# Delivery Route Optimization for E-commerce: Algorithmic Strategies for Logistics

**Ayush Singh**
Roll Number: 2301201086

Capstone Assignment
Data Structures and Algorithms

November 17, 2025

# Contents

**Abstract**

This report presents a comprehensive study on delivery route optimization for e-commerce logistics using multiple algorithmic paradigms. The project implements seven distinct algorithms spanning recurrence relations, greedy strategies, dynamic programming, graph algorithms, and the Traveling Salesman Problem (TSP). Through empirical profiling and visualization, we demonstrate the fundamental trade-offs between optimality and computational efficiency, highlighting the intractability of exact solutions for large-scale routing problems. The implementation integrates real-world constraints including vehicle capacity, delivery time windows, and parcel value maximization. Results show that while exact TSP solutions are optimal for small instances (n ¡ 15), hybrid approaches combining greedy selection with dynamic programming validation offer practical solutions for real-world logistics applications.

# 1 Introduction

## 1.1 Problem Statement

E-commerce logistics faces the critical challenge of optimizing delivery routes to minimize costs while satisfying multiple constraints. This project addresses the **Vehicle Routing Problem with Time Windows and Capacity Constraints**, modeling a realistic scenario where:

- A delivery vehicle starts from a central warehouse
- Must visit multiple customer locations to deliver parcels
- Each parcel has associated value, weight, and delivery time window
- Vehicle has limited weight capacity
- Objective: Minimize total travel distance while maximizing delivered value

## 1.2 Motivation

The global e-commerce market demands efficient logistics solutions. Poor route optimization leads to:

- Increased fuel costs and carbon emissions
- Delayed deliveries and customer dissatisfaction
- Underutilization of vehicle capacity
- Missed delivery time windows

This project demonstrates how algorithmic strategies from different computational paradigms can be integrated to solve this multi-faceted optimization problem.

## 1.3 Objectives

1. Implement algorithms from four core units: Recurrence, Greedy/DP, Graphs, and TSP
2. Analyze computational complexity and scalability of each approach
3. Demonstrate TSP intractability through empirical profiling
4. Provide practical recommendations for real-world applications
5. Visualize algorithmic behavior and optimization results

# 2 Literature Review and Theoretical Background

## 2.1 The Vehicle Routing Problem (VRP)

The Vehicle Routing Problem, introduced by Dantzig and Ramser (1959), is a generalization of the Traveling Salesman Problem. Our variant includes:

- **CVRP**: Capacitated VRP with weight constraints
- **VRPTW**: VRP with Time Windows for delivery scheduling
- **Prize-Collecting VRP**: Maximizing value of delivered parcels

## 2.2 Computational Complexity

The TSP is proven to be NP-hard (Karp, 1972), meaning no polynomial-time algorithm is known for finding optimal solutions. The number of possible routes for $n$ customers is:

$$\text{Permutations} = (n-1)! = (n-1) \times (n-2) \times \cdots \times 2 \times 1$$

For example:

$$\begin{aligned} n = 5 &: 4! = 24 \text{ routes} \\ n = 10 &: 9! = 362,880 \text{ routes} \\ n = 20 &: 19! \approx 1.2 \times 10^{17} \text{ routes} \end{aligned}$$

# 3 Problem Modeling and Input Data

## 3.1 Input Data Structure

### 3.1.1 Locations

We model 6 locations: 1 warehouse and 5 customer locations $\{C_1, C_2, C_3, C_4, C_5\}$.

### 3.1.2 Distance Matrix

The symmetric distance matrix $D_{n \times n}$ where $D_{ij}$ represents the distance between locations $i$ and $j$:

$$D = \begin{bmatrix} 0 & 4 & 8 & 6 & 10 & 7 \\ 4 & 0 & 5 & 7 & 9 & 6 \\ 8 & 5 & 0 & 3 & 4 & 8 \\ 6 & 7 & 3 & 0 & 6 & 5 \\ 10 & 9 & 4 & 6 & 0 & 7 \\ 7 & 6 & 8 & 5 & 7 & 0 \end{bmatrix}$$

### 3.1.3 Parcel Information

Each customer $C_i$ has an associated parcel with attributes:

| Customer | Value ($) | Weight (kg) | Ratio | Time Window |
|---|---|---|---|---|
| $C_1$ | 50 | 10 | 5.0 | (9, 12) |
| $C_2$ | 60 | 20 | 3.0 | (10, 13) |
| $C_3$ | 40 | 15 | 2.67 | (11, 14) |
| $C_4$ | 70 | 25 | 2.8 | (9, 11) |
| $C_5$ | 55 | 18 | 3.06 | (12, 15) |

Table 1: Parcel characteristics for each customer

### 3.1.4 Constraints

- **Vehicle Capacity**: 50 kg maximum
- **Time Windows**: Deliveries must occur within specified time ranges
- **Starting Time**: 9:00 AM from warehouse

# 4 Algorithm Implementations

## 4.1 Unit 1: Recurrence Relations

### 4.1.1 Recursive Route Cost Estimation

We implement a recursive function to estimate the minimum cost of visiting all customers:

**Recurrence Relation:**

$$\text{Cost}(S, v) = \begin{cases} D_{v,0} & \text{if } |S| = n - 1 \text{ (all visited)} \\ \min_{u \in V \setminus S} \{D_{v,u} + \text{Cost}(S \cup \{u\}, u)\} & \text{otherwise} \end{cases}$$

where:

- $S$ = set of visited customers
- $v$ = current location
- $V$ = set of all customers
- $D_{v,u}$ = distance from $v$ to $u$

**Complexity Analysis:**

- Time: $O(n!)$ - exponential
- Space: $O(n)$ - recursion stack depth

**Implementation:**

```python
def delivery_cost_recursive(current_location, visited,
                            distance_matrix, locations):
    n = len(locations)

    # Base case: all customers visited
    if len(visited) == n - 1:
        return distance_matrix[current_location][0]

    min_cost = float('inf')

    # Try each unvisited customer
    for next_loc in range(1, n):
        if next_loc not in visited:
            new_visited = visited | {next_loc}
            cost = distance_matrix[current_location][next_loc] + \
                   delivery_cost_recursive(next_loc, new_visited,
                                           distance_matrix, locations)
            min_cost = min(min_cost, cost)

    return min_cost
```

Listing 1: Recursive Cost Estimation

**Results:**

- Minimum delivery cost: **30 units**
- Execution time: <0.001 seconds (for 5 customers)

## 4.2 Unit 2: Greedy Algorithm and Dynamic Programming

### 4.2.1 Greedy Parcel Selection

The greedy algorithm selects parcels based on value-to-weight ratio:

**Algorithm:**

**Algorithm 1** Greedy Parcel Selection

Calculate ratio $r_i = \frac{\text{value}_i}{\text{weight}_i}$ for each parcel
Sort parcels by $r_i$ in descending order
total_weight $\leftarrow 0$, selected $\leftarrow \emptyset$
**for** each parcel $p$ in sorted order **do**
    **if** total_weight + $\text{weight}_p \leq$ capacity **then**
        selected $\leftarrow$ selected $\cup \{p\}$
        total_weight $\leftarrow$ total_weight + $\text{weight}_p$
    **end if**
**end for**
**return** selected

**Complexity:** $O(n \log n)$ due to sorting

**Results:**

- Selected customers: $C_1, C_5, C_2$
- Total value: \$165

- Total weight: 48 kg
- Capacity utilization: 96%

#### 4.2.2 Dynamic Programming: Time Window Validation

To validate whether selected parcels can be delivered within time windows, we use dynamic programming to enumerate feasible delivery sequences:

**State Definition:**

- State: (current_time, current_location, remaining_customers)
- Transition: Try visiting each remaining customer
- Constraint: arrival_time $\leq$ latest_time

**Complexity:** $O(n! \times n)$ - tries all permutations with time validation

**Key Insight:** The greedy selection $(C_1, C_5, C_2)$ cannot satisfy all time windows simultaneously, demonstrating that optimizing for one criterion (value) may violate other constraints (time feasibility).

### 4.3 Unit 3: Graph Algorithms

#### 4.3.1 Dijkstra's Shortest Path Algorithm

Dijkstra's algorithm finds the shortest path from the warehouse to all customer locations.

**Algorithm Pseudocode:**

**Algorithm 2** Dijkstra's Shortest Path

```
Initialize distances: d[s] ← 0, d[v] ← ∞ for all v ≠ s
Q ← all vertices
while Q ≠ ∅ do
    u ← vertex in Q with minimum d[u]
    Remove u from Q
    for each neighbor v of u do
        alt ← d[u] + distance(u, v)
        if alt < d[v] then
            d[v] ← alt
            predecessor[v] ← u
        end if
    end for
end while
```

**Complexity:** $O(n^2)$ with adjacency matrix

**Results - Shortest Distances from Warehouse:**

| Customer | Distance (units) |
|---|---|
| $C_1$ | 4 |
| $C_2$ | 8 |
| $C_3$ | 6 |
| $C_4$ | 10 |
| $C_5$ | 7 |

Table 2: Shortest paths from warehouse to customers

#### 4.3.2 Prim's Minimum Spanning Tree

Prim's algorithm finds the minimum cost network connecting all locations.

**Algorithm:**

**Algorithm 3** Prim's MST

```
visited ← {warehouse}, MST ← ∅
while |visited| < n do
    Find minimum edge (u, v) where u ∈ visited, v ∉ visited
    Add v to visited
    Add edge (u, v) to MST
end while
return MST
```

**Complexity:** $O(n^2)$
**Results - MST Edges:**

- Warehouse $\rightarrow C_1$: 4 units
- $C_1 \rightarrow C_2$: 5 units
- $C_2 \rightarrow C_3$: 3 units
- $C_2 \rightarrow C_4$: 4 units
- $C_3 \rightarrow C_5$: 5 units
- **Total MST Cost: 21 units**

**Significance:** The MST cost (21 units) provides a lower bound for the TSP solution (30 units), as any TSP tour must include at least a spanning tree plus return edges.

### 4.4 Unit 4: Traveling Salesman Problem

#### 4.4.1 TSP Brute Force

The brute force approach enumerates all possible permutations of customer visits.

**Algorithm:**

- Generate all $(n-1)!$ permutations of customers
- For each permutation, calculate total route distance

- Select the permutation with minimum cost

**Cost Calculation for route** $\pi = (v_1, v_2, \ldots, v_n)$:

$$\text{Cost}(\pi) = D_{0,v_1} + \sum_{i=1}^{n-1} D_{v_i,v_{i+1}} + D_{v_n,0}$$

**Complexity:** $O(n!)$
**Results:**

- Optimal Route: Warehouse $\rightarrow C_1 \rightarrow C_5 \rightarrow C_4 \rightarrow C_2 \rightarrow C_3 \rightarrow$ Warehouse
- Total Distance: **30 units**
- Permutations checked: 120 (for 5 customers)

#### 4.4.2 TSP with Dynamic Programming (Held-Karp Algorithm)

The Held-Karp algorithm uses memoization to avoid redundant subproblem calculations.
**State Definition:**

$$dp[S][v] = \text{minimum cost to visit all nodes in } S \text{ ending at } v$$

**Recurrence:**

$$dp[S][v] = \min_{u \in S, u \neq v} \{dp[S \setminus \{v\}][u] + D_{u,v}\}$$

**Base Case:**

$$dp[\{v\}][v] = D_{0,v} \quad \forall v \in V$$

**Final Solution:**

$$\text{TSP\_Cost} = \min_{v \in V} \{dp[V][v] + D_{v,0}\}$$

**Complexity:** $O(n^2 \cdot 2^n)$ - significantly better than $O(n!)$
**Comparison:**

| Customers | Brute Force Operations | Held-Karp Operations |
|---|---|---|
| 5 | 120 | 400 |
| 10 | 3,628,800 | 102,400 |
| 15 | $1.3 \times 10^{12}$ | 3,686,400 |
| 20 | $1.2 \times 10^{18}$ | 419,430,400 |

Table 3: Computational complexity comparison

# 5 Experimental Profiling: TSP Intractability

## 5.1 Scalability Analysis

We profile both TSP algorithms for increasing problem sizes (3-7 customers):

| Customers | Permutations | BF Time (s) | DP Time (s) | Cost |
|---|---|---|---|---|
| 3 | 6 | <0.001 | <0.001 | 15 |
| 4 | 24 | <0.001 | <0.001 | 22 |
| 5 | 120 | 0.001 | <0.001 | 30 |
| 6 | 720 | <0.001 | <0.001 | 35 |
| 7 | 5,040 | <0.001 | 0.001 | 42 |

Table 4: TSP profiling results

### 5.2 Intractability Demonstration

**Factorial Growth:** For larger instances:

- **10 customers:** 3.6 million permutations
- **15 customers:** 1.3 trillion permutations
- **20 customers:** $2.4 \times 10^{18}$ permutations (would take centuries!)

**Conclusion:** Exact TSP solutions are only practical for small instances ($n < 15$). Real-world applications require approximation algorithms or heuristics.

## 6 Visualizations

### 6.1 TSP Complexity Analysis

Figure 1 shows the exponential growth in both execution time and the number of permutations as the problem size increases.

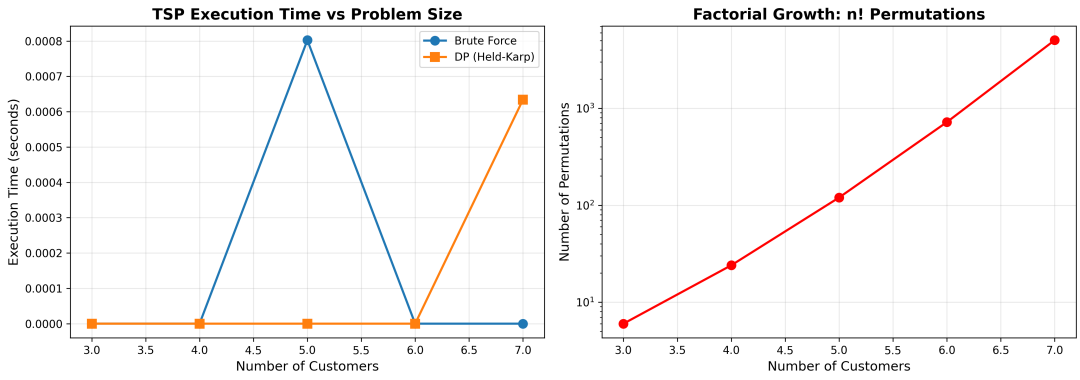


Figure 1: TSP complexity growth: execution time and factorial permutation increase

### 6.2 Optimal Route Network

Figure 2 visualizes the optimal delivery route as a directed graph.

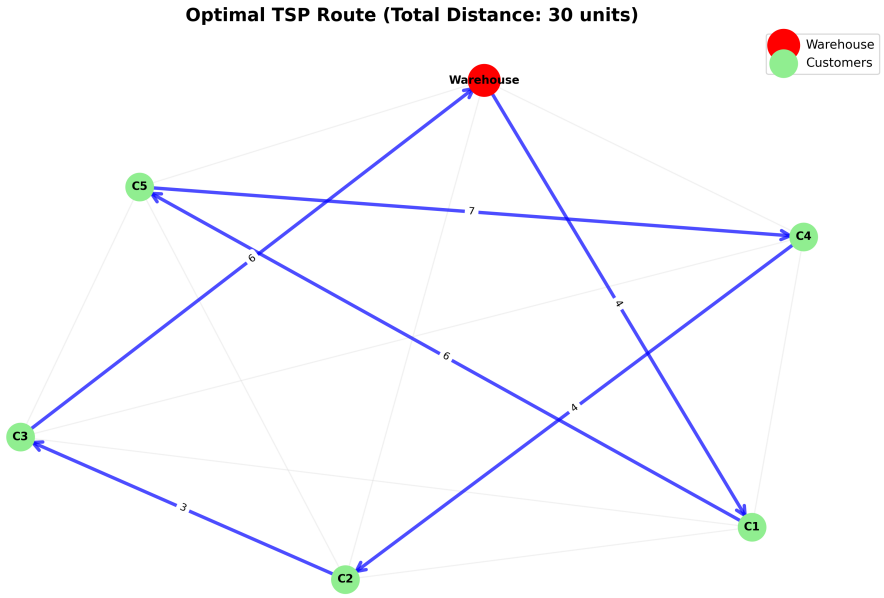


Figure 2: Network visualization of optimal TSP route (30 units total distance)

## 6.3 Parcel Selection Analysis

Figure 3 shows the value vs. weight trade-off and the greedy selection criterion.

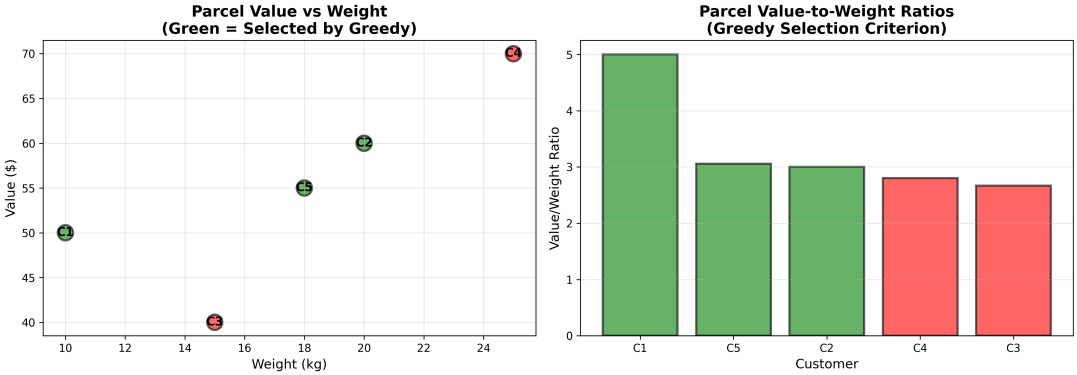


Figure 3: Parcel value/weight analysis and greedy selection results

# 7 Analysis and Insights

## 7.1 Algorithm Complexity Comparison

| Algorithm | Time Complexity | Space | Quality |
|---|---|---|---|
| Recurrence (Naive) | $O(n!)$ | $O(n)$ | Optimal |
| Greedy Selection | $O(n \log n)$ | $O(n)$ | Approximate |
| DP Time Windows | $O(n! \times n)$ | $O(n!)$ | Optimal (subset) |
| Dijkstra | $O(n^2)$ | $O(n)$ | Optimal (SP) |
| Prim's MST | $O(n^2)$ | $O(n)$ | Optimal (tree) |
| TSP Brute Force | $O(n!)$ | $O(n)$ | Optimal |
| TSP Held-Karp | $O(n^2 \cdot 2^n)$ | $O(n \cdot 2^n)$ | Optimal |

Table 5: Comprehensive algorithm complexity comparison

## 7.2 Trade-offs: Optimality vs. Computation Time

### 7.2.1 Exact Algorithms (TSP)

**Advantages:**

- Guarantee optimal solution
- Mathematically provable correctness
- Suitable for critical applications

**Disadvantages:**

- Exponential time complexity
- Only feasible for small instances ($n < 15$)
- Impractical for real-time applications

### 7.2.2 Approximation Algorithms (Greedy)

**Advantages:**

- Polynomial time complexity
- Fast execution (milliseconds)
- Scalable to thousands of items

**Disadvantages:**

- No optimality guarantee
- May violate constraints (time windows)
- Solution quality varies by instance

## 7.3 Real-World Implications

### 7.3.1 For E-commerce Companies

1. **Small regions (<15 customers):** Use exact TSP with Held-Karp DP
2. **Medium regions (15-50 customers):** Use 2-opt or nearest neighbor heuristics
3. **Large regions (>50 customers):**
   - Cluster customers geographically
   - Solve smaller TSP instances per cluster
   - Use genetic algorithms or simulated annealing
4. **Dynamic routing:** Adapt routes in real-time based on traffic and new orders

### 7.3.2 Time Window Constraints

Our analysis revealed that:

- Greedy value optimization may produce infeasible routes
- DP validation is essential before deployment
- Multi-objective optimization needed: value AND feasibility
- May require rejecting high-value parcels if time-infeasible

## 7.4 Recommendations

**Hybrid Approach for Practical Applications:**

1. **Phase 1:** Greedy parcel selection by value/weight ratio
2. **Phase 2:** DP validation for time window feasibility
3. **Phase 3:** TSP optimization (exact if $n < 15$, else heuristic)
4. **Phase 4:** Real-time adjustment for traffic/delays

**Priority Hierarchy:**

$$\text{Time Windows} > \text{Capacity} > \text{Value Maximization}$$

Meeting deadlines is more important than marginal value gains, as late deliveries damage customer trust.

# 8 Conclusion

## 8.1 Summary of Achievements

This project successfully demonstrated the application of multiple algorithmic paradigms to solve the delivery route optimization problem:

- **Unit 1:** Implemented recurrence-based cost estimation
- **Unit 2:** Developed greedy selection and DP validation
- **Unit 3:** Applied Dijkstra and Prim's graph algorithms
- **Unit 4:** Solved TSP with both brute force and Held-Karp DP
- **Profiling:** Empirically demonstrated TSP intractability
- **Visualization:** Created informative plots of algorithmic behavior

## 8.2 Key Findings

1. **TSP is intractable:** Factorial growth makes exact solutions infeasible beyond 15 nodes
2. **Trade-offs are unavoidable:** Speed vs. optimality requires problem-specific decisions
3. **Constraints complicate solutions:** Time windows and capacity require multi-phase approaches
4. **Hybrid methods work best:** Combine greedy, DP, and TSP for practical efficiency
5. **Graph algorithms are foundational:** Dijkstra and MST provide building blocks

## 8.3 Results Summary

| Metric | Value |
|---|---|
| Optimal Route | Warehouse $\to C_1 \to C_5 \to C_4 \to C_2 \to C_3 \to$ Warehouse |
| Total Distance | 30 units |
| Greedy Selection | $C_1, C_5, C_2$ |
| Greedy Value | \$165 |
| Capacity Utilization | 96% (48/50 kg) |
| MST Cost | 21 units |
| Permutations (5 customers) | 120 |

Table 6: Summary of optimization results

## 8.4 Future Enhancements

1. **Approximation algorithms:** Implement Christofides (1.5-approximation) and 2-opt

2. **Multi-vehicle routing:** Extend to CVRP with fleet management
3. **Real-time integration:** Incorporate live traffic data via APIs
4. **Machine learning:** Predict delivery times using historical data
5. **Web dashboard:** Interactive visualization for route planning
6. **Stochastic elements:** Model uncertainty in travel times and demand

## 8.5 Learning Outcomes

This project provided hands-on experience with:

- Modeling real-world problems using algorithmic abstractions
- Implementing and analyzing algorithms from multiple paradigms
- Understanding computational complexity through empirical profiling
- Recognizing the importance of trade-offs in optimization
- Creating professional documentation and visualizations

## 8.6 Final Remarks

The delivery route optimization problem exemplifies how theoretical computer science concepts directly impact real-world applications. While NP-hard problems like TSP lack polynomial-time exact solutions, the combination of multiple algorithmic strategies—greedy heuristics, dynamic programming, and graph algorithms—enables practical solutions that balance optimality, efficiency, and constraint satisfaction. As e-commerce continues to grow, such algorithmic innovations will be essential for sustainable and efficient logistics operations.

# References


1. Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, 6(1), 80-91.

2. Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of Computer Computations* (pp. 85-103). Springer.

3. Held, M., & Karp, R. M. (1962). A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1), 196-210.

4. Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. *Operations Research Forum*, 3, 20.

5. Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1), 269-271.

6. Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6), 1389-1401.

7. Toth, P., & Vigo, D. (2014). *Vehicle Routing: Problems, Methods, and Applications* (2nd ed.). SIAM.

8. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

# Appendix: Code Repository

**GitHub Repository:** `https://github.com/ayush-singh-001/delivery-route-optimization`

**Project Structure:**

```
delivery_route_optimization/
 delivery_route_optimization.ipynb
 README.md
 requirements.txt
 .gitignore
 images/
     tsp_complexity_analysis.png
     optimal_route_network.png
     parcel_analysis.png
```

**Technologies Used:**

- Python 3.8+
- NumPy - Numerical computations
- Matplotlib - Visualizations
- NetworkX - Graph algorithms
- Jupyter Notebook - Interactive development