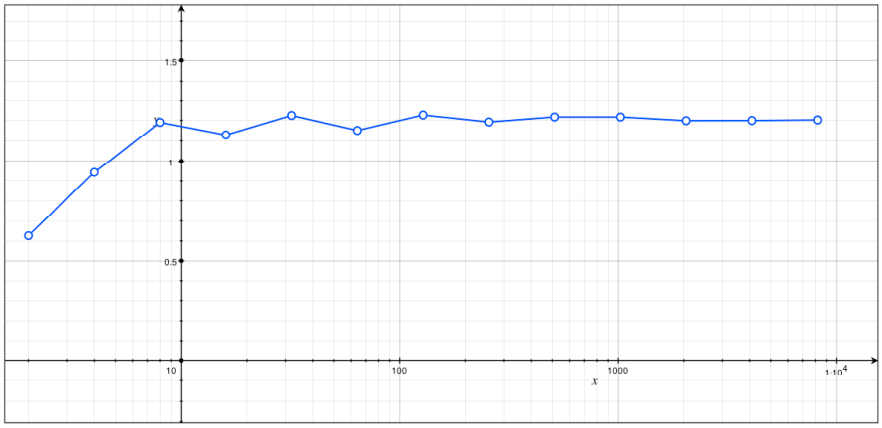
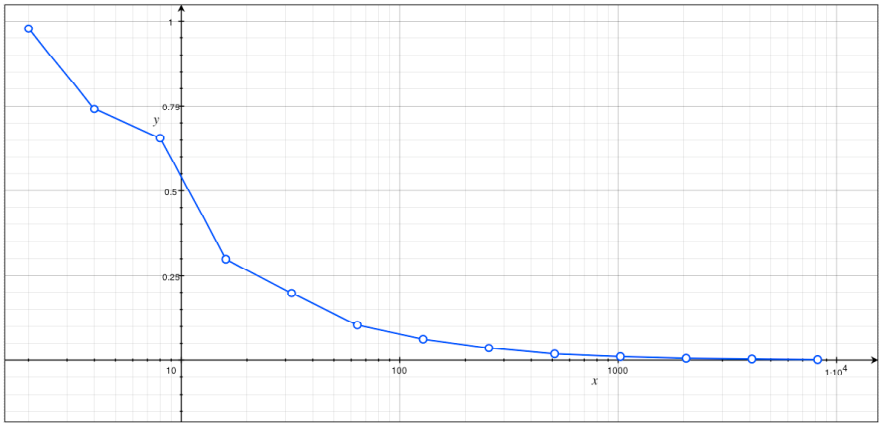
Programming Assignment 1, REPORT, Russell Lowke March 13th 2004.

1) TABLE or graph listing average tree size for several values of n

```
When weights are randomized:
Av. tree weight for n = 2     av. over 10 tries: 0.62613  largest edge = 0.977158
Av. tree weight for n = 4     av. over 10 tries: 0.943949 largest edge = 0.740818
Av. tree weight for n = 8     av. over 10 tries: 1.18944  largest edge = 0.655816
Av. tree weight for n = 16    av. over 10 tries: 1.12869  largest edge = 0.298851
Av. tree weight for n = 32    av. over 10 tries: 1.22276  largest edge = 0.197784
Av. tree weight for n = 64    av. over 10 tries: 1.15055  largest edge = 0.104108
Av. tree weight for n = 128   av. over 10 tries: 1.22498  largest edge = 0.0620191
Av. tree weight for n = 256   av. over 10 tries: 1.19155  largest edge = 0.0353309
Av. tree weight for n = 512   av. over 10 tries: 1.21556  largest edge = 0.0178354
Av. tree weight for n = 1024  av. over 10 tries: 1.21546  largest edge = 0.0100831
Av. tree weight for n = 2048  av. over 10 tries: 1.19786  largest edge = 0.00508966
Av. tree weight for n = 4096  av. over 10 tries: 1.19768  largest edge = 0.00303103
Av. tree weight for n = 8192  av. over 10 tries: 1.20157  largest edge = 0.00123695
    tree weight for n = 30000     over  1   try: 1.20443  largest edge = 0.000348025
    tree weight for n = 35000       over  1   try: 1.19882  largest edge = 0.000274436
```

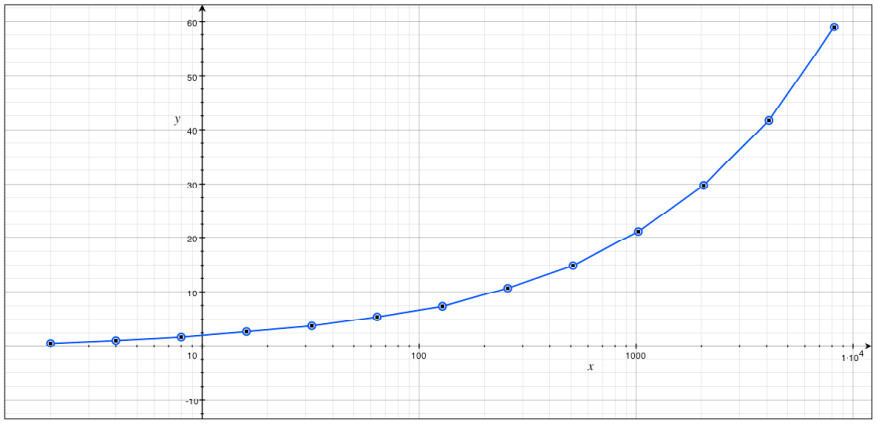


n v average tree weight for randomized weight *

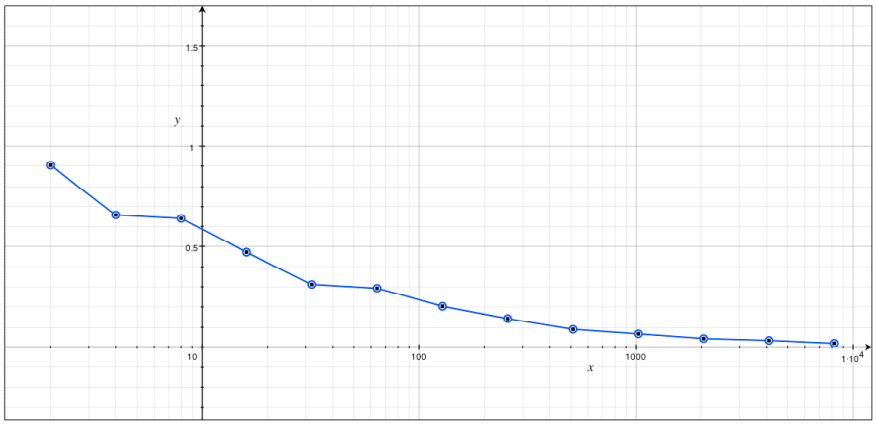


n v largest edge for randomized weight *

```
When weights are distances:
Av. tree weight for n = 2      av. over 10 tries: 0.540596    largest edge = 0.904742
Av. tree weight for n = 4      av. over 10 tries: 1.07053     largest edge = 0.65782
Av. tree weight for n = 8      av. over 10 tries: 1.7213      largest edge = 0.650102
Av. tree weight for n = 16     av. over 10 tries: 2.73812     largest edge = 0.471555
Av. tree weight for n = 32     av. over 10 tries: 3.7886      largest edge = 0.311492
Av. tree weight for n = 64     av. over 10 tries: 5.45684     largest edge = 0.292485
Av. tree weight for n = 128    av. over 10 tries: 7.58164     largest edge = 0.203531
Av. tree weight for n = 256    av. over 10 tries: 10.686      largest edge = 0.143603
Av. tree weight for n = 512    av. over 10 tries: 14.8958     largest edge = 0.0901973
Av. tree weight for n = 1024   av. over 10 tries: 21.1243     largest edge = 0.0673661
Av. tree weight for n = 2048   av. over 10 tries: 29.7679     largest edge = 0.0436024
Av. tree weight for n = 4096   av. over 10 tries: 41.6676     largest edge = 0.03428
Av. tree weight for n = 8192   av. over 10 tries: 58.9475     largest edge = 0.0198867
    tree weight for n = 30000       over  1   try: 112.443    largest edge = 0.0112098
    tree weight for n = 35000       over  1   try: 121.665    largest edge = 0.0113445
```



n v average tree weight for weight by distance *



n v largest edge for weight by distance *

*NOTE: Plotted n axis is logarithmic

2) A description of guess for the function f(n)

I was unable to curve fit the graph, but by plotting the n vs. the weight I could see that it might be exponential, and have the form A * e ^ (b n). But this curve wasn't the actual value of n... it was log n. So the actual form of the function is f(n) = A * e ^ (b (e ^ n)).

3) Briefly discuss

I chose to use Kruskal's algorithm, manly due to the methodology described in class for dealing with disjoint sets using path compression which optimizes Kruskal's and results in a rapid finding, linking and union of sets. This methodology of finding the set of a Vertex is log*(n) which is nice to have in the algorithm.

Aside from having to sort Edges, Kruskal's is a fast algorithm at O(*m* log**n*), which is close to linear. To deal with the sorting of edges [the largest bottleneck of Kruskal's algorithm] I have adapted a version of Quicksort as outlined in "C Programming - A Modern Approach" by K.N.King pages 173-75.

The growth rates (f(n)) are very surprising. When the edge weight represented the distance between points, it was surprising to see how quickly the average tree weight increased. I plotted log n vs. the average weight and the graph was still an exponential, which was further surprising. As the edge weight represented random values in the [0, 1] range, it seemed to quickly taper off and hang at around 1.2. I expected the value to increase indefinitely (even if slowly).

I seeded the random number generator using the current time, but not feel the generator is all that random, particularly as
there is an obvious anomaly at n = 8 in the graph of "n v largest edge for randomized weight."

Much time was spent optimizing the Quicksort routine to allow for greater values of n to be computed. Oddly, the most dramatic speed enhancement occurred after specifying

```
bool operator< (Edge& other) {return edgeWeight < other.edgeWeight;}
```

in edge.hpp on line 19. This allowed the compiler to optimize the Edge comparisons and gave a factor of 12 increase in speed. Before this improvement the largest value of n I could feasibly compute 20 times in 40 minutes was 2048. The majority of the time in my Kruskal's algorithm was tested and shown to be the sorting portion.

After the time constraint was taken care of it became evident that there was a memory constraint at n > 8192. To relieve this problem I started omitting Edges. Looking at the data the largest edge size at n = 8192 was 0.0198867, so I felt it was safe to remove any edge larger .11, which is nearly a factor of 10 higher than this. With this implemented I could compute values of n as high as 35000.

Russell Lowke