Hudson Cho, Ryan Wilson, Jesse Washburn, Colin Shuster, Samhith Patibandla
COSC 336
03/4/2025

Assignment 3

Instructions.

1. Due date and time: As indicated on Blackboard.

2. This is a team assignment. Work in teams of 3-4 students. Submit on Blackboard one assignment per team, with the names of all students making the team.

3. The exercises will not be graded, but you still need to present your best attempt to solve them. If you do not know how to solve an exercise, say it. This will give me feedback about your understanding of the theoretical concepts.

4. Your programs must be written in Java.

5. Write your programs neatly - imagine yourself grading your program and see if it is easy to read and understand.

   Comment your programs reasonably: there is no need to comment lines like "i++" but do include brief comments describing the main purpose of a specific block of lines.

6. You will submit on **Blackboard** 3 files.

   The **1-st file** is a pdf file (produced ideally with latex and Overleaf) and it will contain the following:

   (a) The solution to the Exercises (see the remark above).

   (b) A short description of your algorithms for the Programming Task1, where you explain your algorithms. Focus on how you have modified MERGE (for task 1), and on the relations between subproblems for the dynamic algorithm (for task 2)..

   (c) Tables clearly labeled with the results your programs give for the data sets indicated for the programming tasks.

   (d) The java code (so that the grader can make observations) of the 2 programs (for task 1 and for task 2).

   The **2-nd file** is the .java file containing the java source code for Programming Task 1.

   The **3-rd file** is the .java file containing the java source code for Programming Task 2.

Exercise 1. Analyze the following recurrences using the method that is indicated. In case you use the Master Theorem, state what the corresponding values of a, b, and $f(n)$ are and how you determined which case of the theorem applies.

- $T(n) = 3T\left(\frac{n}{4}\right) + 3$. Use the Master Theorem to find a $\Theta()$ evaluation, or say "Master Theorem cannot be used", if this is the case.
- $T(n) = 2T\left(\frac{n}{2}\right) + 3n$. Use the Master Theorem to find a $\Theta()$ evaluation, or say "Master Theorem cannot be used", if this is the case.
- $T(n) = 9T\left(\frac{n}{3}\right) + n^2 \log n$. Use the Master Theorem to find a $\Theta()$ evaluation, or say "Master Theorem cannot be used", if this is the case.

Answers:

1. For $T(n) = 3T\left(\frac{n}{4}\right) + 3$:

   $$a = 3, \quad b = 4, \quad f(n) = 3.$$

   We compute

   $$n^{\log_4 3}.$$

   Since $f(n) = 3 = \Theta(1)$, and $\Theta(1) = O\left(n^{\log_4 3 - \epsilon}\right)$ for any $\epsilon > 0$, we are in Case 1 of the Master Theorem. So,

   $$\boldsymbol{T(n) = \Theta\left(n^{\log_4 3}\right)}$$

2. For $T(n) = 2T\left(\frac{n}{2}\right) + 3n$:

   $$a = 2, \quad b = 2, \quad f(n) = 3n.$$

   We compute

   $$n^{\log_2 2} = n.$$

   Since $f(n) = \Theta(n)$, which is the same as $n^{\log_2 2}$, we are in **Case 2** of the Master Theorem. So,

   $$\boldsymbol{T(n) = \Theta(n \log n).}$$

3. For $T(n) = 9T\left(\frac{n}{3}\right) + n^2 \log n$:

   $$a = 9, \quad b = 3, \quad f(n) = n^2 \log n.$$

   We compute

   $$n^{\log_3 9} = n^{\log_3 (3^2)} = n^2.$$

   Comparing $f(n)$ with $n^{\log_b a}$:

   $$\frac{f(n)}{n^2} = \frac{n^2 \log n}{n^2} = \log n.$$

   Since the extra factor is only $\log n$ (which is not a polynomial factor), the difference between $f(n)$ and n^2 is less than a polynomial factor.

The standard Master Theorem requires that $f(n)$ be either polynomially smaller or larger than $n^{\log_b a}$ (i.e., differing by a factor of n^ϵ for some $\epsilon > 0$). Since $\log n$ grows slower than any positive power of n, **the standard Master Theorem cannot be used.**

Exercise 2.

- $T(n) = 2T(n-1)+1$, with $T(0) = 1$. Use the iteration method to find a $\Theta()$ evaluation for $T(n)$.
- $T(n) = T(n-1)+1$, with $T(0) = 1$. Use the iteration method to find a $\Theta()$ evaluation for $T(n)$.
- Give a $\Theta(\cdot)$ evaluation for the runtime of the following code:

```
i = n
while(i >= 1) {
    for (j = 1; j <= n; j++)
        x = x + 1
    i = i/2
}
```

- Give a $\Theta(\cdot)$ evaluation for the runtime of the following code:

```
i = n
while(i >= 1) {
    for (j = 1; j <= i; j++)
        x = x + 1
    i = i/2
}
```

Answers:

1. For $T(n) = 2T(n-1) + 1$, with $T(0) = 1$:

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2\Big[2T(n-2) + 1\Big] + 1 = 2^2 T(n-2) + 2 + 1 \\ &= 2^3 T(n-3) + 2^2 + 2 + 1 \\ &\vdots \\ &= 2^n T(0) + \sum_{i=0}^{n-1} 2^i \\ &= 2^n + (2^n - 1) \\ &= 2^{n+1} - 1. \end{aligned}$$

   So,

$$\boldsymbol{T(n) = \Theta(2^n).}$$

2. For $T(n) = T(n-1) + 1$, with $T(0) = 1$:

$$\begin{aligned} T(n) &= T(n-1) + 1 \\ &= T(n-2) + 1 + 1 \\ &\vdots \\ &= T(0) + n \\ &= 1 + n. \end{aligned}$$

   So,

$$\boldsymbol{T(n) = \Theta(n).}$$

3. For the following code:

```
i = n
while(i >= 1) {
    for (j = 1; j <= n; j++)
        x = x + 1
    i = i/2
}
```

   the outer loop runs $\Theta(\log n)$ times (since i is halved each time) and the inner loop runs $\Theta(n)$ times per iteration. So, the total runtime is:

$$\boldsymbol{\Theta(n \log n).}$$

4. For the following code:

```
i = n
while(i >= 1) {
    for (j = 1; j <= i; j++)
        x = x + 1
    i = i/2
}
```

the outer loop iterates $\Theta(\log n)$ times with $i = \frac{n}{2^k}$ in the kth iteration, and the inner loop executes $\Theta\left(\frac{n}{2^k}\right)$ operations. So, the total work is:

$$\sum_{k=0}^{\lfloor \log_2 n \rfloor} \frac{n}{2^k} = n + \frac{n}{2} + \frac{n}{4} + \cdots,$$

a geometric series with first term $a = n$ and common ratio $r = \frac{1}{2}$ that sums to $\frac{n}{1-\frac{1}{2}} = 2n$, so:

$$\boldsymbol{T(n) = \Theta(n).}$$

Programming Task 1.: The input is an array $a_1, a_2, \ldots, a_n$ of numbers. A *UP-pair* is a pair (a_i, a_j) so that $1 \leq i < j \leq n$ and $a_i < a_j$. The task is to count the number of UP-pairs in the array. The ideal solution is to use a modified merge-sort algorithm that counts the number of UP-pairs as it merges the elements. The code first calls a wrapper method, countUpPairs(A), which calculates the initial values of p and r for array *A[]*. The code then recursively calls upon the divideAndCount(*A, p, r*) method, first calculating the index of the midpoint q as $p + \frac{r-p}{2}$. The method recursively calls upon the left and right half to split them until $p >= r$, or until there is only 1 element in the given segment of the array (No UP-pairs in an array with length 1 or 0). After the array has been split down to its smallest segments, divideAndCount() calls upon mergeAndCount(A, p, q, r) to count the number of UP-pairs in a given segment. It works the same as the standard merge sort algorithm the main difference here is that while merging, if $L[i] < R[j]$ count is incremented by $nR - j$ where nR is the length of the right array segment and j is the current index of the right array segment. This works because in a given sorted array, if $L[i] < R[j]$, in otherwords *L[i]* forms a UP-pair with *R[j]*, then all other elements of *R[j]* also form a UP-pair with *L[i]* because *R[j]* is sorted and all elements of R come after the element *L[i]*. This way the algorithm runs in $O(n \log n)$ time as opposed to the approach using nested loops which runs in $O(n^2)$.

Input	**Output**
7, 3, 8, 1, 5	4 UP-pairs [1, 3, 5, 7, 8]
input-3.4.txt	248339 UP-pairs [0, 1, 3, 4, 4, 5, 8, 9, . . . , 984, 987, 990, 995, 999]
input-3.5.txt	24787869 UP-pairs [2, 3, 3, 4, 4, 6, 8, 9, . . . , 9996, 9997, 9998, 9999, 9999]

Raw Code for Programming Task 1

```
import java.util.*;
import java.io.*;

public class Asgmnt3Task1 {
    public static void main(String[] args) throws FileNotFoundException  {
        int[] arr = {7, 3, 8, 1, 5};
        System.out.println("Data Set 1:");
        // countUpPairs sorts the array and returns the number of UP-pairs.
        System.out.println("UP-Pairs count: " + countUpPairs(arr));
        System.out.println("Sorted array: " + Arrays.toString(arr));
        System.out.println();

        // Prompt the user to enter filenames for additional data sets
        Scanner console = new Scanner(System.in);
        System.out.println("Enter filenames (separated by spaces):");
        String inputLine = console.nextLine().trim();
        if (!inputLine.isEmpty()) {
            // Split the input string into individual filenames
            String[] filenames = inputLine.split("\\s+");
            for (String filename : filenames) {
                try {
                    Scanner fileScanner = new Scanner(new File(filename));
                    // The first integer in the file is the number of
    elements
                    int n = fileScanner.nextInt();
                    int[] fileInputArr = new int[n];
                    // Read the next n integers from the file into the array
                    for (int i = 0; i < n; i++) {
                        fileInputArr[i] = fileScanner.nextInt();
                    }
                    fileScanner.close();
                    // Process the array: sort it and count the UP-pairs
    using the modified merge sort
                    int count = countUpPairs(fileInputArr);
                    System.out.println("File: " + filename);
                    System.out.println("UP-Pairs count: " + count);
                    System.out.println("Sorted array: " + Arrays.toString(
    fileInputArr));
                    System.out.println();
                } catch (FileNotFoundException e) {
                    System.err.println("File not found: " + filename);
                }
            }
        }
    }

    /**
     * countUpPairs:
     * Initiates the modified merge sort that counts UP-pairs.
     *
     * @param A the input array of numbers.
     * @return the total number of UP-pairs in the array.
     */
    public static int countUpPairs(int[] A) {
```

```
52          return divideAndCount(A, 0, A.length - 1);
53      }
54
55      /**
56       * divideAndCount:
57       * Recursively divides the array into halves, counts the UP-pairs in
58       * each half, and then counts the UP-pairs that cross the two halves
        during the merge.
59       *
60       * @param A the array to process.
61       * @param p the starting index.
62       * @param r the ending index.
63       * @return the number of UP-pairs in the subarray A[p..r].
64       */
65      public static int divideAndCount(int[] A, int p, int r) {
66          if (p >= r)
67              return 0;
68          int q = p + ((r - p) / 2);
69          // Recursively count UP-pairs in the left half, right half, and
        across the halves.
70          return divideAndCount(A, p, q)
71                  + divideAndCount(A, q + 1, r)
72                  + mergeAndCount(A, p, q, r);
73      }
74
75      /**
76       * mergeAndCount:
77       * Merges two sorted subarrays A[p...q] and A[q+1...r] while counting UP
        -pairs.
78       * If an element in the left subarray (L) is less than an element in the
         right subarray (R),
79       * then all remaining elements in R will form UP-pairs with that element
         from L.
80       *
81       * @param A the array containing the two subarrays.
82       * @param p the starting index of the first subarray.
83       * @param q the ending index of the first subarray.
84       * @param r the ending index of the second subarray.
85       * @return the number of UP-pairs counted during the merge.
86       */
87      public static int mergeAndCount(int[] A, int p, int q, int r) {
88          int nL = q - p + 1; // Length of left subarray
89          int nR = r - q;     // Length of right subarray
90
91          int[] L = new int[nL];
92          int[] R = new int[nR];
93
94          // Copy A[p...q] into L and A[q+1...r] into R
95          for (int i = 0; i < nL; i++) {
96              L[i] = A[p + i];
97          }
98          for (int j = 0; j < nR; j++) {
99              R[j] = A[q + j + 1];
100         }
101         int i = 0, j = 0, k = p, count = 0;
```

```
102         // Merge the two sorted arrays back into A while counting UP-pairs.
103         // When L[i] < R[j], all elements after R[j] form an UP-pair with L[
    i].
104         while (i < nL && j < nR) {
105             if (L[i] < R[j]) {
106                 count += (nR - j);  // All remaining elements in R are
    greater than L[i]
107                 A[k++] = L[i++];
108             } else {
109                 A[k++] = R[j++];
110             }
111         }
112         // Copy any remaining elements of L into A.
113         while (i < nL) {
114             A[k] = L[i];
115             i++;
116             k++;
117         }
118         // Copy any remaining elements of R into A.
119         while (j < nR) {
120             A[k] = R[j];
121             j++;
122             k++;
123         }
124         // Return the count of UP-pairs found during the merge.
125         return count;
126     }
127 }
```

Programming Task 2.

The input is a 2-dimensional matrix cost[][] of integer numbers, with n rows labeled $0, 1, 2, \ldots, n-1$ and m columns labeled $0, 1, 2, \ldots, m-1$, and also a target cell (i, j). The task is to calculate the minimum cost path to reach cell (i, j) from cell (0, 0). Each cell of the matrix represents a cost to traverse through that cell. The total cost of a path to reach (i,j) is the sum of all the costs on that path (including both source and destination). We can only traverse down, right, and diagonally lower from a given cell. Formally, from a given cell (i, j) one can move to one of the following cells:

(i+1, j) (down)

(i, j+1) (right)

and the diagonal moves are to (i+1, j+1), or to (i+1,j-1).

Example: the input is the matrix cost below and the target cell $(2, 1)$.

1	2	3
4	8	1
1	5	3

Then the minimum cost to go from cell (0,0) to the target cell (2,1) is 9, corresponding to the path (0,0) – (0,1) – (1,2) – (2,1).

Design a dynamic programming algorithm that solves this problem. The input should have the following format

line 1 consists of 4 numbers: n, m, i and j.

This is followed by n rows each one containing m numbers, giving the cost table.

For instance for the above problem the input is

3,3,2,1

1,2,3

4,8,1

1,5,3

Test your program and report in a table the results for the following data sets.

Data set 1: The above example

Data set 2: The numbers from the file input-3.6, available on Blackboard.

Data set 3: The numbers from the file input-3.7, available on Blackboard.

Programming Task 2.: The input is a 2-dimensional matrix $cost[][]$ of integers with n rows and m columns, along with a target cell (i, j). Each cell represents the cost to traverse that cell, and the total cost of a path is the sum of the costs along the path from the source cell $(0, 0)$ to the target cell. The goal is to compute the minimum cost path using a dynamic programming algorithm. To achieve this, the algorithm constructs a cost matrix $minInc[][]$, where each entry $minInc[r][c]$ stores the minimum cumulative cost to reach cell (r, c) from $(0, 0)$. Initially, $minInc[0][0]$ is set to $cost[0][0]$, and the first row is filled by accumulating the costs from left to right, since only rightward moves are possible. For each subsequent row, the algorithm computes the minimum cost for each cell as the minimum among the cost of moving from directly above, diagonally from above-left, diagonally from above-right, and (for cells not in the first column) from the left. The final result is given by $minInc[i][j]$, which represents the minimum cost to reach the target cell. Input is provided either via a hard-coded matrix or through an input file (where the first line specifies n, m, i, and j, followed by the matrix rows). This dynamic programming approach runs in $O(nm)$ time.

'Assignment3_T2'

Integer Matrix Size : [3][3]

[1, 2, 3]

[4, 8, 1]

[1, 5, 3]

Min cost to cell (2, 1) : 9

Input File Name : input-3.6.txt

Integer Matrix Size : [6][6]

[3, 1, 1, 1, 1, 1]

[1, 4, 2, 3, 5, 1]

[9, 1, 2, 3, 4, 5]

[1, 7, 2, 5, 4, 4]

[1, 1, 1, 1, 1, 1]

[1, 7, 1, 7, 1, 7]

Min cost to cell (5, 5) : 16

Input File Name : input-3.7.txt

Integer Matrix Size : [8][5]

[1, 2, 3, 4, 5]

[5, 4, 3, 2, 1]

[1, 2, 3, 4, 5]

[5, 4, 3, 2, 1]

[1, 2, 3, 4, 5]

[5, 4, 3, 2, 1]

[1, 2, 3, 4, 5]

[5, 4, 3, 2, 1]

Min cost to cell (7, 4) : 20

Task 2

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.util.Scanner;
4
5  public class Assignment3_T2 {
6      public static int rowF = 0;
7      public static int colF = 0;
8
9      public static void main(String[] args) throws FileNotFoundException {
10         int[][] temp = new int[][]{ {1, 2, 3}, {4, 8, 1}, {1, 5, 3}};
11         rowF = 2;
12         colF = 1;
13         print(temp, temp.length, temp[0].length);
14         System.out.println("\nOptimize path (0, 0) to (" + rowF + ", " +
       colF + ")\n");
15         System.out.println("\nMin cost to cell  (" + rowF + ", " + colF + ")
        : " + costOpti(temp)+"\n");
16
17
18         int[][] x = fileInts();
19         print(x, x.length, x[0].length);
20         System.out.println("\nOptimize path (0, 0) to (" + rowF + ", " +
       colF + ")\n");
21         System.out.println("\nMin cost to cell  (" + rowF + ", " + colF + ")
        : " + costOpti(x)+"\n");
22
23         x = fileInts();
24         print(x, x.length, x[0].length);
25         System.out.println("\nOptimize path (0, 0) to (" + rowF + ", " +
       colF + ")\n");
26         System.out.println("\nMin cost to cell  (" + rowF + ", " + colF + ")
        : " + costOpti(x)+"\n");
27
28     }
29
30     public static int costOpti(int[][] x) {
31
32         int minInc[][] = new int[x.length][x[0].length];
33
34         // Fill edge row with right only movements//
35         minInc[0][0] = x[0][0];
36         for (int i = 1; i < x[0].length; i++) {// Case only right moves: Top
       //
37             int rightMoves = minInc[0][i - 1] + x[0][i];
38             minInc[0][i] = rightMoves;
39         }
40
41         // Process array by layer as each element in a row > 0 has group of
42         // Moves to choosen: cell0:(2), cell1:(4), cell2:(4), cell3:(4),
       ..., Last
43         // index:(3)
44
45         for (int k = 1; k < x.length; k++) {// all other rows//
46
```

```
47              for (int j = 0; j < x[k].length; j++) {// in line//
48
49                  if (j == 0) {// First index of row//
50                      int downMove = x[k][j] + minInc[k - 1][j];
51                      int diag = x[k][j] + minInc[k - 1][j + 1];
52                      minInc[k][j] = Math.min(downMove, diag);
53                  }
54
55                  else if (j < x[k].length - 1) {// Not at last index of row//
56                      int downMove = x[k][j] + minInc[k - 1][j];
57                      int diagUpRight = x[k][j] + minInc[k - 1][j + 1];
58                      int diagUpLeft = x[k][j] + minInc[k - 1][j - 1];
59                      int leftMoves = x[k][j] + minInc[k][j - 1];
60
61                      minInc[k][j] = Math.min(Math.min(diagUpRight, diagUpLeft
    ), Math.min(downMove, leftMoves));
62                  }
63
64                  else {// Last index of row//
65                      int downMove = x[k][j] + minInc[k - 1][j];
66                      int diagUpLeft = x[k][j] + minInc[k - 1][j - 1];
67                      int leftMoves = x[k][j] + minInc[k][j - 1];
68
69                      minInc[k][j] = Math.min(diagUpLeft, Math.min(downMove,
    leftMoves));
70
71                  }
72
73              }
74          }
75          // Actually reviews all moves for each square
76          // Does full array incase of negative cost areas...
77
78          System.out.println("Min cost path summations processed...");
79          print(minInc, minInc.length, minInc[0].length);
80
81          return minInc[rowF][colF];
82      }
83
84      // Done : builds array from file correctly//
85      public static int[][] fileInts() throws FileNotFoundException {
86          Scanner scnr = new Scanner(System.in);
87
88          System.out.print("Input File Name : ");
89          String userVar = scnr.next();
90
91          try {
92              while (!new File(userVar).exists()) {
93                  System.out.print("Input File Name : ");
94                  userVar = scnr.next();
95              }
96              System.out.println();
97              Scanner scnrX = new Scanner(new File(userVar));
98
99              // Grab size row x col first//
```

```
100             int n = scnrX.nextInt();
101             int m = scnrX.nextInt();
102             // Public static goal index//
103             rowF = scnrX.nextInt();
104             colF = scnrX.nextInt();
105 
106             int x[][] = new int[n][m];
107 
108             for (n = 0; n < x.length; n++) {
109                 for (m = 0; m < x[0].length; m++) {
110                     if (scnrX.hasNextInt()) {
111                         x[n][m] = scnrX.nextInt();
112                     } else {// Out Of Room//
113                         n = x.length;
114                         m = x[0].length;
115                     }
116                 }
117             }
118 
119             return x;
120         } catch (FileNotFoundException e) {
121             System.out.println("Error File Not Found...");
122             return null;
123         }
124     }
125 
126     // Print 2D int array//
127     public static void print(int[][] y, int n, int m) {
128         System.out.println("Integer Matrix Size : [" + n + "]" + "[" + m + "
    ]");
129 
130         int i;
131         int j;
132 
133         for (i = 0; i < n; i++) {
134             System.out.print("[ ");
135 
136             for (j = 0; j < m; j++) {
137 
138                 if (j != 0) {
139                     System.out.print(", ");
140                 }
141                 System.out.print(y[i][j]);
142 
143             }
144 
145             System.out.println(" ]");
146 
147         }
148 
149     }
150 }
```