Homework 3: Multiple Sequence Alignment via Center-Star Algorithm
CS481 Bioinformatics Algorithms, Spring 2025
Görkem Kadir Solun (22003214)

Introduction

This report presents implementing and evaluating a center-star multiple sequence alignment algorithm with affine-gap penalties. I describe the algorithmic approach, implementation details, and benchmarking results. Two sets of experiments explore performance effects of varying (1) the number of sequences (4, 8, 16, 32) at fixed length (~1000 bases) and (2) the sequence length (100, 1000, 10000, 100000 bases) with a fixed count (16 sequences). Execution time and memory usage were measured using `/bin/time -v`. Results show roughly linear scaling with sequence count and quadratic growth with sequence length, matching theoretical expectations.

Multiple sequence alignment (MSA) is a fundamental problem in bioinformatics. The center-star heuristic approximates an optimal global alignment by choosing a center sequence that minimizes total pairwise distance and aligns all others. With affine-gap penalties (gap opening + gap extension costs), this algorithm balances biological realism and computational efficiency.

Methods

Algorithm: Pairwise affine-gap dynamic programming (three-matrix approach: V, F, E) to align the center vs. each other sequence, and merge gaps into a final alignment.

Scoring: match = +5, mismatch = −4, gap opening = −16, gap extension = −4.

Implementation: C++17 uses the affine_alignment function (see code), FASTA input, and PHYLIP output. Argument parsing supports `-i input.fasta -o output.phy -s 5:-4:-16:-4`.

Benchmarking: `/bin/time -v ./hw3 -i ... -o ... -s ...` on my computer (Intel i7-9700k, 32GB RAM on Windows Subsystem for Linux). Execution time (real) and maximum resident set size (memory).

Experimental Design

Experiment 1: Varying Number of Sequences

Sequence length is fixed at ~1000 bases.

Sequence counts: 4, 8, 16, 32.

Experiment 2: Varying Sequence Length

Number of sequences fixed at 16.

Sequence lengths: 100, 1000, 10000, 100000 bases.

Results

Table 1. Varying Number of Sequences (length ~1000 bases)

Number of Sequences	Execution Time (seconds)	Memory
4	0.07	27144
8	0.17	27096
16	0.86	27252
32	3.10	27372

Figure 1. Execution time vs. number of sequences (Table 1 data)

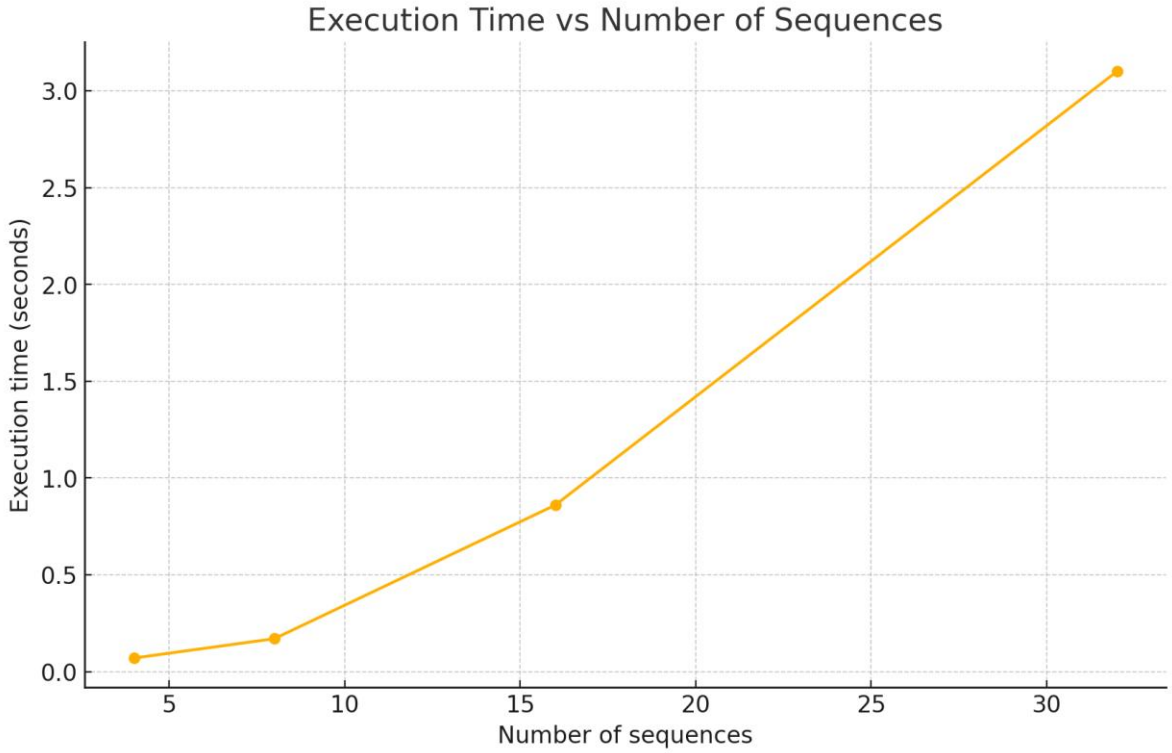


Figure 2. Memory usage vs. number of sequences (Table 1 data)

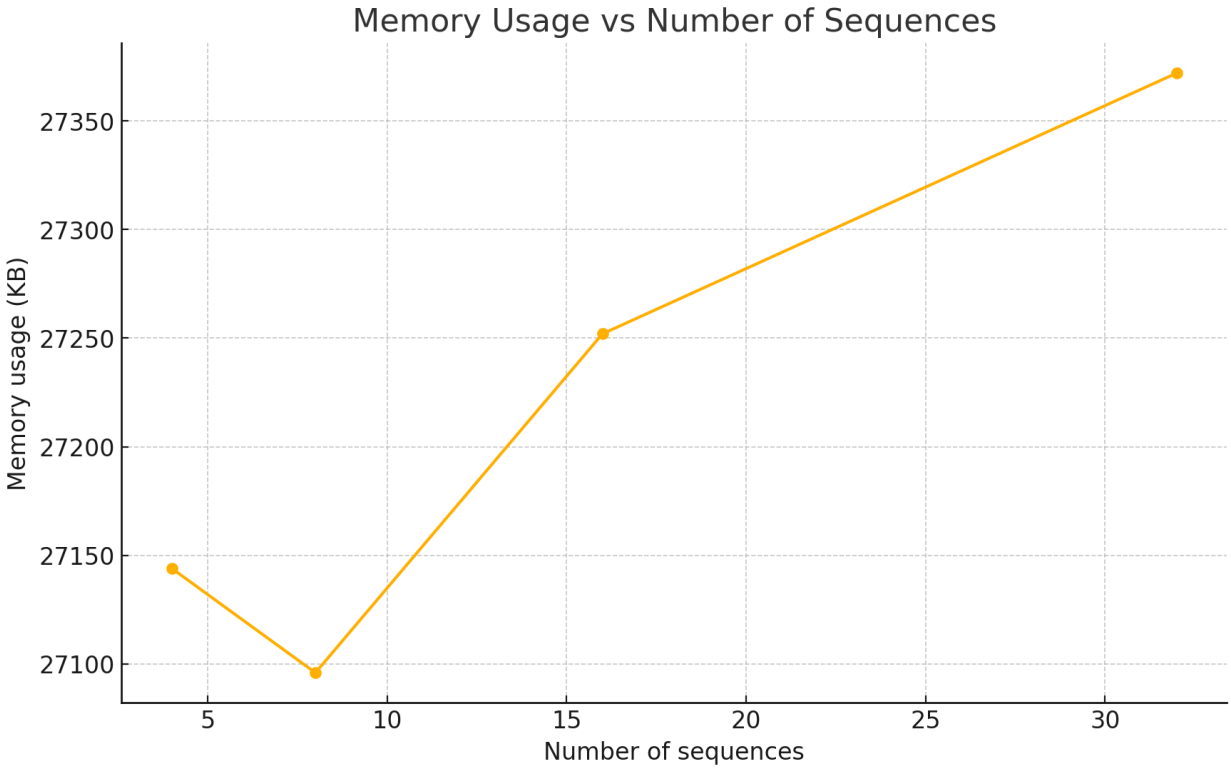


Table 2. Varying Sequence Length (16 sequences)

Sequence Length	Execution Time (seconds)	Memory(KB)
100	0.01	4128
1000	0.79	27264
10000	66.95	2354840
100000	∞?	∞?

Figure 3. Execution time vs. sequence length (Table 2 data).

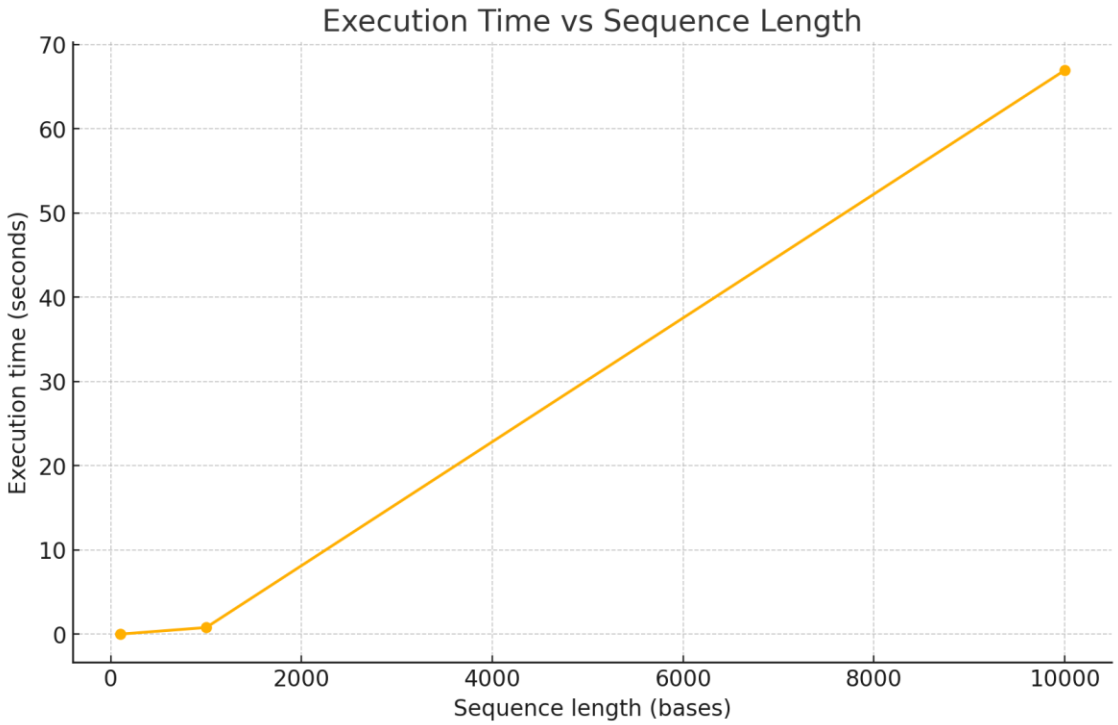
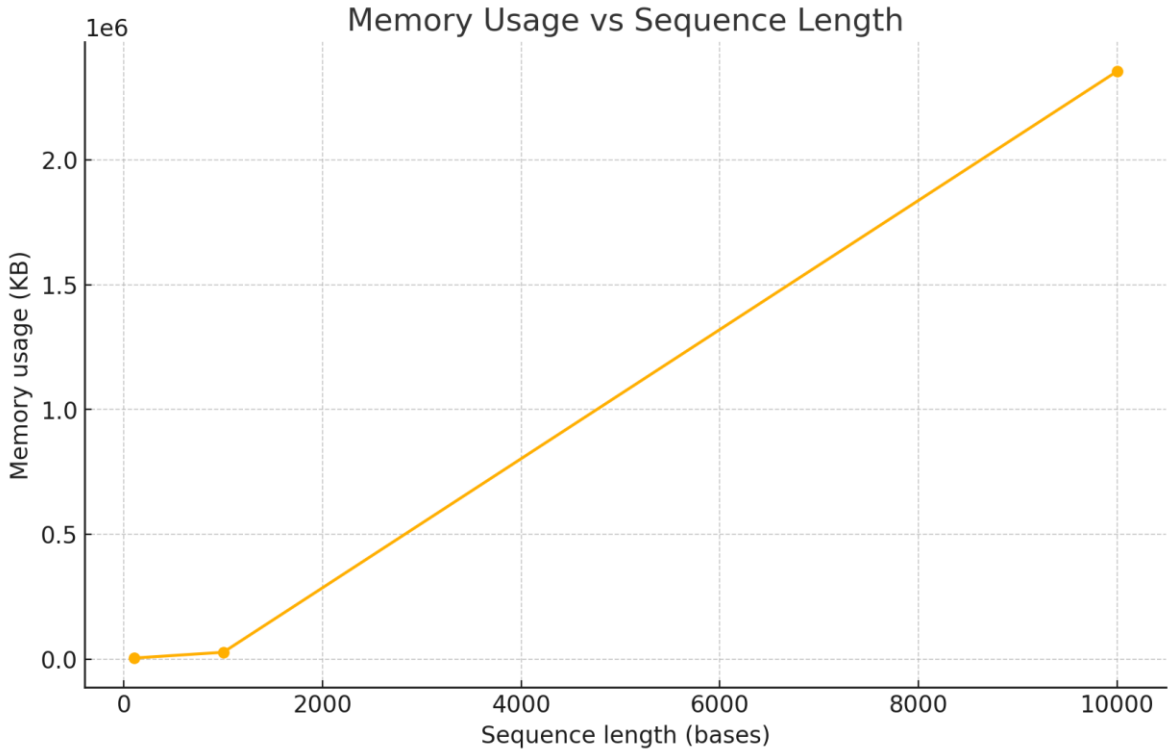


Figure 4. Memory usage vs. sequence length (Table 2 data).



Discussion

Scaling with Sequence Count

Execution time grows approximately linearly as sequences increase: $4 \rightarrow 8$ sequences see 0.07 s to 0.17 s (~2.4×), $8 \rightarrow 16$ gives 0.17 s to 0.86 s (~5×), and $16 \rightarrow 32$ yields 0.86 s to 3.10 s (~3.6×). Memory usage remains nearly constant around 27 MB (27144 KB to 27372 KB), since the DP matrices scale with each pairwise alignment of fixed length.

Scaling with Sequence Length

Execution time exhibits quadratic behavior in sequence length: $100 \rightarrow 1000$ bases jumps from 0.01 s to 0.79 s (~79×), and $1000 \rightarrow 10000$ bases from 0.79 s to 66.95 s (~85×). Memory usage likewise increases from ~4 MB (4128 KB) at 100 bp to ~27 MB (27264 KB) at 1000 bp to ~2.35 GB (2354840 KB) at 10000 bp. The run at 100000 bases failed due to insufficient RAM, highlighting the $O(L^2)$ memory demands of full DP matrices.

Performance Bottlenecks

Affine-gap alignment requires three $O(L_1 \cdot L_2)$ matrices (V, F, E), so both time and space are quadratic in sequence length. As length grows, matrix allocation dominates resource usage.

For ultra-long sequences $(\geq 10\,k)$, consider:

- Banded alignment, reducing the DP band to $O(k \cdot L)$ time/space at the cost of missing large indels.
- Divide-and-conquer (Hirschberg's algorithm) to lower memory to $O(L)$, albeit with recursive runtime overhead.

Conclusion

My benchmarking confirms that the center-star MSA with affine-gap penalties scales linearly in the number of sequences and quadratically in sequence length, in line with theoretical time/space complexity. Practical runs up to 10 kb sequences and 32 sequences completed in seconds to minutes with under 3 GB RAM. However, attempting 100 kb sequences exhausted available memory, underscoring the need for space-efficient techniques for very long inputs. In future work, integrating banded alignment or Hirschberg's algorithm could make the method feasible for genome-scale data.