in Figure 6.28. The weights are the numbers drawn inside the nodes. The goal in this question is to solve the following problem: Find an independent set in a path $G$ whose total weight is as large as possible.

(a) Give an example to show that the following algorithm does not always find an independent set of maximum total weight.

(b) Give an example to show that the following algorithm also does not always find an independent set of maximum total weight.

(c) Give an algorithm that takes an $n$-node path $G$ with weights and returns an independent set of maximum total weight. Analyze the algorithm. The running time should be polynomial in $n$, independent of the values of the weights.
Problem 4. [5 points]

Given the code for Dijkstra’s algorithm shared with you in class, and in your textbook, find the cost of the shortest paths from node s to the rest of the nodes in the graph shown below. Make sure to trace the algorithm exactly as shown in the algorithm. To help you trace it, I have provided you with this table to complete with every iteration.

<table>
<thead>
<tr>
<th>S</th>
<th>V − S</th>
<th>d′(v), ∀v ∈ V − S</th>
<th>Node to be added to S</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>{s, A, B, C, D, t}</td>
<td>d′(s) = 0, d′(A) = d′(B) = d′(C) = d′(D) = d′(t) = ∞</td>
<td>s</td>
</tr>
<tr>
<td>{s}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Hint:** An initial dummy iteration is already done for you to get you started, with all temporary distances initially set to ∞.

Finally, fill out the distances table below.
Problem 5. [10 points]
Suppose you’re running a lightweight consulting business—just you, two associates, and some rented equipment. Your clients are distributed between the East Coast and the West Coast, and this leads to the following question.

Each month, you can either run your business from an office in New York (NY) or from an office in San Francisco (SF). In month i, you’ll incur an operating cost of Ni if you run the business out of NY; you’ll incur an operating cost of Si if you run the business out of SF. (It depends on the distribution of client demands for that month.)

However, if you run the business out of one city in month i, and then out of the other city in month i + 1, then you incur a fixed moving cost of M to switch base offices.

Given a sequence of n months, a plan is a sequence of n locations—each one equal to either NY or SF—such that the ith location indicates the city in which you will be based in the ith month. The cost of a plan is the sum of the operating costs for each of the n months, plus a moving cost of M for each time you switch cities. The plan can begin in either city.

The problem. Given a value for the moving cost M, and sequences of operating costs N1, . . . , Nn and S1, . . . , Sn, find a plan of minimum cost. (Such a plan will be called optimal.)

(a) Show that the following algorithm does not correctly solve this problem, by giving an instance on which it does not return the correct answer.

```
For i = 1 to n
    If Ni < Si then
        Output "NY in Month i"
    Else
        Output "SF in Month i"
End
```

(b) Design and write the pseudocode of an efficient algorithm that takes values for n, M, and arrays of operating costs [N1, . . . , Nn] and [S1, . . . , Sn], and returns the cost of an optimal plan.

(c) Implement the code you designed in part (b). You are given a JAVA file that contains the starter code for this part with an example. You can access that starter code from the CS231 downloads directory. All instructions on the input and output format for this code can be found within that JAVA file.

Problem 6. [5 points]
Let us say that a graph \( G = (V, E) \) is a near-tree if it is connected and has at most \( n + 8 \) edges, where \( n = |V| \). Give an algorithm with running time \( O(n) \) that takes a near-tree \( G \) with costs on its edges, and returns a minimum spanning tree of \( G \). You may assume that all the edge costs are distinct. You have to explain why the running time of your algorithm is \( O(n) \).