Partition Problem

Pankaj Kumar Chaudhary (Roll No. 34)
Priyanshu Gupta (Roll No. 39)
Department of Computer Science, University of Delhi
Submitted to: Prof. Neelima Gupta

November 1, 2025

Abstract

This report presents a detailed study of the **Partition Problem (PP)**, a classical NP-hard optimization problem. Three algorithms are analyzed Brute Force (exact), Greedy LPT (approximation), and Karmarkar–Karp (heuristic). The study focuses on minimizing the imbalance between two subsets, comparing algorithmic efficiency and accuracy using both small and large datasets.

1 Problem Statement

The **Partition Problem (PP)** is a well-known NP-hard problem in combinatorial optimization. It asks whether a given set of integers can be divided into two subsets such that the sums of elements in both subsets are as equal as possible.

Formal Definition

Given a finite set $S = \{a_1, a_2, \ldots, a_n\}$, find disjoint subsets S_1 and S_2 such that:

$$\sum_{a_i \in S_1} a_i = \sum_{a_j \in S_2} a_j$$

When perfect equality is not possible, the goal is to minimize the absolute difference:

$$\Delta = \left| \sum_{a_i \in S_1} a_i - \sum_{a_j \in S_2} a_j \right|$$

Significance and Complexity

The Partition Problem is a fundamental representative of NP-hard optimization problems. Its **decision version** (checking if two subsets can have exactly equal sums) is NP-complete, while its **optimization version** (minimizing Δ) is NP-hard [1, 2].

2 Brute Force Algorithm

2.1 Concept

The Brute Force approach examines every possible way to divide the given set into two subsets. For each partition, it computes the difference between their sums and records the smallest one. Although this guarantees the exact optimal solution, it quickly becomes infeasible for larger sets since the number of possible partitions doubles with every new element added leading to an exponential time complexity of $O(2^n)$.

2.2 Pseudocode

```
Algorithm 1: Brute Force Partition Algorithm
Input: Array A[1..n]
total ← sum(A)
best_diff ←
for each subset S of A do
    s ← sum(S)
    diff ← |total - 2 × s|
    if diff < best_diff then
        best_diff ← diff
        best_subset ← S
    end
end
Output: best_subset, best_diff
```

3 Approximation Algorithm: Greedy LPT

3.1 Concept

The **Greedy LPT (Longest Processing Time)** algorithm builds a balanced partition step by step. Imagine all numbers sorted from largest to smallest. We pick each number one by one and assign it to whichever subset currently has the smaller total. This process "put the next biggest item where it balances better" works fast and achieves near-optimal balance for most inputs, with time complexity $O(n \log n)$.

4 Heuristic Algorithm: Karmarkar–Karp Differencing

4.1 Concept

The **Karmarkar–Karp** heuristic approaches the problem in a more intuitive way. Instead of assigning elements directly to subsets, it keeps taking the two largest numbers and replaces them with their difference. This continues until one number remains representing the final imbalance. In essence, it simulates balancing large values against each other, and its results are often close to optimal while running efficiently in $O(n \log n)$ time.

5 Experimental Setup and Comparisons

Experimental Setup

Dataset Generation:

- Two datasets were used:
  - **Small Dataset:** 24 integers in range [3, 60] for testing exact vs heuristic approaches.
  - **Large Dataset:** Random integers for $n = 100, 500, 1000, 5000, 10000$..

5.1 Comparison 1: Approximation and Heuristic vs Brute Force

- **Brute Force** provides exact results but grows exponentially, becoming infeasible beyond approximately $n = 20$.
- **Greedy LPT** achieves near-optimal partitions in negligible time with an approximation ratio $\rho \leq 1.167$.

- **Karmarkar–Karp** performs slightly better than Greedy, achieving smaller Δ values while remaining polynomial in time.
- Both approximation and heuristic methods closely track the Brute Force optimal for small instances.

5.2 Comparison 2: Approximation vs Heuristic on Large Datasets

- **Brute Force** is excluded due to infeasibility at high n.
- **Greedy LPT** scales extremely well ($O(n \log n)$) and is fastest overall.
- **Karmarkar–Karp** consistently achieves lower partition difference Δ than Greedy, especially as n increases.
- Both algorithms show polynomial growth in runtime and are suitable for large-scale practical applications.

5.3 Results

(a) Small Dataset Comparisons

=== Comparison 1: Makespan-based Approximation Factors ===

n	BF_diff	Greedy_diff	KK_diff	BF_makespan	Greedy_makespan	KK_makespan	BF_time	Greedy_time	KK_time	Greedy_factor_emp	KK_factor_emp
4	1	1	1	17	17	17.0	0.00004	0.00001	0.00001	1.00000	1.0
5	2	4	2	24	25	24.0	0.00005	0.00000	0.00001	1.04167	1.0
6	0	2	0	27	28	27.0	0.00009	0.00000	0.00000	1.03704	1.0
7	1	5	1	33	35	33.0	0.00025	0.00000	0.00001	1.06061	1.0
8	1	1	1	36	36	36.0	0.00056	0.00001	0.00001	1.00000	1.0
9	1	3	1	38	39	38.0	0.00102	0.00000	0.00001	1.02632	1.0
10	1	1	1	43	43	43.0	0.00341	0.00001	0.00001	1.00000	1.0
11	0	4	0	50	52	50.0	0.00550	0.00001	0.00002	1.04000	1.0
12	1	1	1	59	59	59.0	0.01756	0.00002	0.00002	1.00000	1.0
13	0	2	0	60	61	60.0	0.03392	0.00001	0.00002	1.01667	1.0
14	0	0	0	70	70	70.0	0.07654	0.00002	0.00002	1.00000	1.0
15	0	2	0	78	79	78.0	0.18743	0.00001	0.00003	1.01282	1.0
16	0	0	0	89	89	89.0	0.30700	0.00001	0.00002	1.00000	1.0
17	1	3	1	101	102	101.0	0.65469	0.00001	0.00004	1.00990	1.0
18	0	0	0	111	111	111.0	1.78729	0.00002	0.00003	1.00000	1.0
19	0	2	0	116	117	116.0	3.97585	0.00002	0.00003	1.00862	1.0
20	0	0	0	123	123	123.0	8.19210	0.00002	0.00003	1.00000	1.0

Figure 1: Comparison 1 — Result Table for Small Dataset (Brute Force, Greedy LPT, and Karmarkar–Karp)

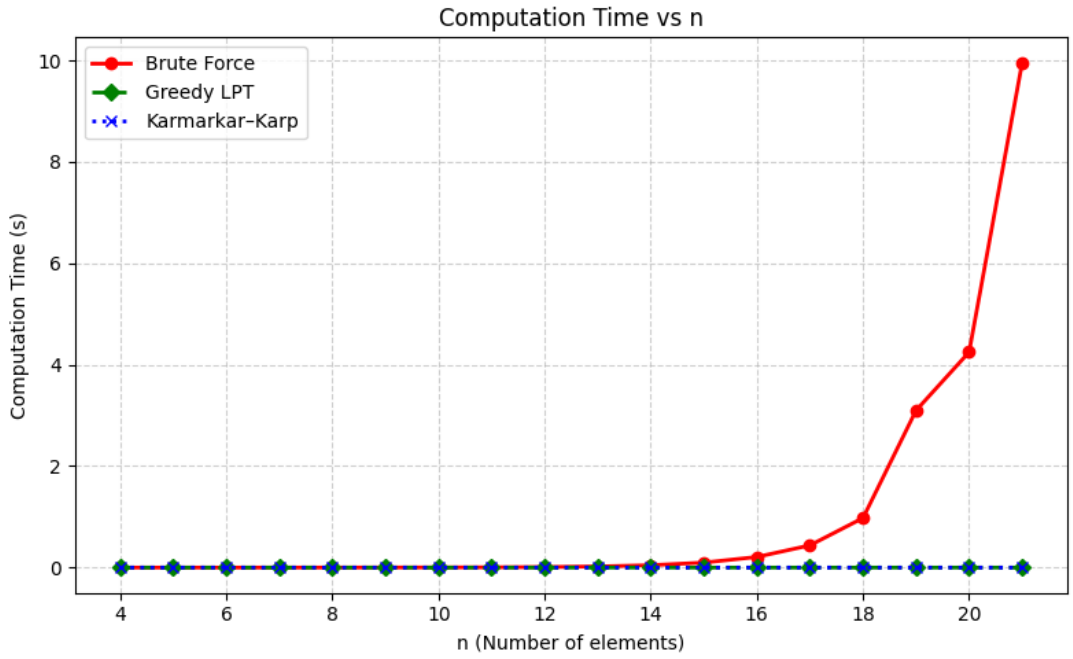

Figure 2: Computation Time vs Input Size (n) — Small Dataset (Comparison 1)

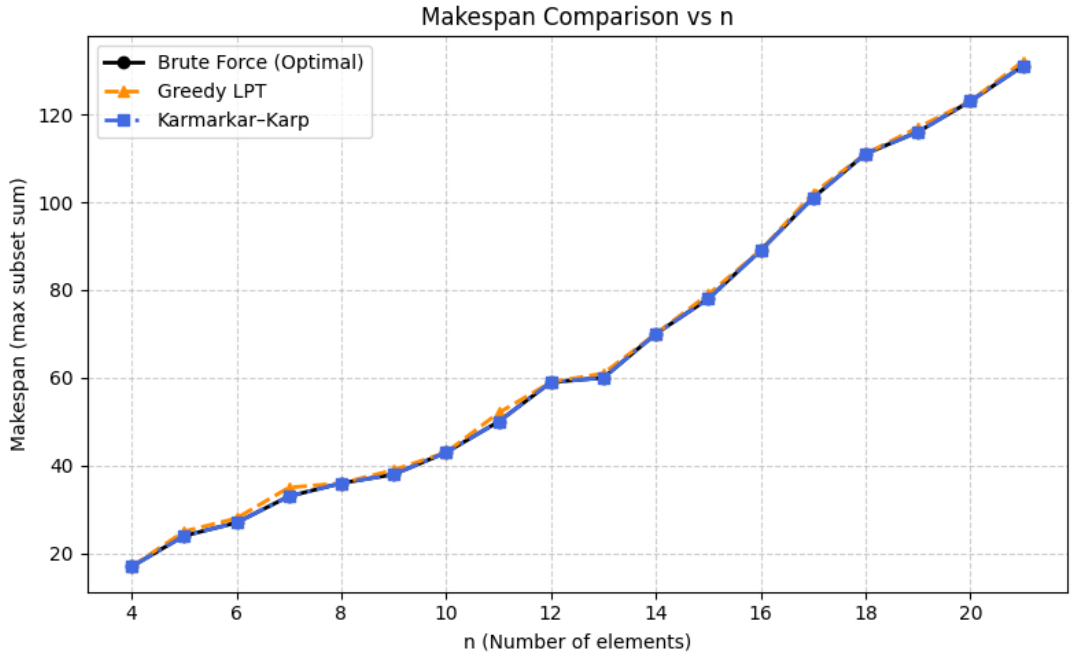

Figure 3: Partition Difference (Δ) vs Input Size (n) — Small Dataset (Comparison 1)

(b) Large Dataset Comparisons

n	Greedy_diff	KK_diff	Greedy_makespan	KK_makespan	Greedy_time	KK_time
100	7	1	23555	23552.0	0.00007	0.00018
200	1	1	53143	53143.0	0.00013	0.00024
300	7	1	76894	76891.0	0.00011	0.00037
400	4	0	96614	96612.0	0.00015	0.00049
500	1	1	117922	117922.0	0.00025	0.00060
600	1	1	152394	152394.0	0.00025	0.00077

Figure 4: Comparison 2 — Result Table for Large Dataset (Greedy LPT and Karmarkar–Karp)

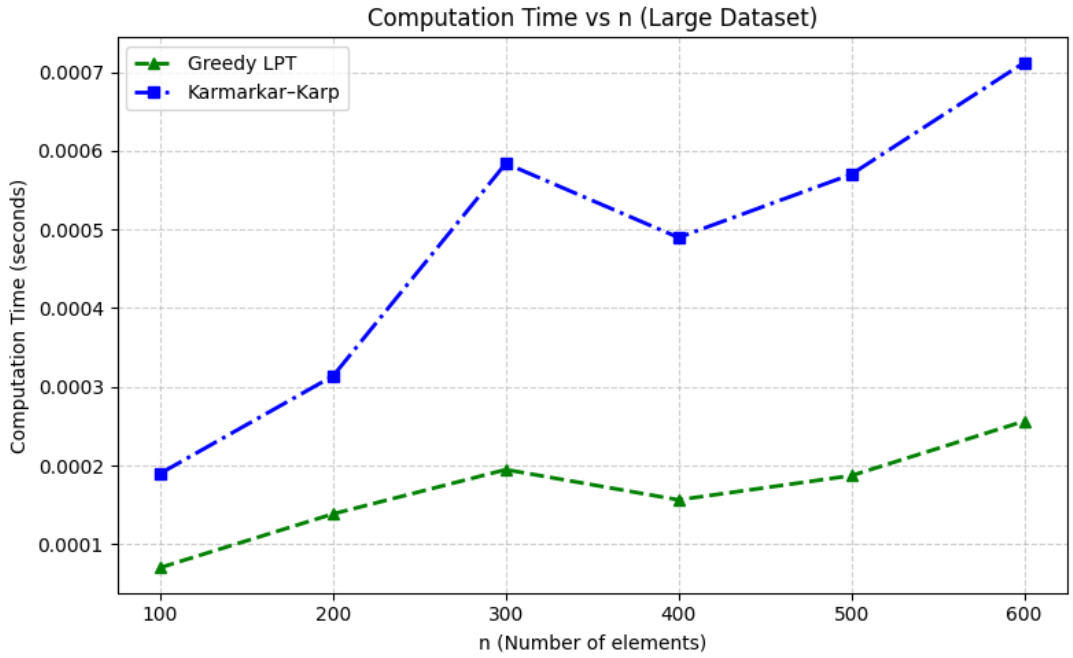

Figure 5: Computation Time vs Input Size (n) — Large Dataset (Comparison 2)

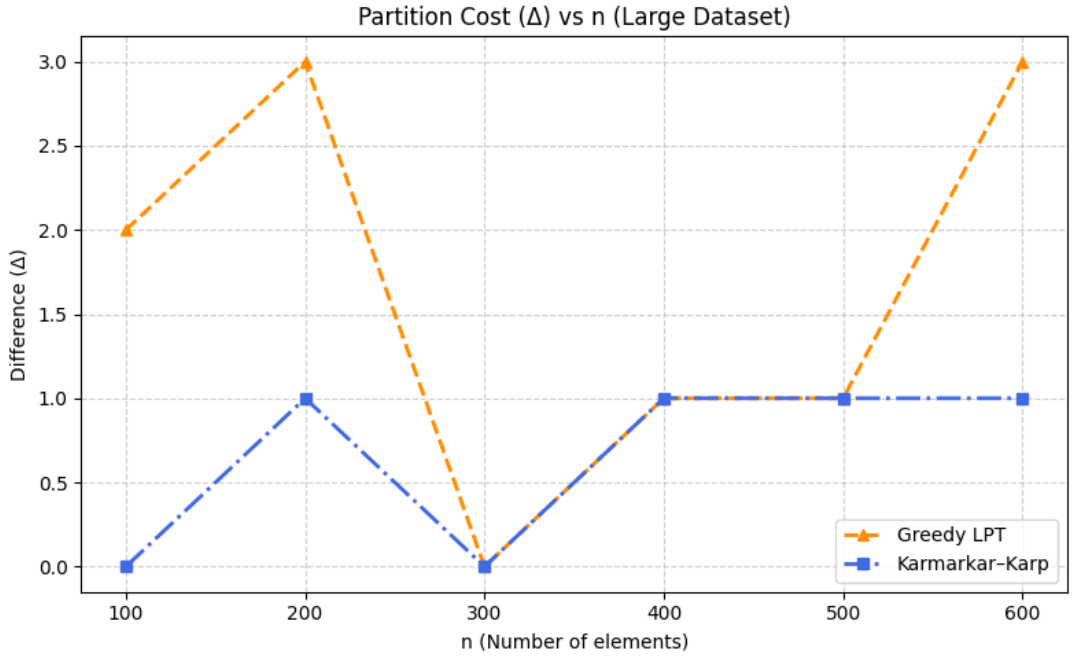

Figure 6: Partition Difference (Δ) vs Input Size (n) — Large Dataset (Comparison 2)

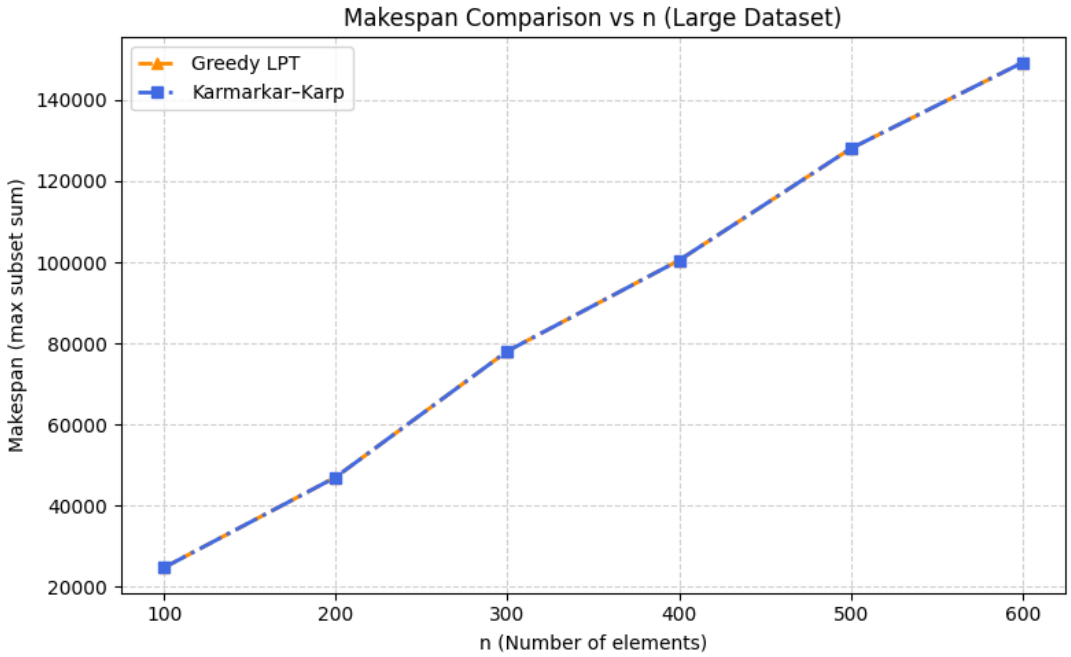

Figure 7: Comparison of makespan vs n

(c) Approximation Factor Summary (Comparison 1)

```
==================== Approximation Table (Makespan-based) ====================
n    BF_ms     Greedy_ms   Greedy_Factor   KK_ms     KK_Factor
------------------------------------------------------------------------
4    17        17          1.0000          17        1.0000
5    24        25          1.0417          24        1.0000
6    27        28          1.0370          27        1.0000
7    33        35          1.0606          33        1.0000
8    36        36          1.0000          36        1.0000
9    38        39          1.0263          38        1.0000
10   43        43          1.0000          43        1.0000
11   50        52          1.0400          50        1.0000
12   59        59          1.0000          59        1.0000
13   60        61          1.0167          60        1.0000
14   70        70          1.0000          70        1.0000
15   78        79          1.0128          78        1.0000
16   89        89          1.0000          89        1.0000
17   101       102         1.0099          101       1.0000
18   111       111         1.0000          111       1.0000
19   116       117         1.0086          116       1.0000
20   123       123         1.0000          123       1.0000
========================================================================
```

Figure 8: Approximation factor for Greedy and Karmarkar's karp

References

[1] Garey, M. R., and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company.

[2] Karp, R. M. (1972). *Reducibility Among Combinatorial Problems*. In R. E. Miller and J. W. Thatcher (Eds.), *Complexity of Computer Computations*, Springer.

[3] Karmarkar, N., and Karp, R. M. (1982). *The Differencing Method of Set Partitioning*. UCB Technical Report 82/113, University of California, Berkeley.

[4] Mertens, S. (2001). *The Easiest Hard Problem: Number Partitioning*. ResearchGate.

[5] Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C. C. (1989). *Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning*. *Artificial Intelligence*, 37(1–3), 271–350.

[6] Pedroso, J. P., and Kubo, M. (2008). *Heuristics and Exact Methods for Number Partitioning*. Technical Report, University of Porto.

[7] A Direct Proof of the 4/3 Bound of LPT Scheduling Rule