**Batch- T8**

**Practical No. 3**

**Divide and Conquer Strategy**

**Name: Siya Pondkule**

**PRN: 23520008**

**Week 3 Assignment**

Part: 1

**Divide and conquer strategy**

**Q1)** Implement algorithm to Find the maximum element in an array which is first increasing and then decreasing, with Time Complexity *O(Logn).*

import java.util.\*;

public class inc\_dec\_array {

    public static int findMax(int[] arr, int low, int high) {

        if (low == high) {

            return arr[low];

        }

        int mid = low + (high - low) / 2;

        if ((mid == 0 || arr[mid - 1] <= arr[mid]) && (mid == arr.length - 1 || arr[mid + 1] <= arr[mid])) {

            return arr[mid];

        } else if (mid < arr.length - 1 && arr[mid] < arr[mid + 1]) {

            return findMax(arr, mid + 1, high);

        } else {

            return findMax(arr, low, mid - 1);

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter number of elements in array:");

        int n = sc.nextInt();

        System.out.println("Insert elements in array:");

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();

        }

        int max = findMax(arr, 0, n - 1);

        System.out.println("Max Element: " + max);

    }

}

**Pseudo code:**

Procedure main:

Prompt user to enter number of elements in array

Read integer n

Prompt user to insert elements in array

Initialize array arr of size n

For each index i from 0 to n-1:

Read integer and assign it to arr[i]

Call findMax(arr, 0, n - 1) and store the result in max

Print "Max Element: " + max

End Procedure

Procedure findMax(arr, low, high):

If low is equal to high:

Return arr[low]

Calculate mid as low + (high - low) / 2

If (mid is 0 or arr[mid - 1] <= arr[mid]) AND (mid is arr.length - 1 or arr[mid + 1] <= arr[mid]):

Return arr[mid]

Else If mid < arr.length - 1 AND arr[mid] < arr[mid + 1]:

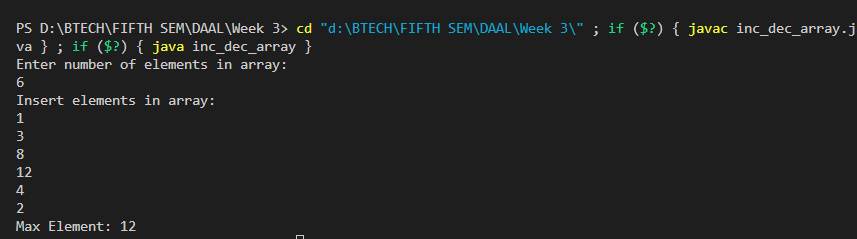
Return findMax(arr, mid + 1, high)

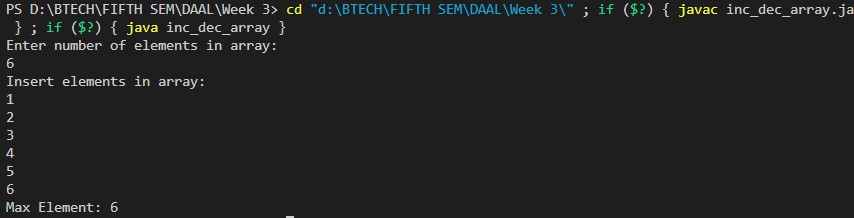
Else:

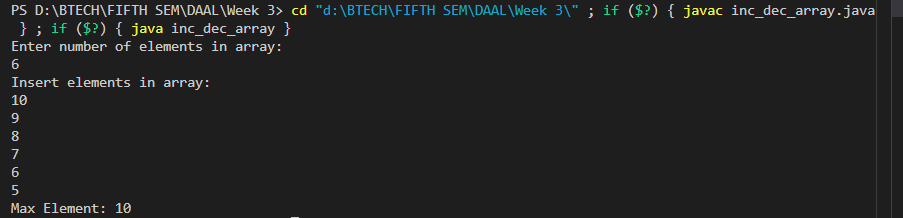
Return findMax(arr, low, mid - 1)

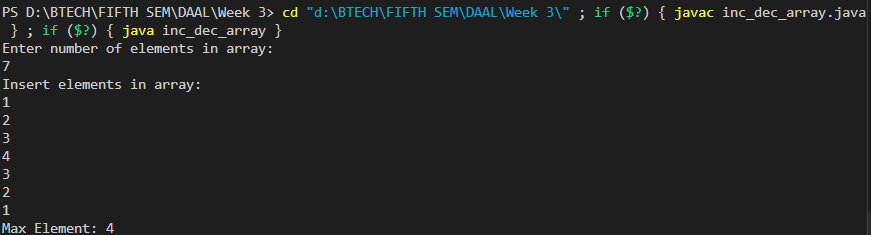
End Procedure

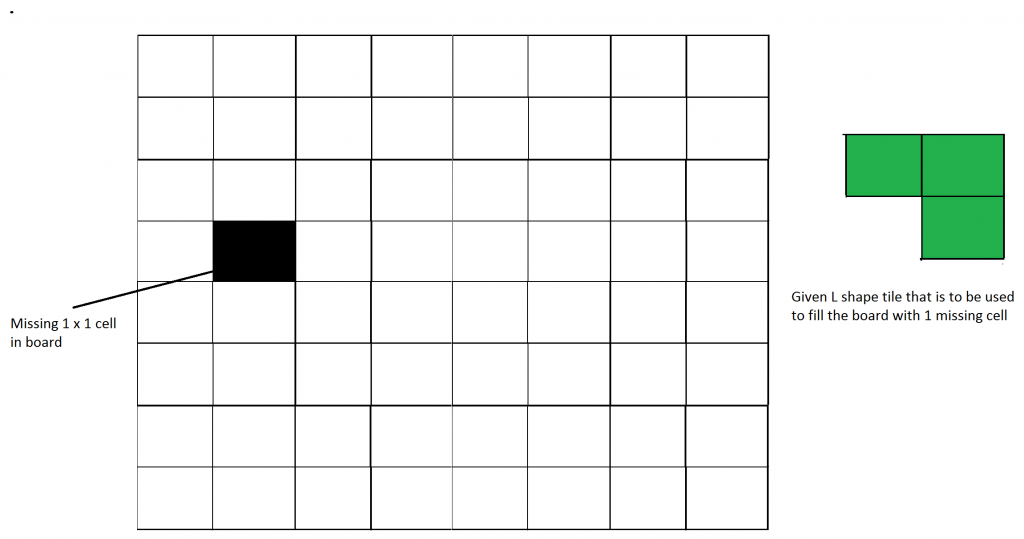
Output:









**Q2)** Implement algorithm for Tiling problem: Given an *n by n* board where n is of form *2k* where *k >= 1* (Basically n is a power of *2* with minimum value as *2*). The board has one missing cell (of size *1 x 1*). Fill the board using L shaped tiles. An *L* shaped tile is a *2 x 2* square with one cell of size *1×1* missing

import java.util.Scanner;

public class TilingProblem {

    static int tileNumber = 1;

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the size of the board (must be a power of 2): ");

        int n = scanner.nextInt();

        System.out.println("Enter the row and column of the missing cell (0-based indexing): ");

        int missingRow = scanner.nextInt();

        int missingCol = scanner.nextInt();

        int[][] board = new int[n][n];

        solveTilingProblem(board, n, missingRow, missingCol, 0, 0);

        printBoard(board);

        scanner.close();

    }

    static void solveTilingProblem(int[][] board, int size, int missingRow, int missingCol, int topRow, int leftCol) {

        if (size == 2) {

            tileNumber++;

            for (int i = 0; i < 2; i++) {

                for (int j = 0; j < 2; j++) {

                    if (!(topRow + i == missingRow && leftCol + j == missingCol)) {

                        board[topRow + i][leftCol + j] = tileNumber;

                    }

                }

            }

            return;

        }

        int subSize = size / 2;

        int centerRow = topRow + subSize;

        int centerCol = leftCol + subSize;

        int quadrant = (missingRow < centerRow ? 0 : 1) \* 2 + (missingCol < centerCol ? 0 : 1);

        tileNumber++;

        if (quadrant != 0) board[centerRow - 1][centerCol - 1] = tileNumber;

        if (quadrant != 1) board[centerRow - 1][centerCol] = tileNumber;

        if (quadrant != 2) board[centerRow][centerCol - 1] = tileNumber;

        if (quadrant != 3) board[centerRow][centerCol] = tileNumber;

        solveTilingProblem(board, subSize,

                quadrant == 0 ? missingRow : centerRow - 1, quadrant == 0 ? missingCol : centerCol - 1,

                topRow, leftCol);

        solveTilingProblem(board, subSize,

                quadrant == 1 ? missingRow : centerRow - 1, quadrant == 1 ? missingCol : centerCol,

                topRow, centerCol);

        solveTilingProblem(board, subSize,

                quadrant == 2 ? missingRow : centerRow, quadrant == 2 ? missingCol : centerCol - 1,

                centerRow, leftCol);

        solveTilingProblem(board, subSize,

                quadrant == 3 ? missingRow : centerRow, quadrant == 3 ? missingCol : centerCol,

                centerRow, centerCol);

    }

    static void printBoard(int[][] board) {

        for (int[] row : board) {

            for (int cell : row) {

                System.out.printf("%2d ", cell);

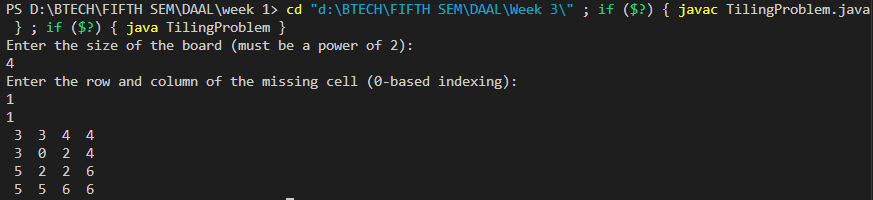
            }

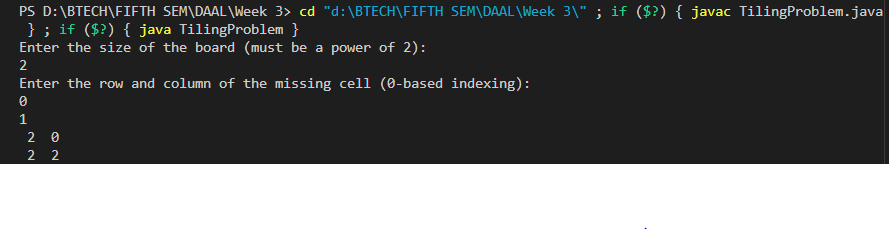
            System.out.println();

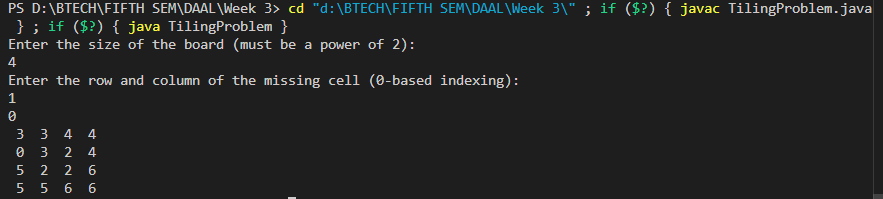
        }

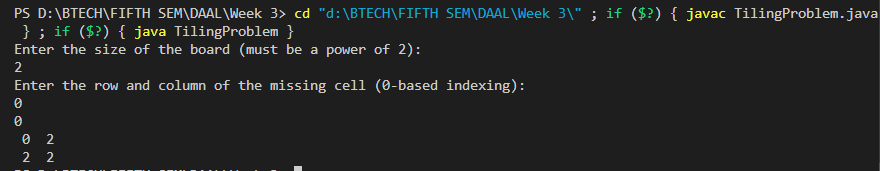
    }

}







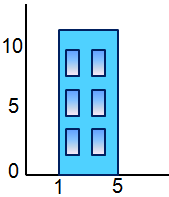


Q3) Implement algorithm for The Skyline Problem: Given n rectangular buildings in a 2-dimensional city, computes the skyline of these buildings, eliminating hidden lines. The main task is to view buildings from a side and remove all sections that are not visible.

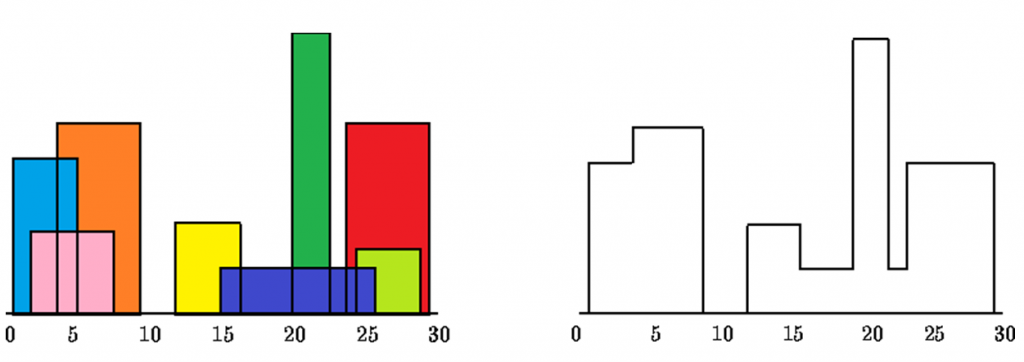
All buildings share common bottom and every **building**is represented by triplet (left, ht, right)

‘left’: is x coordinated of left side (or wall).  
‘right': is x coordinate of right side  
‘ht': is height of building.

For example, the building on right side is represented as *(1, 11, 5)*

[](http://www.geeksforgeeks.org/divide-and-conquer-set-7-the-skyline-problem/building/)

A **skyline**is a collection of rectangular strips. A rectangular **strip**is represented as a pair (left, ht) where left is x coordinate of left side of strip and ht is height of strip.



With Time Complexity *O(nLogn)*

import java.util.\*;

class SkylineProblem {

    private static List<int[]> mergeSkylines(List<int[]> leftSkyline, List<int[]> rightSkyline) {

        List<int[]> mergedSkyline = new ArrayList<>();

        int leftHeight = 0, rightHeight = 0, currentHeight = 0;

        int leftIndex = 0, rightIndex = 0;

        while (leftIndex < leftSkyline.size() && rightIndex < rightSkyline.size()) {

            int x;

            if (leftSkyline.get(leftIndex)[0] < rightSkyline.get(rightIndex)[0]) {

                x = leftSkyline.get(leftIndex)[0];

                leftHeight = leftSkyline.get(leftIndex)[1];

                leftIndex++;

            } else {

                x = rightSkyline.get(rightIndex)[0];

                rightHeight = rightSkyline.get(rightIndex)[1];

                rightIndex++;

            }

            int maxHeight = Math.max(leftHeight, rightHeight);

            if (currentHeight != maxHeight) {

                mergedSkyline.add(new int[]{x, maxHeight});

                currentHeight = maxHeight;

            }

        }

        while (leftIndex < leftSkyline.size()) {

            mergedSkyline.add(leftSkyline.get(leftIndex));

            leftIndex++;

        }

        while (rightIndex < rightSkyline.size()) {

            mergedSkyline.add(rightSkyline.get(rightIndex));

            rightIndex++;

        }

        return mergedSkyline;

    }

    private static List<int[]> findSkyline(int[][] buildings, int left, int right) {

        if (left == right) {

            List<int[]> skyline = new ArrayList<>();

            skyline.add(new int[]{buildings[left][0], buildings[left][1]});

            skyline.add(new int[]{buildings[left][2], 0});

            return skyline;

        }

        int mid = (left + right) / 2;

        List<int[]> leftSkyline = findSkyline(buildings, left, mid);

        List<int[]> rightSkyline = findSkyline(buildings, mid + 1, right);

        return mergeSkylines(leftSkyline, rightSkyline);

    }

    public static List<int[]> getSkyline(int[][] buildings) {

        if (buildings.length == 0) {

            return new ArrayList<>();

        }

        return findSkyline(buildings, 0, buildings.length - 1);

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of buildings:");

        int n = sc.nextInt();

        int[][] buildings = new int[n][3];

        System.out.println("Enter the buildings as (left ht right):");

        for (int i = 0; i < n; i++) {

            buildings[i][0] = sc.nextInt(); // left

            buildings[i][1] = sc.nextInt(); // height

            buildings[i][2] = sc.nextInt(); // right

        }

        List<int[]> skyline = getSkyline(buildings);

        System.out.println("Skyline:");

        for (int[] point : skyline) {

            System.out.println(Arrays.toString(point));

        }

        sc.close();

    }

}

Output:

