**Batch- T8**

**Practical No. 1**

**Sorting Algorithm**

**Name: Siya Pondkule**

**PRN: 23520008**

**Sorting Algorithm**

Q) 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 merge\_AB {

    public static void main(String args[]){

        Scanner sc=new Scanner(System.in);

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

        int n=sc.nextInt();

        System.out.println("Enter elements in A:");

        int[] A=new int[n];

        for(int i=0;i<n;i++){

            A[i]=sc.nextInt();

        }

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

        int m=sc.nextInt();

        System.out.println("Enter elements in B:");

        int[] B=new int[m];

        for(int i=0;i<m;i++){

            B[i]=sc.nextInt();

        }

        int[] result = new int[A.length + B.length];

        int i = 0, j = 0, k = 0;

        while (i < A.length && j < B.length) {

            if (A[i] < B[j]) {

                result [k++] = A[i++];

            }

            else {

                result[k++] = B[j++];

            }

        }

        while (i < A.length) {

            result[k++] = A[i++];

        }

        while (j < B.length) {

            result[k++] = B[j++];

        }

        for (int x = 0; x < result.length; x++) {

            System.out.print(result[x] + " ");

        }

        sc.close();

    }

}

**Pseudo code:**

Procedure mergeArrays(A, B, n, m):

// A has enough space to hold elements of B

i = n - 1 // Pointer for the end of the initial elements in A

j = m - 1 // Pointer for the end of B

k = n + m - 1 // Pointer for the end of the buffer in A

// Merge in reverse order

While i >= 0 and j >= 0:

If A[i] > B[j]:

A[k] = A[i]

i = i - 1

Else:

A[k] = B[j]

j = j - 1

k = k - 1

// Copy remaining elements of B if any

While j >= 0:

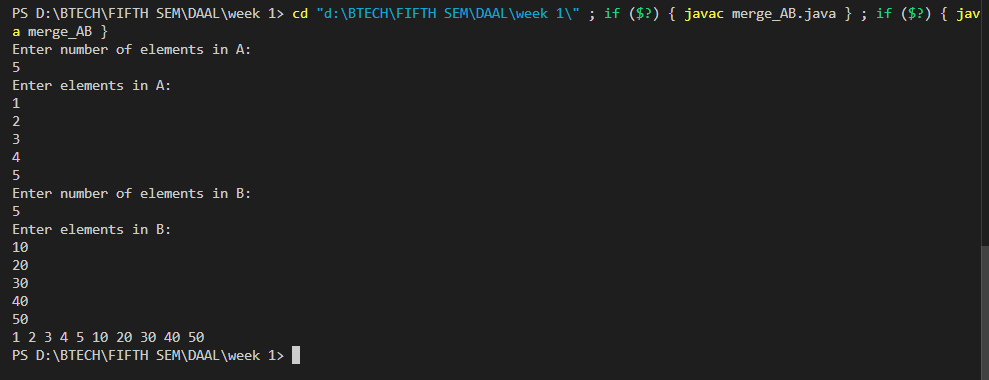
A[k] = B[j]

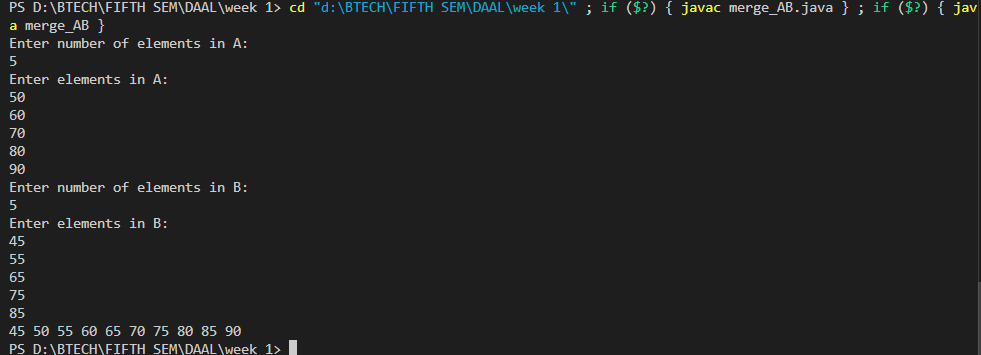
j = j - 1

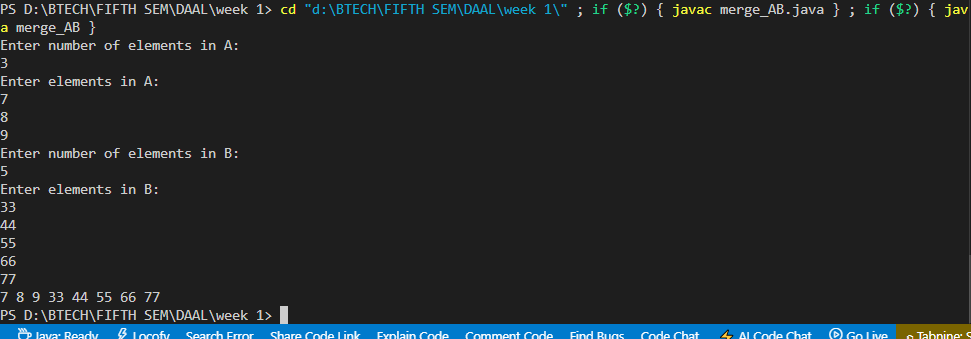
k = k - 1

End Procedure

Output:







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

import java.util.\*;

public class sort\_arr\_anagrams {

    private static String sortCharacters(String s) {

        char[] chars = s.toCharArray();

        Arrays.sort(chars);

        return new String(chars);

    }

    public static void sortAnagrams(String[] array) {

        Arrays.sort(array, new Comparator<String>() {

            @Override

            public int compare(String s1, String s2) {

                return sortCharacters(s1).compareTo(sortCharacters(s2));

            }

        });

    }

    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();

        sc.nextLine();

        System.out.println("Enter strings:");

        String[] array = new String[n];

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextLine();

        }

        sc.close();

        sortAnagrams(array);

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

    }

}

**Pseudo code:**

Procedure sortAnagrams(array):

Define a function sortCharacters(s):

Convert string s to character array chars

Sort chars array

Convert sorted chars array back to string

Return sorted string

Define a custom comparator function for two strings s1 and s2:

sortedS1 = sortCharacters(s1)

sortedS2 = sortCharacters(s2)

Return the result of comparing sortedS1 and sortedS2

Sort array using the custom comparator

End Procedure

Procedure main:

Prompt user to enter number of elements in array

Read number of elements n

Initialize an array of size n

For i from 0 to n-1:

Prompt user to enter a string

Read string and store it in the array[i]

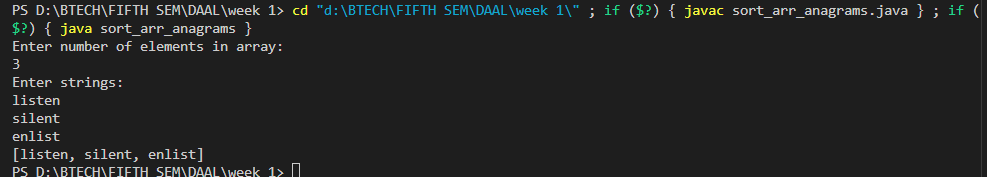
End For

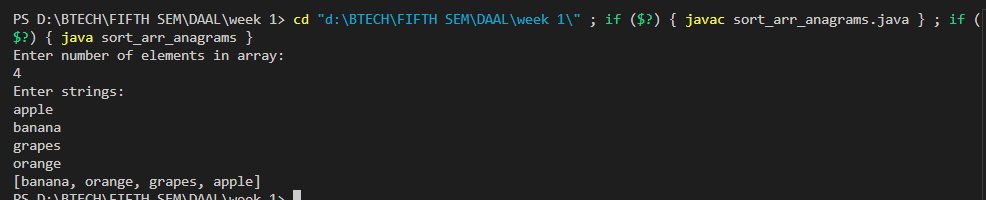
Call sortAnagrams with the array

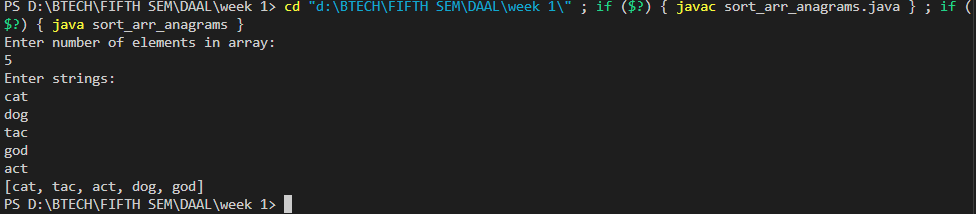
Print the sorted array

End Procedure

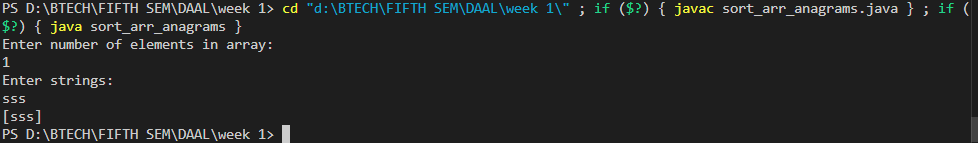
Output:











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.\*;

public class find\_index {

    public static void main(String args[]){

        Scanner sc=new Scanner(System.in);

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

        int n=sc.nextInt();

        int[] arr=new int[n];

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

        for(int i=0;i<n;i++){

            arr[i]=sc.nextInt();

        }

        System.out.println("Enter element to find:");

        int target=sc.nextInt();

        int index=find\_index(arr,target);

        System.out.println("Index of target elements: "+index);

        sc.close();

    }

    public static int find\_index(int[] arr,int target){

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

            if(arr[i]==target){

                return i;

            }

        }

        return -1;

    }

}

**Pseudo code:**

Procedure main:

Prompt user to enter the number of elements

Read the number of elements n

Initialize an array arr of size n

Prompt user to enter the array elements

For i from 0 to n-1:

Read the next element and store it in arr[i]

End For

Prompt user to enter the element to find

Read the target element

Call find\_index with arr and target, and store the result in index

Print "Index of target element: " followed by the value of index

Close the scanner

End Procedure

Function find\_index(arr, target):

For i from 0 to length of arr - 1:

If arr[i] is equal to target:

Return i

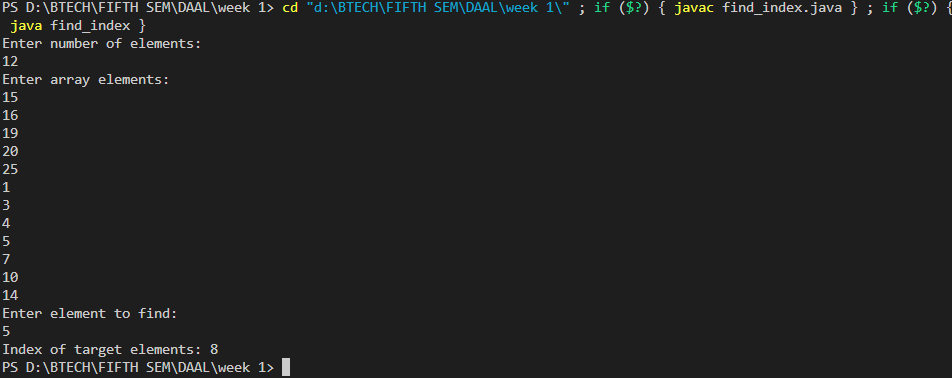
End If

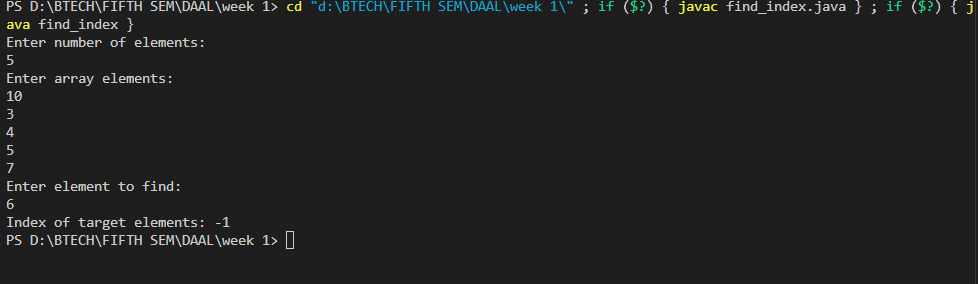
End For

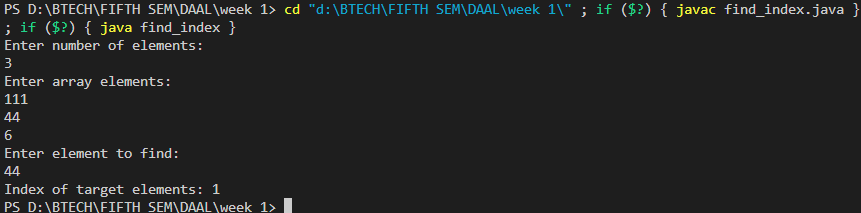
Return -1

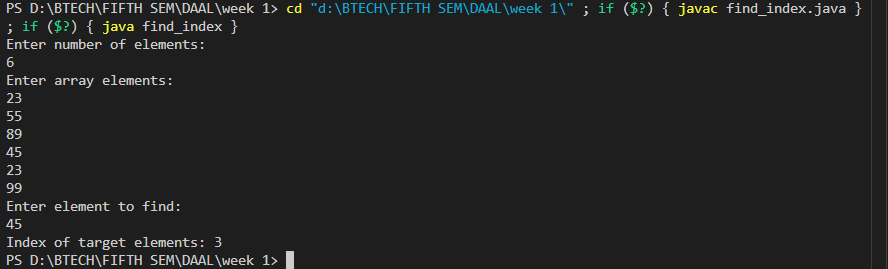
End Function

Output:









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 SparseSearch {

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

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

            return -1;

        }

        return findString(strings, target, 0, strings.length - 1);

    }

    public static int findString(String[] strings, String target, int low, int high) {

        if (low > high) {

            return -1;

        }

        int mid = (low + high) / 2;

        if (strings[mid].isEmpty()) {

            int left = mid - 1;

            int right = mid + 1;

            while (true) {

                if (left < low && right > high) {

                    return -1;

                } else if (right <= high && !strings[right].isEmpty()) {

                    mid = right;

                    break;

                } else if (left >= low && !strings[left].isEmpty()) {

                    mid = left;

                    break;

                }

                right++;

                left--;

            }

        }

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

            return mid;

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

            return findString (strings, target, mid + 1, high);

        } else {

            return findString (strings, target, low, mid - 1);

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

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

        int n = sc.nextInt();

        sc.nextLine();

        String[] strings = new String[n];

        System.out.println("Enter the strings:");

        for (int i = 0; i < n; i++) {

            strings[i] = sc.nextLine();

        }

        System.out.println("Enter the target string:");

        String target = sc.nextLine();

        int index = findString(strings, target);

        System.out.println("Index of '" + target + "': " + index);

        sc.close();

    }

}

Pseudo code:

Procedure findString(strings, target):

If strings is null OR target is null OR target is empty:

Return -1

End If

Return findString(strings, target, 0, length of strings - 1)

End Procedure

Function findString(strings, target, low, high):

If low is greater than high:

Return -1

End If

Calculate mid as (low + high) / 2

If strings[mid] is empty:

Set left to mid - 1

Set right to mid + 1

While True:

If left is less than low AND right is greater than high:

Return -1

Else If right is less than or equal to high AND strings[right] is not empty:

Set mid to right

Break out of the loop

Else If left is greater than or equal to low AND strings[left] is not empty:

Set mid to left

Break out of the loop

End If

Increment right

Decrement left

End While

End If

If strings[mid] is equal to target:

Return mid

Else If strings[mid] is less than target:

Return findString(strings, target, mid + 1, high)

Else:

Return findString(strings, target, low, mid - 1)

End If

End Function

Procedure main:

Prompt user to enter the number of strings in the array

Read number of strings n

Initialize an array of size n

Prompt user to enter the strings

For i from 0 to n-1:

Read the next string and store it in the array at index i

End For

Prompt user to enter the target string

Read the target string

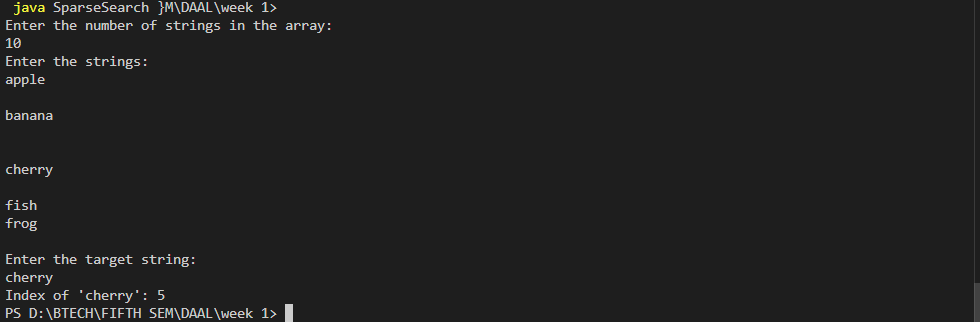
Call findString with the array and target

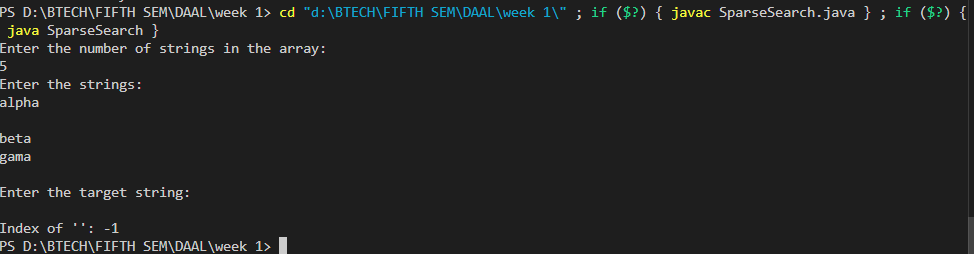
Print "Index of 'target': " followed by the result of findString

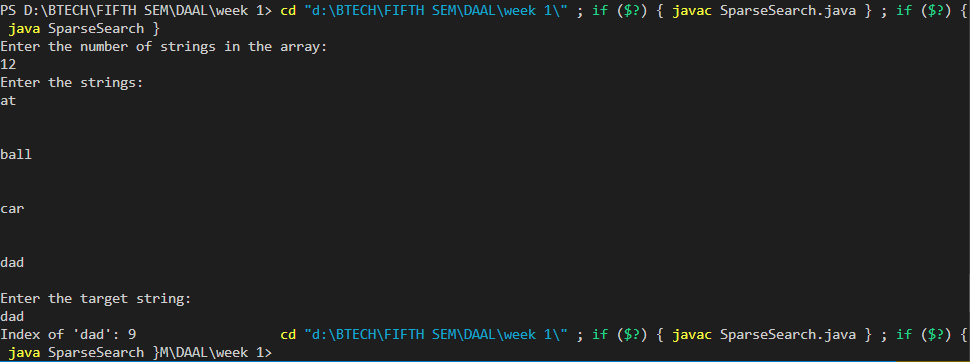
Close the scanner

End Procedure

Output:







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.Scanner;

public class SortedMatrixSearch {

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

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

            return false;

        }

        int rows = matrix.length;

        int cols = matrix[0].length;

        int row = 0;

        int col = cols - 1;

        while (row < rows && col >= 0) {

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

                return true;

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

                col--;

            } else {

                row++;

            }

        }

        return false;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of rows in the matrix:");

        int rows = sc.nextInt();

        System.out.println("Enter the number of columns in the matrix:");

        int cols = sc.nextInt();

        int[][] matrix = new int[rows][cols];

        System.out.println("Enter the elements of the matrix row by row:");

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

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

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

            }

        }

        System.out.println("Enter the target value to search:");

        int target = sc.nextInt();

        boolean result = findElement(matrix, target);

        System.out.println("Element " + target + " found: " + result);

        sc.close();

    }

}

**Pseudo code:**

Procedure findElement(matrix, target):

If matrix is null OR matrix has no rows OR matrix has no columns:

Return false

End If

Set rows to the number of rows in matrix

Set cols to the number of columns in matrix

Set row to 0

Set col to cols - 1

While row is less than rows AND col is greater than or equal to 0:

If matrix[row][col] is equal to target:

Return true

Else If matrix[row][col] is greater than target:

Decrement col

Else:

Increment row

End If

End While

Return false

End Procedure

Procedure main:

Prompt user to enter the number of rows in the matrix

Read the number of rows and store in rows

Prompt user to enter the number of columns in the matrix

Read the number of columns and store in cols

Initialize a matrix of size rows x cols

Prompt user to enter the elements of the matrix row by row

For i from 0 to rows - 1:

For j from 0 to cols - 1:

Read the next element and store it in matrix[i][j]

End For

End For

Prompt user to enter the target value to search

Read the target value

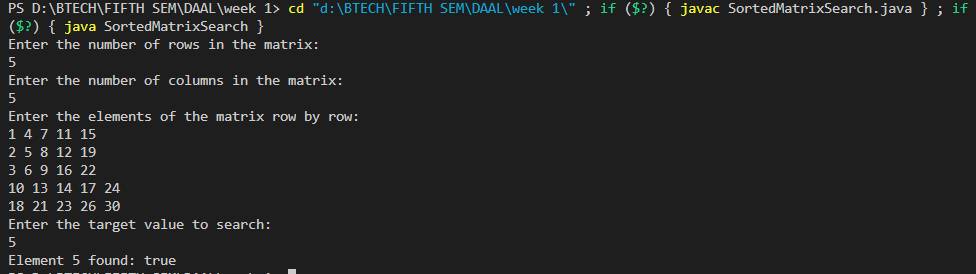
Call findElement with matrix and target, and store the result in result

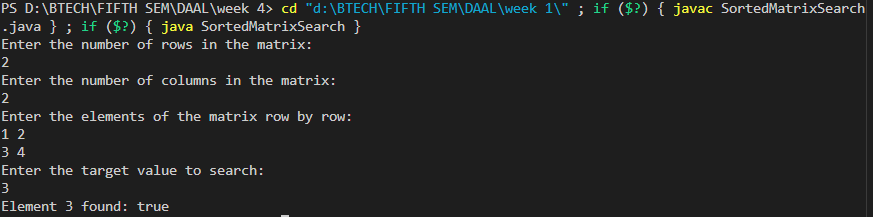
Print "Element target found: " followed by the value of result

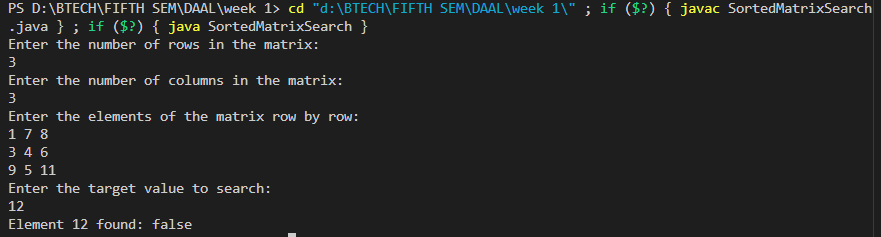
Close the scanner

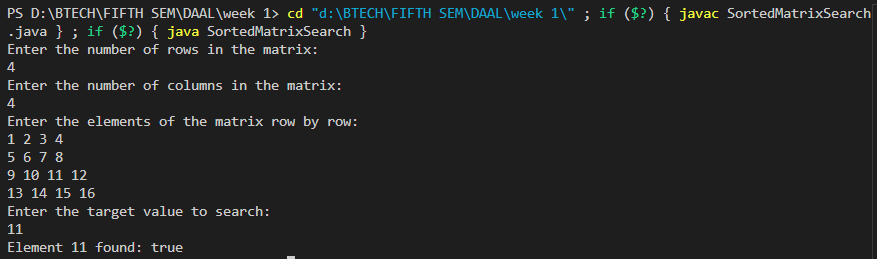
End Procedure

**Output:**









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.\*;

public class CircusTower {

    static class Person {

        int height;

        int weight;

        public Person(int height, int weight) {

            this.height = height;

            this.weight = weight;

        }

        @Override

        public String toString() {

            return "(" + height + ", " + weight + ")";

        }

    }

    public static List<Person> findLongestTower(List<Person> people) {

        people.sort((a, b) -> {

            if (a.height != b.height) {

                return a.height - b.height;

            } else {

                return a.weight - b.weight;

            }

        });

        int n = people.size();

        int[] weights = people.stream().mapToInt(p -> p.weight).toArray();

        int[] dp = new int[n];

        int[] prev = new int[n];

        Arrays.fill(dp, 1);

        Arrays.fill(prev, -1);

        int maxLength = 1;

        int bestEnd = 0;

        for (int i = 1; i < n; i++) {

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

                if (weights[i] > weights[j] && dp[i] < dp[j] + 1) {

                    dp[i] = dp[j] + 1;

                    prev[i] = j;

                }

            }

            if (dp[i] > maxLength) {

                maxLength = dp[i];

                bestEnd = i;

            }

        }

        List<Person> longestTower = new ArrayList<>();

        for (int i = bestEnd; i != -1; i = prev[i]) {

            longestTower.add(people.get(i));

        }

        Collections.reverse(longestTower);

        return longestTower;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

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

        int n = sc.nextInt();

        List<Person> people = new ArrayList<>();

        for (int i = 0; i < n; i++) {

            System.out.println("Enter height and weight for person " + (i + 1) + ":");

            int height = sc.nextInt();

            int weight = sc.nextInt();

            people.add(new Person(height, weight));

        }

        List<Person> longestTower = findLongestTower(people);

        System.out.println("The longest tower is length " + longestTower.size() + " and includes from top to bottom:");

        for (Person p : longestTower) {

            System.out.println(p);

        }

        sc.close();

    }

}

Define Person class:

Attributes:

- height

- weight

Constructor:

Initialize height and weight

Method toString:

Return a string representation of height and weight

Procedure findLongestTower(people):

Sort people by height in ascending order

If two people have the same height, sort by weight in ascending order

Initialize:

- n as the number of people

- weights array to store the weight of each person

- dp array where dp[i] will store the length of the longest tower ending with person i

- prev array to track the predecessor of each person in the tower

- maxLength as 1

- bestEnd as 0

Fill weights array with the weight of each person

Initialize dp array with all values set to 1

Initialize prev array with all values set to -1

For i from 1 to n - 1:

For j from 0 to i - 1:

If weights[i] > weights[j] AND dp[i] < dp[j] + 1:

Update dp[i] to dp[j] + 1

Update prev[i] to j

If dp[i] > maxLength:

Update maxLength to dp[i]

Update bestEnd to i

Initialize longestTower as an empty list

Backtrack from bestEnd to build the longestTower:

Add person at index bestEnd to longestTower

Update bestEnd to prev[bestEnd]

Reverse longestTower to get the correct order from top to bottom

Return longestTower

End Procedure

Procedure main:

Prompt user to enter the number of people

Read the number of people and store in n

Initialize people list

For i from 0 to n - 1:

Prompt user to enter height and weight for person i

Read height and weight

Create a new Person object and add to people list

End For

Call findLongestTower with people list and store the result in longestTower

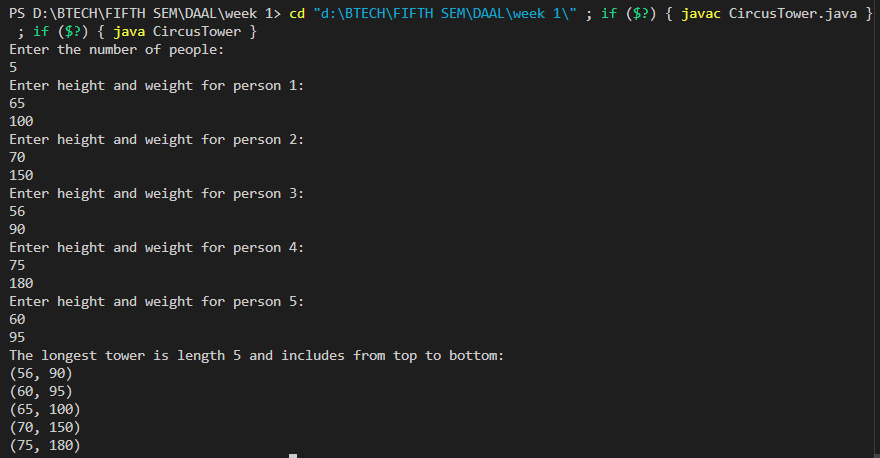
Print the length of longestTower

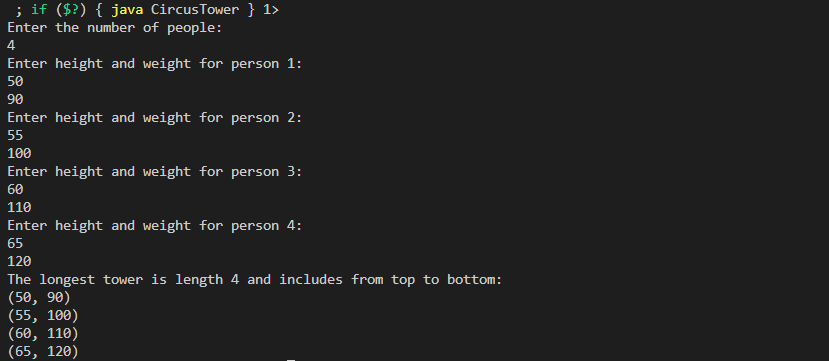
Print each person in longestTower

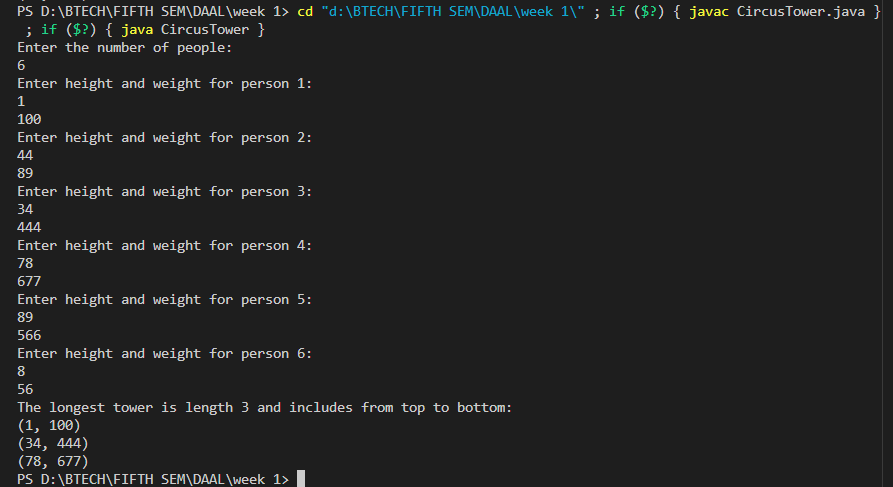
Close the scanner

End Procedure

**Output:**







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.Scanner;

public class StreamRank {

    private static class Node {

        int value;

        int leftSize = 0;

        Node left, right;

        public Node(int value) {

            this.value = value;

        }

    }

    private Node root;

    public void track(int x) {

        if (root == null) {

            root = new Node(x);

        } else {

            insert(root, x);

        }

    }

    public void insert(Node root, int x) {

        if (x <= root.value) {

            root.leftSize++;

            if (root.left == null) {

                root.left = new Node(x);

            } else {

                insert(root.left, x);

            }

        } else {

            if (root.right == null) {

                root.right = new Node(x);

            } else {

                insert(root.right, x);

            }

        }

    }

    public int getRankOfNumber(int x) {

        return getRank(root, x);

    }

    private int getRank(Node root, int x) {

        if (root == null) {

            return -1;

        }

        if (x == root.value) {

            return root.leftSize;

        } else if (x < root.value) {

            return getRank(root.left, x);

        } else {

            int rightRank = getRank(root.right, x);

            if (rightRank == -1) {

                return -1;

            }

            return root.leftSize + 1 + rightRank;

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        StreamRank streamRank = new StreamRank();

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

        int n = sc.nextInt();

        System.out.println("Enter the elements of the stream:");

        for (int i = 0; i < n; i++) {

            int number = sc.nextInt();

            streamRank.track(number);

        }

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

        int q = sc.nextInt();

        for (int i = 0; i < q; i++) {

            System.out.println("Enter the number to get its rank:");

            int number = sc.nextInt();

            System.out.println("getRankOfNumber(" + number + ") = " + streamRank.getRankOfNumber(number));

        }

        sc.close();

    }

}

**Pseudo code:**

Define Node class:

Attributes:

- value

- leftSize (initially 0)

- left (reference to left child node)

- right (reference to right child node)

Constructor:

Initialize value with given value

Procedure track(x):

If root is null:

Create a new Node with value x and assign it to root

Else:

Call insert(root, x) to insert x into the tree

End Procedure

Procedure insert(root, x):

If x <= root.value:

Increment root.leftSize by 1

If root.left is null:

Create a new Node with value x and assign it to root.left

Else:

Call insert(root.left, x) to insert x into the left subtree

Else:

If root.right is null:

Create a new Node with value x and assign it to root.right

Else:

Call insert(root.right, x) to insert x into the right subtree

End Procedure

Procedure getRankOfNumber(x):

Return result of getRank(root, x)

End Procedure

Procedure getRank(root, x):

If root is null:

Return -1 (not found)

If x == root.value:

Return root.leftSize (rank of x)

Else If x < root.value:

Return result of getRank(root.left, x) (rank in left subtree)

Else:

Call getRank(root.right, x) and store result in rightRank

If rightRank is -1:

Return -1 (not found in right subtree)

Return root.leftSize + 1 + rightRank (rank including root)

End Procedure

Procedure main:

Create an instance of StreamRank called streamRank

Prompt user to enter the number of elements in the stream

Read integer n

Prompt user to enter the elements of the stream

For i from 0 to n - 1:

Read integer number

Call streamRank.track(number) to add number to the stream

End For

Prompt user to enter the number of queries

Read integer q

For i from 0 to q - 1:

Prompt user to enter the number to get its rank

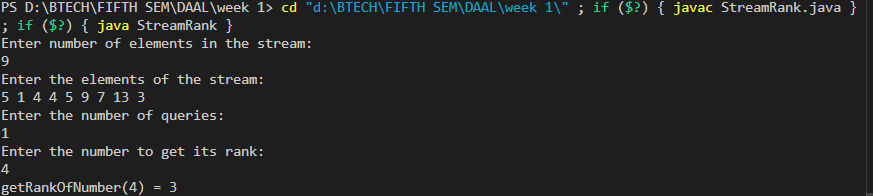
Read integer number

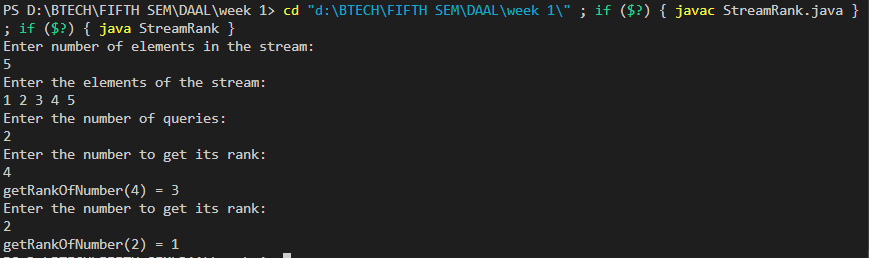
Print "getRankOfNumber(number) = " followed by result of streamRank.getRankOfNumber(number)

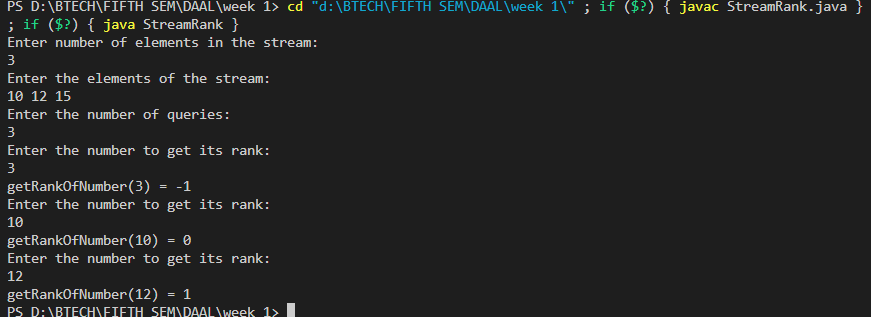
End For

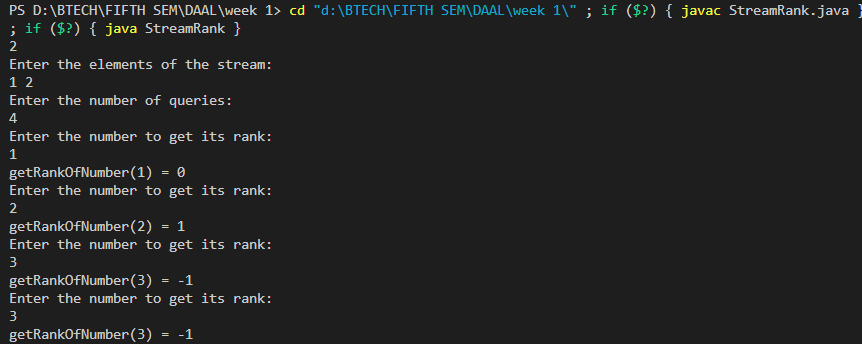
Close the scanner

End Procedure









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

**Algorithm:**

**Step 1: Split the File**: Divide the large file into smaller chunks that can fit into memory. For example, if you have 1GB of RAM, you might choose to create chunks of 500MB each.

**Step 2: Sort Each Chunk**: Read each chunk into memory, sort it using an in-memory sorting algorithm, and write the sorted data back to disk as individual sorted files.

**Step 3: Merge Sorted Chunks**: Merge the sorted chunks using a multi-way merge algorithm. This involves reading the smallest elements from each sorted chunk and writing the result to a new file. You may need to use a priority queue to efficiently find the smallest element among the current elements of each chunk.

import java.io.\*;

import java.util.\*;

public class ExternalSort {

    private static final int CHUNK\_SIZE = 100\_000;

    public static void splitAndSortChunks(String inputFile, String tempDir) throws IOException {

        BufferedReader reader = new BufferedReader(new FileReader(inputFile));

        List<String> lines = new ArrayList<>(CHUNK\_SIZE);

        int chunkCount = 0;

        String line;

        while ((line = reader.readLine()) != null) {

            lines.add(line);

            if (lines.size() == CHUNK\_SIZE) {

                Collections.sort(lines);

                writeChunk(lines, tempDir, chunkCount++);

                lines.clear();

            }

        }

        if (!lines.isEmpty()) {

            Collections.sort(lines);

            writeChunk(lines, tempDir, chunkCount);

        }

        reader.close();

    }

    private static void writeChunk(List<String> lines, String tempDir, int chunkNumber) throws IOException {

        File tempFile = new File(tempDir, "chunk" + chunkNumber + ".txt");

        BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile));

        for (String line : lines) {

            writer.write(line);

            writer.newLine();

        }

        writer.close();

    }

    public static void mergeChunks(String tempDir, String outputFile, int chunkCount) throws IOException {

        PriorityQueue<BufferedReaderWrapper> pq = new PriorityQueue<>();

        List<BufferedReader> readers = new ArrayList<>();

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

            File chunkFile = new File(tempDir, "chunk" + i + ".txt");

            BufferedReader reader = new BufferedReader(new FileReader(chunkFile));

            readers.add(reader);

            pq.add(new BufferedReaderWrapper(reader));

        }

        BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile));

        while (!pq.isEmpty()) {

            BufferedReaderWrapper wrapper = pq.poll();

            writer.write(wrapper.peek());

            writer.newLine();

            if (wrapper.advance()) {

                pq.add(wrapper);

            } else {

                wrapper.close();

            }

        }

        writer.close();

        for (BufferedReader reader : readers) {

            reader.close();

        }

    }

    static class BufferedReaderWrapper implements Comparable<BufferedReaderWrapper> {

        private BufferedReader reader;

        private String currentLine;

        public BufferedReaderWrapper(BufferedReader reader) throws IOException {

            this.reader = reader;

            advance();

        }

        public String peek() {

            return currentLine;

        }

        public boolean advance() throws IOException {

            currentLine = reader.readLine();

            return currentLine != null;

        }

        public void close() throws IOException {

            reader.close();

        }

        @Override

        public int compareTo(BufferedReaderWrapper other) {

            return this.currentLine.compareTo(other.currentLine);

        }

    }

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

        String inputFile = "largefile.txt";

        String tempDir = "D:\\BTECH\\FIFTH SEM\\DAAL Assignments";

        String outputFile = "sortedfile.txt";

        splitAndSortChunks(inputFile, tempDir);

        int chunkCount = new File(tempDir).list().length;

        mergeChunks(tempDir, outputFile, chunkCount);

    }

}