1. Shortest paths are not always unique: sometimes there are two or more different paths between two vertices, with the minimum possible length. Now consider the following problem.

**Input:** A directed, edge-weighted graph $G = (V, E)$, a source vertex $s \in V$

**Output:** A boolean array $usp[\cdot]$ such that for each vertex $u \in V$, the entry $usp[u]$ is True if and only if there is a *unique* shortest path from $s$ to $u$.

**Note:** $usp[s] = True$.

(a) For the following directed, edge-weighted graph and given source vertex $s$, write down the values in the array $usp[\cdot]$.

```
  a  3  d
  |   |
  5   1
  +---+
  |   |
  b   1
  |   |
  3   4
  +---+
  s   c
```

(b) Given a directed graph with $n$ vertices and $m$ edges, describe an algorithm that solves the above problem in $O((m + n) \log n)$ time.

**Hint:** Your starting point should be Dijkstra’s shortest path algorithm.

**Note:** Use the top of the next page for your answer.
Consider a data structure that maintains a binary counter of unspecified length and supports two operations: (i) increment, which increments the counter's value by 1 and (ii) reset, that resets the counter's value to 0.

A simple way to implement a binary counter is to allocate a very large array, say of length $M$, of bits when the data structure is initialized and set all these bits to 0. Then, the increment operation can be implemented as follows:

```plaintext
function increment(B)
    i ← 0
    while (B[i] = 1) do
        B[i] ← 0
        i ← i + 1
    B[i] ← 1
```

Notice that this implementation of increment does not check for an overflow; it just assumes that $M$ is going to be large enough that checking for overflow is unnecessary. In your thinking about this problem, do not worry about the possibility of an overflow.

**Example:** Suppose that after initializing the binary counter data structure, we perform five increment operations. Thus the current value of the counter is 5, which is represented by $B[0] = 1, B[1] = 0, B[2] = 1, B[i] = 0$ for all $i > 2$. Then calling increment once more changes $B[0]$ to 0 and $B[1]$ to 1, leaving all other bits unchanged.

(a) Suppose that the binary counter is initialized as described above. Now consider a sequence of $n$ operations, some of which are increment operations and some of which are reset operations. What is the worst case running time of any one of these operations, as a function of $n$?
(b) Argue that the amortized running time of these operations is $O(1)$.

Hint: Every increment is a sequence of assignments that turn a bunch of bits from 1 to 0 followed by one assignment that turns a bit from 0 to 1. Now notice that every assignment that turns a bit from 1 to 0 can be “charged” to a previous increment operation that turned that bit from 0 to 1.

3. The following statements may or may not be True. In each case, determine if the statement is True or False. If you claim that the statement is True, provide a proof. Otherwise, provide a counterexample.

(a) Let $G = (V, E)$ be an undirected, edge-weighted graph and let $T$ be a minimum spanning tree (MST) of $G$. Now let us increase the weight of every edge in $G$ by 1. $T$ is still an MST of the graph with increased edge-weights.
(b) Let $G = (V, E)$ be a directed, edge-weighted graph and let $P$ be a shortest path in $G$ from a vertex $s \in V$ to a vertex $t \in V$. Now let us increase the weight of every edge in $G$ by 1. $P$ is still a shortest path from $s$ to $t$ in the graph with increased edge-weights.

(c) Suppose a graph $G$ with $n$ vertices has more than $n - 1$ edges and there is a unique heaviest edge. Then this edge cannot be part of any minimum spanning tree of $G$.

(d) Suppose that $G$ is an undirected, edge-weighted graph in which all edge-weights are distinct and positive. Consider a vertex $s \in V$. It is possible for the tree of shortest paths from $s$ (to all vertices in $G$) to not share even a single edge with the minimum spanning tree of $G$. 
4. Consider the following recursive function that takes as arguments an array $L$ and two non-negative integers $\text{first}$ and $\text{last}$, that serve as indices into $L$. Therefore, if $L$ has length $n$, then $\text{first}$ and $\text{last}$ are guaranteed to be in the range 0 through $n - 1$.

```python
function strangeSum(L, first, last)
    if (last < first) then
        return 0
    if (last = first) then
        return L[first]
    if (last = first + 1) then
        return L[first] + L[first+1]
    else
        m ← last - first + 1
        leftSum ← strangeSum(L, first, first + m/2 - 1)
        midSum ← strangeSum(L, first + m/4, first + 3 * m/4 - 1)
        rightSum ← strangeSum(L, first + m/2, last)
        return leftSum + midSum + rightSum
```

(a) What is the value returned by the function call `strangeSum(L, 0, 3)` where $L$ is the array $[1, 4, 2, 3]$.

(b) Write a recurrence relation describing the running time the function call `strangeSum(L, 0, n-1)` on an array $L$ of length $n$.

(c) Solve the recurrence in (b) to obtain the running time of the function call `strangeSum(L, 0, n-1)`, in terms of $n$, the length of the given array $L$. 
5. Here are some problems on NP-completeness and intractability.

(a) The decision version of the Interval Scheduling problem is the following.

**Interval Scheduling Decision (ISD)**

**Input:** A set $I$ of intervals, a positive integer $k$.

**Output:** Is there is subset $I' \subseteq I$ of pairwise non-overlapping intervals of size at least $k$?

The decision version of the Maximum Independent Set problem is the following.

**Maximum Independent Set Decision (MISD)**

**Input:** A set $G = (V, E)$, a positive integer $k$.

**Output:** Is there is an independent set $V' \subseteq V$ of size at least $k$?

Your task is to prove that $\text{ISD} \leq_p \text{MISD}$.

(b) OK, proving that $\text{ISD} \leq_p \text{MISD}$ may not have been that difficult. But, what about showing that $\text{MISD} \leq_p \text{ISD}$? I want you to either prove that $\text{MISD} \leq_p \text{ISD}$ or argue that it is unlikely for there to be a polynomial-time reduction from MISD to ISD.
(c) For a graph $G = (V, E)$, a clique is a set $C \subseteq V$ of vertices such that every pair of vertices in $C$ is connected by an edge. Now consider the following decision problem:

**Maximum Clique Decision (MCD)**

**Input:** A set $G = (V, E)$, a positive integer $k$.

**Output:** Is there a clique $C \subseteq V$ of size at least $k$?

Using the fact that Misd is NP-complete, I want you to show that MCD is NP-complete. Recall that there are two main steps in showing this: (i) Show that MCD $\in$ NP and (ii) Misd $\leq_P$ MCD.