**Batch -** T5

**Assignment No. -** 1

**Title -** Sorting Algorithm

**Student Name -** Sharaneshwar Bharat Punjal

**Student PRN -** 23520011

1) You are given two sorted array, A and B, where A has a large enough buffer at the end to hold B. Write a method to merge B into A in sorted order.

import java.util.Scanner;

public class Q1 {

    public static void merge(int[] A, int[] B, int n, int m) {

        int i = n - 1;

        int j = m - 1;

        int k = n + m - 1;

        while (i >= 0 && j >= 0) {

            if (A[i] > B[j]) {

                A[k] = A[i];

                i--;

            } else {

                A[k] = B[j];

                j--;

            }

            k--;

        }

        while (j >= 0) {

            A[k] = B[j];

            j--;

            k--;

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int m = sc.nextInt();

        int[] A = new int[n + m];

        int[] B = new int[m];

        for (int i = 0; i < n; i++)

            A[i] = sc.nextInt();

        for (int i = 0; i < m; i++)

            B[i] = sc.nextInt();

        merge(A, B, n, m);

        for (int i = 0; i < n + m; i++)

            System.out.print(A[i] + " ");

        System.out.println();

        sc.close();

    }

}

**Test Case 1:**

5 5

1 3 5 7 9

2 4 6 8 10

**Output:**

1 2 3 4 5 6 7 8 9

**Test Case 2:**

5 3

1 3 5 7 9

2 4 6

**Output:**

1 2 3 4 5 6 7 9

**Test Case 3:**

6 5

2 4 6 8 10 12

1 3 5 7 9

**Output:**

1 2 3 4 5 6 7 8 9 10 12

**Test Case 4:**

3 3

10 20 30

15 25 35

**Output:**

10 15 20 25 30 35

2) Write a method to sort an array of string so that all the anagrams are next to each other.

import java.util.Arrays;

import java.util.Scanner;

public class Q2 {

    public static void sortAnagrams(String[] arr) {

        int n = arr.length;

        for (int i = 0; i < n - 1; i++) {

            for (int j = 0; j < n - i - 1; j++) {

                if (canonicalForm(arr[j]).compareTo(canonicalForm(arr[j + 1])) > 0) {

                    String temp = arr[j];

                    arr[j] = arr[j + 1];

                    arr[j + 1] = temp;

                }

            }

        }

    }

    private static String canonicalForm(String s) {

        char[] chars = s.toCharArray();

        for (int i = 0; i < chars.length - 1; i++) {

            for (int j = 0; j < chars.length - i - 1; j++) {

                if (chars[j] > chars[j + 1]) {

                    char temp = chars[j];

                    chars[j] = chars[j + 1];

                    chars[j + 1] = temp;

                }

            }

        }

        return new String(chars);

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        String[] arr = new String[n];

        for (int i = 0; i < n; i++)

            arr[i] = sc.next();

        sc.close();

        sortAnagrams(arr);

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

    }

}

**Test Case 1:**

6

cat dog tac god act odg

**Output:**

[act, cat, tac, dog, god, odg]

**Test Case 2:**

4

listen silent enlist inlets

**Output:**

[enlist, inlets, listen, silent]

**Test Case 3:**

5

abc bca cab xyz zyx

**Output:**

[abc, bca, cab, xyz, zyx]

**Test Case 4:**

7

bat tab cat act tca rat tar

**Output:**

[bat, tab, act, cat, tca, rat, tar]

Q) Given a sorted array of *n* integers that has been rotated an unknown number of times, write code to find an element in the array. You may assume that the array was originally sorted in increasing order.

EXAMPLE

Input: find 5 in {15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14}

Output: 8 (the index of 5 in the array)

import java.util.Scanner;

public class Q3 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++)

            arr[i] = sc.nextInt();

        int target = sc.nextInt();

        sc.close();

        System.out.println(findElement(arr, target));

    }

    public static int findElement(int[] arr, int target) {

        int left = 0;

        int right = arr.length - 1;

        while (left <= right) {

            int mid = left + (right - left) / 2;

            if (arr[mid] == target) {

                return mid;

            }

            if (arr[left] <= arr[mid]) {

                if (target >= arr[left] && target < arr[mid]) {

                    right = mid - 1;

                } else {

                    left = mid + 1;

                }

            } else {

                if (target > arr[mid] && target <= arr[right]) {

                    left = mid + 1;

                } else {

                    right = mid - 1;

                }

            }

        }

        return -1;

    }

}

**Test Case 1:**

12

15 16 19 20 25 1 3 4 5 7 10 14

5

**Output:**

8

**Test Case 2:**

10

4 5 6 7 8 9 1 2 3

3

**Output:**

9

**Test Case 3:**

7

10 20 30 40 50 5 7

50

**Output:**

4

**Test Case 4:**

8

50 60 70 80 90 100 10 20

70

**Output:**

2

Q) Imagine you have a 20GB file with one string per line. Explain how you would sort the file.

The idea is to use the external sort algorithm. We can't bring all the data into memory, so we need to use a divide and conquer approach. We can divide the file into chunks which are x megabytes each, where x is the amount of memory we have available. Each chunk is sorted separately and then saved back to the file system. Once all the chunks are sorted, we merge the chunks one by one. We can use a min heap to keep track of the next element to write to the file. We read the first element of each chunk into the heap and then write the smallest one to the file. We then read the next element from the chunk we took the smallest element from and add it to the heap. We continue this process until all elements have been written to the file. This is the external sort algorithm.

Q) Given a sorted array of string which is interspersed with empty string, write a method to find the location of a given string.

EXAMPLE

Input: find “ball” in {“at”, “”, “”, “ball”, “”, “”, “car”, “”, “”, “dad”, “”,””}

Output: 4

import java.util.Scanner;

public class Q5 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        String[] arr = new String[n];

        sc.nextLine();

        for (int i = 0; i < n; i++)

            arr[i] = sc.nextLine();

        String target = sc.nextLine();

        sc.close();

        System.out.println(findString(arr, target));

    }

    public static int findString(String[] arr, String target) {

        if (arr == null || target == null || target.isEmpty()) {

            return -1;

        }

        return search(arr, target, 0, arr.length - 1);

    }

    private static int search(String[] arr, String target, int left, int right) {

        if (left > right) {

            return -1;

        }

        int mid = left + (right - left) / 2;

        if (arr[mid].isEmpty()) {

            int leftMid = mid - 1;

            int rightMid = mid + 1;

            while (true) {

                if (leftMid < left && rightMid > right) {

                    return -1;

                } else if (rightMid <= right && !arr[rightMid].isEmpty()) {

                    mid = rightMid;

                    break;

                } else if (leftMid >= left && !arr[leftMid].isEmpty()) {

                    mid = leftMid;

                    break;

                }

                rightMid++;

                leftMid--;

            }

        }

        if (arr[mid].equals(target)) {

            return mid;

        } else if (arr[mid].compareTo(target) < 0) {

            return search(arr, target, mid + 1, right);

        } else {

            return search(arr, target, left, mid - 1);

        }

    }

}

**Test Case 1:**

12

at

""

""

ball

""

""

car

""

""

dad

""

""

ball

**Output:**

3

**Test Case 2:**

8

apple

""

banana

""

""

cat

""

dog

banana

**Output:**

2

**Test Case 3:**

10

at

""

ball

""

""

""

car

""

""

cat

car

**Output:**

6

**Test Case 4:**

10

at

""

ball

""

""

""

car

""

""

cat

car

**Output:**

4

Q) Given an M\*N matrix in which each row and each column is sorted in ascending order, write a method to find an element.  
import java.util.Arrays;

import java.util.Scanner;

public class Q6 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int m = sc.nextInt();

        int n = sc.nextInt();

        int[][] matrix = new int[m][n];

        for (int i = 0; i < m; i++)

            for (int j = 0; j < n; j++)

                matrix[i][j] = sc.nextInt();

        int target = sc.nextInt();

        sc.close();

        System.out.println(Arrays.toString(findElement(matrix, target)));

    }

    public static int[] findElement(int[][] matrix, int target) {

        if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {

            return new int[]{-1, -1};

        }

        int row = 0;

        int col = matrix[0].length - 1;

        while (row < matrix.length && col >= 0) {

            if (matrix[row][col] == target) {

                return new int[]{row, col};

            } else if (matrix[row][col] < target) {

                row++;

            } else {

                col--;

            }

        }

        return new int[]{-1, -1};

    }

}

**Test Case 1:**

3 4

1 4 7 11

2 5 8 12

3 6 9 16

8

**Output:**

[1, 2]

**Test Case 2:**

3 4

1 4 7 11

2 5 8 12

3 6 9 16

10

**Output:**

[-1, -1]

**Test Case 3:**

1 1

5

5

**Output:**

[0, 0]

**Test Case 4:**

0 0

5

**Output:**

[-1, -1]

Q) A circus is designing a tower routine consisting of people standing atop one another’s shoulders. For practical and aesthetic reasons, each person must be both shorter and lighter than the person below him or her. Given the heights and weight of each circus, write a method to compute the largest possible number of people in such tower.

EXAMPLE:

*Input(ht,wt):* (65, 100) (70, 150) (56, 90) (75,190) (60, 95) (68, 110).

Output: The longest tower is length 6 and includes from top to bottom:

(56, 90) (60, 95) (65, 100) (68, 110) (70, 150) (75, 190)

import java.util.Arrays;

import java.util.Scanner;

public class Q7 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[][] people = new int[n][2];

        for (int i = 0; i < n; i++) {

            people[i][0] = sc.nextInt();

            people[i][1] = sc.nextInt();

        }

        sc.close();

        System.out.println(findLongestTower(people));

    }

    public static int findLongestTower(int[][] people) {

        if (people == null || people.length == 0) {

            return 0;

        }

        Arrays.sort(people, (p1, p2) -> {

            if (p1[0] != p2[0]) {

                return Integer.compare(p1[0], p2[0]);

            } else {

                return Integer.compare(p1[1], p2[1]);

            }

        });

        int[] dp = new int[people.length];

        Arrays.fill(dp, 1);

        for (int i = 1; i < people.length; i++) {

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

                if (people[i][0] > people[j][0] && people[i][1] > people[j][1]) {

                    dp[i] = Math.max(dp[i], dp[j] + 1);

                }

            }

        }

        return Arrays.stream(dp).max().getAsInt();

    }

}

**Test Case 1:**

6

65 100

70 150

56 90

75 190

60 95

68 110

**Output:**

6

**Test Case 2:**

4

80 200

75 210

85 180

90 170

**Output:**

1

**Test Case 3:**

5

65 100

70 150

56 105

75 130

60 90

**Output:**

3

**Test Case 4:**

8

56 90

60 110

65 100

70 150

75 130

80 190

68 120

66 80

**Output:**

5

Q) Imagine you are reading in stream of integers. Periodically, you wish to be able to look up the rank of number *x* (the number of values less than or equal to *x*). Implement the data structures and algorithms to support these operations. That is, Implement the method *track (int x),* which is called when each number is generated, and the method *getRankOfNumber (int x)*, which return the number of values less than or equal to *x* (not including x itself).

EXAMPLE

Stream (in order of appearance) : 5, 1, 4, 4, 5, 9, 7, 13, 3

*getRankOfNumber(1) = 0*

*getRankOfNumber(3) = 1*

*getRankOfNumber(4) =3*

import java.util.Arrays;

import java.util.Scanner;

public class Q8 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int stream[] = new int[n];

        for (int i = 0; i < n; i++) {

            stream[i] = sc.nextInt();

        }

        sc.close();

        Arrays.sort(stream);

        System.out.println(getRankOfNumber(stream, 1));

        System.out.println(getRankOfNumber(stream, 3));

        System.out.println(getRankOfNumber(stream, 4));

        System.out.println(getRankOfNumber(stream, 9));

    }

    private static int getRankOfNumber(int stream[], int x) {

        int left = 0;

        int right = stream.length - 1;

        while (left <= right) {

            int mid = left + (right - left) / 2;

            if (stream[mid] <= x) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return left - 1;

    }

}

**Test Case 1:**

9

5 1 4 4 5 9 7 13 3

**Output:**

0

1

3

6

**Test Case 2:**

7

2 2 2 2 2 2 2

**Output:**

6

6

6

6

**Test Case 3:**

6

-3 -1 0 2 4 6

**Output:**

0

1

2

5

**Test Case 4:**

1

10

**Output:**

-1

-1

-1

0

**Batch –** T5

**Assignment No. -** 2

**Title -** SearchingAlgorithm

**Student Name -** Sharaneshwar Bharat Punjal

**Student PRN -** 23520011

1) You are an IT company's manager. Based on their performance over the last N working days, you must rate your employee. You are given an array of N integers called workload, where workload[i] represents the number of hours an employee worked on an ith day. The employee must be evaluated using the following criteria:

Rating = the maximum number of consecutive working days when the employee has worked more than 6 hours.

You are given an integer N where N represents the number of working days. You are given an integer array workload where workload[i] represents the number of hours an employee worked on an ith day.

Task: Determine the employee rating

import java.util.Scanner;

public class Q1 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] workload = new int[n];

        for (int i = 0; i < workload.length; i++)

            workload[i] = sc.nextInt();

        sc.close();

        System.out.println(getRating(workload, n));

    }

    private static int getRating(int[] workload, int N) {

        int rating = Integer.MIN\_VALUE;

        int cnt = 0;

        for (int i : workload) {

            if (i > 6)

                cnt++;

            else {

                rating = Math.max(rating, cnt);

                cnt = 0;

            }

        }

        rating = Math.max(rating, cnt);

        return rating;

    }

}

**Test Case 1:**

7

8 7 5 6 10 9 2

**Output:**

2

**Test Case 2:**

5

7 8 9 10 11

**Output:**

5

**Test Case 3:**

4

3 4 5 6

**Output:**

0

**Test Case 4:**

20

5 6 7 8 2 3 4 5 9 10 11 12 1 2 3 6 7 8 1 2

**Output:**

4

2) You have N boxes numbered 1 through N and K candies numbered 1 through K. You put the candies in the boxes in the following order:

• first candy in the first box,

• second candy in the second box,

• .......

• .......

• so up to N-th candy in the Nth box,

• the next candy in (N - 1)-th box,

• the next candy in (N - 2)-th box

• .......

• .......

• and so on up to the first box,

• then the next candy in the second box

• ...... and so on until there is no candy left.

So you put the candies in the boxes in the following order:

Find the index of the box where you put the K-th candy.

import java.util.Scanner;

public class Q2 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int T = sc.nextInt();

        while (T-- > 0) {

            int N = sc.nextInt();

            int K = sc.nextInt();

            System.out.println(findBoxIndex(N, K));

        }

        sc.close();

    }

    private static int findBoxIndex(int N, int K) {

        int cycleLength = 2 \* (N - 1);

        int pos = (K - 1) % cycleLength;

        if (pos < N)

            return pos + 1;

        else

            return N - (pos - N + 1);

    }

}

**Test Case 1:**

4

5 7

4 10

6 20

10 25

**Output:**

3

2

6

6

**Test Case 2:**

3

2 5

3 8

7 14

**Output:**

2

2

7

**Test Case 3:**

4

3 7

5 13

6 18

8 24

**Output:**

1

3

6

8

**Test Case 4:**

3

4 100

7 150

10 200

**Output:**

4

6

10

3) Implement and Explain Tower of Hanoi algorithm.

import java.util.Scanner;

public class Q3 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        sc.close();

        towerOfHanoi(n, 'A', 'C', 'B');

    }

    private static void towerOfHanoi(int n, char from, char to, char aux) {

        if (n == 1) {

            System.out.println("Move disk 1 from rod " + from + " to rod " + to);

            return;

        }

        towerOfHanoi(n - 1, from, aux, to);

        System.out.println("Move disk " + n + " from rod " + from + " to rod " + to);

        towerOfHanoi(n - 1, aux, to, from);

    }

}

**Test Case 1:**

3

**Output:**

Move disk 1 from rod A to rod C

Move disk 2 from rod A to rod B

Move disk 1 from rod C to rod B

Move disk 3 from rod A to rod C

Move disk 1 from rod B to rod A

Move disk 2 from rod B to rod C

Move disk 1 from rod A to rod C

**Test Case 2:**

2

**Output:**

Move disk 1 from rod A to rod B

Move disk 2 from rod A to rod C

Move disk 1 from rod B to rod C

**Test Case 3:**

4

**Output:**

Move disk 1 from rod A to rod B

Move disk 2 from rod A to rod C

Move disk 1 from rod B to rod C

Move disk 3 from rod A to rod B

Move disk 1 from rod C to rod A

Move disk 2 from rod C to rod B

Move disk 1 from rod A to rod B

Move disk 4 from rod A to rod C

Move disk 1 from rod B to rod C

Move disk 2 from rod B to rod A

Move disk 1 from rod C to rod A

Move disk 3 from rod B to rod C

Move disk 1 from rod A to rod B

Move disk 2 from rod A to rod C

Move disk 1 from rod B to rod C

**Test Case 4:**

5

**Output:**

Move disk 1 from rod A to rod C

Move disk 2 from rod A to rod B

Move disk 1 from rod C to rod B

Move disk 3 from rod A to rod C

Move disk 1 from rod B to rod A

Move disk 2 from rod B to rod C

Move disk 1 from rod A to rod C

Move disk 4 from rod A to rod B

Move disk 1 from rod C to rod B

Move disk 2 from rod C to rod A

Move disk 1 from rod B to rod A

Move disk 3 from rod C to rod B

Move disk 1 from rod A to rod C

Move disk 2 from rod A to rod B

Move disk 1 from rod C to rod B

Move disk 5 from rod A to rod C

Move disk 1 from rod B to rod A

Move disk 2 from rod B to rod C

Move disk 1 from rod A to rod C

Move disk 3 from rod B to rod A

Move disk 1 from rod C to rod B

Move disk 2 from rod C to rod A

Move disk 1 from rod B to rod A

Move disk 4 from rod B to rod C

Move disk 1 from rod A to rod C

Move disk 2 from rod A to rod B

Move disk 1 from rod C to rod B

Move disk 3 from rod A to rod C

Move disk 1 from rod B to rod A

Move disk 2 from rod B to rod C

Move disk 1 from rod A to rod C

4) There is a frog initially placed at the origin of the coordinate plane. In exactly 1 second, the frog can either move up 1 unit, move right 1 unit, or stay still. In other words, from position (x, y), the frog can spend 1 second to move to:

• (X + 1, Y)

• (X, Y + 1)

• (X, Y)

After T seconds, a villager who sees the frog reports that the frog lies on or inside a square of side-length s with coordinates (X, Y), (X + S, Y), (X, Y + S), (X + S, Y + S). Calculate how many points with integer coordinates on or inside this square could be the frog's position after exactly T seconds

Input Format:

The first and only line of input contains four space-separated integers: X, Y, S, and T.

Output Format:

Print the number of points with integer coordinates that could be the frog's position after T seconds.

import java.util.Scanner;

public class Q4 {

    public static void main(String args[] ) throws Exception {

        Scanner sc = new Scanner(System.in);

        int X = sc.nextInt();

        int Y = sc.nextInt();

        int s = sc.nextInt();

        int T = sc.nextInt();

        int count = 0;

        for (int i = X; i <= X + s; i++)

            for (int j = Y ; j <= Y + s; j++)

                if (i + j <= T)

                    count++;

        System.out.println(count);

        sc.close();

    }

}

**Test Case 1:**

1 1 2 5

**Output:**

8

**Test Case 2:**

0 0 1 2

**Output:**

4

**Test Case 3:**

2 2 3 6

**Output:**

6

**Test Case 4:**

2 60 95 116

**Output:**

1540

5) Implement linear Search Algorithm.

import java.util.Scanner;

public class Q5 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int arr[] = new int[n];

        for (int i = 0; i < n; i++)

            arr[i] = sc.nextInt();

        int key = sc.nextInt();

        sc.close();

        int index = linearSearch(arr, key);

        if (index == -1) {

            System.out.println("Element not found");

        } else {

            System.out.println("Element found at index: " + index);

        }

    }

    public static int linearSearch(int[] arr, int key) {

        for (int i = 0; i < arr.length; i++) {

            if (arr[i] == key) {

                return i;

            }

        }

        return -1;

    }

}

**Test Case 1:**

25

10 22 35 47 50 65 78 89 90 102 113 124 135 146 157 168 179 190 201 212 223 234 245 256 267

124

**Output:**

Element found at index: 11

**Test Case 2:**

30

5 17 29 41 53 65 77 89 101 113 125 137 149 161 173 185 197 209 221 233 245 257 269 281 293 305 317 329 341 353

305

**Output:**

Element found at index: 25

**Test Case 3:**

22

8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160 168 176

474

**Output:**

Element not found

**Test Case 4:**

24

7 14 21 28 35 42 49 56 63 70 77 84 91 98 105 112 119 126 133 140 147 154 161 168

140

**Output:**

Element found at index: 20

6) Implement Binary Search algorithm.

import java.util.Scanner;

public class Q6 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int arr[] = new int[n];

        for (int i = 0; i < n; i++)

            arr[i] = sc.nextInt();

        int key = sc.nextInt();

        sc.close();

        int index = binarySearch(arr, key);

        if (index == -1) {

            System.out.println("Element not found");

        } else {

            System.out.println("Element found at index: " + index);

        }

    }

    public static int binarySearch(int[] arr, int key) {

        int low = 0;

        int high = arr.length - 1;

        while (low <= high) {

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

            if (arr[mid] == key) {

                return mid;

            }

            if (arr[mid] < key) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        return -1;

    }

}

**Test Case 1:**

15

3 7 11 16 20 25 30 35 40 45 50 55 60 65 70

55

**Output:**

Element found at index: 11

**Test Case 2:**

20

-10 -5 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85

5

**Output:**

Element found at index: 3

**Test Case 3:**

25

100 200 300 400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600 1700 1800 1900 2000 2100 2200 2300 2400 2500

1950

**Output:**

Element not found

**Test Case 4:**

18

2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36

28

**Output:**

Element found at index: 13

**Batch -** T5

**Assignment No. -** 3

**Title -** Divide and Conquer Strategy

**Student Name -** Sharaneshwar Bharat Punjal

**Student PRN –** 23520011

1) Implement algorithm to find the maximum element in an array which is first increasing and then decreasing, with time complexity O (log n).

**Algorithm:**

1. Initialize two pointers: low at the start of the array and high at the end.

2. While low is less than or equal to high:

Calculate the middle index mid.

Check if mid is the peak element

3. If the element at mid is greater than both its neighbours, then arr[mid] is the

maximum element.

4. If the array is increasing at mid, then move the low pointer to mid + 1.

5. Otherwise, move the high pointer to mid - 1.

6. The loop will eventually converge on the maximum element.

**Code:**

import java.util.Scanner;

public class Q1 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++)

            arr[i] = sc.nextInt();

        sc.close();

        System.out.println(findMaximum(arr, n));

    }

    public static int findMaximum(int[] arr, int n) {

        int left = 0;

        int right = n - 1;

        while (left <= right) {

            int mid = left + (right - left) / 2;

            if (left == right)

                return arr[left];

            if (right == left + 1)

                return Math.max(arr[left], arr[right]);

            if (arr[mid] > arr[mid - 1] && arr[mid] > arr[mid + 1])

                return arr[mid];

            if (arr[mid] > arr[mid - 1] && arr[mid] < arr[mid + 1])

                left = mid + 1;

            else

                right = mid - 1;

        }

        return -1;

    }

}

**Test Case 1:**

10

1 3 5 7 9 12 15 10 6 2

**Output:**

15

**Test Case 2:**

15

5 10 15 20 25 30 35 30 25 20 15 10 5 3 1

**Output:**

35

**Test Case 3:**

20

2 5 10 15 20 30 40 50 45 35 25 15 10 5 2 1 0 -1 -2 -3

**Output:**

50

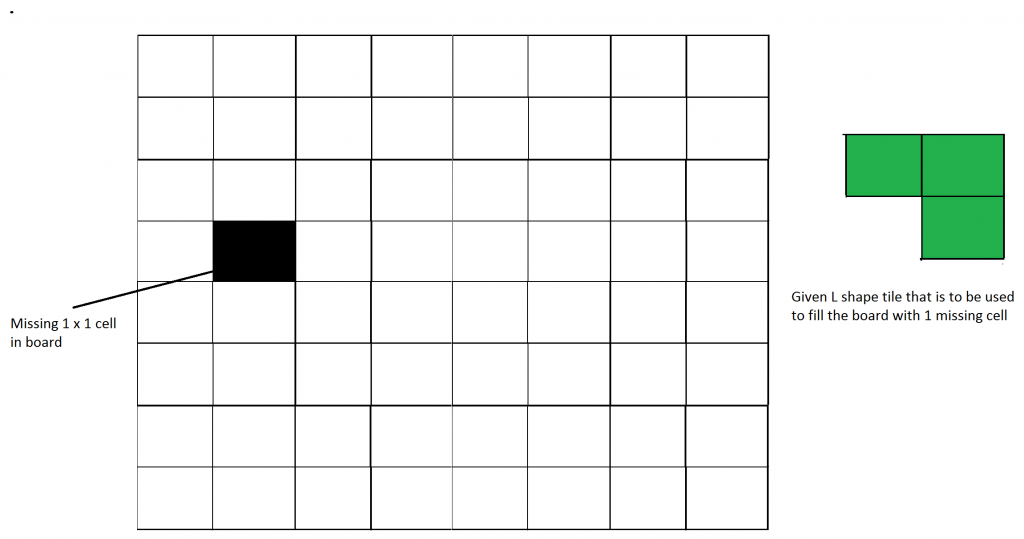
**Test Case 4:**

25

3 6 12 18 27 35 40 38 30 20 15 10 5 2 1 0 -2 -5 -10 -15 -20 -25 -30 -35 -40

**Output:**

40

2) 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.

**Algorithm:**

1. Start with an empty board of size n x n initialized with all zeros.

2. Mark the pre-filled cell (given as input) with -1.

3. Define a recursive function tile(size, r, c, board) to fill the board.

4. If the current sub-board size is 2 x 2, fill it with a unique tile number, ensuring that the missing tile is respected.

5. Find the coordinates of the pre-filled cell within the current sub-board.

6. Based on the location of this pre-filled cell, place an L-shaped tromino in the other three quadrants.

7. Recursively apply the tile function to the four quadrants.

8. The base case is when the size of the sub-board is 2 x 2. In this case, fill the empty cells with the current tile number.

9. After the board is fully tiled, output the board.

**Code:**

import java.util.Scanner;

public class Q2 {

    static int tileNum = 0;

    static char[][] board;

    private static void place(int x1, int y1, int x2, int y2, int x3, int y3) {

        tileNum++;

        board[x1][y1] = '\*';

        board[x2][y2] = '\*';

        board[x3][y3] = '\*';

    }

    private static void tile(int size, int r, int c) {

        if (size == 2) {

            tileNum++;

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

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

                    if (board[r + i][c + j] == ' ') {

                        board[r + i][c + j] = '\*';

                    }

                }

            }

            return;

        }

        int mr = -1, mc = -1;

        for (int i = r; i < r + size; i++) {

            for (int j = c; j < c + size; j++) {

                if (board[i][j] != ' ') {

                    mr = i;

                    mc = j;

                    break;

                }

            }

            if (mr != -1) break;

        }

        if (mr < r + size / 2 && mc < c + size / 2) {

            place(r + size / 2, c + size / 2 - 1, r + size / 2, c + size / 2, r + size / 2 - 1, c + size / 2);

        } else if (mr >= r + size / 2 && mc < c + size / 2) {

            place(r + size / 2 - 1, c + size / 2, r + size / 2, c + size / 2, r + size / 2 - 1, c + size / 2 - 1);

        } else if (mr < r + size / 2 && mc >= c + size / 2) {

            place(r + size / 2, c + size / 2 - 1, r + size / 2, c + size / 2, r + size / 2 - 1, c + size / 2 - 1);

        } else if (mr >= r + size / 2 && mc >= c + size / 2) {

            place(r + size / 2 - 1, c + size / 2, r + size / 2, c + size / 2 - 1, r + size / 2 - 1, c + size / 2 - 1);

        }

        tile(size / 2, r, c + size / 2);

        tile(size / 2, r, c);

        tile(size / 2, r + size / 2, c);

        tile(size / 2, r + size / 2, c + size / 2);

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int mr = sc.nextInt();

        int mc = sc.nextInt();

        sc.close();

        board = new char[n][n];

        for (int i = 0; i < n; i++) {

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

                board[i][j] = ' ';

            }

        }

        board[mr][mc] = '+';

        tile(n, 0, 0);

        for (int i = 0; i < n; i++) {

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

                System.out.print(board[i][j] + " ");

            }

            System.out.println();

        }

    }

}

**Test Case 1:**

4

1 2

**Output:**

\* \* \* \*

\* \* + \*

\* \* \* \*

\* \* \* \*

**Test Case 2:**

8

2 2

**Output:**

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* + \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

**Test Case 3:**

16

4 3

**Output:**

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* + \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

**Test Case 4:**

2

1 1

**Output:**

\* \*

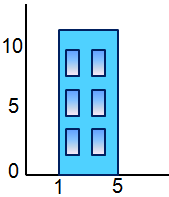
\* +

3) 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)

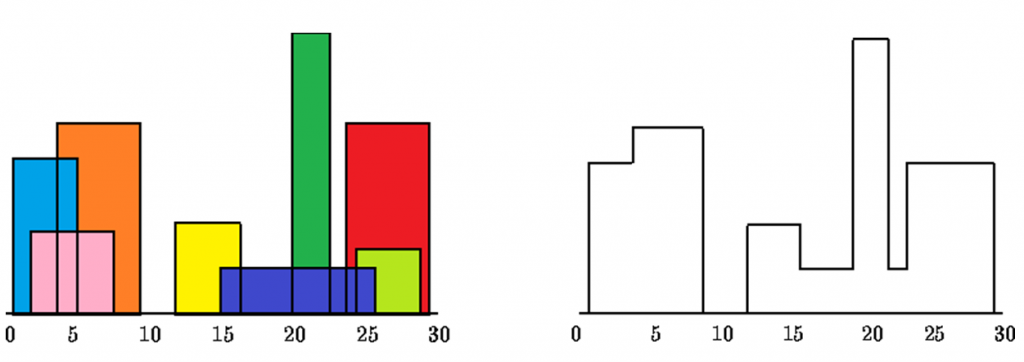
‘left’: is x coordinated of left side (or wall).  
‘right': is x coordinate of right side  
‘ht': is height of building.

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.

Required Time Complexity - O (n log n)



**Algorithm:**

1. The input is a list of buildings, where each building is represented by a triplet [Li, Ri, Hi], where Li is the left coordinate, Ri is the right coordinate, and Hi is the height.

2. For each building, generate two points:

* A start point (Li, -Hi) which represents the beginning of the building with negative height to distinguish start from end.
* An end point (Ri, Hi) which represents the end of the building with positive height.

3. Sort the points. If two points have the same x-coordinate, the point with the

smaller height comes first. If two points have the same height, the start point

(negative height) comes before the end point (positive height).

4. Use a max-heap to keep track of the current heights of the buildings as you iterate through the points. For each point:

* If it’s a start point, add its height to the heap.
* If it’s an end point, remove its height from the heap.
* Compare the current maximum height (top of the heap) with the previous maximum height. If they differ, this means the skyline changes at this point, so record this point in the result.

**Code:**

import java.util.\*;

public class Q3 {

    static class Point {

        int x, height;

        Point(int x, int height) {

            this.x = x;

            this.height = height;

        }

    }

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

        List<List<Integer>> result = new ArrayList<>();

        TreeSet<Integer> heights = new TreeSet<>(Collections.reverseOrder());

        List<Point> points = new ArrayList<>();

        for (int[] building : buildings) {

            points.add(new Point(building[0], -building[2]));

            points.add(new Point(building[1], building[2]));

        }

        points.sort((a, b) -> {

            if (a.x != b.x)

                return Integer.compare(a.x, b.x);

            return Integer.compare(b.height, a.height);

        });

        int ongoingHeight = 0;

        for (Point point : points) {

            int currentPoint = point.x;

            int heightAtCurrentPoint = point.height;

            if (heightAtCurrentPoint < 0) {

                heights.add(-heightAtCurrentPoint);

            } else {

                heights.remove(heightAtCurrentPoint);

            }

            int currentHeight = heights.isEmpty() ? 0 : heights.first();

            if (ongoingHeight != currentHeight) {

                ongoingHeight = currentHeight;

                result.add(Arrays.asList(currentPoint, ongoingHeight));

            }

        }

        return result;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

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

        for (int i = 0; i < n; i++) {

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

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

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

        }

        List<List<Integer>> result = getSkyline(buildings);

        for (List<Integer> r : result) {

            System.out.println(r.get(0) + " " + r.get(1));

        }

        sc.close();

    }

}

**Test Case 1:**

3

2 9 10

3 7 15

5 12 12

**Output:**

2 10

3 15

7 12

12 0

**Test Case 2:**

2

1 2 2

3 4 3

**Output:**

1 2

2 0

3 3

4 0

**Test Case 3:**

4

1 4 4

2 5 6

4 6 2

5 7 3

**Output:**

1 4

2 6

5 2

5 3

7 0

**Test Case 4:**

5

1 3 4

2 5 6

4 7 3

6 8 5

9 10 2

**Output:**

1 4

2 6

5 3

6 5

8 0

9 2

10 0