CS 124 Programming Assignment 1: Spring 2023

Your name(s) (up to two): Jasmine Zhang, Kaitlyn Zhou

Collaborators: n/a

No. of late days used on previous psets: 0
No. of late days used after including this pset: 0

Homework is due Wednesday Feb. 22 at 11:59pm ET. You are allowed up to **twelve** (college)/**forty** (extension school) late days through the semester, but the number of late days you take on each assignment must be a nonnegative integer at most **two** (college)/**four** (extension school).

What to hand in: Besides *submitting a copy of the code you created*, your group should hand in a single well organized and clearly written report describing your results. For the first part of the assignment, this report must contain the following quantitative results (for each graph type):

- A table listing the average tree size for several values of n. (A *graph* is insufficient, although you can have that too; we need to see the actual numbers.)
- **Dimension 0:**

n	Average Weight	Time (milliseconds)
128	1.1478418	4
256	1.1884851	5
512	1.192078	12
1024	1.1949253	34
2048	1.2061161	102
4096	1.2090738	381
8192	1.2165079	1817
16384	1.1964153	6629
32768	1.1983122	29480
65536	1.1981264	119360
131072	1.1992619	510212
262144	1.1958005	11461913

- **Dimension 2:**

n	Average Weight	Time (milliseconds)
128	7.517803	5
256	10.723677	10
512	15.047468	21
1024	20.794521	39
2048	29.538305	113
4096	41.75341	449
8192	58.99131	1781
16384	83.19006	8689
32768	117.5072	166787
65536	166.00212	383224
131072	234.54898	719724
262144	331.70428	6711917

- **Dimension 3:**

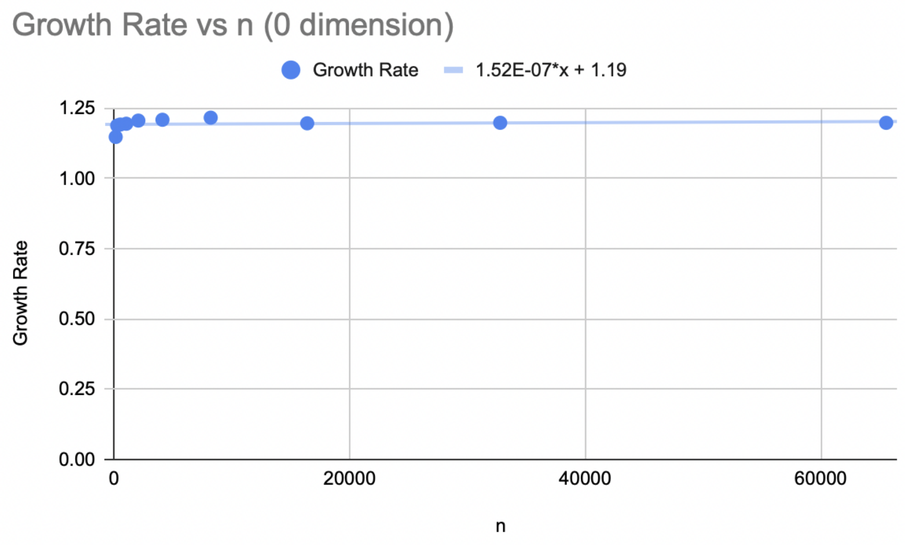
n	Average Weight	Time (milliseconds)
128	17.474815	4
256	27.763723	26
512	43.193542	19
1024	67.97221	32
2048	107.4636	75
4096	168.46289	269
8192	267.28543	1001
16384	422.1049	7215
32768	668.2659	25934
65536	1059.3376	113294
131072	1677.3324	457690
262144	2659.438	5627728

- **Dimension 4:**

n	Average Weight	Time (milliseconds)
128	28.893885	3
256	47.058754	7
512	78.10452	15
1024	130.11275	32
2048	215.89151	76
4096	360.85257	309
8192	603.6745	1237
16384	1008.2189	5292
32768	1688.207	27733
65536	2827.5454	116395
131072	4738.0386	496803
262144	7952.0703	2118567

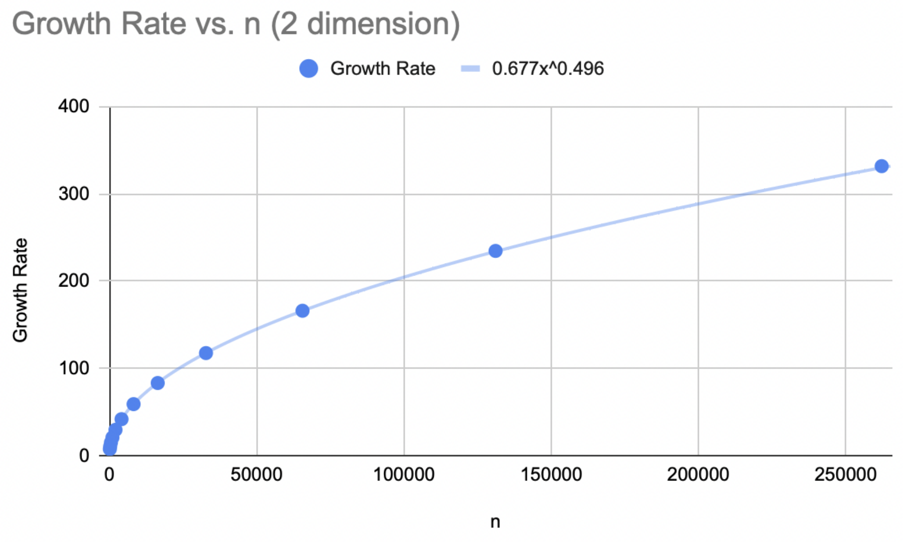
- A description of your guess for the function $f(n)$.

- **Dimension 0:**



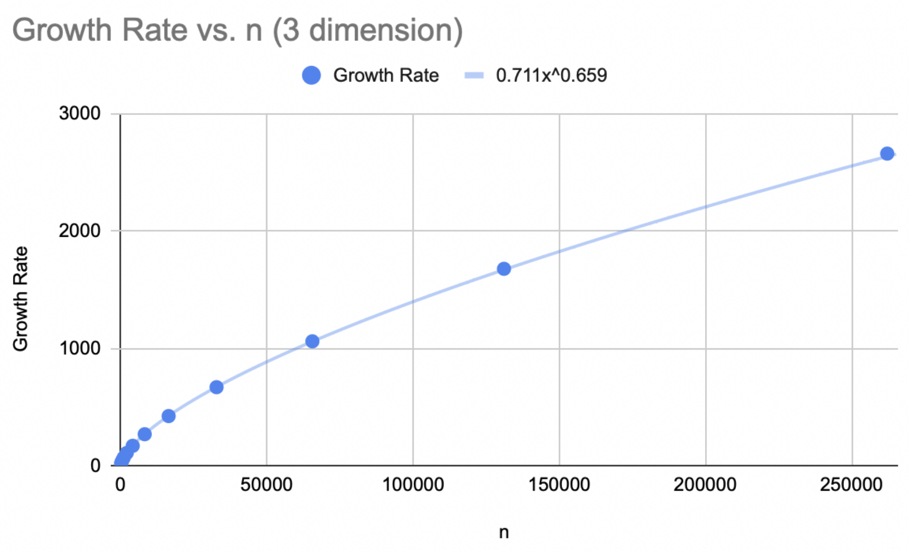
An estimated function for dimension 0 would be $f(n) = 1.19$

- **Dimension 2:**



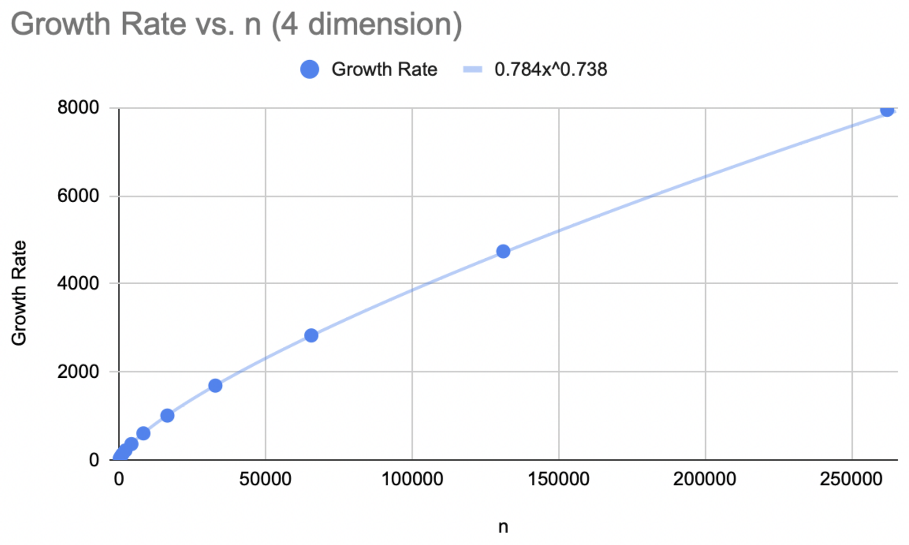
An estimated function for dimension 2 would be $f(n) = 0.677 * n^{1/2}$

- **Dimension 3:**



An estimated function for dimension 3 would be $f(n) = 0.711 * n^{2/3}$

- **Dimension 4:**



An estimated function for dimension 4 would be $f(n) = 0.784 * n^{3/4}$

- To estimate $f(n)$, I first graphed the points that we obtained through our code for each value of n afterwards I tried to find the line/curve of best fit.

In addition, you are expected to discuss your experiments in more depth. This discussion should reflect what you have learned from this assignment. You should, at a minimum, discuss the asymptotic runtime of your algorithm(s) and the correctness of any modifications from the standard algorithms presented in class; otherwise, the actual issues you choose to discuss are up to you. Here are some possible suggestions for the second part:

- Which algorithm did you use, and why?

  - The algorithm that we decided to implement was the Prim's Algorithm. The reason why we used Prim's Algorithm over Kruksal's Algorithm is because Prim's algorithm runs significantly faster than Kruskal's Algorithm without modifications when the graph contains a large amount edges in the graph. Specifically, since we are running on complete graphs, the graph will have n vertices and n^2 edges. Our version of Prim's algorithm will only need to explore each edge once, running in $O(E) = O(n^2)$ time (explained further in the third bullet point). In contrast, Kruksal's algorithm will have to sort all the edges in the graph and then go through all the sorted edges and check whether or not each edge is part of the minimum spanning tree, running in $O(E \log E) = O(n^2 \log n)$ time by lectures notes. Therefore, since we are dealing with very large values of edges, we decided to use Prim's algorithm for our code implementation.

    Moreover, we decided not to use a priority heap for finding the minimum distance vertices since the number of edges in the graph was very high. From lecture, we know that running Prim's using a binary heap has the same runtime as Dijkstra's algorithm using a binary heap, so it would have a runtime of $O((V + E) \log V) = O(n^2 \log n)$, so still worse that $O(n^2)$ time. Instead, we just iterated through the distance array to find the minimum distance vertex that is not explored with the help of a visited array. This does not change the correctness of the algorithm since the distance array stores the weight of the edge with the minimum distance connecting a vertex to the MST, which is what we add to our running total weight and is correctly updated every time we visit a new vertex.

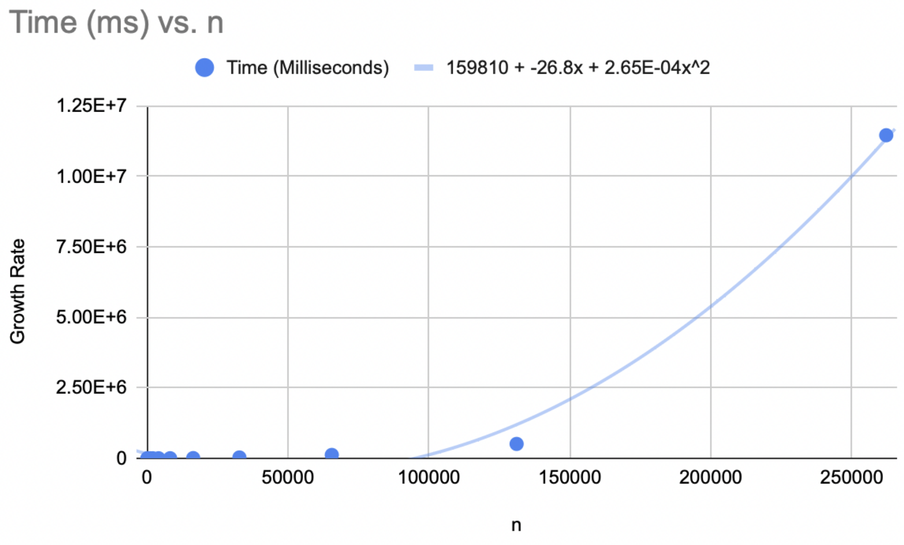
    Moreover, with this implementation, we did not need to store the weight of each edge since we only explore each edge once. Thus, we can randomly generate or calculate the distance as we explore each edge and never need to store the information since we never need it again. Since we were only asked about the total weight of the minimum spanning tree, we did not need to store each individual weight, saving us a lot of memory when n got big.

- Are the growth rates (the $f(n)$) surprising? Can you come up with an explanation for them?

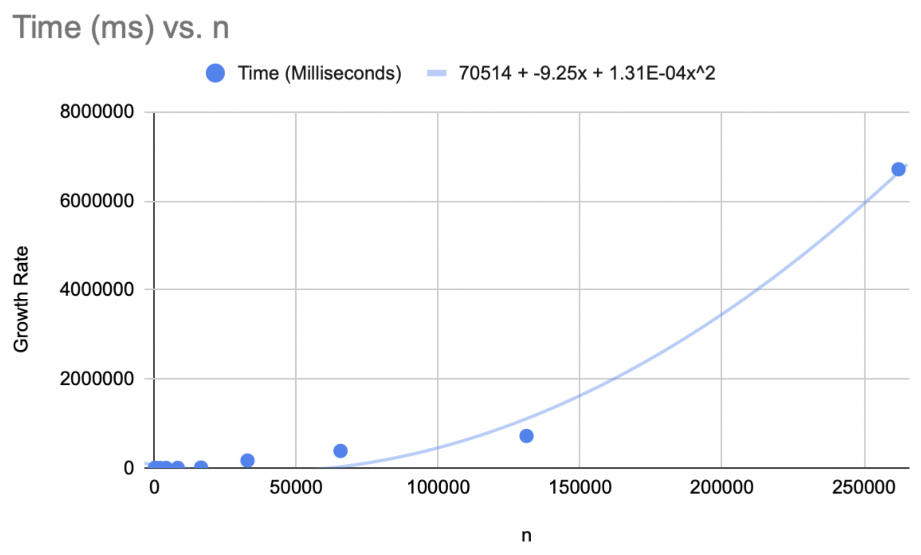
  - We believe that the growth rates were not surprising.
  - We saw that for dimension 0, there was a constant function and thus a constant growth rate (we estimated that $f(n)$ would be $f(n) = 1.19$). We believe this was the case because the more nodes that we will have in the graph will means that we will have more edges in the graph as well, this means that we will have a greater likely-hood of including small value edges in our minimum spanning tree. This means that as we continue to grow our vertices and edges, the sum of the edges of the minimum spanning tree will converge to a similar value.
  - As we increasing the dimensions of the graph, the largest possible distance for an edge between two vertices would also increase (from 1 (dimension 0) to $\sqrt{2}$ (dimension 2) to $\sqrt{3}$ (dimension 3) to 2 (dimension 4)). First, we can calculate the largest possible distance between two edges as if we are calculating the distance between two corner vertices of a square that is diagonal to each other. Therefore, we will use the distance formula and we find that the maximum distance would be $\sqrt{1^2 + 1^2} = \sqrt{2}$. For the largest possible distance between two vertices in three dimension, it would be calculating the distance between two corner vertices of a cube that is facing diagonal of each other. Thus, our calculation would be $\sqrt{1^2 + 1^2 + 1^2} = \sqrt{3}$. This will also be similar to calculate the max distance between two vertices in two dimension, which would be $\sqrt{1^2 + 1^2 + 1^2 + 1^2} = 2$. Thus, as shown through these calculations, since the max distances between two nodes increases as we increase the dimensions. This means that as the edge weights for the graph on average will also increase between dimensions. Thus, the sum of the edges weights of the minimum spanning tree would also increase as we increase the

dimensions from dim 0 to 2D to 3D to 4D. That is why our growth rates increase as we increase the dimensions.

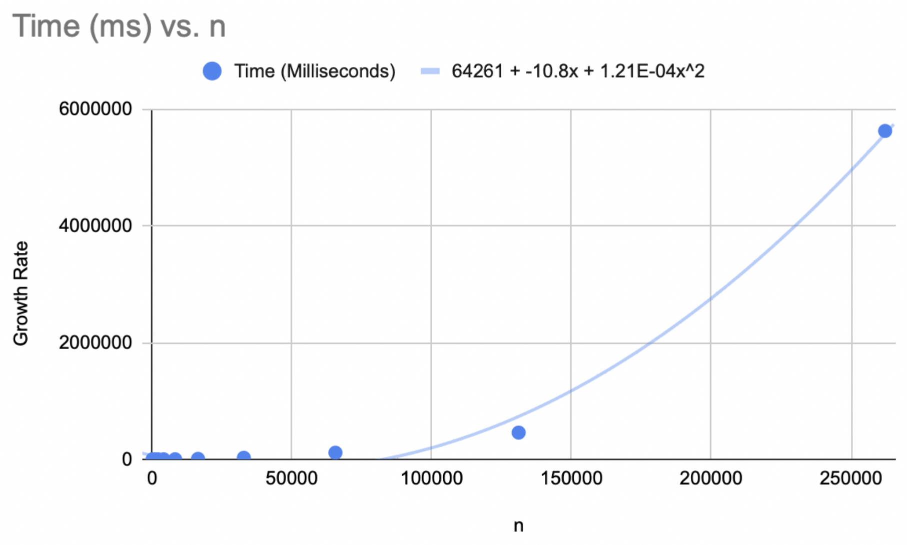
- How long does it take your algorithm to run? Does this make sense? Do you notice things like the cache size of your computer having an effect?
  - Our algorithm will take $O(n^2)$ time. This is due to the fact that since we have n vertices in the graph, we will have $O(n^2)$ edges due to the fact that we have a complete graph. Therefore, there will be $\binom{n}{2}$ edges, which simplifies to $O(n^2)$ edges. Observe that for finding the MST, Prim will iterate through all n vertices in the order that they are added to the MST. For each vertex, we follow Prim's algorithm and update the distances of all vertices to the tree by exploring all of the edges of the current vertex we are on, which takes $O(n)$ time. Next, we will iterate through all of the vertices again and find the one with the minimum distance from the tree, performing an $O(1)$ check to see if it has not been visited, so this step takes a total time of $O(n)$. We then set the minimum distance vertex to be the next one we explore, and add its distance to the total weight, which are all $O(1)$ operations, so we perform a total of $O(n)$ operations for each vertex. Thus, our algorithm has a total asymptotic runtime of $O(n * n) =)(n^2)$. Through plotting how long the algorithm took for each input of n, I saw that each dimension closely matched $O(n^2)$. These graphs can be shown below:
  - **Dimension 0:**



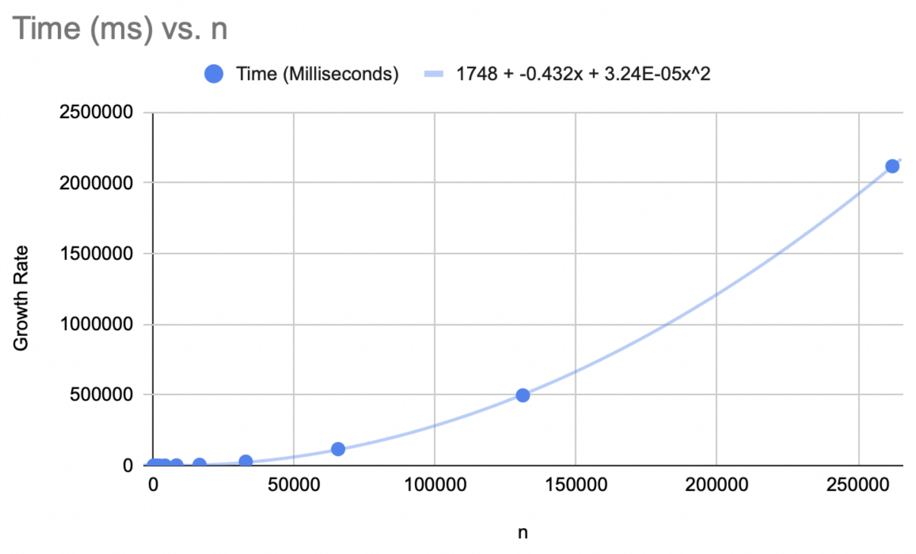
  - **Dimension 2:**



– **Dimension 3:**



– **Dimension 4:**



As shown through all the curve of best fits that are pictured above, we saw that the the growth rate of the time was running in polynomial time (more specifically, around n^2) Therefore, the algorithm's run-time will take around $O(n^2)$ time, which makes sense through testing how long the algorithm take for each input and graphing the times and figuring out the curve of best fit as well as knowing that there are n^2 edges and the algorithm will run through each edge once.

- Did you have any interesting experiences with the random number generator? Do you trust it?
  - For our algorithm, we used the Random class in Java to help us generator random numbers. We did not have any interesting experiences with the random number generator that we chose.
  - We trust the Random class in Java because we know that although this Random class is not truly random, we believe that this class is good enough for this situation. This is due to that fact that the Random class contains features that ensures that there is a random distribution where every value has an equal probability of being generated and that all numbers generated are non-deterministic.

Your grade will be based primarily on the correctness of your program and your discussion of the experiments. Other considerations will include the size of n your program can handle. Please do a careful job of solid writing in your writeup. Length will not earn you a higher grade, but clear descriptions of what you did, why you did it, and what you learned by doing it will go far.