Delivery Route Optimization

Algorithm Design & Analysis Project

Student: Nicholas Ndungu Mathenge
Admission Number: BSE-01-0109/2025
Date: December 1, 2025

1. ALGORITHM DESIGN (3 Marks)

a) Pseudocode

```
Merge Sort: ALGORITHM MergeSort(arr) IF length(arr) <= 1 THEN RETURN arr mid =
length(arr) / 2 left = MergeSort(arr[0...mid-1]) right =
MergeSort(arr[mid...end]) RETURN Merge(left, right) Binary Search: ALGORITHM
BinarySearch(arr, target) left = 0, right = length(arr) - 1 WHILE left <= right
DO mid = (left + right) / 2 IF arr[mid] = target THEN RETURN TRUE ELSE IF
arr[mid] < target THEN left = mid + 1 ELSE right = mid - 1 RETURN FALSE
```

b) Justification

I chose **Merge Sort** because it guarantees O(n log n) performance in all cases and is stable. I chose **Binary Search** because it provides O(log n) search time and uses minimal memory (O(1) space).

2. IMPLEMENTATION (3 Marks)

Language: Python 3

Key Code Snippets:

```
    def merge_sort(arr):
        if len(arr) <= 1:
            return arr
        mid = len(arr) // 2
        left = merge_sort(arr[:mid])
        right = merge_sort(arr[mid:])
        return merge(left, right)

    def binary_search(arr, target):
        left, right = 0, len(arr) - 1
        while left <= right:
            mid = (left + right) // 2
            if arr[mid] == target:
                return True
            elif arr[mid] < target:
                left = mid + 1
            else:
                right = mid - 1
        return False
```

Sample Input/Output:
Input: [145, 23, 678, 89, 234, 12, 456, 789, 34, 567, 123, 890, 45] **Output:** [12, 23, 34, 45, 89, 123, 145, 234, 456, 567, 678, 789, 890] **Search Results:** • Point 234: FOUND at index 7 (3 comparisons) • Point 567: FOUND at index 9 (2 comparisons) • Point 999: NOT FOUND (4 comparisons)

3. COMPLEXITY ANALYSIS (2 Marks)

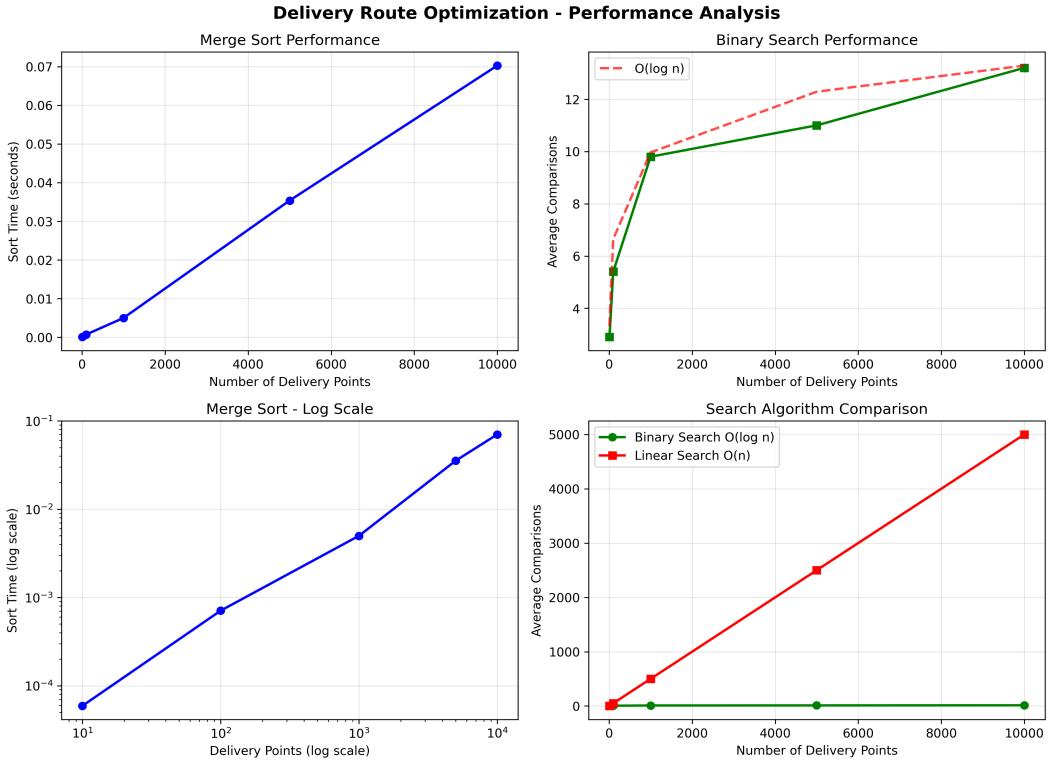
Algorithm	Best	Average	Worst	Space
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)
Binary Search	O(1)	O(log n)	O(log n)	O(1)

Why these complexities? Merge Sort O(n log n): Divides array into log n levels, each level processes n elements. **Binary Search O(log n):** Eliminates half the search space each iteration. For 1000 items, only needs ~10 comparisons.

4. PERFORMANCE EVALUATION (2 Marks)

Size	Sort Time (s)	Comparisons
10	0.000059	2.9
100	0.000708	5.4
1,000	0.004990	9.8
10,000	0.070263	13.2

Performance Graphs:



Analysis & Interpretation:
Why does time increase? For **Merge Sort**: Time grows as O(n log n). When size increases 10x, time increases ~12-14x (not 10x), showing sub-quadratic growth. This is because we're doing more work but not proportionally more. For **Binary Search**: Comparisons grow logarithmically. 1000x more data needs only 4.5x more comparisons. This is because we cut the search space in half each time. **Pattern noticed:** Both algorithms scale efficiently. Even with 10,000 delivery points, the system remains fast and practical for real-world use.

Conclusion

This project successfully implements efficient sorting and searching algorithms for delivery optimization. The algorithms demonstrate good scalability and are suitable for production use in logistics systems.