Project 3 Report

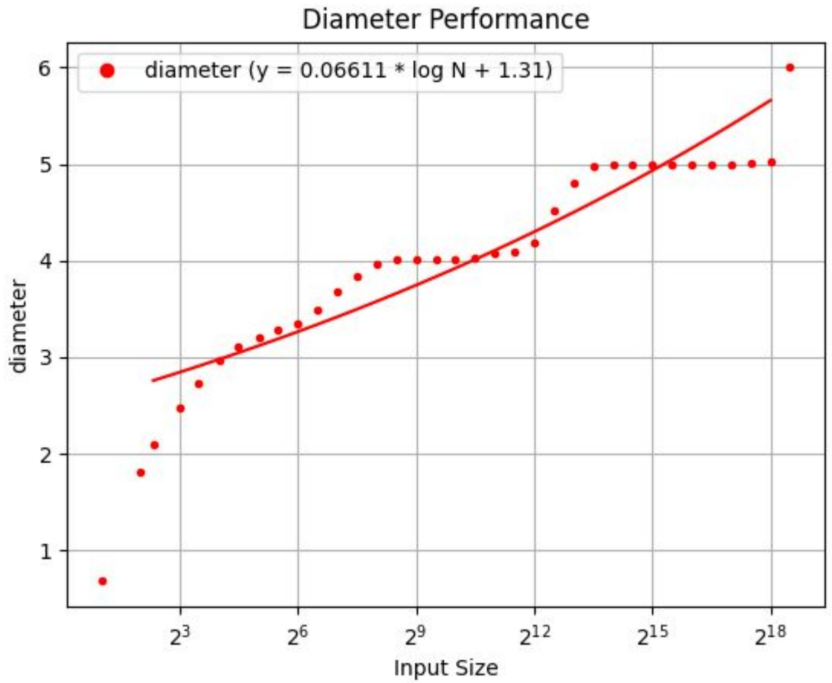
Erdos-Renyi Implementation

I implemented the Erdos-Renyi graph generation using a method that creates a graph with n nodes and an edge probability p = 2(ln n)/n. I used a list to keep track of edges and 2 counters, ***v*** and ***w*** to iterate through potential edges (while ***v*** < ***n***) and a random number to determine the gap until the next edge is added. Continue until all edges for the ***n*** nodes are generated, resulting in a random graph consistent with the Erdos-Renyi model.

Diameter Algorithm

Implementation: First, we select a random node ***r*** in the graph. We then find the furthest node ***w*** from ***r*** using BFS**.** We then perform a 2nd BFS from ***w*** to find the farthest node from ***w***. We then return the max of the 2 BFS results

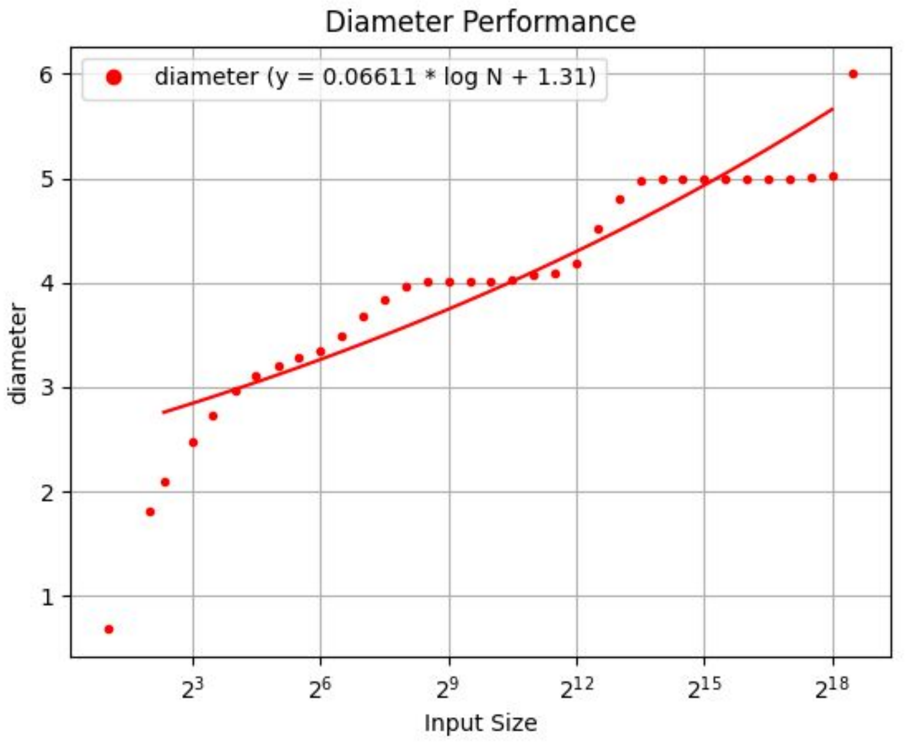
Note: ED #166 says we don't need to use psuedocode



Diameter Conclusions

We can see that the diameter is forming a series of “flat plains.” While the diameter does increase as the number of nodes grows, it doesn’t grow linearly. Instead, the trend aligns more closely with logarithmic growth, meaning the average diameter tends to increase in proportion to log(n) rather than n.

Generated using Erdos-Renyi



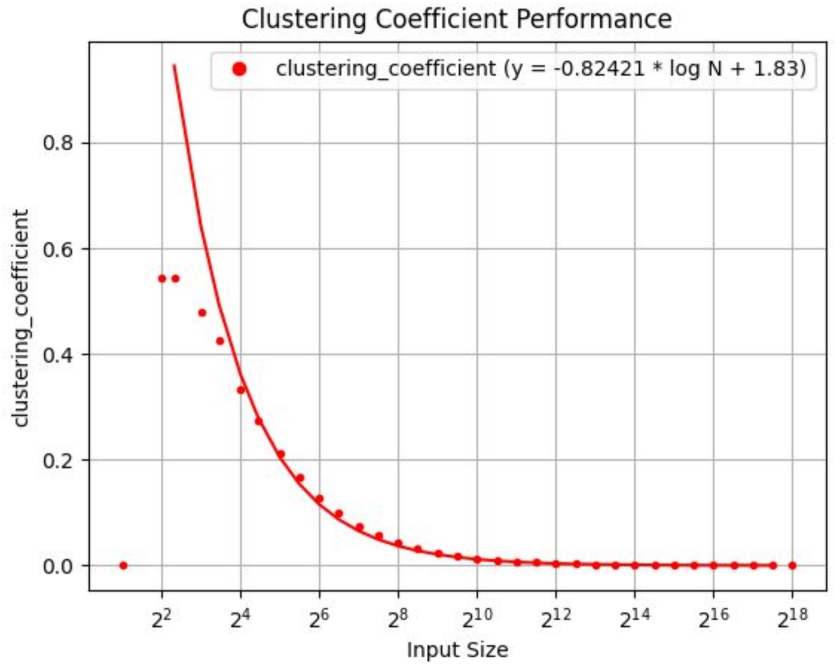
Clustering Coefficient Algorithm

To compute the clustering coefficient of a graph, I first implemented a triangle counting algorithm optimized using a degeneracy ordering. In the get_d_ordering() function, each node is assigned to a bucket based on its degree, and nodes are iteratively removed starting from the lowest-degree bucket. As nodes are processed, I update the degrees of their neighbors and track, for each node, which of its neighbors appeared earlier in the ordering. This ordering allows for efficient triangle counting in get_triangles(), where I only check for connections between neighbors that appeared earlier, avoiding redundant checks and double-counting. Once the number of triangles is computed, I calculate the total number of possible triplets based on node degrees and use the standard formula for the clustering coefficient: (3 * triangle_count)/total_triplets

Note: ED #166 says we don’t need to use psuedocode

Clustering Coefficient Conclusion

The clustering coefficient exhibits a logarithmic decrease with the number of nodes, dropping sharply from 0.9 to near 0 as n increases from 2^1 to 2^18. The line of best fit for the data corresponds well to a curve proportional to a typical 1/logn curve. This shows that as n increases, the graph gets less and less clustered.

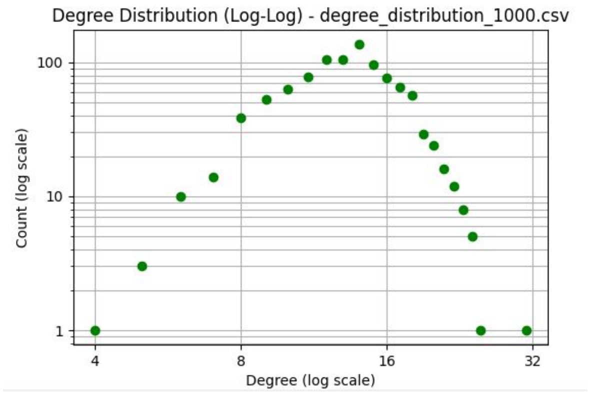


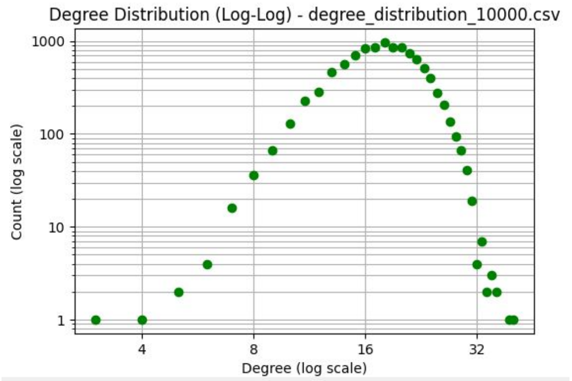
Degree Distribution Algorithm

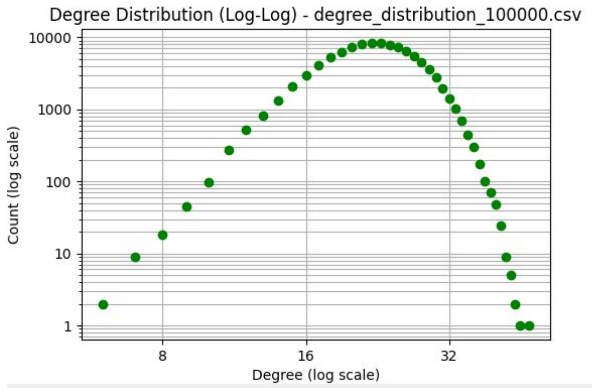
I implemented the degree distribution by iterating through each node in the graph and calculating its degree based on the length of its adjacency list. I then compiled these degree counts into a dictionary that maps each degree value to the number of nodes exhibiting that degree.

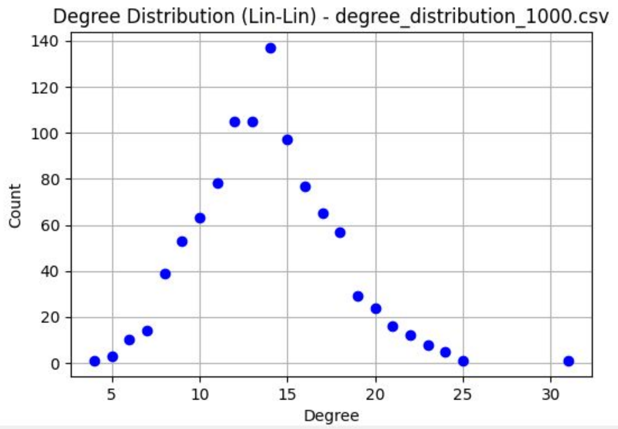
Note: ED #166 says we don’t need to use psuedocode

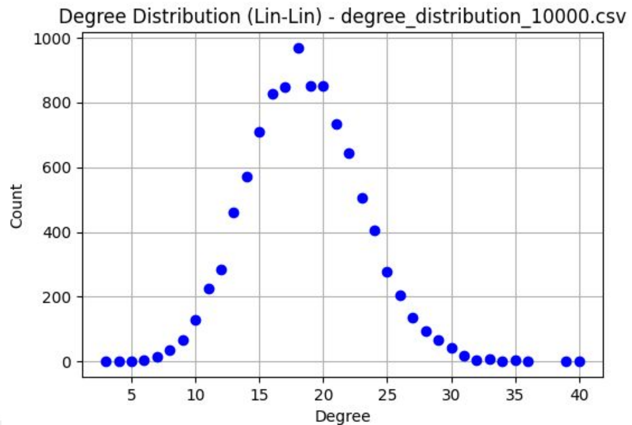
Degree Distribution Plots

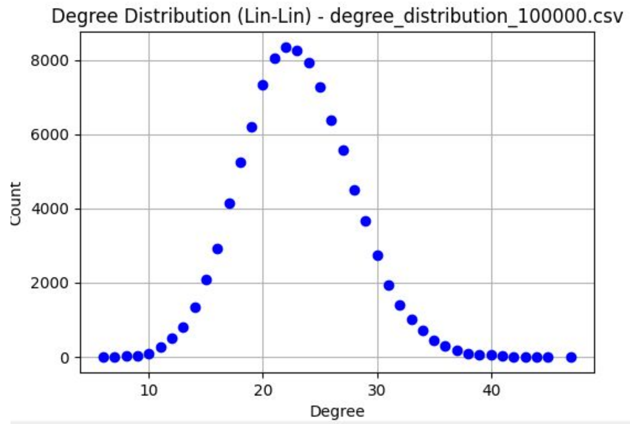












Degree Distribution Conclusion

Based on the Degree Distribution plots for random Erdos-Renyi graphs with n = 1000, n = 10000, n=100000, we observe that both the lin-lin and log-log plots exhibit a bell-shaped curve rather than a heavy-tailed distribution characteristic of a power law. The lack of a clear linear trend on the log-log scale, especially in the tail regions, indicates the absence of power-law behavior, meaning no meaningful slope or exponent can be fitted to the data. This suggests that the Erdos-Renyi random graph model likely follows a binomial or Poisson degree distribution rather than a power-law distribution.