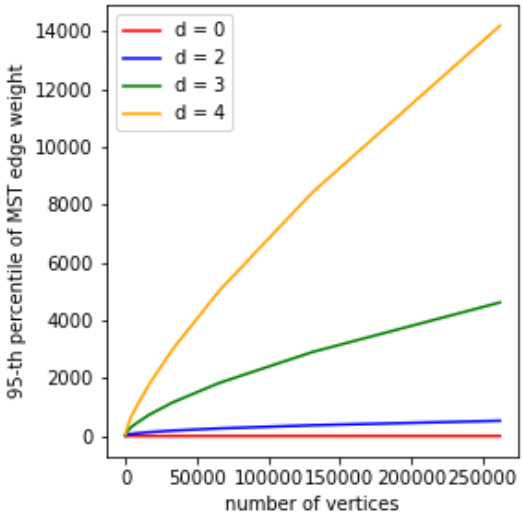
# Programming set 1

Names: Yuting Liu
Collaborators: N/A
No. of late days: 0
No. of late days already used: 0

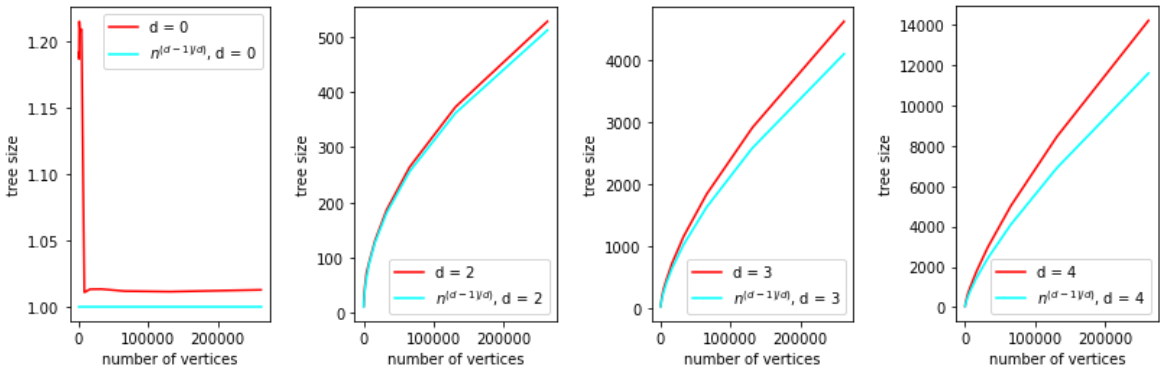
## 1. MST quantitative result

The table and graph listing the average tree size for several n values are shown below (under each condition 5 trials):

| n | 0 | 2 | 3 | 4 |
|---|---|---|---|---|
| **128** | 1.191887 | 12.382567 | 31.077404 | 49.303428 |
| **256** | 1.192258 | 18.182777 | 49.014863 | 84.581214 |
| **512** | 1.187116 | 25.211807 | 77.970324 | 139.927868 |
| **1024** | 1.215457 | 36.339929 | 122.929543 | 235.118182 |
| **2048** | 1.208593 | 50.952632 | 197.978402 | 399.972942 |
| **4096** | 1.209612 | 72.652047 | 312.155410 | 673.878702 |
| **8192** | 1.010766 | 93.529337 | 458.532787 | 1055.735091 |
| **16384** | 1.013140 | 131.726636 | 727.563707 | 1771.018437 |
| **32768** | 1.013256 | 186.581085 | 1155.429740 | 2980.151945 |
| **65536** | 1.011742 | 263.994819 | 1834.150939 | 5018.292217 |
| **131072** | 1.011301 | 373.090821 | 2908.738010 | 8436.555777 |
| **262144** | 1.012660 | 527.891869 | 4620.787844 | 14195.169182 |




My guess of $f(n) = cn^{(d-1)/d}$ where c is a dimension-related constant, n is the number of vertices, and $d \in \{1(i.e. dimension = 0 \ in \ this \ problem \ set), 2, 3, 4\}$. This is an asymptotic estimation: when n is smaller, the difference between the algorithm solution and this function is large and unstable, especially d = 0 in this problem set.



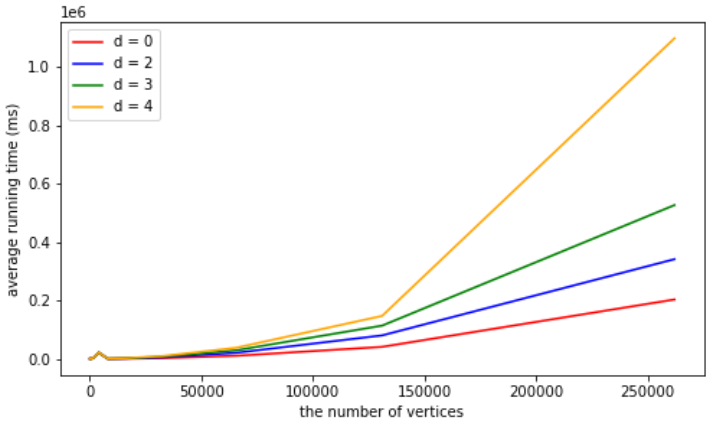

## 2. MST algorithm

I used Kruskal algorithm for MST solution. This is because after pruning the dense graph with a threshold, the number of edges decreases dramatically. As I tested, when n is very

large, $O(E\ log\ V)$ is much smaller than $O(V^2)$ and $O(V\ log\ V + E)$ (even if using Fibonacci heaps).

3. **Convergence of f(n)**
   For each dimension, the slope of f(n) becomes slightly smoother with the increase of n, so f(n) is asymptotically convergent. This is reasonable: when n tends to positive infinity, the graph is so dense that the weight sum increase from V to V ∪{v} tends to zero because the weight of the edge e crossing between the S and V \ S tends to zero in this dense graph. Therefore, the tree size converges, with the slope becoming smaller and smaller.

4. **Running time**
   Because I use Python, the running time was terribly long. I recorded the running time for each n and dimension and the plot of average running time is shown below. The running time increases dramatically. It makes sense just as the time complexity of the Kruskal algorithm.




   Random number generation is time-consuming. Use matrix instead of element-wise generation accelerate calculation significantly.

5. **Random number generator**
   I used numpy.random.rand() as the random number generator. It is a pseudorandom number generator, which means it is not truly random and relies on the seed that I set. To make the generator 'more random', I change the seed manually when repeating trials with same n and same dimension, but in each trial, the seed is identical using different n and dimensions.

6. **Dense graph**
   When n > = 16384, the fully connected graph is so dense that the memory was overloaded. Several procedures were taken to reduce the memory use.
   a. Take the adjacency list instead of adjacency matrix for graph representation. There exists only one of the paired undirected edges: (u, v, weight) due to u<v.
   b. Take a threshold k(n) to trim the graph when n > 5000. When n <= 5000, the graph is not trimmed and running smaller n is feasible. This is because the asymptotic property is practically useful for very large samples. If the weight of one edge is larger than this threshold, this edge will be not added into the final graph.

Statistically I checked the empirical distribution of weights in MST: it is highly skewed to zero, with few outliners having relatively large values. Therefore, if the weight of one edge is larger than a specific outliner threshold, this edge will be extremely unlikely an edge of MST.

How to measure the extremeness? I took a very naive estimation: if 95% of MST edges are smaller than this value, then larger weights are relatively impossible to be selected. The threshold should be larger but very close to 95-th quantile value.

Intuitively, I choose a power function for this threshold with a constant: $k(n) = c * n^{-1/d}$, where d means dimension and c depends on dimensions, which is different with c in f(n). This function has a similar curve with the 95-th quantile value with n. For each dimension, c is estimated using for loop: the selected one is the smallest one larger than 95-th quantile of all n values.