Healthcare Application: DNA Sequence Alignment for Genetic Matching

1. Problem Description

Given two DNA sequences (e.g., from a patient and a reference genome, or from two patients with a genetic disease), find the **optimal way to align them** — identifying matches, mismatches, insertions, and deletions. This tells us:

- Genetic similarity
- Disease-causing mutations
- Evolutionary relationships
- Donor-recipient compatibility (e.g., bone marrow transplant)

2. Algorithm Design

Selected paradigm =>Dynamic Programming

Classic DP Algorithm: Needleman–Wunsch (Global Alignment)

Scoring Scheme

- Match: `+1`
- Mismatch: `-1`
- Gap penalty (insertion/deletion): `-2`

DP Table Definition

Let `dp[i][j]` = **maximum alignment score** between:

- First `i` characters of sequence A
- First `j` characters of sequence B

Recurrence Relation

t

```
dp[i][j] = max(
    dp[i-1][j-1] + score(A[i], B[j]),   # match/mismatch
    dp[i-1][j] + gap_penalty,            # gap in B (deletion in A)
```

```
    dp[i][j-1] + gap_penalty            # gap in A (insertion in B)
)
```

Base Cases

```
text
dp[0][0] = 0
dp[i][0] = i * gap_penalty   # all gaps
dp[0][j] = j * gap_penalty
```

Concrete Healthcare Example: BRCA1 Gene Mutation Detection

Patient DNA segment:
A T G C T A G C

Reference BRCA1 segment:
A T G G T A G C
(healthy)

DP Table Calculation

	-	A	T	G	G	T	A	G	C
-	0	-2	-4	-6	-8	-10	-12	-14	-16
A	-2	1	-1	-3	-5	-7	-9	-11	-13
T	-4	-1	2	0	-2	-4	-6	-8	-10
G	-6	-3	0	3	1	-1	-3	-5	-7
C	-8	-5	-2	1	2	0	-2	-4	-4
T	-10	-7	-4	-1	1	3	1	-1	-3
A	-12	-9	-6	-3	-1	1	4	2	0
G	-14	-11	-8	-5	-2	0	2	5	3
C	-16	-13	-10	-7	-4	-1	1	3	6

Result:
Optimal alignment score = **6** (max at bottom-right)

Pseudocode:

```
ALGORITHM NeedlemanWunsch(seqA, seqB, matchScore, mismatchPenalty, gapPenalty)
    INPUT:  seqA, seqB - DNA strings (A, T, G, C)
            matchScore - reward for matching characters (e.g., +1)
            mismatchPenalty - penalty for mismatches (e.g., -1)
            gapPenalty - penalty for insertions/deletions (e.g., -2)
    OUTPUT: alignmentScore - optimal alignment score
            alignedA, alignedB - aligned sequences with gaps
```

```
m = LENGTH(seqA)
n = LENGTH(seqB)

// Initialize DP table with dimensions (m+1) x (n+1)
CREATE TABLE dp[m+1][n+1]

// Base cases: aligning with empty sequence = all gaps
FOR i = 0 TO m:
    dp[i][0] = i * gapPenalty

FOR j = 0 TO n:
    dp[0][j] = j * gapPenalty

// Fill DP table
FOR i = 1 TO m:
    FOR j = 1 TO n:
        // Case 1: align seqA[i] with seqB[j]
        IF seqA[i] == seqB[j]:
            diagScore = dp[i-1][j-1] + matchScore
        ELSE:
            diagScore = dp[i-1][j-1] + mismatchPenalty

        // Case 2: gap in seqB (delete from seqA)
        gapInB = dp[i-1][j] + gapPenalty

        // Case 3: gap in seqA (insert into seqA)
        gapInA = dp[i][j-1] + gapPenalty

        dp[i][j] = MAX(diagScore, gapInB, gapInA)

alignmentScore = dp[m][n]

// Traceback to reconstruct alignment
i = m
j = n
alignedA = ""
alignedB = ""

WHILE i > 0 OR j > 0:
    IF i > 0 AND j > 0 AND dp[i][j] == dp[i-1][j-1] +
       (matchScore IF seqA[i] == seqB[j] ELSE mismatchPenalty):
        // Diagonal move = match or mismatch
        alignedA = seqA[i] + alignedA
        alignedB = seqB[j] + alignedB
        i = i - 1
        j = j - 1

    ELSE IF i > 0 AND dp[i][j] == dp[i-1][j] + gapPenalty:
        // Up move = gap in seqB
        alignedA = seqA[i] + alignedA
        alignedB = "-" + alignedB
        i = i - 1

    ELSE:
```

```
        // Left move = gap in seqA
        alignedA = "-" + alignedA
        alignedB = seqB[j] + alignedB
        j = j - 1

    RETURN alignmentScore, alignedA, alignedB
```

3. Analysis and Justification

- **Overlapping subproblems**: Aligning prefixes of the sequences can be reused.
- **Optimal substructure**: Best alignment of whole sequences = best alignment of prefixes + last character decision.
- **Handles gaps**: Insertions/deletions (indels) are naturally modeled.

Algorithm	Time Complexity	Space Complexity	Best Use Case
Needleman-Wunsch	O(mn)	O(mn)	Short genes, mutation detection

4. Self-Learning Reflection

- What additional resources you used to learn independently.

    1. *Biological Sequence Analysis* by Durbin, Eddy, Krogh, Mitchison – definitive resource for pairwise alignment algorithms
    2. *Introduction to Algorithms (CLRS)* – Chapter 15 (Dynamic Programming) – theoretical foundation
    3. **B. Needleman and Christian D. Wunsch in 1970** in the Journal of Molecular Biology (Volume 48, Issue 3, pages 443–53)

What challenges you faced while designing the algorithm.

1. Understanding the DP Table Filling Logic

- **The problem**: I kept mixing up rows and columns. When filling `dp[i][j]`, I wasn't sure if `i` was from sequence A or sequence B.
- **How I solved it**: Drew a small 3x3 table on paper and manually filled it step by step. This made the indexing clear.

2. Getting the Base Cases Right

- **The problem**: Forgot to initialize the first row and column with gap penalties. My alignments were missing gaps at the beginning.
- **How I solved it**: Traced through the code with a debugger and saw `dp[0][2]` was 0 instead of `2 * gap_penalty`.

3. Traceback Was Confusing

- **The problem**: After filling the table, I didn't know how to walk backwards to get the actual alignment. I kept going in circles.
- **How I solved it**: Used three separate if-statements (diagonal, up, left) instead of trying to combine them. Also stored directions in a separate table.

What new knowledge or skills you gained.

Problem-Solving Lessons

1. **Start simple, then improve** – Got basic DP working first, then added optimizations
2. **Visualize to debug** – Printing DP tables as grids helped find off-by-one errors
3. **Test with fake data first** – Generated artificial DNA with known mutations to verify correctness
4. **Real data is messy** – Always add input validation