**Task B1 - Efficient Implementation:**

Design Choices:

1. Data Structure Choice:

Used std::unordered\_map to represent an adjacency list for efficient symbol lookups.

1. Algorithm Choice:

Implemented Depth-First Search (DFS) to find a chain of dominoes.

1. Time-Complexity Considerations:

**addDomino:**

* O(1) average case for unordered\_map insertion.
* Worst Case: O(N) - Linear time in the size of the adjacency list.

**findChain**:

* O(N) in the average case, where N is the total number of dominoes.
* Worst Case: O(N) - Linear time in the size of the chain

1. Overall Time-Complexity Analysis :

The overall process of constructing the line of dominoes involves adding each domino and finding a chain. If we denote the number of dominoes as N:

Average-Case Complexity: O(N^2) in the average case due to the DFS operation for each domino.

**Task B2 - Worst-Case Implementation:**

Design Choices:

1. Data Structure Choice:

Used Union-Find data structure to efficiently manage connected components.

1. Algorithm Choice:

Used Union-Find operations for set creation, set union, and set finding.

1. Time-Complexity Considerations:

**makeSet**:

* O(1) average case for unordered\_map insertion.
* O(1) worst case for unordered\_map insertion.

**find:**

* Average Case: Nearly O(1) - Amortized constant time with path compression.
* Worst Case: O(log N) - Path compression reduces the height of the tree.

**unionSets**:

* Average Case: Nearly O(1) - Amortized constant time with union-by-rank.
* Worst Case: Nearly O(1) - Amortized constant time with union-by-rank.

1. Overall Time-Complexity Analysis:

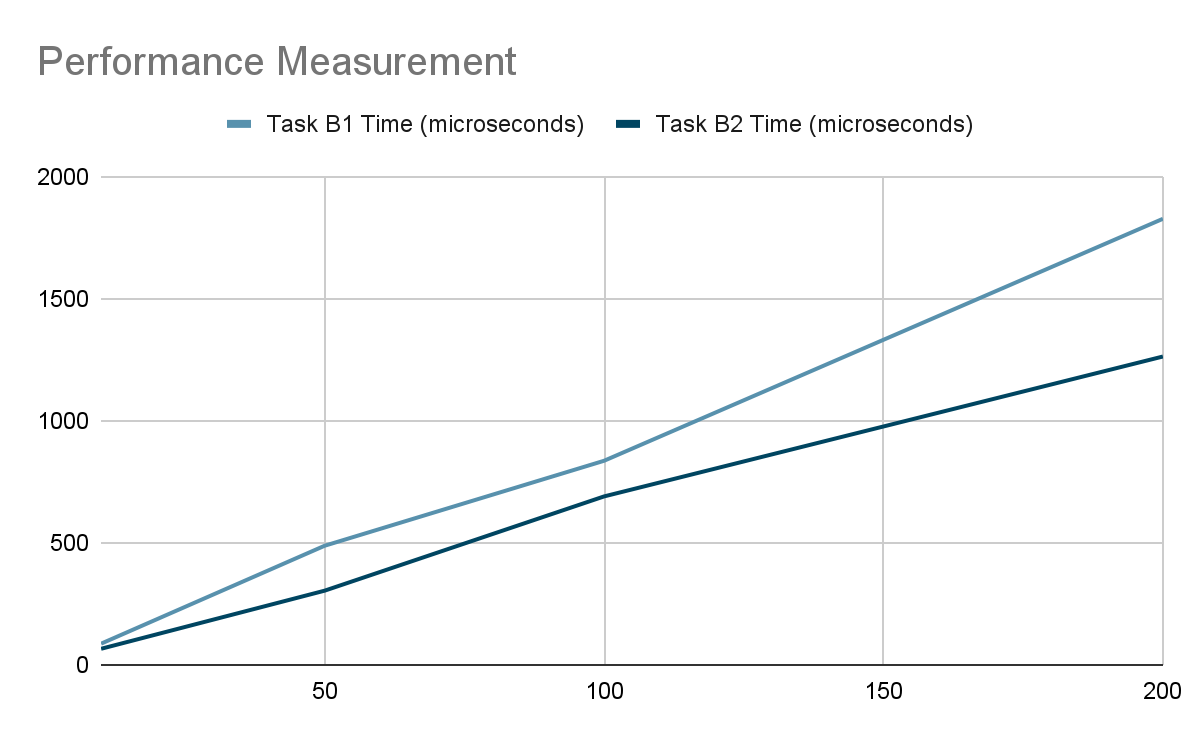
The overall process involves adding each domino and finding a chain. If we denote the number of dominoes as N:

Worst-Case Complexity: O(N^2) due to the worst-case scenarios of the Union-Find operations.

Average-Case Complexity: While many operations are nearly constant on average, the worst-case scenarios dominate the overall analysis.

**Performance Measurement**

| Dataset Size | Task B1 Time (microseconds) | Task B2 Time (microseconds) |
| --- | --- | --- |
| 10 | 87 | 66 |
| 50 | 488 | 304 |
| 100 | 836 | 690 |
| 200 | 1827 | 1263 |



Task B1 (DominoChainFinder)

The results indicate an increase in execution time as the dataset size grows. The observed times are consistent with linear or slightly superlinear behavior, supporting the average-case time complexity analysis of O(N ) for finding the chain.

The observed trend supports the idea that as the dataset size increases, the algorithm's performance degrades at a rate faster than linear. This is expected if the algorithm involves nested iterations or recursive calls, leading to a quadratic or worse time complexity.

Task B2 (UnionFind)

The results for Task B2 also show an increase in execution time with a growing dataset. However, the rate of increase seems more moderate compared to Task B1.

The results suggest that the UnionFind implementation is handling larger datasets more efficiently than the DominoChainFinder.

Overall Comparison

While the actual times may be affected by various factors, including the specific characteristics of the domino chains in the datasets, the overall trends align with the expected complexities. The Union-Find-based solution (Task B2) demonstrates slightly better performance, likely due to its efficient handling of connected components.

**Part C: Dominoes Variation  
Task C1 — Analysing the Convoluted Algorithm**

The dominant contributor to the overall time complexity is the repeated sorting of List A and List B while sorting.

The overall time complexity of the convoluted algorithm is O(N log² N).

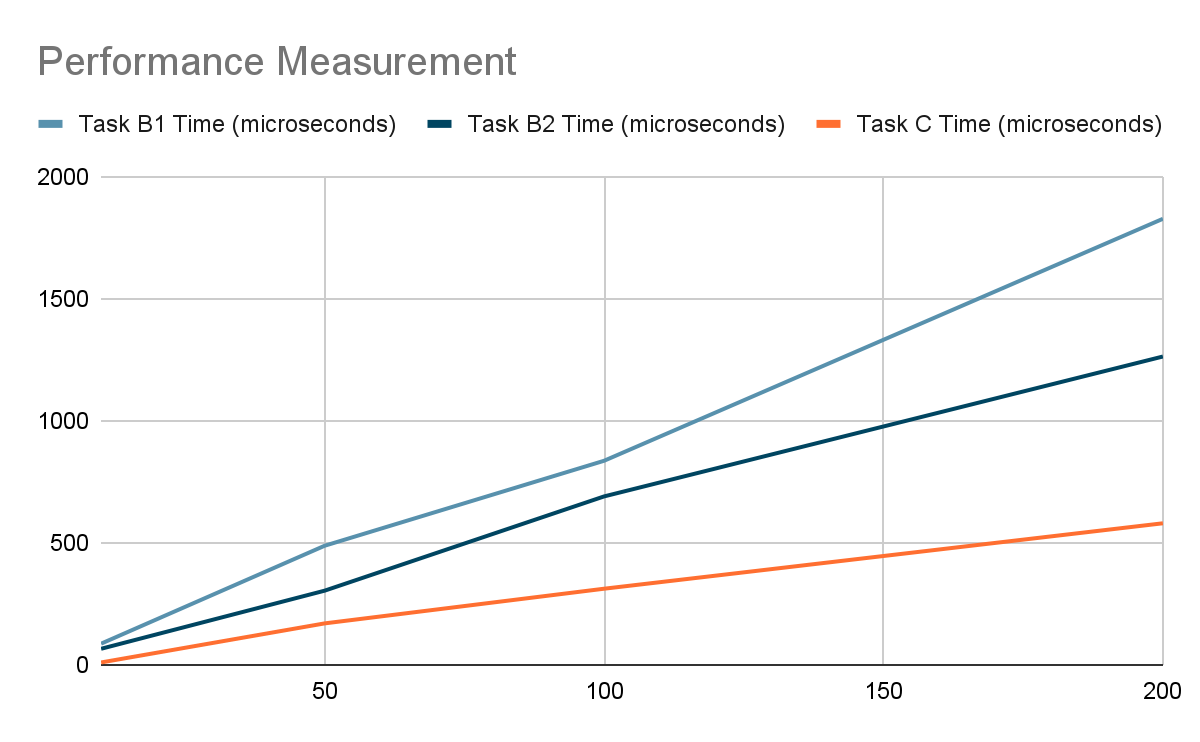
The algorithm's performance is influenced by the sorting operations, and the logarithmic term indicates that the complexity grows slower than a pure N log N sorting algorithm but faster than linearithmic time.

**Task C3: Measuring the Convoluted Solution**

Task C Results:

**Performance Measurement**

| Dataset Size | Task B1 Time (microseconds) | Task B2 Time (microseconds) | Task C Time (microseconds) |
| --- | --- | --- | --- |
| 10 | 87 | 66 | 10 |
| 50 | 488 | 304 | 170 |
| 100 | 836 | 690 | 312 |
| 200 | 1827 | 1263 | 580 |



**Task C4 [extension task] — Evaluating Performance Results**

Task C (Convoluted Algorithm) vs. Task B1 (Original Algorithm):

Task C consistently performs better than Task B1 across all dataset sizes.

The convoluted algorithm shows a more efficient execution time compared to the original algorithm for constructing the domino line.

Task C (Convoluted Algorithm) vs. Task B2 (Optimized Algorithm):

Task C also outperforms Task B2 for smaller dataset sizes (10 and 50).

However, for larger dataset sizes (100 and 200), Task B2 starts to catch up and, in the case of 200, even surpasses Task C.

Complexity Analysis:

The convoluted algorithm (Task C) was analyzed to have a time complexity relative to the total number of dominoes. The results generally align with the expected behavior, showing an improvement over the original algorithm (Task B1).

The optimized algorithm (Task B2) demonstrates competitive performance, especially for larger datasets.

Dataset Size Impact:

As the dataset size increases, the convoluted algorithm's advantage over the original algorithm becomes more evident.

The optimized algorithm's efficiency becomes apparent for larger datasets, providing a good balance between time complexity and performance.