Sequence Alignment: Linear Space

Q. Can we avoid using quadratic space?

- Easy. Optimal value in $O(m + n)$ space and $O(mn)$ time.
  - Compute $OPT(i, *)$ from $OPT(i-1, *)$.
  - No easy way to recover alignment itself.

- Optimal longest common subsequence in $O(m + n)$ space and $O(mn)$ time [Hirschberg (1975)].
  - Clever combination of divide-and-conquer and dynamic programming.

- Application to sequence alignment:

Divide and Conquer Algorithms

- Divide problem into sub-problems
- Conquer by solving sub-problems recursively. If the sub-problems are small enough, solve them in brute force fashion
- Combine the solutions of sub-problems into a solution of the original problem

Sorting

- Given: an unsorted array
  \[ 5 2 4 7 1 3 2 6 \]
- Goal: sort it
  \[ 1 2 3 4 5 6 7 \]

Mergesort: Divide

\[ 5 2 4 7 1 3 2 6 \]
\[ 5 2 4 7 1 3 2 6 \]
\[ 5 2 4 7 1 3 2 6 \]
\[ 5 2 4 7 1 3 2 6 \]

$log(n)$ divisions to split an array of size $n$ into single element arrays

Mergesort: Conquer

\[ 5 2 4 7 1 3 2 6 \]
\[ 2 5 4 7 1 3 2 6 \]
\[ 2 4 5 7 1 3 2 6 \]
\[ 2 4 5 7 1 3 2 6 \]
\[ 1 2 3 4 5 6 7 \]
\[ 1 2 3 4 5 6 7 \]

$log(n)$ iterations, each iteration takes $O(n)$ time
Total time: $O(n \log n)$
**MergeSort**

**Merge Step**

Merging:

1. 2 4 5 7 1 2 3 6
2. 1 2 3 4 5 6 7
3. Etcetera...

- 2 sorted arrays of size \(n\) and \(m\) can be merged in \(O(n+m)\) time to form a sorted array of size \(n+m\).

**MergeSort Algorithm**

```plaintext
MergeSort(c):
    n ← size of array c
    if n = 1
        return c
    left ← list of first \(n/2\) elements of c
    right ← list of last \(n-n/2\) elements of c
    sortedLeft ← MergeSort(left)
    sortedRight ← MergeSort(right)
    sortedList ← Merge(sortedLeft, sortedRight)
    return sortedList
```

**MergeSort: Example**

- **Divide**
  - 20 4 7 6 1 3 9 5
  - 1 2 3 4

- **Conquer**
  - 4 6 7 20
  - 1 2 3 5 9

**MergeSort: Running Time**

- in the \(i\)th iteration we do \(O(n)\) work
- number of iterations is \(O(\log n)\)
- running time: \(O(n \log n)\)

**The Problem: Computing Alignment Path**

- Requires Quadratic Memory

**Alignment Path**

- Space complexity for computing alignment path for sequences of length \(n\) and \(m\) is \(O(nm)\)
- We need to keep all backtracking references in memory to reconstruct the path (backtracking)
Computing Alignment Score using Linear Memory

- Space complexity of computing just the score itself is $O(n)$
- Only need the previous column to calculate the current column

Finding the Middle Point

And Again

Crossing the Middle Line

Define:

\[
\text{score}(i) = \text{the score of the optimal path from } (0, 0) \text{ to } (i, m) \text{ that passes through } (i, m/2)
\]

\[
\text{mid} = \arg\min_{0 \leq i \leq n} \text{score}(i)
\]

(mid, m/2): the position where the optimal path crosses the middle column.
Crossing the Middle Line

score(i) = prefix(i) + suffix(i)

prefix(i): score of the optimal alignment of a length m/2 prefix of y to a prefix of x (takes a path from (0,0) to (\(i, m/2\))
suffix(i): score of the optimal alignment of a length m/2 suffix of y to a suffix of x (takes a path from (\(i, m/2\)) to (\(n, m\)))

Computing prefix(i)

• prefix(i): length of the longest path from (0,0) to (\(i, m/2\))
• Compute prefix(i) by dynamic programming in the left half of the matrix

Finding the Middle Point

store prefix(i) column

And Again

store suffix(i) column

And Again
**Time = Area: First Pass**
- On first pass, the algorithm covers the entire area
  \[ \text{Area} = n \times m \]

**Time = Area: Second Pass**
- On second pass, the algorithm covers only 1/2 of the area
  \[ \text{Area}/2 \]

**Geometric Reduction At Each Iteration**
- \[ 1 + \frac{1}{2} + \frac{1}{4} + \ldots + \left(\frac{1}{2}\right)^k \leq 2 \]
- Runtime: \( O(\text{Area}) = O(nm) \)

**Geometric Reduction At Each Iteration**
- First pass: 1
- Second pass: 1/2
- Third pass: 1/4
- Fourth pass: 1/8
- Fifth pass: 1/16

**Run Time Analysis**
- Let \( T(m, n) \) = max running time of algorithm on strings of length \( m \) and \( n \).
- \( O(nm) \) time to compute \( \text{prefix}(\bullet, m/2) \) and \( \text{suffix}(\bullet, m/2) \) and find midpoint \( q \).
- \( T(q, m/2) + T(n - q, m/2) \) time for two recursive calls.
- Choose constant \( c \) so that:
  - \( T(m, 2) \leq 2cm \)
  - \( T(2, n) \leq 2cn \)
  - \( T(m, n) \leq cmn + T(q, m/2) + T(n-q, m/2) \)

- Claim: \( T(m, n) \leq 2cmn \) (proof by induction)
Is it Possible to Align Sequences in Subquadratic Time?

- Dynamic programming takes $O(n^2)$ for various alignment methods
- Can we do better?
- Yes: The Four-Russians Speedup (works for LCS but not for general sequence alignment problem) $O(n^2/\log n)$