Travis Robinson
CS325
Homework 4

1)
a. False; it may not be efficient, but a possible algorithm would be to check every potential route
b. True; because P != NP, there are no polynomial algorithms that can solve it, so there are no efficient algorithms
c. Because we assume that P != NP, there are no polynomial time algorithms, so there is no efficient way of solving
d. True, since P!=NP, and TSP is NP, then TSP is not in P
e. False, P!= NP, and since TSP is in NP, and not P, there are no poly time algorithms for it
f. False, there are no poly-time algorithms, but that doesn't mean that they all run in exponential time; there's also n! time, etc

2)
a. False; we could infer that Y is NP hard, but not NP complete
b. False; if X is no harder than Y, then Y must be at least as hard as X, so if X is NPC, then Y is NP hard
c. False
d. True; if X is NP complete, Y is NP hard. If Y is also NP hard, the it's at the overlap, so by definition is NP complete
e. False; if X is NP complete, Y is NP complete as long as it is in NP
f. False; X is no harder than Y, but that doesn't mean Y isn't harder than X. Y could be NP
g. True; X is no harder than Y, since Y is in P, X must be at least as easy, so it's in P

3)
We know that HAM-Cycle can be used to solve for Ham-Path, because if there's a hamiltonian cycle, there must also be a path. This means that Ham-Path reduces to Ham-Cycle, which makes Ham-Path NP-hard because Ham-Cycle is NP-complete. We also know that Ham-Path is in NP because it can be verified in polynomial time by following the certificate. Since Ham-path is both NP-hard and in NP, we know that it is NP-complete

4)
Long-path, since it finds a simple path (a path that uses each vertex once) would be a reduction of Ham-path. We know from question 3 that ham-path is NP-complete. This makes long-path NP-hard. It would be easy to verify the result returned by long path by simply following the certificate and verifying that each vertex is used once; this makes it poly-time, so it's an NP problem, making long-path NP-complete

5)
a. Graph-color-recur(country,color)
```
{
	if(this.color != color)
	{
		return false
	}
	else if (color == blue)
	{
		this.color == red
	}
	else
	{
		this.color == blue
	}
	for (i=0;i<#_of_neighbors)
	{
		if(graph-color-recur(neighbor[i],this.color)==false)
		{
			return false
		}
	}
	return true
}
```

b.
K-COLOR is no harder than the algorithm from part a; they both go through the graph painting the vertices; the difference is one of these is to return a true or false value if there is a valid coloring, and the other is to actually color them. This means that K-COLOR is no harder than Graph-color-recur, and Graph-color-recur is no harder than K-COLOR. Therefore one both not harder and not easier than the other, so they must be equal in difficutlty. This means that if one is in P, they're both in P.
K-COLOR(country,color)
```
{
	if(this.color != color)
	{
		return false
	}
	else if (color == blue)
	{
		this.color == red
	}
	else
	{
		this.color == blue
	}
	for (i=0;i<#_of_neighbors)
	{
		if(graph-color-recur(neighbor[i],this.color)==false)
		{
```

```
                    return false
                }
            }
        return true
}
```

c.
Because K-COLOR is in P, it by definition is also in NP; this is because it can be solved in P time, so to verify that a certificate is correct, it simply needs to be solved again. Therefore is in NP

d. Because 3-COLOR is NP-complete, 4-COLOR is NP hard because 4-COLOR is a derivative of 3-COLOR. To then show that 4-COLOR is also NP complete, we need to show that it is also in NP, or that it can be solved in poly-time. This can be done by going through the certificate and confirming that no two neighboring vertices are the same color, which is a poly-time algorithm (each vertex needs to only be gone through once to confirm this, making it linear which is poly-time with exponent of 1). Therefore 4-COLOR is in NP, and because it's also NP hard, it's NP-Complete