Rick Sullivan
Professor Tran
Algorithms
5 June 2013

Homework #7

1. Idea: We can negate the weights of each edge and then use the MST algorithm. This will cause the algorithm to choose the heavier weights first. We can then simply negate the weights of the returned MST to find the maximum ST.

```
void Graph::addEdge(vertex from, vertex to, int weight);
int  Graph::edgeWeight(vertex from, vertex to);

Graph negate(Graph G) {
    N = new Graph(G.n());    //Create an empty graph of size G.n()
    for(i = 0; i < G.n(); i++) {
        for each u in G.i { //Move through the adjacency list
            //Add the negative edge to the new graph
            N.addEdge(i, u, -G.edgeWeight(i, u));
        }
    }
    return N;
}

Graph MaxST(Graph G) {
    return negate( MST( negate(G) ) );
}
```

2. Idea: Move through the given list, checking that no vertices are repeated and the path is sufficiently long.

```
// Returns true iff the given path in G is a simple path with
// at least l edges.
bool verify_longestPath(Graph G, vector<vertex> path, unsigned int l) {
    //Path must be at least the length of the input l
    if (path.length() < l)
        return false;

    //Mark all vertexes as unvisited.
    for (i = 0; i < G.n(); i++) {
        G[i].visited = false;

    for (i = 0; i < l; i++) {
        //The next vertex must be a neighbor of the current one.
        //Also, each vertex cannot have been previously visited.
        if (!G.isEdge(path[i], path[i+1]) || G[path[i]].visited)
            return false;

        //Set the status of the current vertex
        G[i].visited = true;
    }

    return true;
}
```