CS 1240 Programming Assignment 1: Random MSTs

Aisha Ahmed

Spring 2026

1 Overview

We implemented minimum spanning tree (MST) algorithms and ran experiments on the random graph models described in the assignment. For each n, we generated multiple independent random instances and recorded the average MST weight. We then fit a simple function $f(n)$ describing how the expected MST weight grows with n.

Program interface: `./randmst 0 numpoints numtrials dimension`

Dimensions:

- Dimension 0: complete graph with i.i.d. weights in $[0, 1]$
- Dimension 1: hypercube graph with i.i.d. weights in $[0, 1]$
- Dimensions 2,3,4: complete graphs on random points in $[0, 1]^d$ with Euclidean weights

2 Algorithms and Implementation

Deterministic Edge Weights (Dimensions 0 and 1)

For dimensions 0 and 1, we do not store edge weights explicitly. Instead, we compute weights on demand using a deterministic hash of (seed, u, v). For a fixed seed, this defines a fixed weighted graph, ensuring consistency within a trial while avoiding storage of all edges.

Dimension 0: Dense Complete Graph

We implemented the standard array-based version of Prim's algorithm.

At each iteration:

1. We scan all vertices not yet in the tree to find the smallest connecting edge.
2. We update the best known connection for every remaining vertex.

Runtime per trial: $O(n^2)$ Memory usage: $O(n)$

Dimension 1: Hypercube Graph

Each vertex v is connected to vertices of the form $v \pm 2^i$ for powers of two $2^i < n$. Thus each vertex has at most $\log_2 n$ neighbors, and the graph contains $\Theta(n \log n)$ edges.

We implemented Prim's algorithm using a custom binary min-heap with decrease-key support. Each vertex appears exactly once in the heap. When a better connecting edge is found, we update its key and restore heap order.

Since each vertex has $O(\log n)$ neighbors and each decrease-key costs $O(\log n)$ time, the total runtime per trial is:

$$O(n(\log n)^2).$$

Dimensions 2–4: Geometric Complete Graphs

We generate n random points in $[0,1]^d$. Distances are computed on demand using the Euclidean metric.

We use the dense array-based version of Prim's algorithm, resulting in:

$$O(n^2)$$

runtime per trial.

3 Experimental Results

Dimension 0

n	trials	average MST weight
128	5	1.13502
256	5	1.17091
512	5	1.22843
1024	5	1.23670
2048	5	1.21850
4096	5	1.19779
8192	5	1.19745
16384	5	1.19904
32768	5	1.20348

Fit: The average converges to a constant near

$$f(n) \approx 1.20.$$

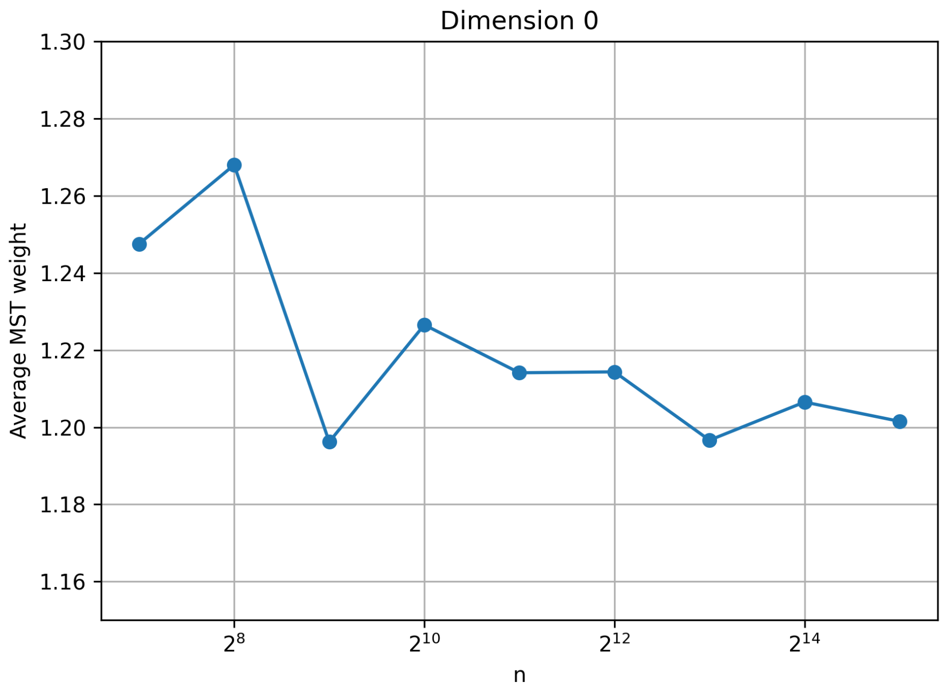

Figure 1: Average MST weight vs. n for Dimension 0 (complete graph with i.i.d. weights).

Dimension 1

n	trials	average MST weight
128	5	12.2081
256	5	21.3285
512	5	37.8052
1024	5	66.3007
2048	5	121.461
4096	5	219.791
8192	5	404.408
16384	5	742.212
32768	5	1376.88
65536	5	2575.18
131072	5	4842.08
262144	5	9102.08

Fit:

$$\frac{9102.08}{262144} \approx 0.0347, \qquad f(n) \approx 0.035n.$$

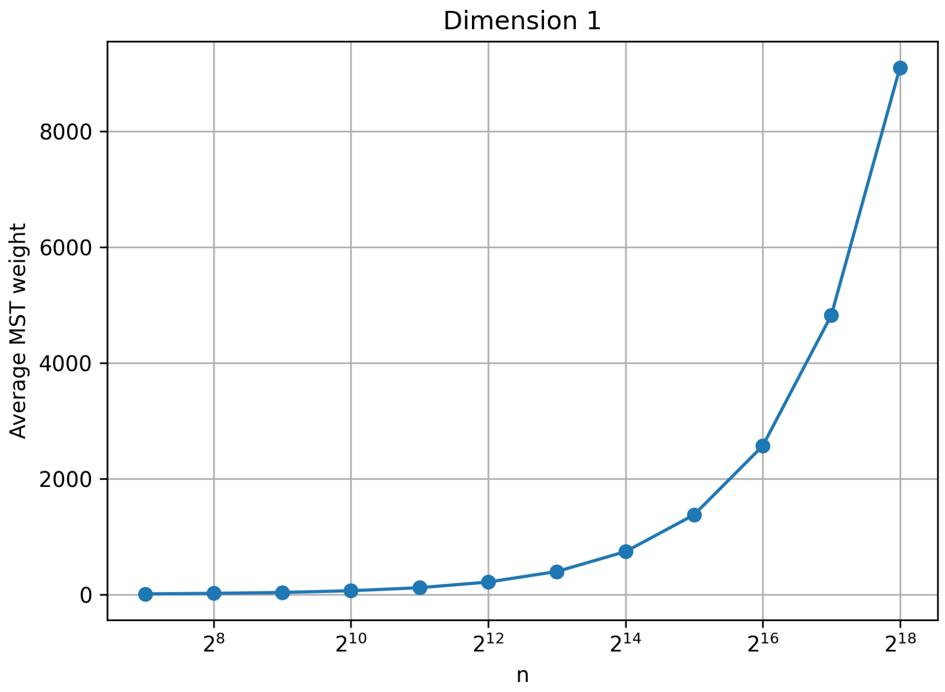

Figure 2: Average MST weight vs. n for Dimension 1 (hypercube graph).

Dimension 2

n	trials	average MST weight
128	5	7.78429
256	5	10.7068
512	5	14.9969
1024	5	21.1132
2048	5	29.6338
4096	5	41.6704

Fit:

$$f(n) \approx 0.66\sqrt{n}.$$

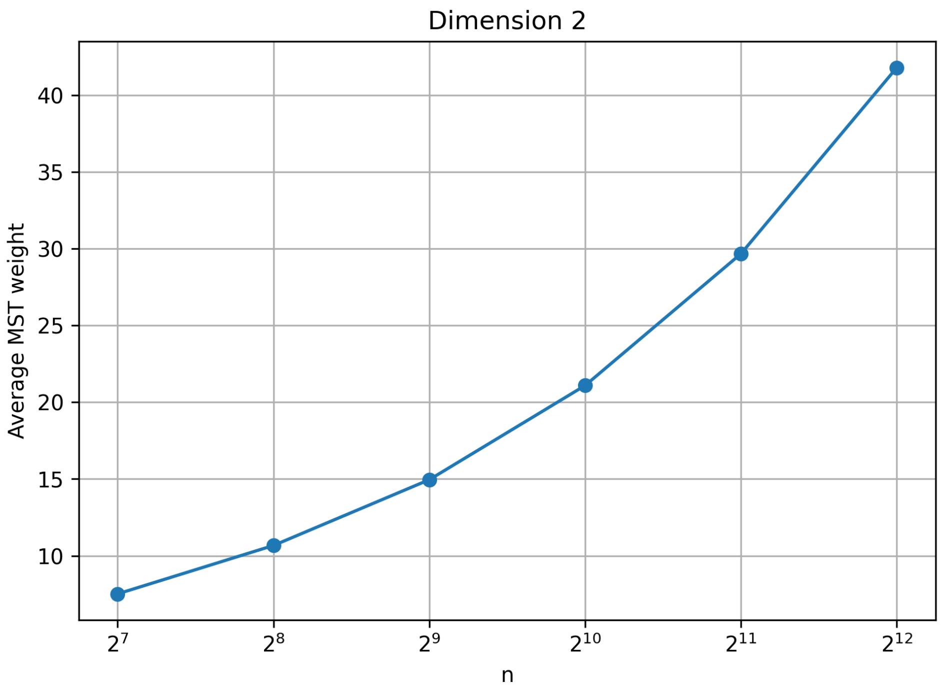

Figure 3: Average MST weight vs. n for Dimension 2 (random geometric graph).

Dimension 3

n	trials	average MST weight
128	5	17.6731
256	5	28.1207
512	5	43.0517
1024	5	67.8171
2048	5	107.185
4096	5	169.016

Fit:

$$f(n) \approx 0.68n^{2/3}.$$

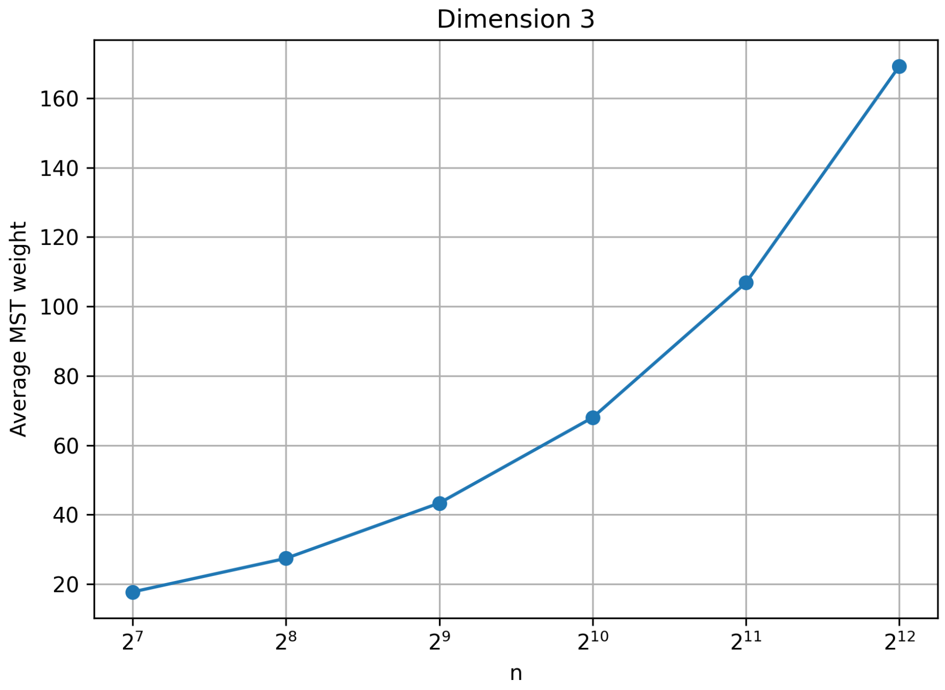

Figure 4: Average MST weight vs. n for Dimension 3 (random geometric graph).

Dimension 4

n	trials	average MST weight
128	5	28.3054
256	5	46.6302
512	5	78.2834
1024	5	129.474
2048	5	216.211
4096	5	361.996

Fit:

$$f(n) \approx 0.73n^{3/4}.$$

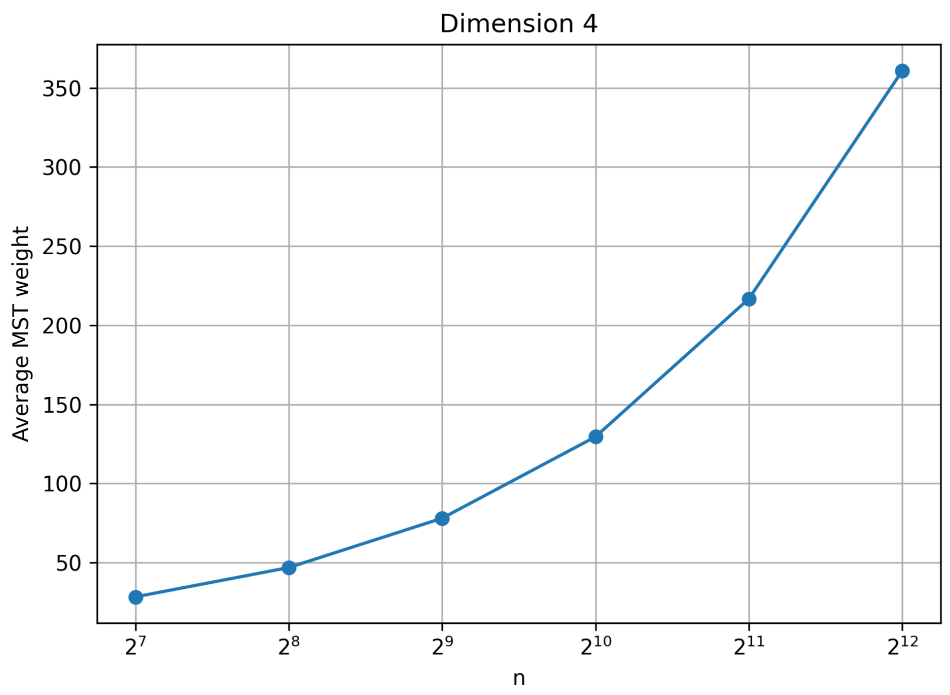

Figure 5: Average MST weight vs. n for Dimension 4 (random geometric graph).

4 Discussion

Runtime and Scalability

The running time of the implementation depends fundamentally on the structure of the underlying graph.

Dense graphs (dimensions 0, 2–4). For dimensions 0, 2, 3, and 4, the graph is complete. A complete graph on n vertices contains

$$\frac{n(n-1)}{2} = \Theta(n^2)$$

edges. Even though we do not store these edges explicitly, the dense structure forces us to consider $O(n)$ potential connections at each step of Prim's algorithm.

In our array-based implementation of Prim's algorithm, each iteration consists of two operations:

1. Scanning all vertices not yet in the tree to find the minimum key value. This requires $O(n)$ time.
2. Updating the best-known connection value for every remaining vertex. This also requires $O(n)$ time.

Since these operations are performed for each of the n vertices added to the tree, the total runtime per trial is

$$O(n^2).$$

This quadratic growth is unavoidable for dense graphs in this implementation model, because each vertex potentially connects to all others.

For geometric graphs (dimensions 2–4), each relaxation requires computing a Euclidean distance. While this increases the constant factor (due to arithmetic operations and a square root), it does not change the asymptotic complexity.

Memory usage in these cases remains $O(n)$, as we store only:

- A boolean array indicating membership in the tree,
- An array of best-known edge weights,
- For geometric graphs, the list of n points.

Hypercube graph (dimension 1). In contrast, the hypercube graph is sparse. A vertex v is connected only to vertices of the form $v \pm 2^i$ for powers of two $2^i < n$. The number of such powers is at most $\lfloor \log_2 n \rfloor + 1$, so each vertex has $O(\log n)$ neighbors.

Thus, the total number of edges is

$$m = \Theta(n \log n),$$

which is asymptotically much smaller than n^2.

For this case, we implemented Prim's algorithm using a custom binary min-heap with decrease-key support. Each vertex appears exactly once in the heap. When a better connecting edge is discovered, we perform a decrease-key operation, restoring heap order in $O(\log n)$ time.

Each vertex has $O(\log n)$ neighbors, so the total number of relaxation attempts is $O(n \log n)$. Since each successful relaxation requires a decrease-key operation costing $O(\log n)$ time, the overall runtime per trial is

$$O(n(\log n)^2).$$

This asymptotic improvement over $O(n^2)$ is what allows the algorithm to scale to $n = 262{,}144$.

Correctness

All minimum spanning trees were computed using Prim's algorithm. We now justify its correctness rigorously.

Prim's algorithm maintains a set $S \subseteq V$ of vertices already included in the growing tree. At each iteration, it selects the minimum-weight edge that crosses the cut

$$(S, V \setminus S).$$

The correctness of this choice follows from the *cut property*:

Cut Property. Let $(S, V \setminus S)$ be any cut of a weighted graph. If e is a minimum-weight edge crossing that cut, then there exists a minimum spanning tree that contains e.

At each step of Prim's algorithm, the selected edge is the minimum-weight edge crossing the current cut. By the cut property, that edge is safe to add — meaning that it can be extended to a full minimum spanning tree.

Since the algorithm adds exactly $n - 1$ edges and never forms a cycle, it constructs a spanning tree. Because every added edge is safe, the resulting tree is a minimum spanning tree.

Deterministic weight generation. For dimensions 0 and 1, edge weights are not stored but computed via a deterministic hash function of (seed, u, v). For a fixed seed, this defines a fixed weighted graph. Therefore, within each trial, Prim's algorithm operates on a well-defined weighted graph and produces its exact MST.

Different trials correspond to different seeds and therefore independent random graphs.

Interpretation of Growth Rates

The experiments reveal distinct asymptotic behaviors that depend on graph structure.

Dimension 0 (complete graph with i.i.d. weights). In a complete graph with independent weights uniformly distributed in $[0, 1]$, the minimum edge incident to each vertex becomes increasingly small as n grows. Classical results in probabilistic combinatorics show that the expected MST weight converges to a constant (specifically, $\zeta(3) \approx 1.202$ for uniform $[0, 1]$ weights).

Our experiments show convergence toward approximately 1.20, consistent with this theoretical prediction.

Dimension 1 (hypercube graph). The hypercube graph is sparse, with only $O(\log n)$ neighbors per vertex. Unlike the complete graph, increasing n does not dramatically increase the number of candidate edges available to reduce edge weights. As a result, the average edge weight in the MST does not shrink rapidly with n.

Since the MST contains $n - 1$ edges and the typical edge weight remains bounded away from zero at a rate proportional to $1/\log n$, the total MST weight grows approximately linearly in n. Empirically, we observe

$$f(n) \approx 0.035n.$$

Geometric graphs (dimensions 2–4). For n random points uniformly distributed in $[0, 1]^d$, the typical nearest-neighbor distance scales on the order of

$$n^{-1/d}.$$

An MST contains $n - 1$ edges, and most edges connect nearby points at roughly this scale. Therefore, the total MST weight scales as

$$(n - 1) \cdot n^{-1/d} = \Theta(n^{(d-1)/d}).$$

This matches the empirical behavior:

$$\sqrt{n} \quad (d = 2), \qquad n^{2/3} \quad (d = 3), \qquad n^{3/4} \quad (d = 4).$$

The experimental constants remain stable as n increases, indicating convergence toward the asymptotic growth rate.