[redacted]
CS165
June 9, 2020

Project 3 Phase 2: Erdos-Renyi and Barabasi-Albert Analysis with Extra Credit

My Testing Environment and Input

I utilized several libraries that I used from Project 1 and Project 2. Most notably, I once again used the *Chrono* and the *Random* libraries. For the *Chrono* library, I used its *High-Resolution Clock* to seed my number generation. For number generation, I used the *Random* library's *Mersenne Twister* algorithm from its *mt19937* class. I used random number generation as part of a heuristic and also in drawing random values for each type of graph generation.

For calculating the diameter and clustering coefficient, I ran each algorithm with a max input size of 2^17, or 131072 different vertices. I also averaged the results of these input over five different generated graphs at each input size.

As per criteria for the degree distribution requirements, I only used one sample for each of *N=1000, N=10000*, and *N=100000*. Therefore, these are not averaged.

Pseudocode/Implementation Choices

a. Data Structures

My graph generation data structures use a combination of hash tables and vectors. In the *Node Class*, I use a *vector<int>* to store the neighbors of a given node. These neighbors are easily added. Whenever I make an edge, I simply add each end of the corresponding edge as a neighbor to the other. I also use a hash table adjacency list via an *unordered_set<int>**. When an edge is made, the table is indexed by the starting node of the edge as the key, and the value of the ending node of the edge is inserted into that position extending the list. This adjacency list suites most of my project's needs. For example, when finding a diameter through *Breadth-First Search*, each key(node) of the adjacency list is iterated for visits.

To sum up, while there are other structures used such as a *map<int,int>* for the degree distribution histogram, and a *list<int>* for the *L* list, my project mostly evolves around utilizing these two main data structures for heavy lifting.

b. Graph Calculation Algorithms

Let us look at the three main algorithms that this project requires starting with *get_clustering_coefficient*. To get the clustering coefficient, I utilize graph degeneracy in order to count the triangles in a reasonable time. For my graph degeneracy algorithm, I followed the provided pseudocode nearly one-to-one. First, I make a *D* list, *N* list, and define a *K* value. I then begin the first outer-loop iteration with a lower bound of 0, iterating through an upper bound that is the total nodes in the graph. I then get the smallest non-empty *i* at *D[i]*, set *K* to *max(K, i)*, get a vertex from *D[i]*, add that to the *L* list, remove it from *D[i]*, and then test all neighbors of that vertice in an inner loop. I continue to follow the algorithm testing each neighbor of the retrieved vertices to ultimately create my *N* list which I use for counting triangles. I use the size of the *L* list as an upper bound for an outer loop in the triangle count, and I go through the entire *L* list testing each neighbor of *L* cross referenced to the created *N* list. Once the entire *L* list has been iterated, I have tested all possible vertices that could contain any triangles and return that total count found.

The second of the main graph algorithms I will cover is *get_diameter*. As we know, this algorithm is used for calculating the diameter of the graph. Here, I approach this calculation by using the "Heuristic Idea 2" from the class notes. In the *get_diameter* function, I first get a random vertex *r* and set *d_max*=0. Then, while a loop holds true, I get a value *w* that is equal to the furthest node found from a breadth-first search on *r*. I then calculate the shortest connected distance between *r* and *w* with a *get_shortest_distance* function I created that finds the shortest distance between two nodes in a graph. If the returned distance is greater than *d_max,* I set *d_max* equal to that distance, set *r=w*, and then continue this loop until the returned distance is less than *d_max.* This is when we have found the diameter of the graph and that value is returned to the caller. Using this algorithm meets the needs of calculating the diameter for large sets of *N* in a reasonable time.

The last of the main algorithms to be covered is the *get_degree_distribution* algorithm. This algorithm is much simpler in complexity than calculating the diameter and clustering coefficient. First, I call a function *compute_degrees* that uses a *vector<int>* as a data structure for holding the total degrees of each node. I follow the provided pseudocode at this point, making a histogram of type *map<int,int>* of size *n*, and initialize each *H[i]=0*. Then, for each *H[deg[v]],* I increment that index. The created *map<int,int>* histogram is then returned.

c. Graph Generation Models

1. Erdos-Renyi

For Erdos-Renyi graph generation, I connect edges based upon the probability value *p=(2*ln(number of nodes))/number of nodes*. First, all vertices are created with *N* total vertices. Then, the *p* value is calculated. I then use the total vertices in the graph as an upper bound for a loop for adding edges starting from the first vertice *v=starting vertice[0]*. In the loop, I draw a uniformly random value between [0,1), get a neighbor of *v* by w=*w+1+(floor(ln(1-random)/log(1-probability)))*, and then add edges accordingly to these calculated values requiring an *O(n+m)* running time. When the outer loop is finished, all edges are added to the graph.

2. Barabasi-Albert

For Barabasi-Albert graph generation, I connect edges by preferential attachment to hopefully simulate real world connections related to areas such as social networks more accurately. Similar to Erdos-Renyi, I first generate all vertices with *N* total vertices. However, I also use the variation of Barabasi-Albert where each new vertex initially starts with *D=5* minimum degrees. I create an array *M[2*num_vertices*D]*, and then begin an outer loop with an upper bound of *num_vertices-1*. In this loop I create an inner loop with upper bound *D-1* (our minimum *D value)*, and add edges by first drawing random values uniformly in the range *{0,...,(2*v*d+i)}*. Edges are then added in a second inner loop with an upper bound *nd-1*.

Plotting Degree Distributions and Determining if They Follow a Power Law.

1. Erdos-Renyi

The degree distribution of graphs built using the Erdos-Renyi edge generation algorithm do not exhibit a power law. As we can see, for each number of nodes *N* where *N=1000*, *N=10000*, and *N=100000*, we have a bell-shaped distribution. As each size of *N* grows, Erdos-Renyi continues towards a smooth and predictable number of expected degree frequency.

In fact, the graphs exemplify the highest expected number of degrees N*P. When *N=1000*, we have a probability of *0.0138155* with an expected highest degree frequency to be *1000*0.0138155 ~13.9*. For *N=10000*, we have a probability of *0.00184207* with an expected highest degree frequency to be *10000*0.00184207 ~18.4*. For *N=100000*, we have a probability of *0.000230259* with an expected highest degree frequency *100000*0.000230259~23*. All sizes of *N* show the degree frequency peak in the correct and expected ranges. ***See Figures 1.1-1.6.***

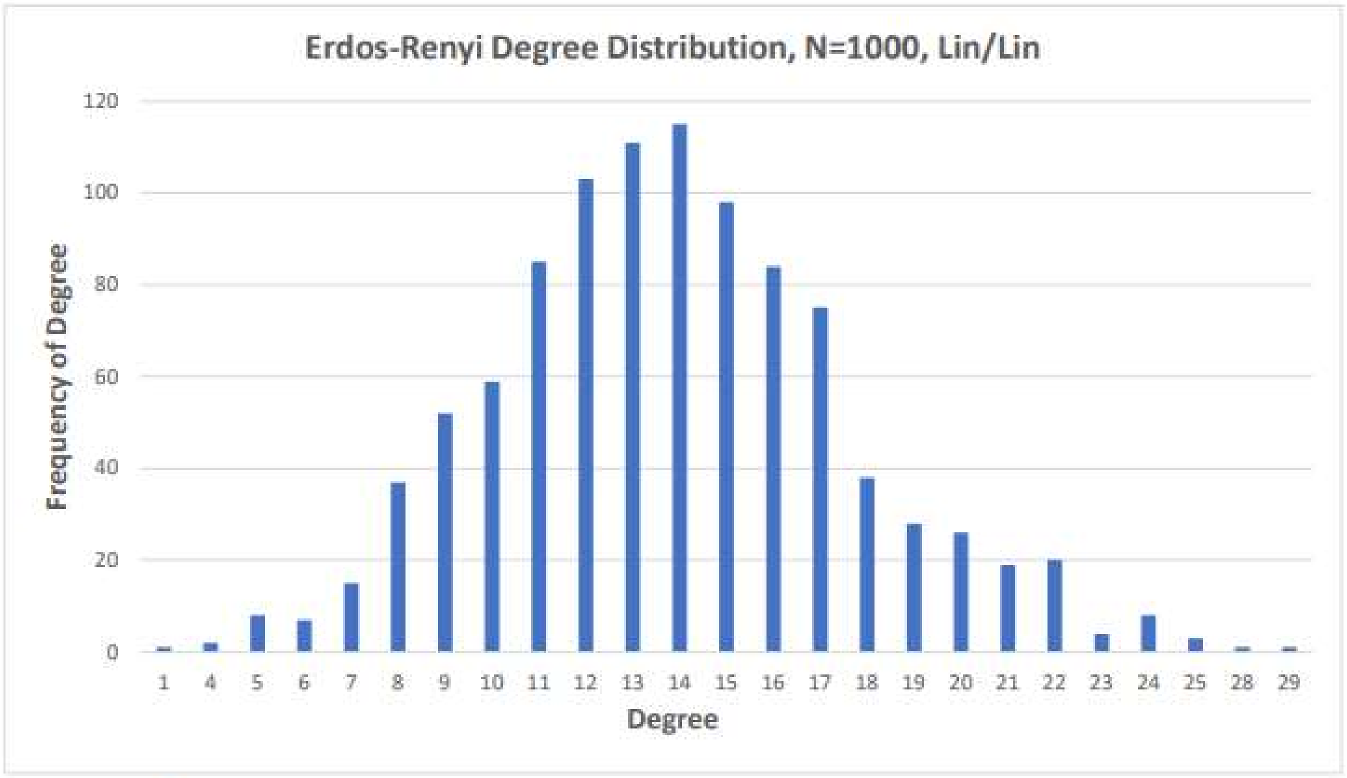

Figure 1.1: *For Erdos-Renyi, when N=1000, the highest degree frequency of a node is at 14 matching the expected N*P value of 13.9.*

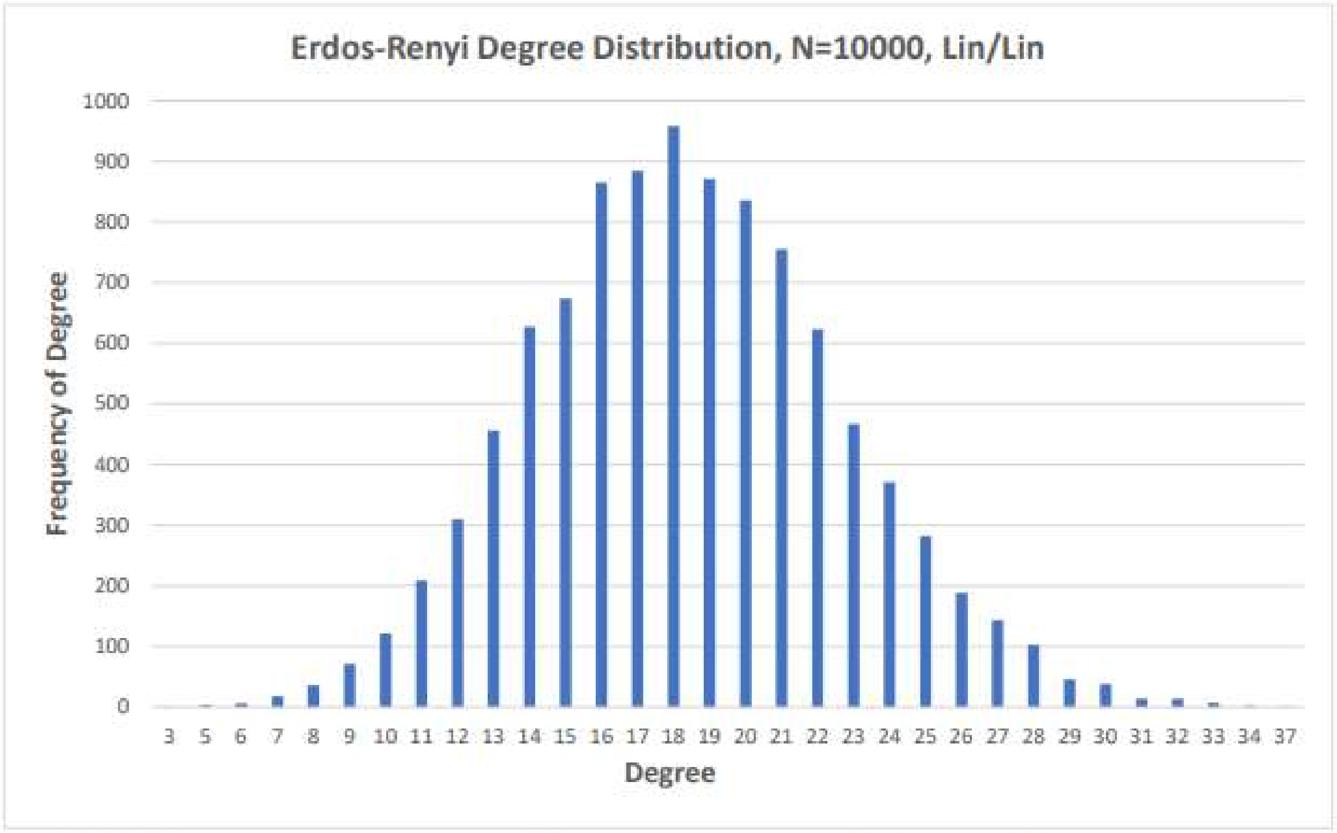

Figure 1.2: *For Erdos-Renyi, when N=10000, the highest degree frequency of a node is at 18 matching the expected N*P value of 18.4. The distribution evens with a higher input size.*

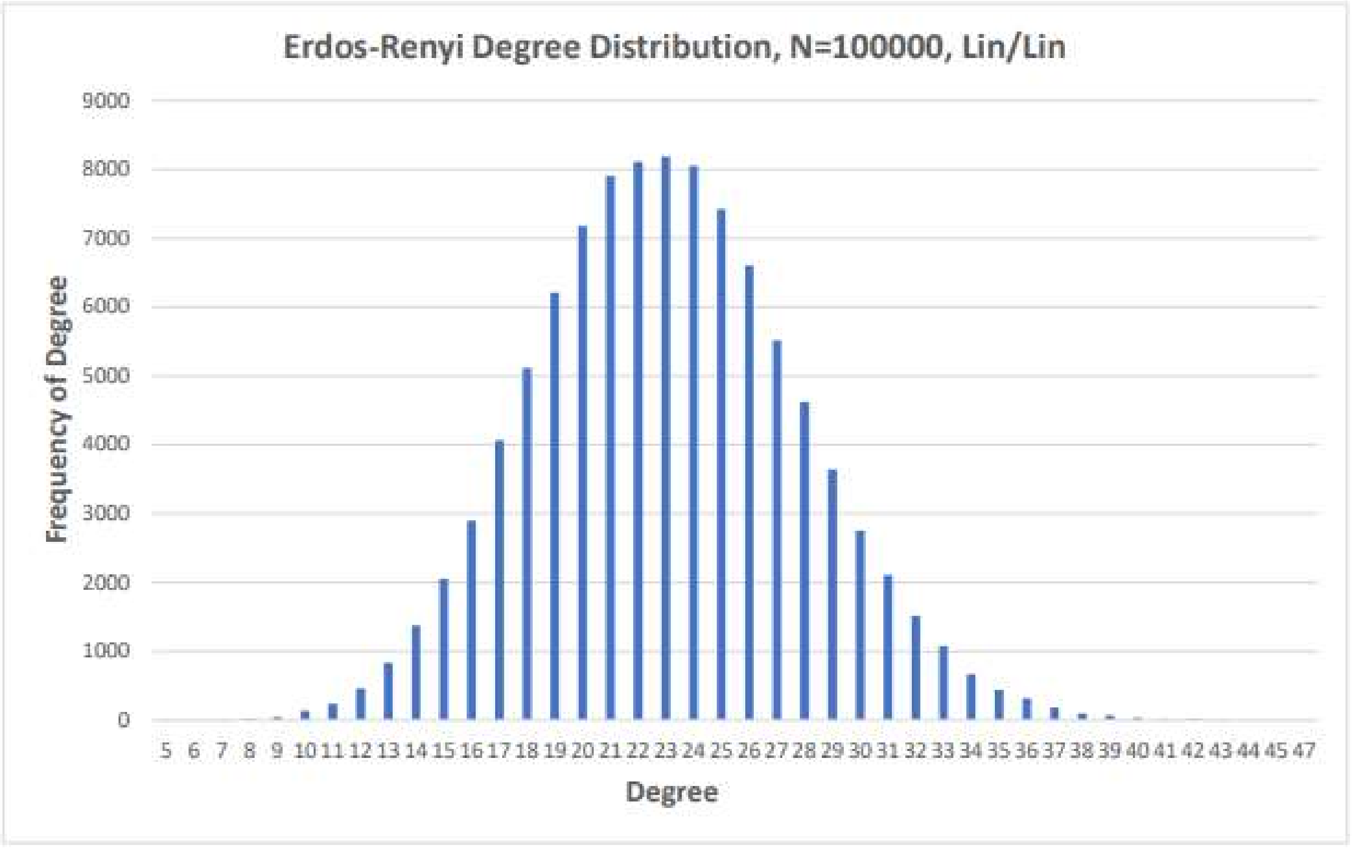

Figure 1.3: *For Erdos-Renyi, when N=100000, highest degree frequency of a node is at 23 matching the expected N*P value of 23. The distribution continues to grow evenly with increasing values of n.*

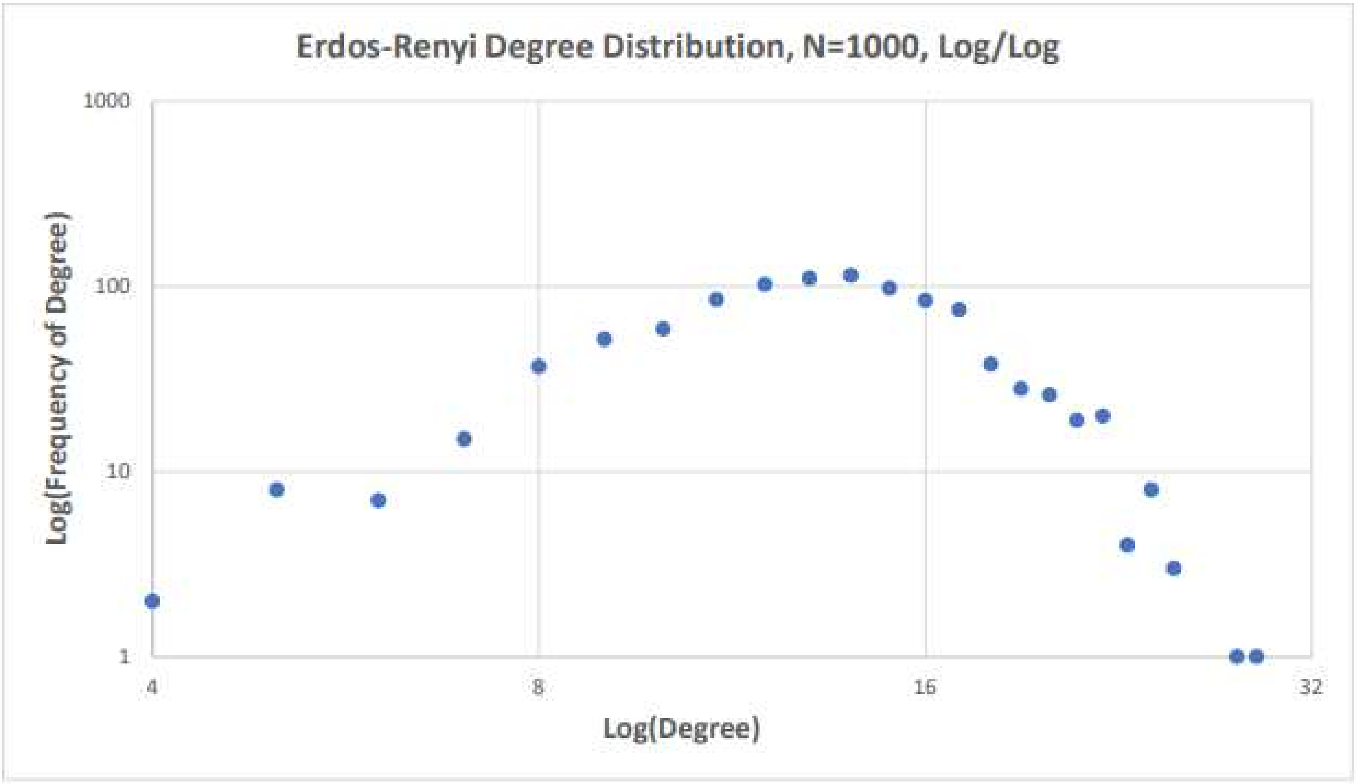

Figure 1.4: *For Erdos-Renyi, there is no power law reflected for N=1000 and degree frequency is relatively evenly distributed.*

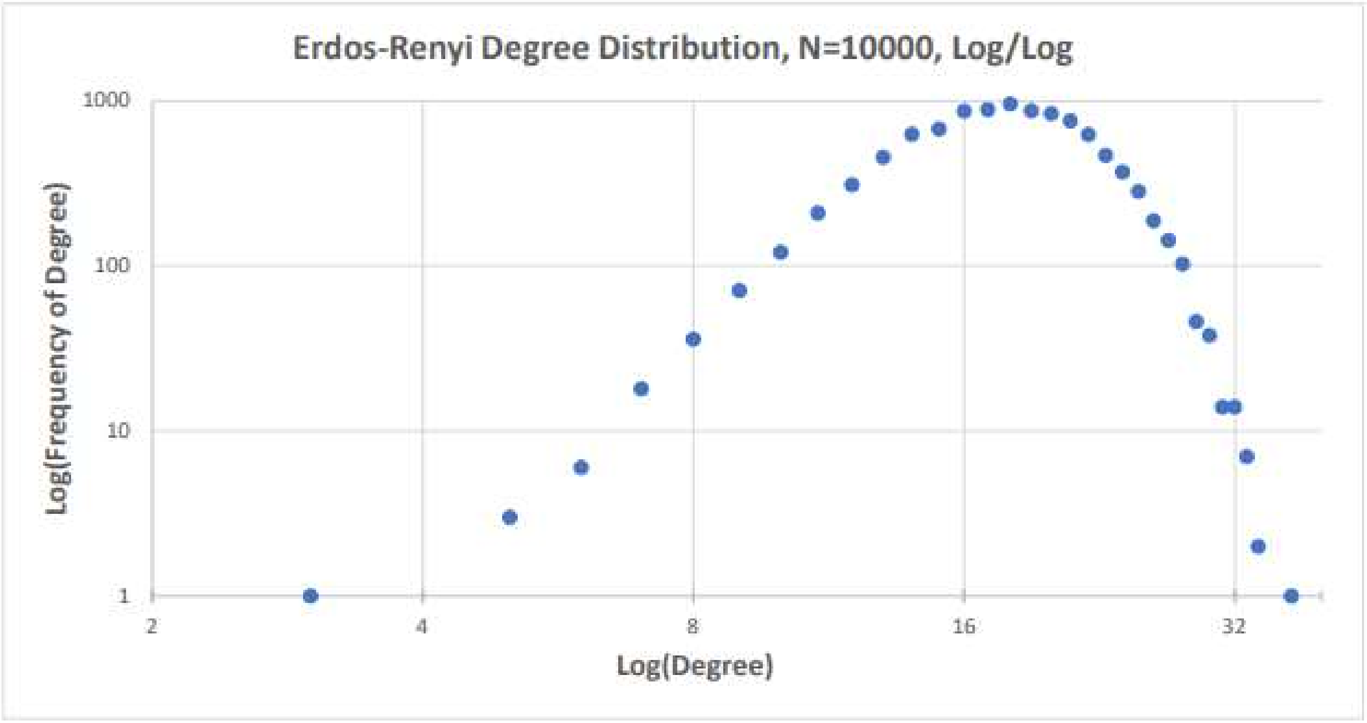

Figure 1.5: *For Erdos-Renyi, there is no power law reflected for N=10000. Variation of degree frequency continues to evenly distribute.*

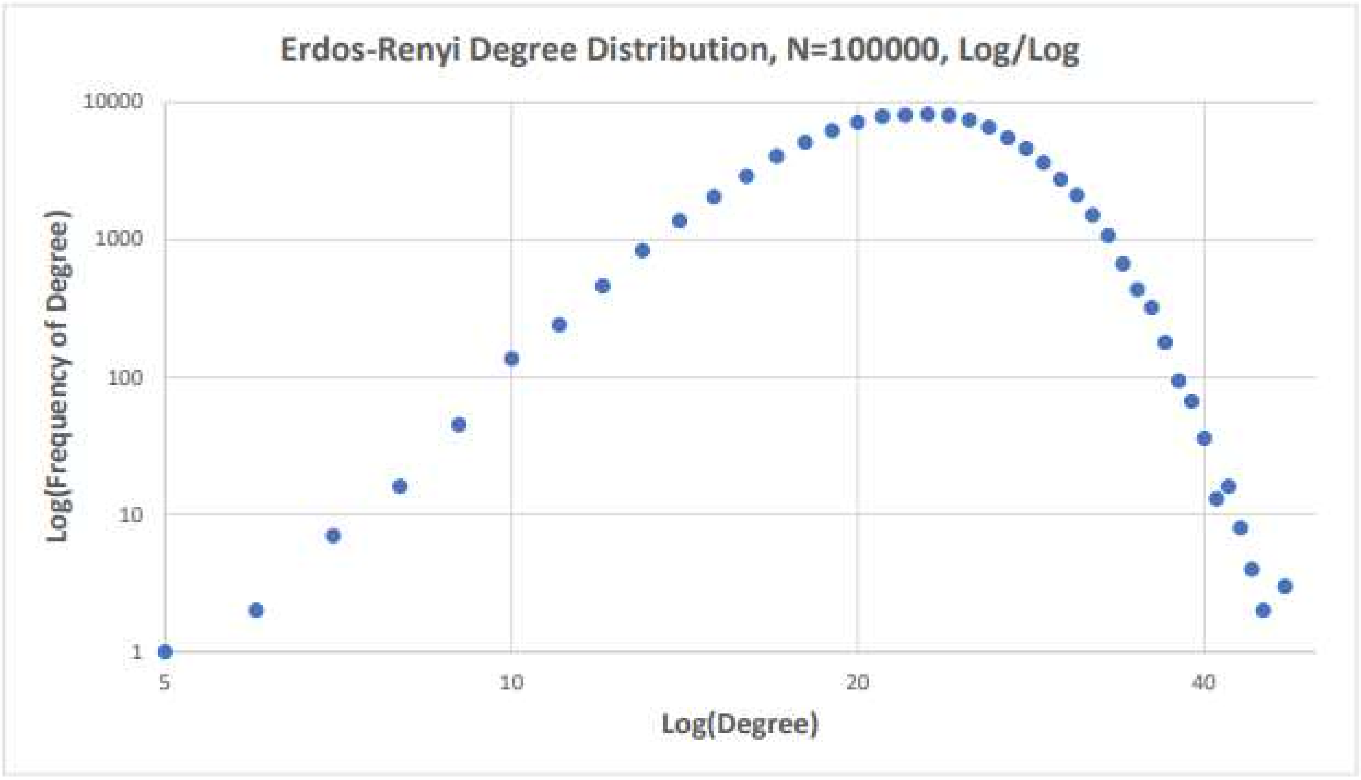

Figure 1.6: *For Erdos-Renyi, no power law reflected for N=100000. The trend continues with input size and the variation in degree frequency stays well distributed. Nearly a perfect arc is formed.*

Plotting Degree Distributions and Determining if They Follow a Power Law.

b. Barabasi-Albert

The degree distribution of the Barabasi-Albert graphs do exhibit a power law that is indicative of the "long-tail" result. As we can see, for each number of nodes *N* where *N=1000*, *N=10000*, and *N=100000*, the "long-tail" is present at each input size. As each size of *N* grows, Barabasi-Albert edge generation continues to preference the highest connected vertices growing the tail of the chart. Moreover, the chart shows that the amount of vertices that never increase beyond the starting degree of 5 continues to grow by the coefficient of the input size. For example, at *N=1000,* there are approximately *300* vertices with the minimum 5 degrees. At *N=10*(1000)=10000*, there are approximately *3000* vertices with the minimum 5 degrees. The "starving" vertices starve at exponential rates as the "rich" become richer. ***See Figures 1.7-1.12***

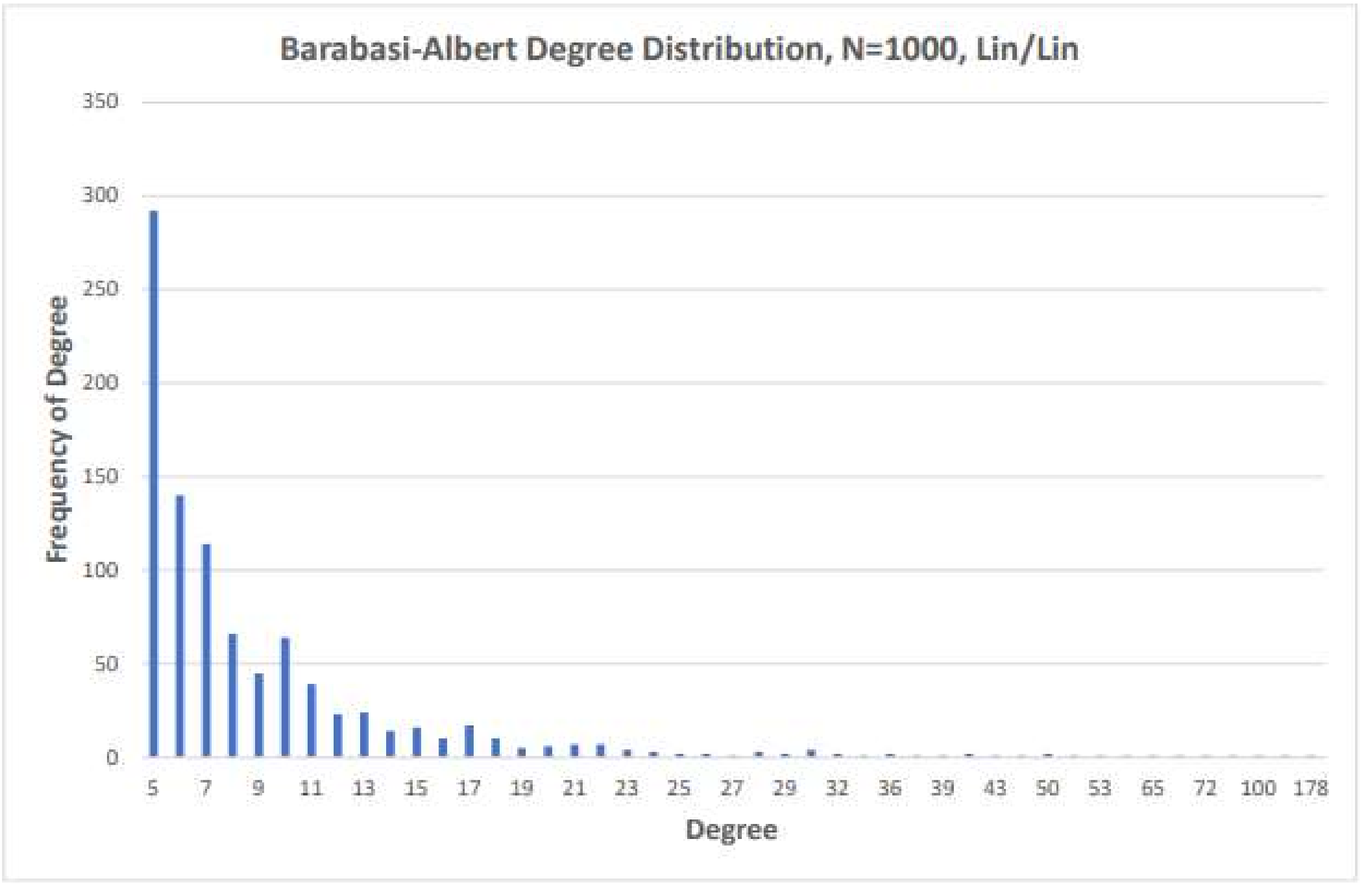

Figure 1.7: *For Barabasi-Albert, with N=1000 the amount of vertices with a degree of 5 are almost 300 with only a small selection of vertices having a degree of up to 178.*

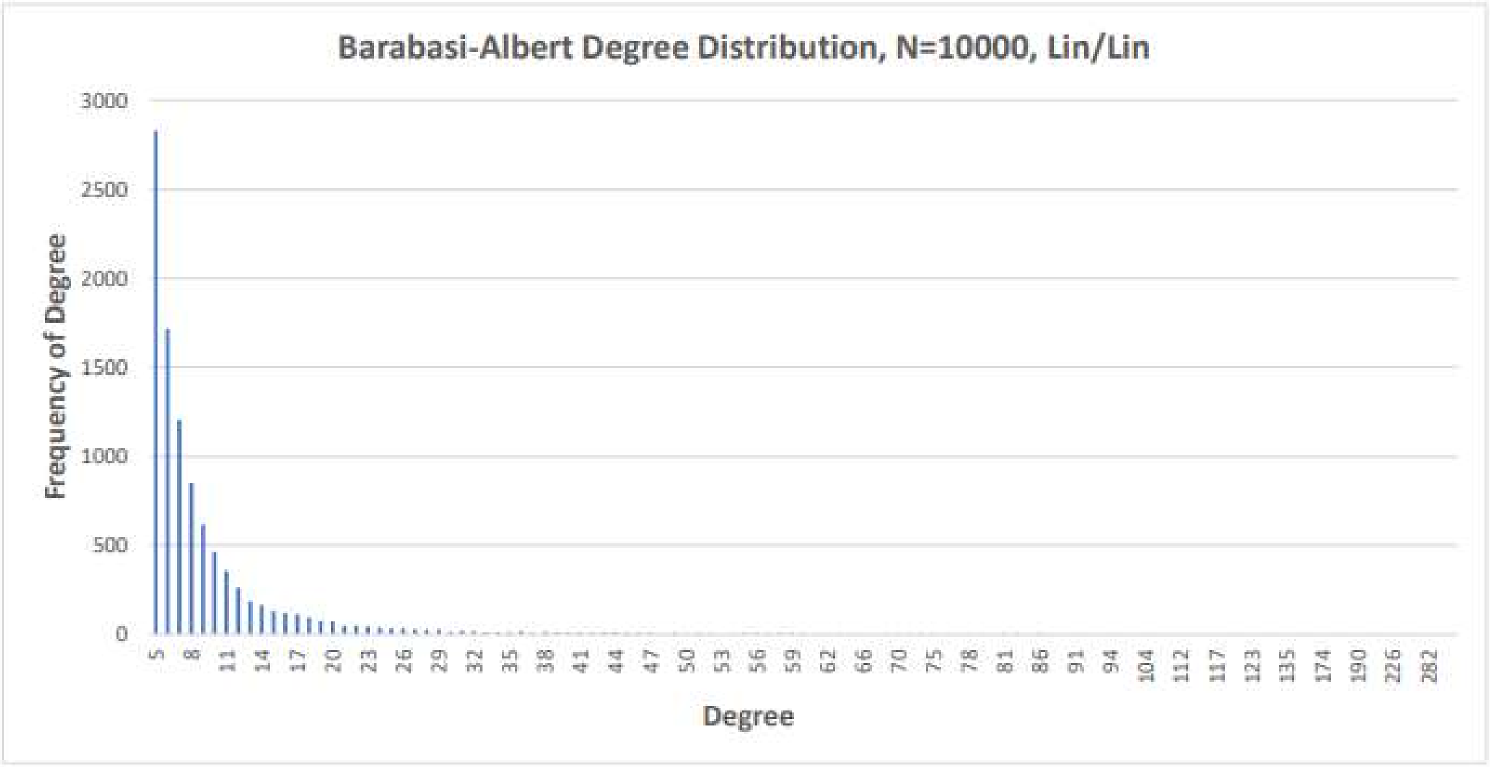

Figure 1.8: *For Barabasi-Albert, with N=10000 the amount of vertices with a degree of 5 are almost 3000 with an even smaller selection vertices having a degree of up to 282. The tail continues to grow longer.*

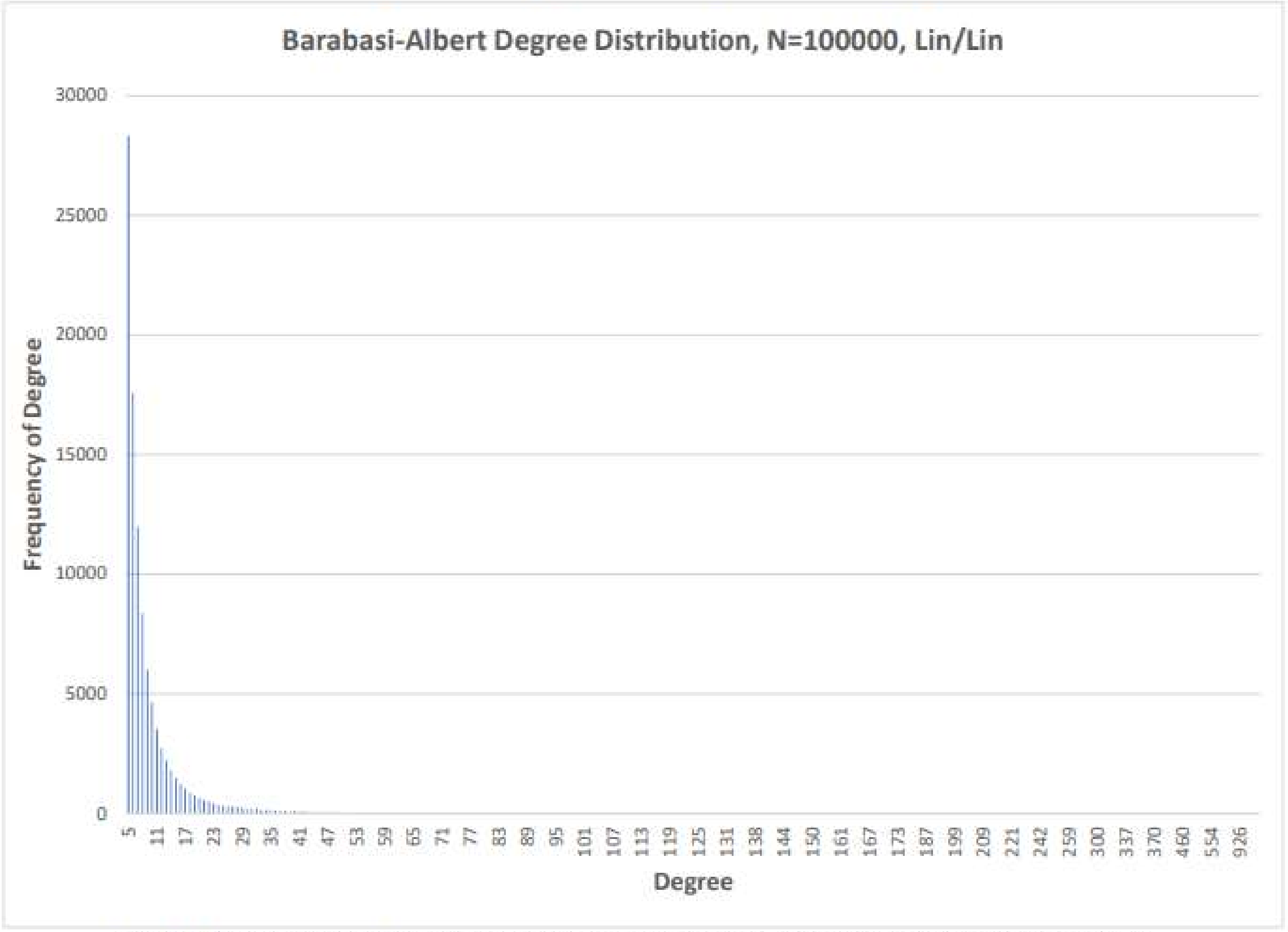

Figure 1.9: *For Barabasi-Albert, with N=100000 the amount of vertices with a degree of 5 are almost 30000 with an even smaller selection of vertices having a degree of up to 926. The discrepancy only increases.*

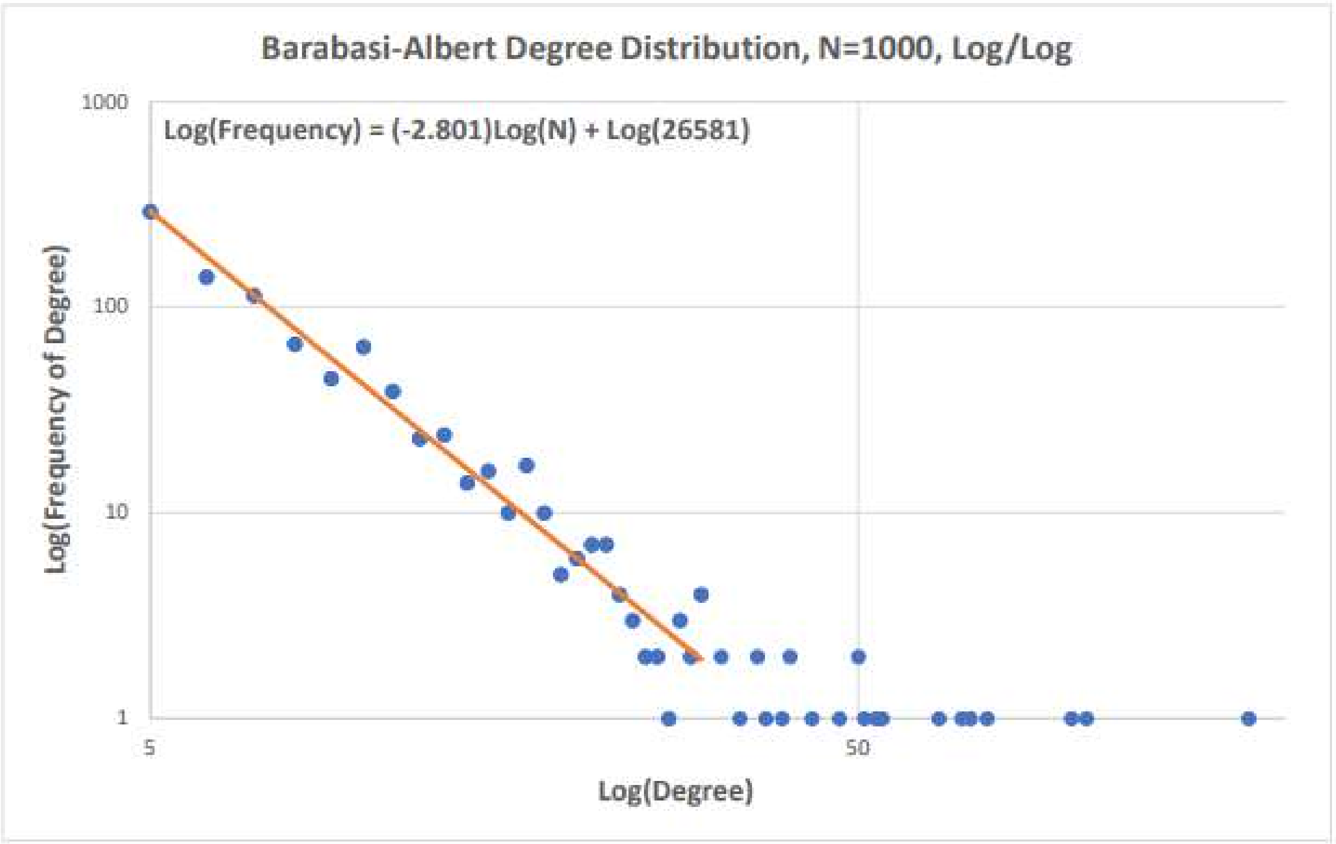

Figure 1.10: *With a slope of -2.801, we can see the frequency of degree drastically reduce as the degrees increase. While there is more variation as we continue right, the amount of vertices holding that variation are close to zero.*

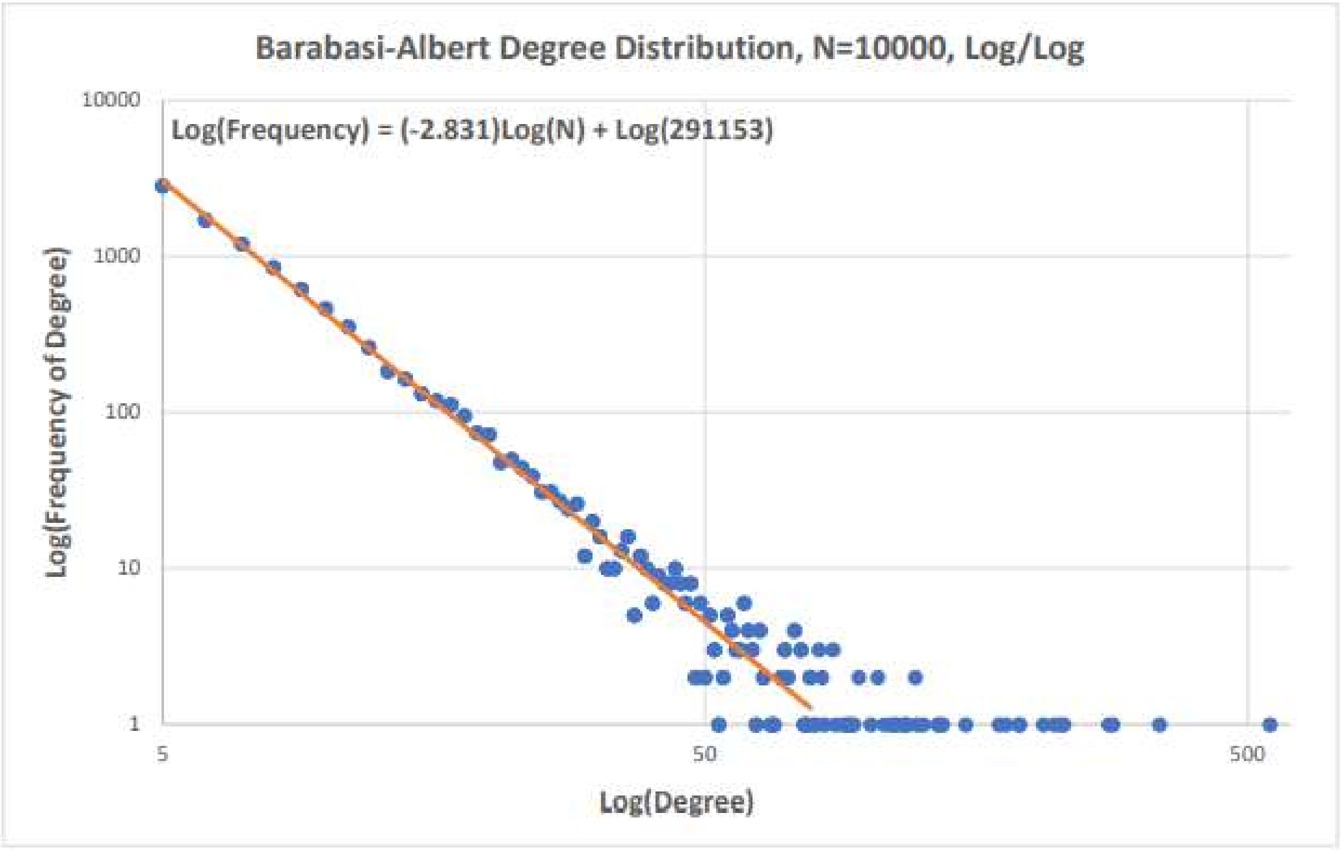

Figure 1.11: *The slope decreases to -2.831 and the frequency of degree drastically reduces as N grows.*

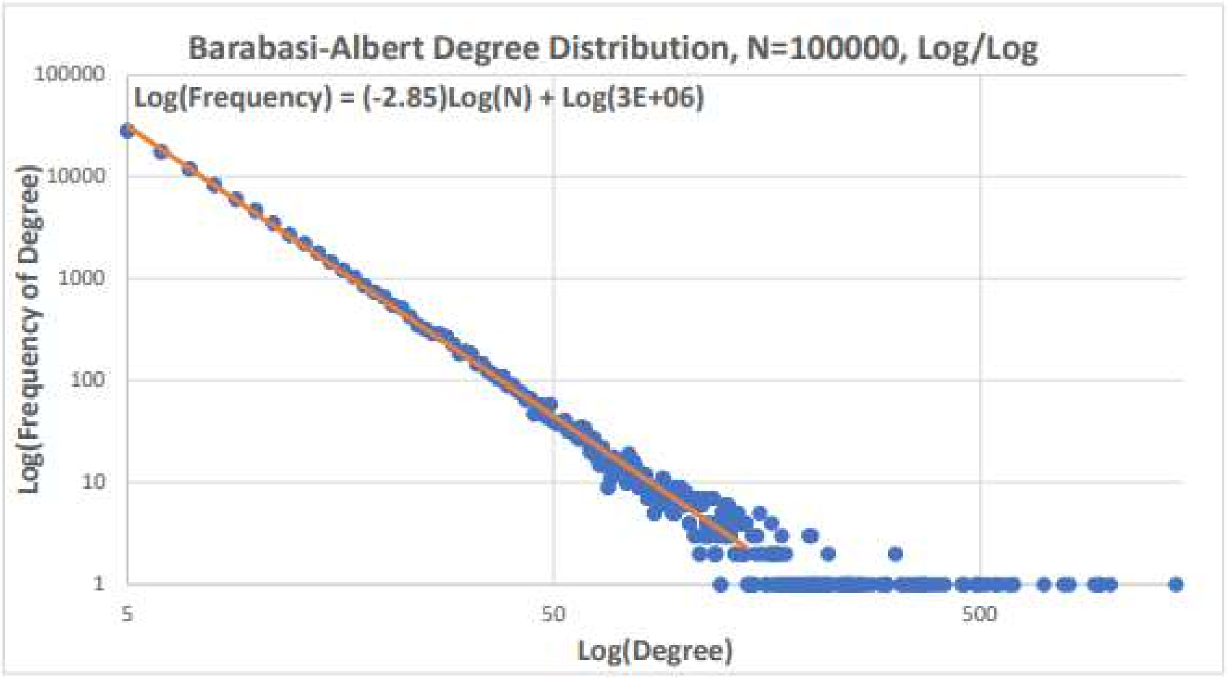

Figure 1.12: *The slope decreases to -2.85 continuing the trend of growing the few rich vertices.*

Plotting Clustering Coefficients and Determining if They Are Changing as a Function of n.

a. Erdos-Renyi

As we can see, for Erdos-Renyi, the clustering coefficients decrease as a function of *n* with a *slope of -0.084* for the range with the greatest change in clustering coefficient. The range here is *2^3 to 2^8* nodes. Once there are *2^9* input values, the clustering coefficient begins to flatten out continuing to approach zero as *n* increases. By *n=2^14*, the clustering coefficient is so small that the graph becomes too hard to determine any further change. Therefore, Erdos-Renyi has created increasingly sparse graphs as *n* grows through my testing. ***See Figure 2.1.***

b. Barabasi-Albert

With Barabasi-Albert, the result was a bit surprising. Although Erdos-Renyi and Barabasi-Albert produce completely opposite graphs with regard to degree frequency, they still both produce sparse graphs as *N* grows. The clustering coefficient is also low for Barabasi-Albert. The only real difference to that of Erdos-Renyi is clustering starts low with Barabasi-Albert and high with Erdos-Renyi. This is likely due to the initial requirement for Barabasi-Albert that each vertice has a minimum degree of 5. Erdos-Renyi, therefore, "catches up" to Barabasi-Albert with a steeper negative slope. At input size of *2048*, both graph models have a clustering coefficient of approximately *0.01.* At this point, the graphs maintain similar results. ***See Figure 2.2.***

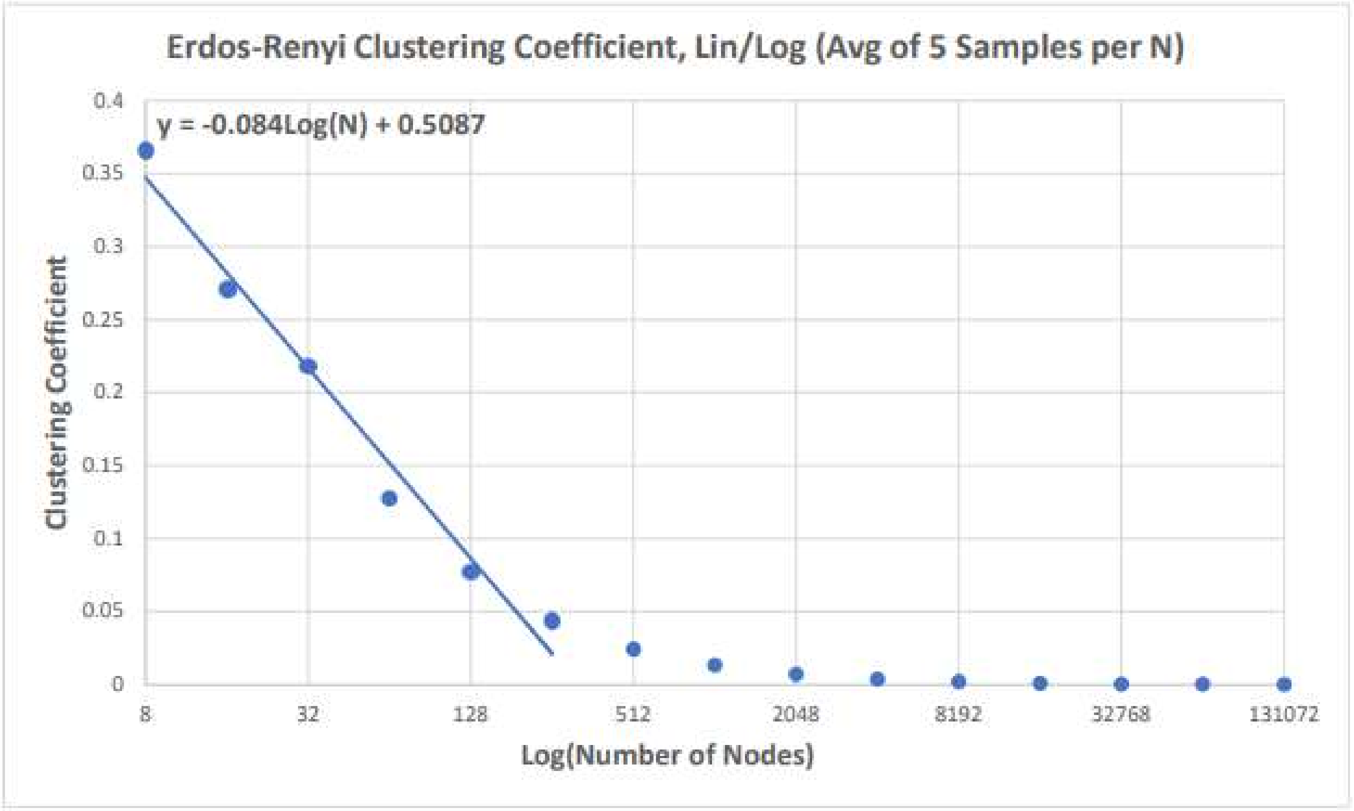

Figure 2.1: *The Erdos-Renyi model starts with high clustering and quickly has an increasingly low clustering coefficient as a function of n as n grows.*

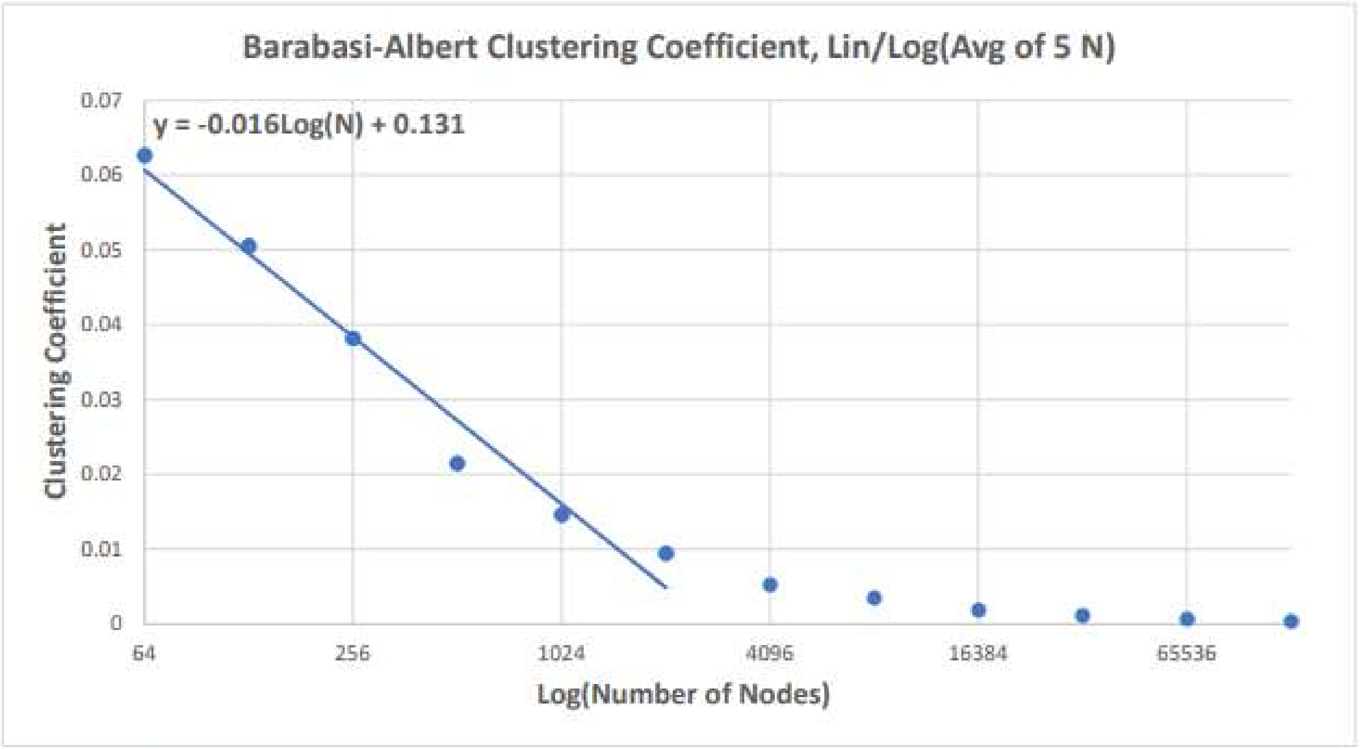

Figure 2.2: *Unlike the Erdos-Renyi modek, Barabasi-Albert starts with a very low clustering coefficient. However, it approaches zero as n grows similar to Erdos-Renyi.*

Plotting the Diameters and Determining if They Are Changing as a Function of n.

a. Erdos-Renyi

The diameter results of the Erdos-Renyi model show that it does increase as *n* grows. However, we notice that the diameter grows at a slow rate with a slope of only 0.2525. This was the most surprising result of all my testing and is an empirical reflection of the "Small World" phenomenon. With a graph of size *2^3*, we already have a diameter of 3. However, on a graph of size *2^17*, we have only a diameter of approximately *5½*. Therefore, the diameter grows as a function of n, but very slowly. ***See Figure 3.1.***

b. Barabasi-Albert

The diameter results of the Barabasi-Albert model are fairly similar to that of Erdos-Renyi. The differences from my testing are that we started with a slightly lower diameter than Erdos-Renyi but ended up with a slightly higher diameter. Therefore, the slope is higher for Barabasi-Albert at approximately *.15* more. Both models exhibit the "small world" phenomenon, but Erdos-Renyi does perform better here. ***See Figure 3.2.***

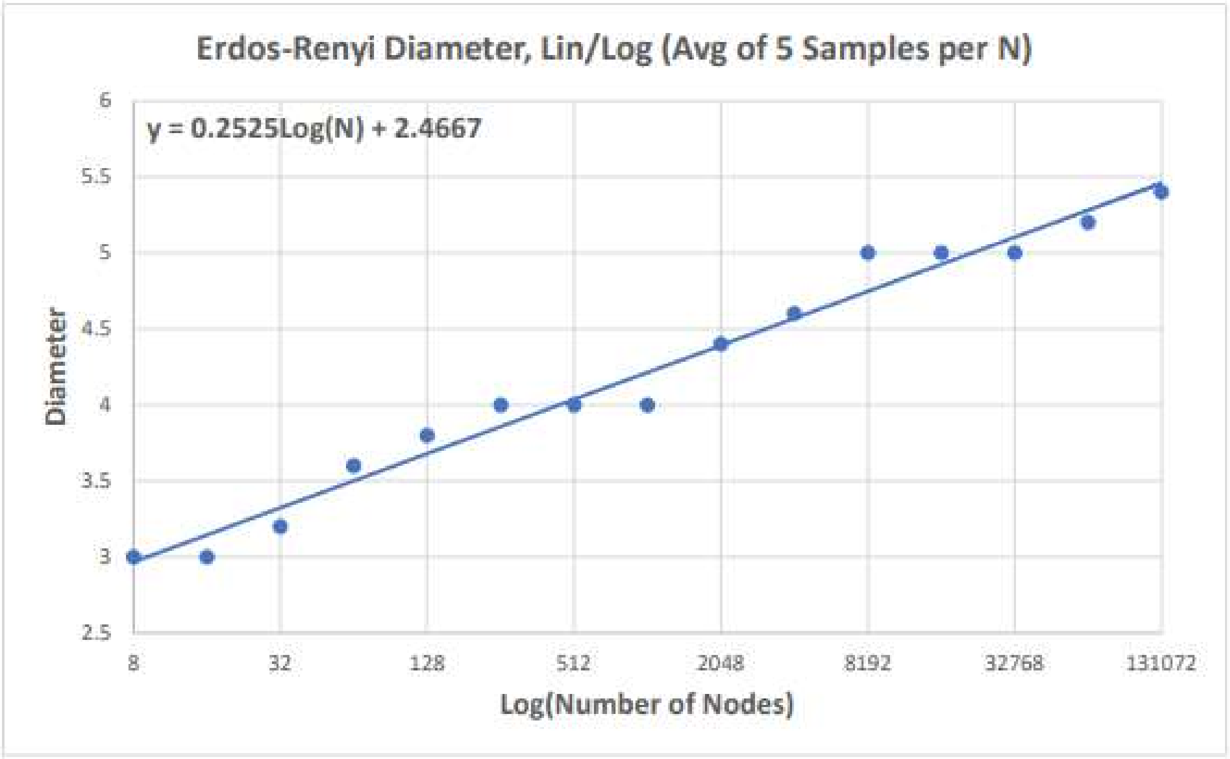

Figure 3.1: *The Erdos-Renyi model displays the "small world" phenomenon with a diameter of only ~ 5½ at input size of 2^17.*

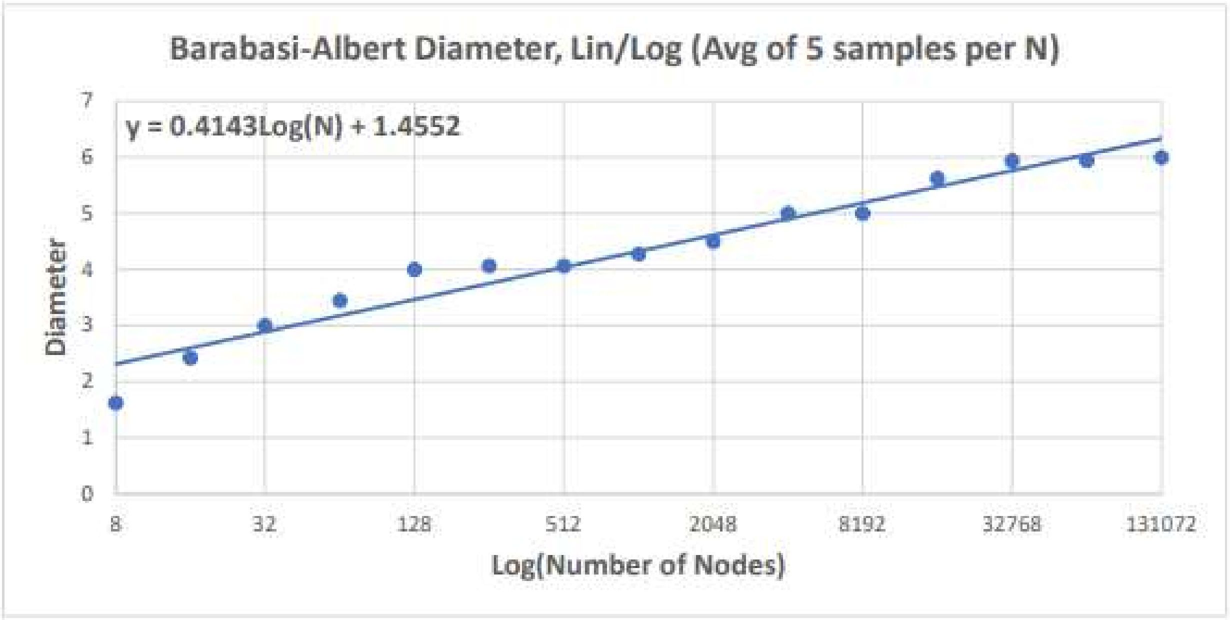

Figure 3.2: *The Barabasi-Albert also displays the "small world" phenomenon with a diameter of ~6 at input size of 2^17. We see it starts with a smaller diameter than Erdos-Renyi, but has a higher diameter as n grows.*

Report on performance for both Erdos-Renyi and Barabasi-Albert graph models

Erdos-Renyi and Barabasi-Albert models are completely different in their degree distribution. Erdos-Renyi tries to be "fair" to the vertices of the graph attempting to share connectivity evenly. As input size grows, degree distributions continue to level out with a nearly perfect bell-shaped curve when modeled. Barabasi-Albert, on the other hand, shows a stark contrast to this happening. It is a greedy model that rewards the vertices who gain momentum with their degree frequency. The results show that the majority of vertices suffer from having few connections where a select few garner hundreds at an input size of only *N=100000*. Barabasi-Albert further exhibits the power law here where when *N=1000, ~300* vertices have the minimum 5 degrees. At *N=10000, ~3000* vertices have the minimum 5 degrees. Finally, at *N=100000, ~30000* have the minimum 5 degrees. There is no power law exhibited for Erdos-Renyi.

These differences in degree distribution affect the resulting diameter of the graphs. The Erdos-Renyi diameter should grow at a smaller rate since there are fewer outliers in degree distribution in comparison to Barabasi-Albert. The outliers of Barabasi-Albert only continue to grow due to the exhibited power law and will therefore always maintain select vertices that are further away from the mass increasing the diameter.

The clustering coefficient of both graphs are their closest result. For me, it was expected that Erdos-Renyi graphs should have a low clustering coefficient due to the evenness of the model, however, I did not expect the Barabasi-Albert model to produce this result. After inspection, the phenomena is logical. Barabasi-Albert starts with a low clustering coefficient and continues to connect vertices to the already balanced model. While the majority of vertices "starve" for connections, there are so many of them that the graph remains sparse. On the contrary, Erdos-Renyi starts with a much higher clustering coefficient and quickly approaches zero as input size grows. Therefore, both Erdos-Renyi and Barabasi-Albert models produce sparse graphs with low clustering coefficients as *n* grows.