Hardik Sangwan
CX 4140
Assignment 2

Problem 1:

a)

(a) $\Theta(g)$
(b) $\Theta(g)$
(c) $\Omega(g)$
(d) $O(g)$
(e) $\Omega(g)$
(f) $\Theta(g)$

b)

Outer loop iterates from 0 through n. Time complexity O(n).
Inner loop iterates $\log_3 n$ times. Time complexity O(logn).
Total Time Complexity O(n*logn).
The algorithm computes and prints a number and the (log base 3 of that number + 1) floored.

Problem 2:

Greedy Solution:

Leaks as points l_1 to l_n. Strip of size s.
Place a strip on l_i starting with i = 1 therefore covering all leaks l_i to lj between {l_i, l_i + s}, where $l_j <= l_i+s$
Update i to j + 1
Repeat until i > n

Runtime: O(n)

Correctness:
Greedy Solution G = {G1, G2..Gn} where Gi is point where strip s starts and Gi < Gj for i < j.
Optimal Solution O = {O1, O2..Ox}, with same notation.
By definition of greedy property O1<= G1 since G1 = l1.
Let O' be the solution obtained by replacing O1 with G1 in O. Since l1 is the first leak and is covered by G1, G1 covers all the leaks that O1 covers. Now moving on to the next leak/strip O2, a similar case can be made where G2 covers all leaks covered by O2 if we assume that our problem is to now cover all the leaks to the right of our leftmost leak l1. By repeating this process, we can covert O into G. Therefore G is optimal.

Problem 3:

Bag with weight limit L.
n minerals with value of v_i and weight w_i for i^{th} mineral.
Maximize bag value.

Greedy Solution:
Calculate the value-per-pound $p_i = v_i/w_i$ for i = 1, 2, . . . , n minerals.
Sort the items by decreasing p_i
Let k be the current weight limit (Initially, l = L).
In each iteration, we choose item i from the head of the unselected sorted list.
If $l \geq w_i$,

$x_i = 1$ (whole Item i taken)

$l = l - w_i$ (current weight limit updated)

If l < wi,

$x_i = k/w_i$ (fraction of Item i taken)

terminate/break out of algorithm.

Run time is O(n * logn) (-> Run time of sorting algorithm)

Solution Correctness:
Let the greedy solution be $G = \langle x_1, x_2, ..., x_k \rangle$
x_i indicates fraction of item i taken (all $x_i = 1$, except possibly for i = k).
Consider any optimal solution $O = \langle y_1, y_2, ..., y_n \rangle$
y_i indicates fraction of item i taken in O (for all i , $0 \leq y_i \leq 1$).
Knapsack is full in both G and O: For all i Sum x_i*w_i = K = Sum y_i*w_i for all i.
Consider the first item i where the two selections differ.
By definition, solution G takes a greater amount of item i than solution O (because the greedy solution always takes as much as it can). Let x = xi – yi .

Consider the following new solution O' constructed from O:
For j < i, keep $y_j' = y_j$.
Set $y_i' = x_i$.
In O, remove items of total weight x*w_i from items i + 1 to n, resetting the y_j' appropriately. The total value of solution O' is greater than or equal to the total value of solution O. Since O is largest possible solution and value of O' cannot be smaller than that of O, O and O' must be equal. Thus solution O' is also optimal. By repeating this process, we will eventually convert O into G, without changing the total value of the selection. Therefore G is also optimal

Problem 4:

Kruskal's Algorithm implemented.
Union Find Data structure used.
Time complexity for Union Find is O(log n) (n= number of elements) and for m nodes (m operations) it's O(mlogn).

m->edges; n->nodes;

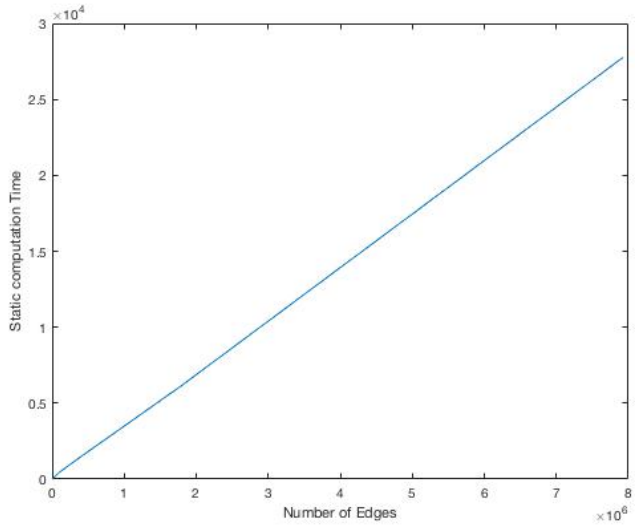
Time complexity for computeMST(): O(m*log n)

Sort Edge List: O(m*log n) from
O(m* log m) where O(log m) belongs to O(log n) since $m \leq n^2$

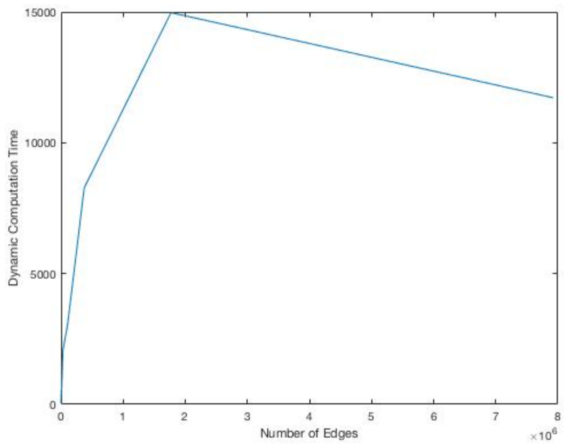
Union Find: O(n+m) from
O(n + m*log n) where log n is ~constant.

Time Complexity for recomputeMST(): O(m * log n)
Shortest Path: O(m*log n) (networkx implementation uses Dijkstra's)

Plots:



Static Computation
Log n close to constant therefore O(m*log n) = O(m) = linear graph



Dynamic computation

Slope of graph changes due to change in the ratio of initial edges to number of edges being added. Initially with a larger ratio the time complexity due to the union find is dominant (O(n+m*logn)) but as the number of edges increases the shortest path computation (O(m * log n) requires a longer time.

Reference:

Union Find Implementation: http://www.ics.uci.edu/~eppstein/PADS/UnionFind.py