Lab 2

Activity Selection: Recursive vs Iterative

The Activity Selection problem involves selecting the maximum number of activities that don't overlap, given their start and finish times. Here, we compare the recursive and iterative approaches for solving this problem.

How To Run

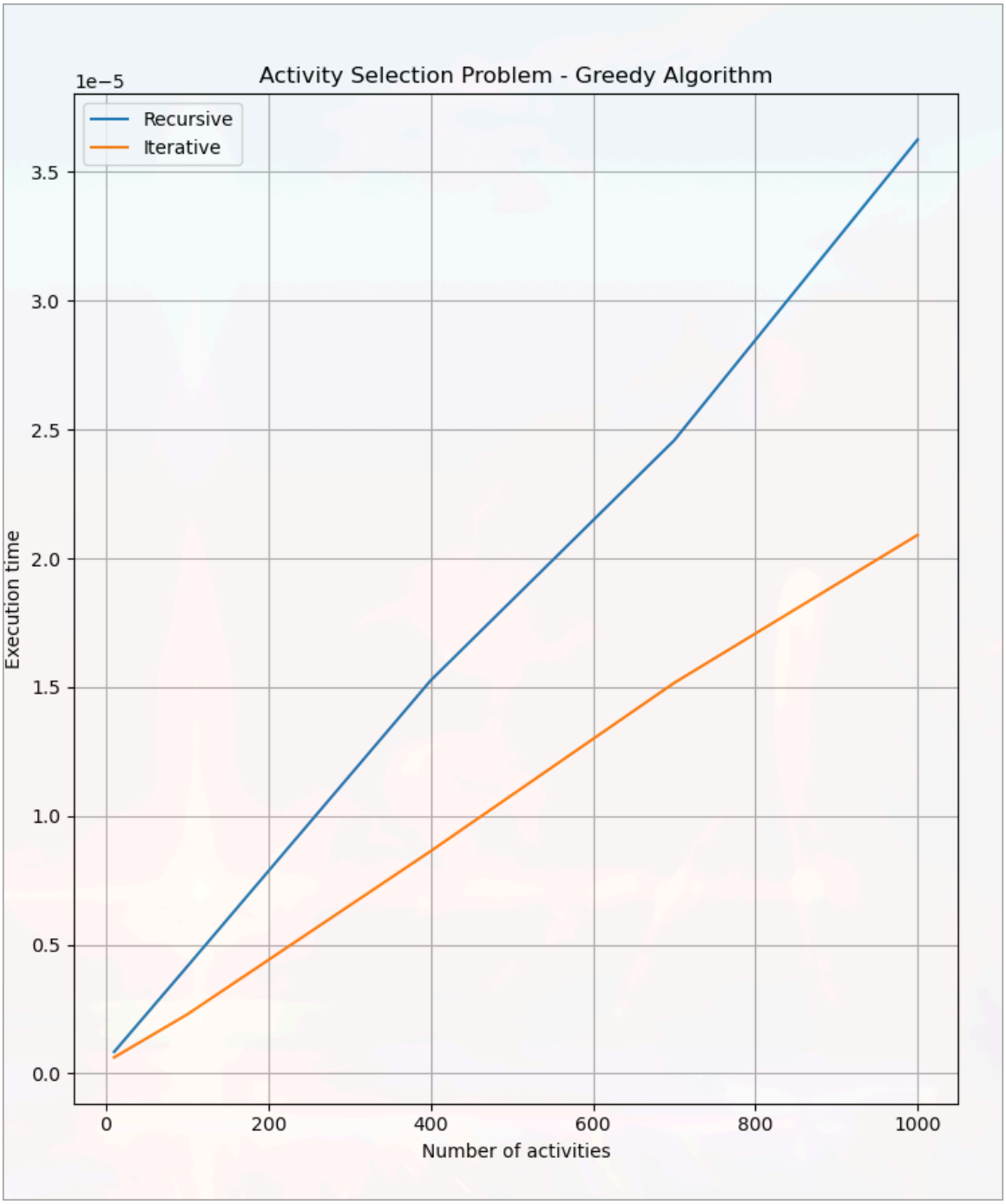
Prerequisites

- python
- matplotlib

Command to run

```
python -m lab2.activity_selection
```

Output



Here we can see that recursive one takes a bit longer than iterative process.

Aspect	Recursive Approach	Iterative Approach
Implementation	Requires recursive function calls with base cases.	Uses loops to iteratively find non-overlapping activities.
Time Complexity	O(n) (with sorted input)	O(n) (with sorted input)

Space Complexity	O(n) (due to recursion stack)	O(1) (no recursion)
Use Case	Useful for small datasets or when recursion is preferred.	Preferred for larger datasets due to better space efficiency.

Prim's vs Kruskal's Algorithm

Prim's and Kruskal's algorithms are used to find the Minimum Spanning Tree (MST) of a graph. Prim's algorithm grows the MST one vertex at a time, while Kruskal's algorithm grows it edge by edge.

How To Run

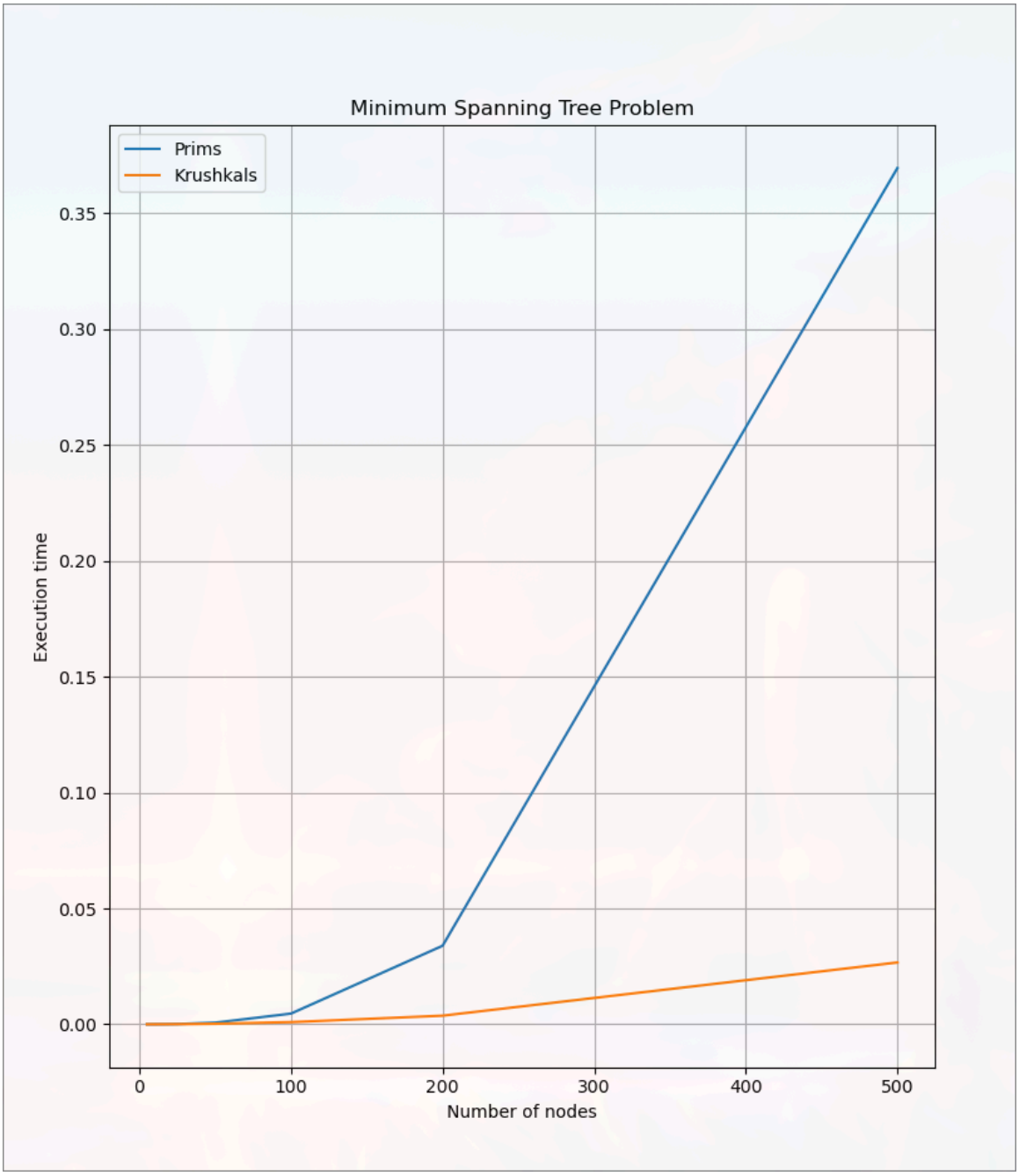
Prerequisites

- python
- matplotlib

Command to run

```
python -m lab2.min_span
```

Output



Here, we can see that Prim's algorithm has a n^2 graph, but Krushkal's algorithm is more efficient, having a graph of n log n function.

Aspect	Prim's Algorithm	Kruskal's Algorithm
Approach	Grows MST vertex by vertex.	Grows MST edge by edge.
Graph Representation	Requires adjacency matrix or list.	Edge list is sufficient.

Time Complexity	O(V^2) (with adjacency matrix), O(E + log(V)) (with min-heap)	O(E log E) (using Union-Find).
Space Complexity	O(V)	O(E)
Use Case	Efficient for dense graphs.	Efficient for sparse graphs.

N-Queens: Backtracking vs Bruteforce

The N-Queens problem involves placing N queens on an N×N chessboard such that no two queens threaten each other. Here, we compare the backtracking approach with a brute force solution.

How To Run

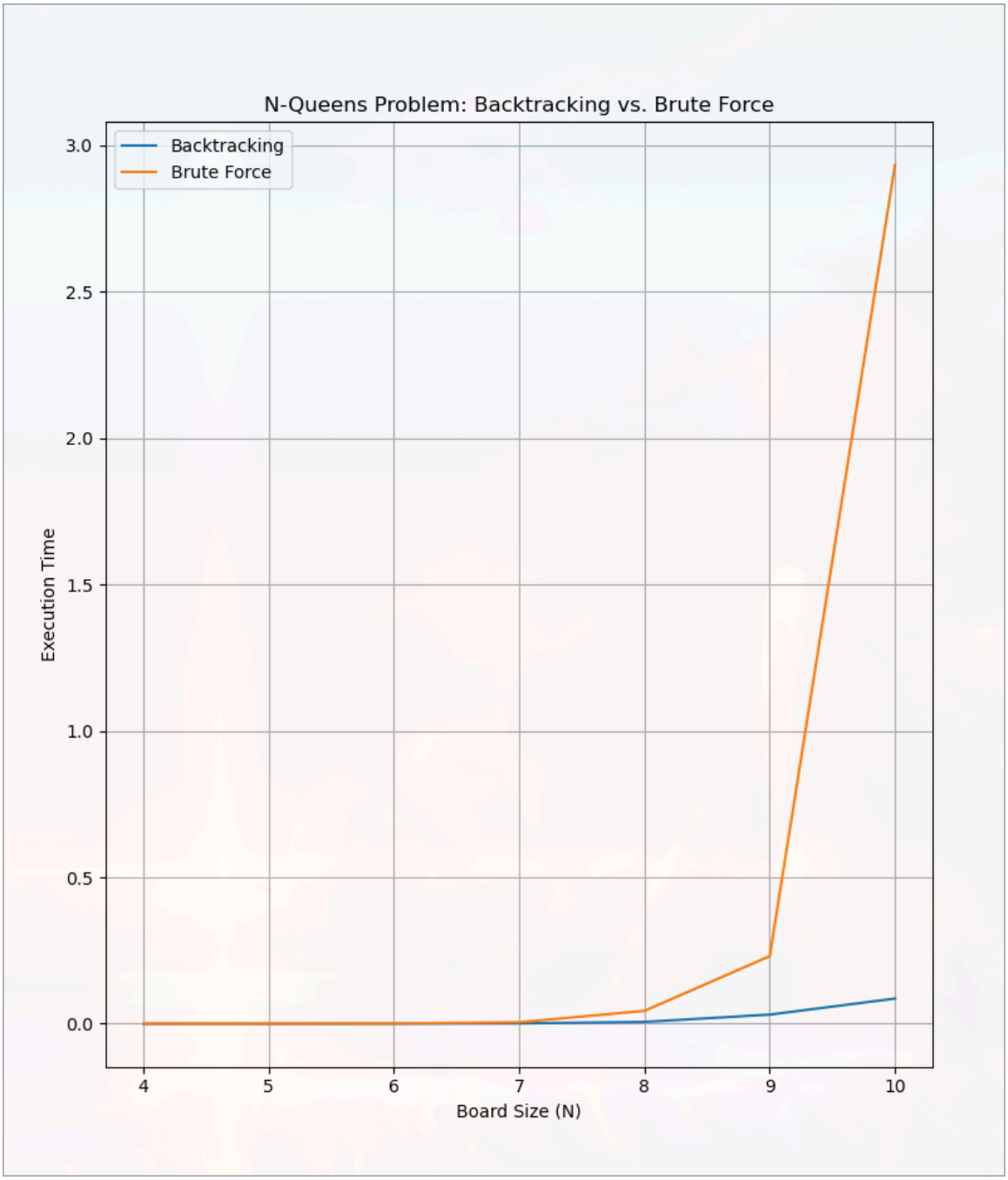
Prerequisites

- python
- matplotlib

Command to run

```
python -m lab2.n_queens
```

Output



As we can see from the graph above, brute force method is not feasible for a large value of n, since it has n! graph. But bruteforce on the other hand, has a polytime graph.

Aspect	Backtracking Approach	Bruteforce Approach
Implementation	Recursively place queens, backtracking when conflicts arise.	Generate all possible board configurations and check each one.

Time Complexity	O(N!) (pruned search space)	O(N^N) (all configurations).
Space Complexity	O(N) (due to recursion stack)	O(1) (no recursion).
Use Case	Efficient for solving N-Queens problem directly.	Rarely used due to exponential time complexity.

Conclusion

Each algorithm and approach has its own strengths and weaknesses. The choice of method depends on the problem, data size, and computational constraints. Understanding these differences helps in selecting the most efficient solution.