MTH-681 Analysis and Design of Algorithms
Assignment (3): Divide and Conquer
The Master Theorem

Mostafa Hassanein

31 March 2025

3-5

Consider an $m \times n$ matrix of numbers, A, having the property that, for every $1 \leq i < j \leq m$ and $1 \leq k < l \leq n$, $A_{ik} + A_{jl} \leq A_{il} + A_{jk}$.
Let $lm(i)$ be the index of the column containing the leftmost minimum number of row i.

a) Prove that, for every $1 \leq i < m, lm(i) \leq lm(i+1)$.

b) Describe a $O(m + nlog(m))$ divide-and-conquer algorithm that computes lm for all rows of A.

Solution:

a)

Proof. (By Contradiction)
Suppose for the sake of contradiction that $\forall i, j, k, l$:

$$1 \leq i < j \leq m \quad \wedge \quad 1 \leq k < l \leq n \implies A_{ik} + A_{jl} \leq A_{il} + A_{jk}$$

but $lm(i) > lm(i+1)$.

Let $j = i + 1$, $k = lm(i+1) = lm(j)$, $l = lm(i)$

$\implies$ A_{il} is the minimum element of row i, and A_{jk} is the minimum element of row j.

$\implies A_{il} \leq A_{ik} \quad \wedge \quad A_{jk} \leq A_{jl}$

$\implies A_{il} + A_{jk} \leq A_{ik} + A_{jl}$

This is a contradiction to the assumption that $A_{ik} + A_{jl} \leq A_{il} + A_{jk}$.
Therefore, we must conclude that: $lm(i) \leq lm(i+1)$.

□

b)

```
// @code{matrix} is an mxn matrix with the propery stated at the
    problem statement.
int[] leftMostMins(int[][] matrix) {
  int[] result = new int[matrix.length];
  return helper(matrix, 0, matrix.length - 1, 0, matrix[0].length - 1,
      result);
}

// Computes the left-most mins for @code{matrix} by recursing on the
    rows of the matrix and collecting the results into the
    @code{result} variable.
int[] helper(int[][] matrix, int lowRowIndex, int highRowIndex, int
    lowColumnIndex, int highColumnIndex, int[] result) {
  // Base case 1.
  if (lowRowIndex > highRowIndex) {
    return result;
  }

  // Base case 2.
  if (lowRowIndex == highRowIndex) {
    result[lowRowIndex] =
        findLeftMostMinIndexforSequence(matrix[lowRowIndex],
        lowColumnIndex, highColumnIndex);
    return result;
  }

  // Recursive step.
    // 1. Divide: find the index of the leftMostMin for midRow
  int midRowIndex = (lowRowIndex + highRowIndex) / 2;
  int midRowLeftMostMinIndex =
      findLeftMostMinIndexforSequence(matrix[midRowIndex],
      lowColumnIndex, highColumnIndex);

    // 2. Conquer: solve 2 smaller sub-problems
  helper(matrix, lowRowIndex, midRowIndex-1, lowColumnIndex,
      midRowLeftMostMinIndex, result);
  helper(matrix, midRowIndex+1, highRowIndex, midRowLeftMostMinIndex,
      highColumnIndex, result);


    // 3. Combine
  result[midRowIndex] = midRowLeftMostMinIndex;

  return result;
}
```

```
// Any O(searchEndIndex - searchStartIndex) linear search algorithm.
int findLeftMostMinIndexforSequence(int[] sequence, int
    searchStartIndex, int searchEndIndex) {
  int result = searchEndIndex;
  int min = Integer.MAX_VALUE;
  for (int i = searchEndIndex; i >= searchStartIndex; i--) {
    if (sequence[i] <= min) {
      min = sequence[i];
      result = i;
    }
  }
  return result;
}
```

Correctness

Proof. (By Strong Induction)

Base case 1: $m = 0$

In this case the top call to helper is passed $lowRowIndex = 0$ and $highRowIndex = -1$, which hits base case 1 and correctly returns an empty array.

Base case 2: $m = 1$

In this case the top call to helper is passed $lowRowIndex = 0$ and $highRowIndex = 0$, which hits base case 2.

Inside base case 2, it correctly finds and sets the left-most min for that single row and returns the result.

Inductive step: $m > 1$

In the recursive step, the top call to helper is passed $lowRowIndex = 0$ and $highRowIndex > 0$, which hits the recursive step.

It then solves the problem at the middle row of the matrix, correctly sets the result for the middle row, and recurses on 2 smaller sub-problems, which by the induction hypothesis also set the result correctly for the remaining rows.

□

Average Case Time Complexity Analysis: $A_t(m, n)$

For simplicity, we do an average-case analysis assuming that $midRowLeftMostMinIndex$ is equally likely to fall anywhere between the $lowColumnIndex$ and $highColumnIndex$, and thus the left and right sub-problems are of nearly equal sizes.

The solution is then described by the following recurrence equation:

$$T(m, n) = 2T(\frac{m}{2}, \frac{n}{2}) + \Theta(n) + \Theta(1).$$

Notes:

1. The $\Theta(n)$ is contributed by the divide part of the algorithm.

2. The $\Theta(1)$ is included because we have to touch each row at least once.

The Master theorem does not apply straightforwardly to this formula, but it does apply to:

$$T_1(m, n) = 2T_1(\frac{m}{2}, \frac{n}{2}) + \Theta(n).$$

In this case: $a = 2$, $b = 2$, $f(n) = \Theta(n)$
$\implies \lg_b a = \lg_2 2 = 1$
$\implies n^{\lg_b a} = n^1 = n.$
$\implies f(n) = \Theta(n^{\lg_b a})$
$\implies$ (Case 2)
$\implies T_1(m, n) = \Theta(n^{\lg_b a} \lg m) = \Theta(n \lg m).$

Finally, we add the effect of the $\Theta(1)$ term in the original recurrence equation which gives an additional $\Theta(m)$ term in the solution:

$$T(m, n) = \Theta(n \lg m) + \Theta(m).$$