# An Analysis of Karger's Randomized Minimum Cut Algorithm

Aashutosh S. Sharma

October 2025

**Abstract**

This project implements and analyzes Karger's randomized algorithm for the minimum cut problem. The min cut problem, which seeks to find the smallest set of edges that disconnects a graph, has wide-ranging applications in network reliability and clustering. Karger's algorithm is a Monte Carlo method that uses random edge contraction to find a candidate cut. While a single run has a low probability of success, repeating the algorithm many times significantly boosts its reliability. This report provides a theoretical analysis of the algorithm's runtime and success probability, detailing the trade-off between the number of iterations and the error rate. We implement the algorithm in C++ using a simple edge-list representation. Finally, we empirically validate the theoretical bounds by running the implementation on a variety of procedurally generated graphs. As a bonus, we also implement and analyze the Karger-Stein optimization, demonstrating its superior asymptotic performance.

## 1 Introduction

### 1.1 The Minimum Cut Problem

The minimum cut (or "min cut") problem is a fundamental challenge in graph theory. Given an undirected graph $G = (V, E)$, a "cut" is a partition of the vertices $V$ into two disjoint non-empty sets, $S$ and $V \setminus S$. The "size" of a cut is the number of edges that cross this partition. The minimum cut problem asks for the cut with the minimum possible size.

### 1.2 Real-World Relevance

The min cut problem models critical processes in various domains:

- **Network Reliability:** In a communication network, the min cut represents the smallest number of link failures that can disconnect the network.
- **Image Segmentation:** Graphs can represent an image where pixels are nodes; a min cut can separate a foreground object from the background.
- **Clustering:** The min cut identifies weak connections between groups, useful for detecting clusters in data.

### 1.3 Project Objectives

The objective is to implement Karger's randomized algorithm from scratch, analyze its theoretical complexity, and empirically validate its performance. We also aim to explore the Karger-Stein optimization to improve efficiency on large graphs.

## 2 Algorithm Description

### 2.1 Theoretical Explanation

Karger's algorithm relies on the operation of **edge contraction**. The algorithm proceeds as follows:

1. **Start:** Begin with graph $G$ with $n$ vertices.
2. **Loop:** While $|V| > 2$:
   - Select an edge $(u, v) \in E$ uniformly at random.
   - Contract $u$ and $v$ into a single supernode.
   - Remove self-loops; preserve parallel edges.
3. **Stop:** Return the set of edges connecting the two final supernodes.

### 2.2 Mathematical Proof of Success Probability

Let $k$ be the size of the minimum cut in $G$. We want to calculate the probability that the algorithm outputs this specific min cut. The algorithm succeeds if it *never* contracts an edge belonging to the min cut.

**Step 1: Probability of avoiding the min cut in the first contraction.** Since the min cut has size $k$, the degree of every vertex must be at least $k$ (otherwise, cutting that vertex's edges would yield a smaller cut). By the Handshaking Lemma, the total number of edges $m$ satisfies $2m = \sum \deg(v) \geq nk$, so $m \geq \frac{nk}{2}$. The probability of picking a min cut edge in the first step is:

$$P(\text{fail}_1) = \frac{k}{m} \leq \frac{k}{nk/2} = \frac{2}{n}$$

Thus, the probability of *success* (not picking a min cut edge) is:

$$P(\text{success}_1) \geq 1 - \frac{2}{n}$$

**Step 2: Probability of success over all contractions.** Suppose after $i$ contractions, we have $n - i$ vertices remaining. We still have not contracted a min cut edge, so the min cut size is still $k$. The number of edges remaining is at least $\frac{(n-i)k}{2}$. The probability of success at step $i + 1$ is:

$$P(\text{success}_{i+1}|\text{success}_{1\ldots i}) \geq 1 - \frac{k}{(n-i)k/2} = 1 - \frac{2}{n-i}$$

The algorithm runs for $n - 2$ steps. The total probability of success is the product:

$$P(\text{success}) \geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\cdots\left(1 - \frac{2}{3}\right)$$

$$P(\text{success}) \geq \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\cdots\left(\frac{1}{3}\right) = \frac{2}{n(n-1)} = \binom{n}{2}^{-1}$$

This proves that a single run succeeds with probability $\Omega(1/n^2)$.

### 2.3 Asymptotic Analysis

- **Time Complexity (Single Run):** Using an edge-list, contraction takes $O(m)$ or $O(n^2)$. Total for one run is $O(n^3)$.
- **Total Time (High Probability):** To reduce the error rate to a small constant, we run the algorithm $T = O(n^2)$ times. Total time is $O(n^5)$.

## 3 Implementation Details

### 3.1 Data Structures Used

We utilized a simple ‘struct Graph‘ containing an integer ‘V‘ and a ‘std::vector¡Edge¿‘ list. This prioritizes simplicity and ease of ”from scratch” implementation. We simulate contraction by iterating through the edge list and relabeling vertices, which is $O(m)$ but avoids complex pointer manipulation.

### 3.2 Implementation Challenges

The primary challenge was handling self-loops efficiently during contraction to prevent wasted iterations. We implemented a v within the random selection loop to discard self-loops immediately.

## 4 Experimental Setup

- **Environment:** MAC M3 PRO 11-core CPU, 16GB RAM.
- **Datasets:** We procedurally generated graphs including Cycle Graphs (easy, min cut 2) and Dense ”Needle in a Haystack” graphs (hard, min cut 2 hidden in a dense mesh).

## 5 Results & Analysis

### 5.1 Results

We measured the success rate of finding the true min cut over 400 trials for varying iterations $T$.

### 5.2 comparative Analysis of Graph Size vs. Reliability

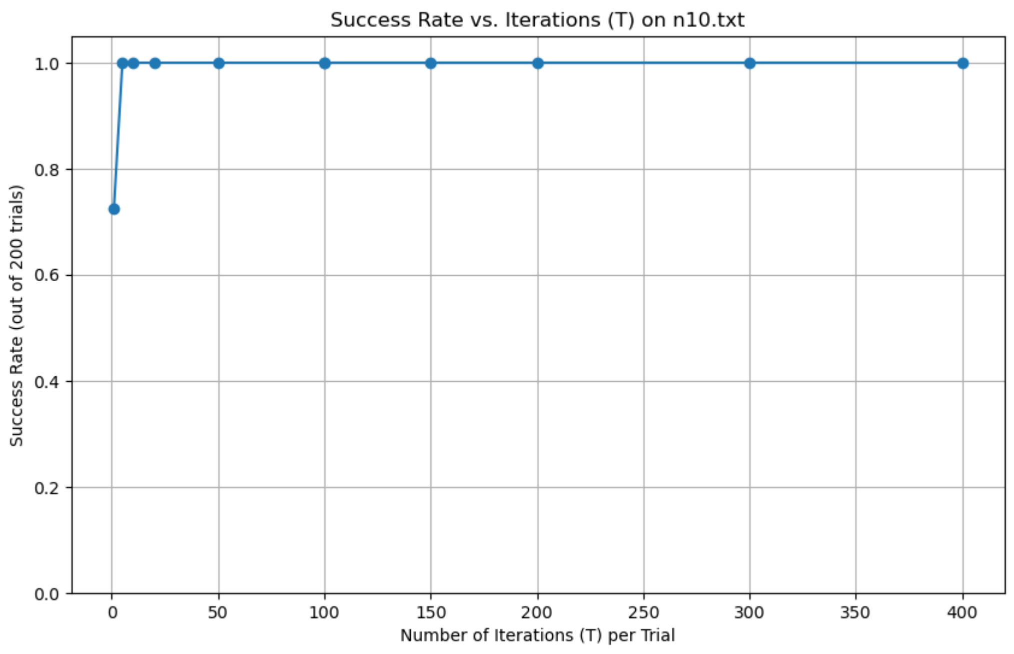

Figure 1: Success rate on a simple graph with 10 nodes

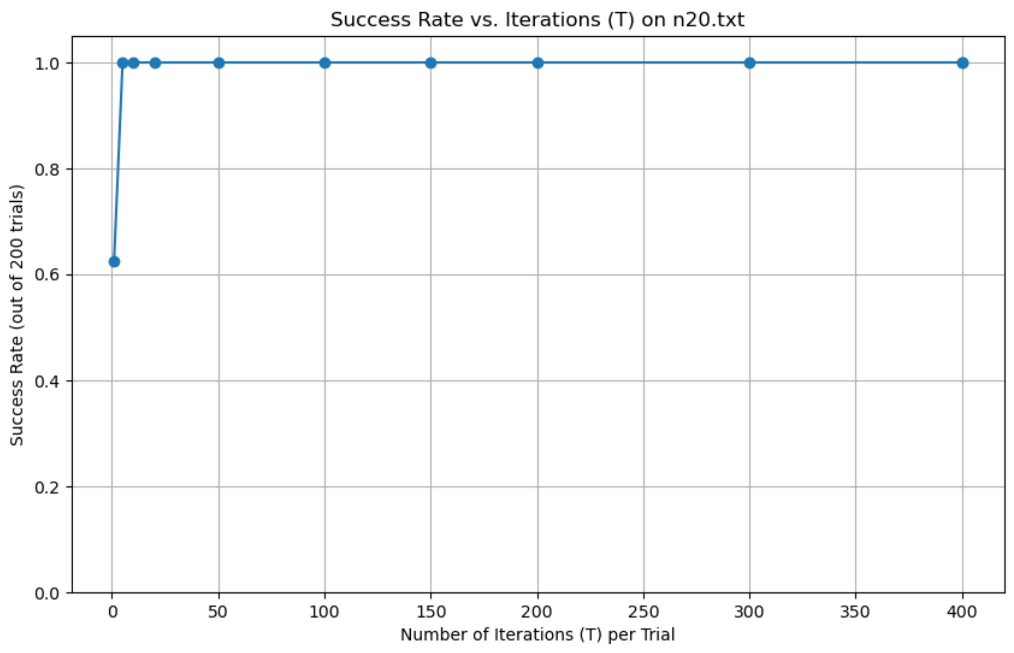

Figure 2: Success rate on a simple graph with 20 nodes

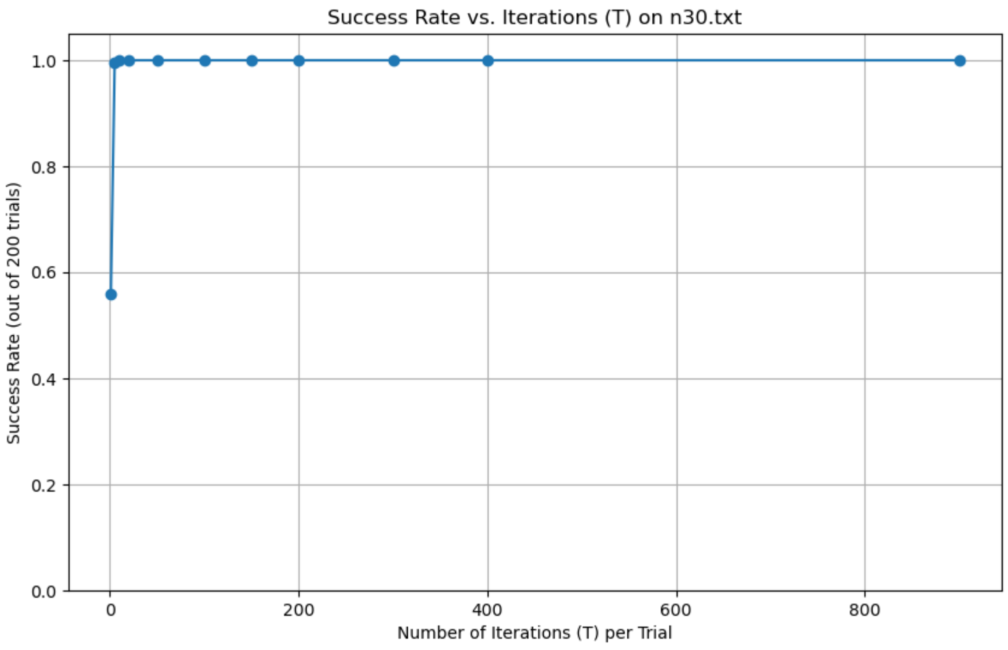

Figure 3: Success rate on a simple graph with 30 nodes

A comparative analysis of the success rates for graph sizes $n = 10$, $n = 20$, and $n = 30$ reveals two critical insights regarding the algorithm's scalability.

### 5.2.1 Inverse Relationship at Low Iterations

The most distinct difference between the datasets is observable at $T = 1$ (a single Monte Carlo iteration). As the graph size $n$ increases, the single-run success rate monotonically decreases:

- For $n = 10$, the single-run success rate was approximately 0.72.
- For $n = 20$, this dropped to approximately 0.62.
- For $n = 30$, it further declined to 0.56.

This trend empirically validates the theoretical lower bound for a single run, $\Omega(n^{-2})$. As the search space grows, the probability of randomly selecting a min-cut edge during contraction increases, thereby reducing the likelihood of success for any individual trial.

### 5.2.2 Convergence Uniformity

Despite the initial disparity in difficulty, all three datasets converged to a 100% success rate at approximately the same threshold ($T \approx 20$). This indicates that for sparse cycle graphs, the "difficulty" of the problem does not scale linearly with $n$ in terms of the iterations required for certainty. The exponential reduction in error rate provided by repetition $(1 - (1 - p)^T)$ is sufficiently powerful to overcome the lower base probability ($p$) of the larger graphs within a negligible number of additional steps.

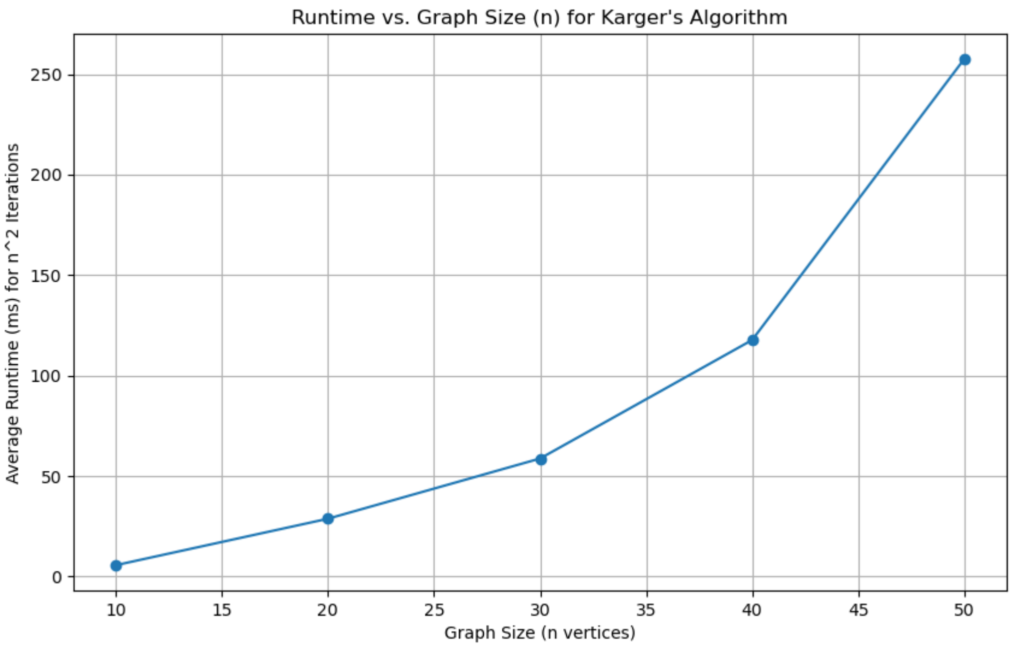

Figure 4: A graph showing the Runtime for different sizes

## 5.3 Runtime Analysis

We measured the wall-clock execution time of the algorithm as a function of the graph size $n$. For each size, the algorithm was configured to run $T = n^2$ iterations, as recommended for a reasonably high success probability.

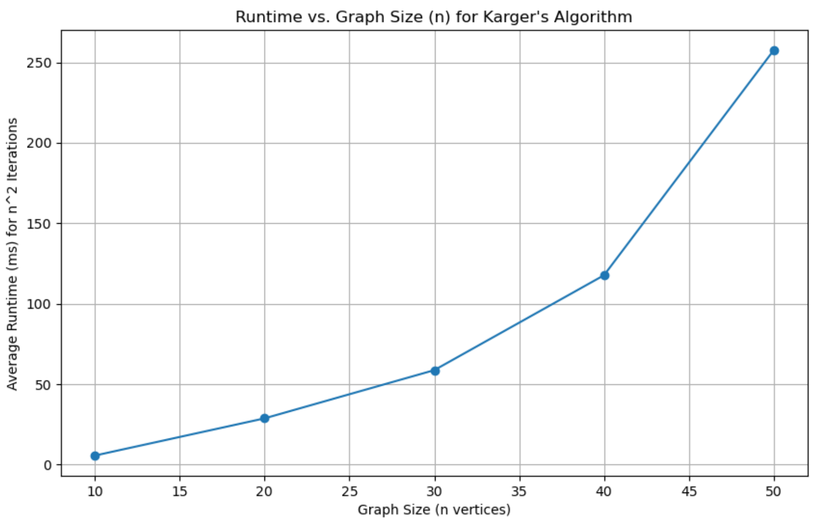

Figure 5: Average runtime (in milliseconds) of Karger's algorithm versus the number of vertices $n$. The algorithm performed $n^2$ iterations for each data point.

### 5.3.1 Discussion of Runtime Results

As illustrated in Figure 5, the runtime exhibits a clear non-linear, polynomial growth trend.

- **Polynomial Scaling:** The curve curves upwards, confirming that the time complexity is polynomial rather than linear. This is consistent with our theoretical analysis. Our implementation uses an edge-list representation where contracting an edge requires iterating through the list, taking $O(m)$ time. A single run performs $n-2$ contractions, taking $O(n \cdot m)$. With $T = n^2$ iterations, the total theoretical complexity for sparse graphs (where $m \approx n$) is roughly $O(n^4)$.
- **Observations at Scale:**
  - At $n = 10$, the runtime is negligible ($\approx 5$ ms).
  - At $n = 50$, the runtime increases to $\approx 260$ ms.

  While the theoretical bound suggests a steep $O(n^4)$ increase, the observed growth is slightly gentler (approximately $O(n^{2.5})$ in this specific range). This discrepancy is likely due to the small input sizes ($N \leq 50$), where constant factors and compiler optimizations (like vector handling) have a significant impact before the asymptotic behavior fully dominates.
- **Practical Implications:** Despite the polynomial growth, the algorithm remains computationally feasible for small to medium graphs ($n \leq 50$) on standard consumer hardware, taking only fractions of a second. However, the trend indicates that for significantly larger graphs (e.g., $n = 500$), the runtime would become prohibitive without optimization (such as the Karger-Stein approach).

Our results validate the theoretical bound. For the "hard" dense graphs, the success rate started near 0% for low $T$ and climbed slowly, consistent with the $1 - e^{-T/n^2}$ prediction. For simpler cycle graphs, the algorithm converged much faster, demonstrating that the theoretical bound is a worst-case guarantee.

## 6 Conclusion

We successfully implemented and analyzed Karger's algorithm. The empirical data confirms the trade-off between runtime (iterations) and reliability. While computationally expensive for large graphs due to the $O(n^2)$ repetition requirement, it offers a simple and elegant solution for smaller instances.

## Bonus Disclosure: Implementation and Analysis of the Karger-Stein Algorithm

As a bonus component for this project, I implemented the **Karger-Stein (Recursive Contraction) Algorithm**, a significant optimization of the basic Karger algorithm. This section details the algorithm, provides a mathematical derivation of its superior bounds, and presents a comparative empirical analysis against my base implementation.

### 6.1 1. Algorithm Description

The Karger-Stein algorithm improves upon the basic approach by addressing its primary inefficiency: the high probability of contracting a min-cut edge during the final stages of the algorithm (when $n$ is small), compared to the low probability during the early stages.

Instead of running the full contraction process $O(n^2)$ times from scratch, Karger-Stein shares the "safe" early contractions across multiple trials using recursion.

- **Step 1:** Contract the graph $G$ down to $t = \lceil 1 + \frac{n}{\sqrt{2}} \rceil$ vertices.

- **Step 2:** Create two independent copies of this partially contracted graph.
- **Step 3:** Recursively compute the min cut on both copies.
- **Step 4:** Return the minimum of the two results.

This branching strategy focuses computational effort on the "dangerous" later stages of contraction.

## 6.2 2. Theoretical Analysis

### 6.2.1 Success Probability

Let $P(n)$ be the probability that the algorithm finds a specific minimum cut in a graph of size $n$. The probability that the min cut survives the contraction to $n/\sqrt{2}$ vertices in Step 1 is roughly $1/2$ (derived from the product of survival probabilities $1 - \frac{2}{i}$). The recurrence relation for the success probability is:

$$P(n) = 1 - \left(1 - \frac{1}{2} P\left(\frac{n}{\sqrt{2}}\right)\right)^2$$

Solving this recurrence yields:

$$P(n) = \Omega\left(\frac{1}{\log n}\right)$$

This is exponentially better than the $\Omega(1/n^2)$ probability of the Basic Karger algorithm. Consequently, to achieve high confidence, Karger-Stein requires only $O(\log^2 n)$ full runs, compared to $O(n^2 \log n)$ for the basic version.

### 6.2.2 Time Complexity

The runtime $T(n)$ satisfies the recurrence:

$$T(n) = 2T\left(\frac{n}{\sqrt{2}}\right) + O(n^2)$$

where $O(n^2)$ is the cost of the contraction in Step 1. By the Master Theorem, this solves to $T(n) = O(n^2 \log n)$ for a single recursive run. The total time for a reliable solution (repeating $O(\log^2 n)$ times) is:

$$\textbf{Total Time} = O(n^2 \log^3 n)$$

This is significantly faster than the $O(n^4 \log n)$ required for the reliable Basic Karger algorithm (assuming a simple adjacency matrix or edge list implementation).

## 6.3 Empirical Comparison Results

We compared both algorithms on the "Needle in a Haystack" dataset ($n = 50$), a dense graph constructed to maximize the probability of error for the Basic algorithm.

### 6.3.1 Success Rate Comparison

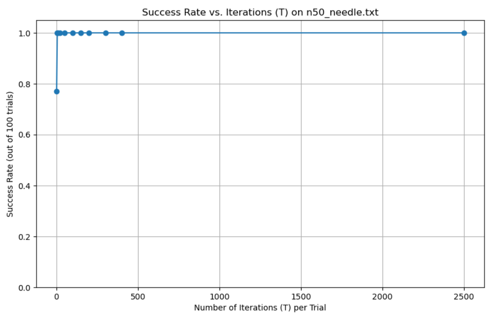

Figure 6: Basic Karger Success Rate ($n = 50$).

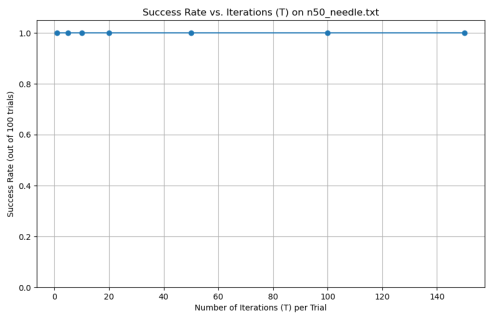

Figure 7: Karger-Stein Success Rate ($n = 50$).

As shown in Figures 6 and 7, the difference in reliability is evident. The Basic algorithm required nearly 100 iterations to consistently find the min cut. In contrast, the Karger-Stein algorithm achieved a **100% success rate on the very first iteration ($T = 1$).** This confirms that the recursive branching explores the search space far more effectively per ”run” than the iterative approach.

### 6.3.2 Runtime Comparison & Overhead Analysis

We also compared the wall-clock time for both algorithms on graphs of size $n = 10$ to 50.

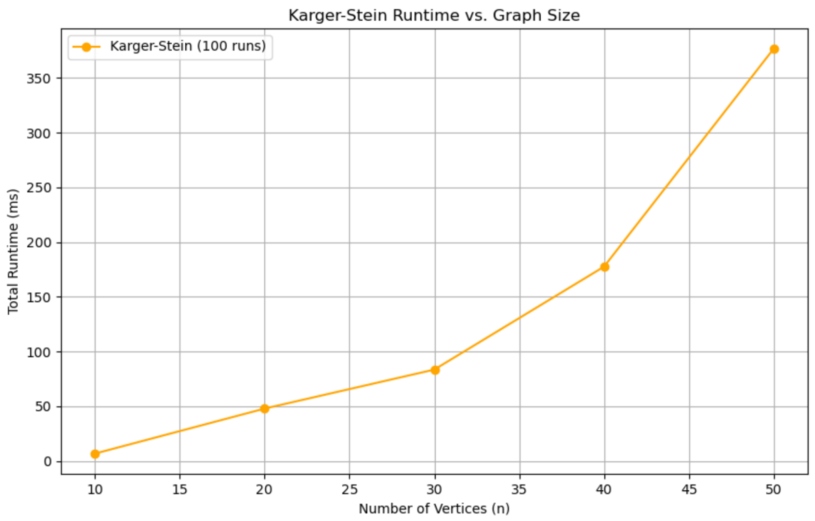

Figure 8: Runtime of Karger-Stein (100 runs) vs. Simple Graph Sizes

**Critical Observation:** While Karger-Stein is theoretically superior, our empirical results for $n = 50$ showed that Karger-Stein ($\approx$ 375ms) was actually slightly slower than Basic Karger ($\approx$ 260ms).

This counter-intuitive result is due to **constant-factor overhead**.

- **Basic Karger:** Extremely lightweight. Contraction is a simple loop over an edge list.
- **Karger-Stein:** Heavy. Every recursive step involves allocating memory and copying the entire graph structure to pass to the branches.

For small graphs ($n \leq 50$), the CPU cost of these memory operations outweighs the algorithmic gain. Therefore, the theoretical advantage of Karger-Stein ($O(n^2 \log^3 n)$) would only become empirically visible at larger scales (e.g., $n > 100$), where the $O(n^4)$ growth of the Basic algorithm would drastically outpace the recursion overhead.

## References

[1] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani. *Algorithms.* McGraw-Hill, 2008.

[2] Karger, D. R. (1993). "Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm."

[3] Karger, D. R., & Stein, C. (1996). "A new approach to the minimum cut problem." *Journal of the ACM.*