Contents

[**Mathematics Problems:** 3](#_Toc70013202)

[Add Fraction 3](#_Toc70013203)

[Dot Product 3](#_Toc70013204)

[Is Power of 10 3](#_Toc70013205)

[Power of Expo ( Math.pow) 4](#_Toc70013206)

[Prime Factorization 4](#_Toc70013207)

[Square root 4](#_Toc70013208)

[Decimal Conversion 4](#_Toc70013209)

[**String & Pattern Problems:** 5](#_Toc70013210)

[Dist. Between Strings 5](#_Toc70013211)

[Longest Word 5](#_Toc70013212)

[Apache Log Pattern 6](#_Toc70013213)

[First NonRepeating Character 7](#_Toc70013214)

[Group Anagrams 7](#_Toc70013215)

[Longest Uniform Substring 7](#_Toc70013216)

[Run Length Encoding 8](#_Toc70013217)

[Pangram 8](#_Toc70013218)

[Reverse String 8](#_Toc70013219)

[Reverse String Bug 9](#_Toc70013220)

[**Numbers/Numeric Problems:** 9](#_Toc70013221)

[Smallest Number 9](#_Toc70013222)

[Second Smallest 9](#_Toc70013223)

[**Data Structure Implementation:** 10](#_Toc70013224)

[Deque 10](#_Toc70013225)

[HashMap 10](#_Toc70013226)

[**Tree** 11](#_Toc70013227)

[Search Tree 11](#_Toc70013228)

[Largest Tree 13](#_Toc70013229)

[**Arrays:** 14](#_Toc70013230)

[Median Two Sorted Arrays 14](#_Toc70013231)

[SubArray Exceeding Sum 15](#_Toc70013232)

[**Dynamic Programming:** 15](#_Toc70013233)

[Student Election Program 15](#_Toc70013234)

[Walking Robot 16](#_Toc70013235)

[Optimal Path 17](#_Toc70013236)

[Staircase 17](#_Toc70013237)

[Train Map 18](#_Toc70013238)

[**Miscellaneous:** 19](#_Toc70013239)

[Count Length Of Cycle 19](#_Toc70013240)

[Magic Potion 20](#_Toc70013241)

[Pascals Triangle 20](#_Toc70013242)

[Unique Tuples 21](#_Toc70013243)

[Best Average Grade 21](#_Toc70013244)

[Snowpack 22](#_Toc70013245)

# **Mathematics Problems:**

## Add Fraction

|  |
| --- |
| Problem Statement-  Given two fractions passed in as int arrays,  returns the fraction which is result of adding the two input fractions  Fraction is represented as a two-element array - [ numerator, denominator ]  The returned fraction has to be in its simplest form.  Signature:  int[] addFractions ( int[] fraction1, int[] fraction2 ){  }  Test Cases:  INPUT:  fraction1 -[2, 3];  fraction2 -[1, 2];  OUTPUT: Hint- {2/3 +1/2}  result = [7,6] |

## Dot Product

|  |
| --- |
| Problem Statement-  two arrays of integers, returns the dot product of the arrays.  Signature:  long dotProduct( int[] array1, int array2[] ){  }  Test Cases: INPUT:  array1 = [1, 2];  array2 = [2, 3];  OUTPUT: hint {1\*2+2\*3}  Result =[8] |

## Is Power of 10

|  |
| --- |
| Problem Statement-  Returns true if x is a power-of-10. Otherwise returns false.  Signature:  bool isPowerOf10(int x){  }  Test Cases:  INPUT:  Input1: 3  Output1: false  Input1: 10  Output1: true |

## Power of Expo ( Math.pow)

|  |
| --- |
| Problem Statement-  Given base and integer exponent, compute value of base raised to the power of exponent.  Signature:  public static double power(double base, int exp) {  }k  Test Cases:  Input: 2.0  4  Output: 16.0 |

## Prime Factorization

|  |
| --- |
| Problem Statement:  Return an array containing prime numbers whose product is x  Examples:  primeFactorization( 6 ) == [2,3]  primeFactorization( 5 ) == [5]  primeFactorization( 12 ) == [2,2,3]  Signature:  public static ArrayList<Integer>primeFactorization(int x) {  }  Test Cases- Input: 6  Output:[2,3] |
|  |

## Square root

|  |
| --- |
| Problem Statement-  Returns square root of the given double  Signature:  double squareRoot( double x ){  }  Test Cases:  Input1: 4  Ouput1:2  Input2:2  Output2:1.41421 |

## Decimal Conversion

|  |
| --- |
| Problem Statement:  Implement the method that provided numerator and denominator will return a string representing fraction's decimal form.  Some fractions in decimal form have cyclic decimal points.  public static String vulgarToDecimal(Long numerator, Long denominator) {  }  Test Cases:  vulgarToDecimal(1l, 2l).equals("0.5");  vulgarToDecimal(1l, 3l).equals("0.(3)");  vulgarToDecimal(1l, 30l).equals("0.0(3)");  vulgarToDecimal(1l, 75l).equals("0.01(3)");  vulgarToDecimal(4l, 7l).equals("0.(571428)"); |

# **String & Pattern Problems:**

## Dist. Between Strings

|  |
| --- |
| Problem Statement –  Given two words returns the shortest distance between their two midpoints in number of characters, words can appear multiple times in any order and should be case insensitive.  Signature :  public static double shortestDistance(String document, String word1, String word2)  {  }  Test Cases:  String Document – “In publishing and graphic design, lorem ipsum is a filler text commonly used to demonstrate the graphic elements”.  shortestDistance( document, "is", "a" ) == 2.5 |

## Longest Word --

|  |
| --- |
| Problem Statement-  Given a string of letters and a dictionary, the function longestWord should  find the longest word or words in the dictionary that can be made from the letters  Input: letters = "oet", dictionary = {"to","toe","toes"}  Output: {"toe"}  Only lowercase letters will occur in the dictionary and the letters  The length of letters will be between 1 and 10 characters  The solution should work well for a dictionary of over 100,000 words  Signature- Set<String>longestWord(String letters, Dictionary dictionary) {  }  Test Cases-  Dictionary dict = new Dictionary(new String[]{"to", "toe", "toes", "doe", "dog", "god", "dogs", "book", "banana"});  Input- toe  Output- toe  Input –oetdg  Output- "doe", "toe", "dog", "god" |

## Apache Log Pattern

|  |
| --- |
| Problem Statement-  Given an Apache log file, return IP address(es) which accesses the site most often.  our log is in this format (Common Log Format). One entry per line.  10.0.0.1 - frank [10/Dec/2000:12:34:56 -0500] "GET /a.gif HTTP/1.0" 200 234  Log file entries are passsed as an array.  NOTE: In case of tie, this returns a comma-separated list of the IPaddresses. Tie is not mentioned explicitly in the exercise on purpose.  Signature:  String findTopIpaddress(String[] lines){  }  Test Cases:  Input:  String lines[] = new String[]{  "10.0.0.1 - frank [10/Dec/2000:12:34:56 -0500] \"GET /a.gif HTTP/1.0\" 200 234",  "10.0.0.1 - frank [10/Dec/2000:12:34:57 -0500] \"GET /b.gif HTTP/1.0\" 200 234",  "10.0.0.2 - nancy [10/Dec/2000:12:34:58 -0500] \"GET /c.gif HTTP/1.0\" 200 234"};  Output :10.0.0.1  Input:  String lines[] = new String[]{  "10.0.0.1 - frank [10/Dec/2000:12:34:56 -0500] \"GET /a.gif HTTP/1.0\" 200 234",  "10.0.0.1 - frank [10/Dec/2000:12:34:57 -0500] \"GET /b.gif HTTP/1.0\" 200 234",  "10.0.0.2 - nancy [10/Dec/2000:12:34:58 -0500] \"GET /c.gif HTTP/1.0\" 200 234",  "10.0.0.2 - nancy [10/Dec/2000:12:34:59 -0500] \"GET /c.gif HTTP/1.0\" 200 234",  "10.0.0.3 - logan [10/Dec/2000:12:34:59 -0500] \"GET /d.gif HTTP/1.0\" 200 234",};  Output - 10.0.0.1,10.0.0.2 |

## First NonRepeating Character

|  |
| --- |
| Problem Statement –  Finds the first character that does not repeat anywhere in the input string  If all characters are repeated, return 0  Given “apple”, the answer is “a”  Given “racecars”, the answer is “e"  Signature :  char findFirst(String input){  }  Test Cases-  Input: apple  Output: a  Input – xxyyzz  Output - 0 |

## Group Anagrams

|  |
| --- |
| Problem Statement:  Given a list of words, group them by anagrams  Input: List of "cat", "dog", "god"  Output: A Set of Sets of anagrams: {{‘cat'}, {‘dog', ‘god'}}  Signature :Set<Set<String>> group(List<String> words){  }  Test Cases –  Input :["cat", "dog", "god"]  Output: [[“cat”],[“dog”,”god”]] |

## Longest Uniform Substring

|  |
| --- |
| Problem Statement-  This method should return an integer array with two elements that correctly identifies the location of the longest uniform substring within the input string. The first element of the array should be the starting index of the longest  substring and the second element should be the length.  input: “abbbccda" the longest uniform substring is “bbb” (which starts at index 1 and is 3 characters long.  Signature :  int[] longestUniformSubstring(String input) {  }  -Test Cases –  Input :aabbbbbCdAA  Output – [hi] |

## Run Length Encoding

|  |
| --- |
| Problem Statement-  For a string input the function returns output encoded as follows:  "a" -> "a1"  "aa" -> "a2"  "aabbb" -> "a2b3"  "aabbbaa" -> "a2b3a2"  Signature :  String rle(String input) {  }  Test Case:  Input :aaabbbaad  Output: a3b3a2d1 |

## Pangram

|  |
| --- |
| Problem Statement-  The sentence “The quick brown fox jumps over the lazy dog" contains  every single letter in the alphabet. Such sentences are called pangrams.  write a function findMissingLetters, which takes a String “sentence,  and returns all the letters it is missing (which prevent it from  being a pangram). You should ignore the case of the letters in sentence,  and your return should be all lower case letters, in alphabetical order.  you should also ignore all non US-ASCII characters.  Signature:  String findMissingLetters(String sentence) {  }  Test Cases:  Input:The slow purple oryx meanders past the quiescent canine  Output:bfgjkvz |

## Reverse String

|  |
| --- |
| Problem Statement :  Takes String str and returns a new String  such that the characters are in reversed order.  Example: reverseStr(str) where str is “abcd" returns “dcba".  Signature: public static String reverseStr( String str ){  }  Test Cases:  Input :abcd  Output: dcba |

## Reverse String Bug

|  |
| --- |
| Problem Statement :  Takes String str and returns a new String  such that the characters are in reversed order.  Example: reverseStr(str) where str is “abcd " returns “dcba".  Signature: public static String reverseStr( String str ){  }  Test Cases:  Input :abcd  Output: dcba |

# **Numbers/Numeric Problems:**

## Smallest Number

|  |
| --- |
| Problem Statement-  Returns the smallest number in array that has been rotated  For example - Array {3,4,5,6,1,2} returns 1  Input array was originally sorted in increasing orders  FindMinInArray must have O(log n) runtime  Input array does not have any duplicates  Signature:  public static int FindMin(int a[]){  }  Test Cases:  Input: [3, 4, 5, 6, 1, 2]  Output: 1  Input: [2, 1]  Output:1 |

## Second Smallest

|  |
| --- |
| Problem Statement-  Returns second smallest element in array x. If x has fewer than 2 elements returns 0.  Signature:  int secondSmallest(int[] x)  Test Cases:  Input:[-1, 0, 1, -2, 2]  Output: -1  Input:[0, 1]  Output: 1 |

# **Data Structure Implementation:**

## Deque

|  |
| --- |
| Problem Statement:  Implement a double-ended queue (abbreviated to deque) that stores strings.  A deque is a data structure that has characteristics of both a queue and a stack.  Elements can be added or removed from either the front or back.  Signature:  Create Class Deque |

## HashMap

|  |
| --- |
| Problem Statement:  Create class MyHashMap  Associates a key-value pair in memory such that lookups  and inserts can be performed in 0(1) time for a reasonably  small set of data, and scales linearly (at worst) for larger  sets of key-value pairs.  Each unique key is associated with one single value.  Signature :  Create Class MyHashMap |

# **Tree**

## Search Tree

|  |
| --- |
| Problem Statement:  Implement the “put" and “contains” methods.  Fix the "“inOrderTraversal” method.  public class Problem {  static class BST {  private Node root;  public BST() {  this.root = new Node();  }  public void put(int value) {  // TODO: implement me  }  public booleancontains(int value) {  // TODO: implement me  return false;  }  public List<Integer>inOrderTraversal() {  final ArrayList<Integer> acc = new ArrayList<>();  inOrderTraversal(root, acc);  return acc;  }  private void inOrderTraversal(Node node, List<Integer> acc) {  if (node == null) {  return;  }  inOrderTraversal(node.left, acc);  inOrderTraversal(node.right, acc);  acc.add(node.val);  }  private static class Node {  Integer val;  Node left;  Node right;  }  }  public static void testBST() {  final BST searchTree = new BST();  searchTree.put(3);  searchTree.put(1);  searchTree.put(2);  searchTree.put(5);  assertFalse(searchTree.contains(0));  assertTrue(searchTree.contains(1));  assertTrue(searchTree.contains(5));  assertFalse(searchTree.contains(6));  assertEquals(Arrays.asList(1, 2, 3, 5), searchTree.inOrderTraversal());  }  private static void assertFalse(booleanrez) {  if (rez) {  throw new RuntimeException("Test failed");  } else {  System.out.println("Test passed");  }  }  private static void assertTrue(booleanrez) {  if (!rez) {  throw new RuntimeException("Test failed");  } else {  System.out.println("Test passed");  }  }  private static void assertEquals(List<Integer> expected, List<Integer> result) {  if (!expected.equals(result)) {  System.out.println(String.format("Test failed \"%s\" not equeals to \"%s\"", expected, result));  } else {  System.out.println("Test passed");  }  }  // TODO: write some more tests  public static void main(String[] args) {  testBST();  }  } |

## Largest Tree

|  |
| --- |
| Problem statement:  Given a forest ( one or more disconnected trees ), find the root of largest tree  and return its Id. If there are multiple such roots, return the smallest Id of them.  Complete the largestTree function in the editor below.  It has one parameter, immediateParent, which is a map containing key-value pair indicating  child -> parent relationship. The key is child and value is the corresponding  immediate parent.  Constraints  Child cannot have more than one immediate parent.  Parent can have more than one immediate child.  The given key-value pair forms a well-formed forest ( a tree of n nodes will have n-1 edges )  Example:  Input:  {{1->2}, {3 -> 4} }  Expected output: 2  Explanation: There are two trees one having root of Id 2 and another having root of Id 4.  Both trees have size 2. The smaller number of 2 and 4 is 2. Hence the answer is 2.  Signature :  public static Integer largestTree(final Map<Integer, Integer>immediateParent) {  } |

# **Arrays:**

## Median Two Sorted Arrays

|  |
| --- |
| Problem Statement:  Find the median of the two sorted arrays  Signature:  public static double findMedianSortedArrays(int[] A, int[] B) {  }  Test Cases:  Input:  Arr1= [1,3];  Arr2 = [2,4];  Output:  2.5  Input:  Arr1 = [1,3]  Arr2= [2]  Output:2.0 |

## SubArray Exceeding Sum

|  |
| --- |
| Problem Statement:  Your task is ultimately to implement a function that takes in an array and a integer.  You want to return the \*LENGTH\* of the shortest subarray whose sum is at least the integer,  and -1 if no such sum exists.  Signature:  public static int subArrayExceedsSum(int arr[], int target) {  }  Test Cases:  Input:[1,2,3,4,] , k=6  Output :2  Input:[1,2,3,4,] , k=-1  Output :12 |

# **Dynamic Programming:**

## Student Election Program

|  |
| --- |
| Problem Statement:  A group of students are sitting in a circle. The teacher is electing a new class president.  The teacher does this by singing a song while walking around the circle. After the song is finished the student at which the teacher stopped is removed from the circle.  Starting at the student next to the one that was just removed, the teacher resumes singing and walking around the circle.  after the teacher is done singing, the next student is removed. The teacher repeats this until only one student is left.  A song of length k will result in the teacher walking past k students on each round. The students are numbered 1 to n. The teacher starts at student 1.  For example, suppose the song length is two (k=2). And there are four students to start with (1,2,3,4). The firststudent to go would be “2°, after that “4°, and after that ~3>. Student ~1° would be thenext president in this example.  Signature:  public static int whoIsElected(int n, int k) {  }  Test Cases:  Input :1, 1  Output :1  Input : 2, 2  Output :1  Input : 4,2  Output :1 |

## Walking Robot

|  |
| --- |
| Problem Statement:  Implement the ‘walk" method. This method takes in a string, path,  where each character in the string corresponds to a potential movement  of the robot. The robot can move up, down, left, and right represented  by the characters ‘U', 'D', "L', and 'R' respectively. All other  characters may be ignored. Assume the robot's initial position  is at (0,0). The output of this method is the robot's final x and y  coordinates relative to the initial position  Signature:  public static Integer[] walk(String path) {  }  UUUDLR  x=0  y=0;  case 'U': y++  case 'D' : y--  case 'L': x--  case 'R':x++  Test Cases:  Input: “”(Blank)  Output: [0,0]  Input: “L”  Output: [-1,0]  Input: “UUU”  Output: [0,3]  Input: “ULDR”  Output: [0,0] |

## Optimal Path

|  |
| --- |
| Problem Statement:  You are an avid rock collector who lives in southern California. Some rare  and desirable rocks just became available in New York, so you are planning  a cross-country road trip. There are several other rare rocks that you could  pick up along the way.  You have been given a grid filled with numbers, representing the number of  rare rocks available in various cities across the country. Your objective  is to find the optimal path from So\_Cal to New\_York that would allow you to  accumulate the most rocks along the way.  Note: You can only travel either north (up) or east (right).  2) Consider adding some additional tests in doTestsPass().  3) Implement optimalPath() correctly.  4) Here is an example:  ^  {{0, 0, 0, 0, 5}, New\_York (finish) N  {0, 1, 1, 1, 0}, < W E >  So\_Cal (start) {2, 0, 0, 0, 0}} S  v  The total for this example would be 10 (2 + 0 + 1 + 1 + 1 + 0 + 5).  Signature:  public static Integer optimalPath(Integer[][] grid) {  }  Test Cases:  Input :  {{0, 0, 0, 0, 5},  {0, 1, 1, 1, 0},  {2, 0, 0, 0, 0}};  Output: 10 |

## Staircase

|  |
| --- |
| Problem Statement:  There is a staircase with ‘n' number of steps. A child  walks by and wants to climb up the stairs, starting at  the bottom step and ascending to the top.  Of course, the child wants to have fun, too, so instead  of taking 1 step at a time, it will vary between taking  either 1, 2 or 3 steps at a time.  Please complete the ‘countSteps' method below so that  given 'n' number of steps it will return the number of  unique combinations the child could traverse.  An example would be countSteps(3) == 4:  Signature :  public static Integer countSteps(Integer n) {  }  Test Cases:  Input : 3  Output: 4  Input : 1  Output: 1  Input : 2  Output: 2  Input : 10  Output: 274  Input : -5  Output: 0 |

## Train Map

|  |
| --- |
| Problem Statement:  shortestPath(self, fromStationName, toStationName)  method to find shortest path between 2 stations  /\*  \* Visual representation of the Train map used  \*  \* King's Cross St Pancras --- Angel ---- Old Street  \* | \ |  \* | \ |  \* | \ |  \* Russell Square Farringdon --- Barbican --- Moorgate  \* | /  \* | /  \* | /  \* Holborn --- Chancery Lane --- St Paul’s --- Bank  \*/  private static class TrainMap {  private HashMap<String, Station>stations;  public TrainMap() {  this.stations = new HashMap<>();  }  public TrainMapaddStation(String name) {  Station s = new Station(name);  this.stations.putIfAbsent(name, s);  return this;  }  public Station getStation(String name) {  return this.stations.get(name);  }  public TrainMapconnectStations(Station fromStation, Station toStation) {  if (fromStation == null) {  throw new IllegalArgumentException("From station is null");  }  if (toStation == null) {  throw new IllegalArgumentException("To station is null");  }  fromStation.addNeighbour(toStation);  toStation.addNeighbour(fromStation);  return this;  }  public List<Station>shortestPath(String from, String to) {  /\*  \* TODO Implement  \*/  return Collections.emptyList();  }  public static String convertPathToStringRepresentation(List<Station> path) {  if (path.isEmpty()) {  return "";  }  return path.stream().map(Station::getName).reduce((s1, s2) -> s1 + "->" + s2).get();  }} |

# **Miscellaneous:**

## Count Length Of Cycle

|  |
| --- |
| Problem Statement:  You are given an integer array of size N.  Every element of the array is greater than or equal to 0.  Starting from arr[startIndex], follow each element to the index it points to.  Continue to do this until you find a cycle.  Return the length of the cycle. If no cycle is found return -1  Examples:  Signature:  Int countLengthofcycle(arr, startIndex){  }  Test Cases:  Input :  arr: [1,0]  startIndex: 0  Output :2  Input :  arr: [1,2,0]  startIndex: 0  Output :2 |

## Magic Potion --

Brute Force

`

|  |
| --- |
| Problem Statement:  Hermione is preparing a cheat-sheet for her final exam in Potions class.  To create a potion, one must combine ingredients in a specific order, any of which may be repeated.  As an example, consider the following potion which uses 4 distinct ingredients  (A,B,C,D) in 11 steps: A, B, A, B, C, A, B, A, B, C, D  Hermione realizes she can save tremendous space on her cheat-sheet by introducing a  special instruction, ‘\*', which means “repeat from the beginning”.  Using these optimizations, Hermione is able to encode the potion above using only 6 characters: A,B,\*,C,\*,D  Your job is to write a function that takes as input an un-encoded potion and returns the  minimum number of characters required to encode the potion on Hermione’s Cheat Sheet.  A B A B C A B A B C D  AB\*C\*D  A B \* C A  ABABC\*D  Signature:  private Integer minimalSteps(String ingredients) {  }  Test Case:  Input:ABCDABCE  Output:8  Input:ABCABCE  Output: 5 |
|  |

## Pascals Triangle

|  |
| --- |
| Problem Statement:  Pascals Triangle exhibits the following behaviour:  The first and last numbers of each row in the triangle are 1  Each number in the triangle is the sum of the two numbers above it.  Dp[][]  {  }  Example:  1  1 1  1 2 1  1 3 3 1  1 4 6 4 1  1 5 10 10 5 1  1 6 15 20 15 6 1  Please Complete the ‘pascal’ function below so that given a  col and a row it will return the value in that position.  Example, pascal(1,2) should return 2  Col,  Signature:  public static int pascal(int col, int row){  }  Test Cases:  Input : 0,0  Output:1  Input :1,2  Output:2  Input :4,8  Output:70 |
|  |
|  |

## Unique Tuples

|  |
| --- |
| Problem Statement:  Given a string and size of the tuples, extracts all unique tuples(substrings) of the given size.  Signature:  HashSet<String>uniqueTuples( String input, int len ){  }  Test Cases:  Input :abbccde, 2  Output :  ["ab"  "bb",  "bc",  "cc",  "cd",  "de"] |

## Best Average Grade

|  |
| --- |
| Problem Statement:  Given a list of student test scores, find the best average grade.  Each student may have more than one test score in the list.  Complete the bestAverageGrade function in the editor below.  It has one parameter, scores, which is an array of student test scores.  Each element in the array is a two-element array of the form [student name, test score]  e.g. [ "Bobby", "87" ].  Test scores may be positive or negative integers.  If you end up with an average grade that is not an integer, you should  use a floor function to return the largest integer less than or equal to the average.  Return 0 for an empty input.  Example:  Input:  [ [ "Bobby", "87" ],  [ "Charles", "100" ],  [ "Eric", "64" ],  [ "Charles", "22" ] ].  Expected output: 87  Explanation: The average scores are 87, 61, and 64 for Bobby, Charles, and Eric,  respectively. 87 is the highest.  Signatue:  public static Integer bestAverageGrade(String[][] scores) {  }  Test Cases:  Input :  {{"Sarah", "91"},  {"Goldie", "92"},  {"Elaine", "93"},  {"Elaine", "95"},  {"Goldie", "94"},  {"Sarah", "93"}}  Output: 94 |

## Snowpack

|  |
| --- |
| Problem Statement:  Given an array of non-negative integers representing the elevations  from the vertical cross section of a range of hills, determine how  many units of snow could be captured between the hills.  See the example array and elevation map below.  \_\_\_  \_\_\_ | | \_\_\_  | | \_\_\_ | |\_\_\_ | |  \* \_\_\_| | \_\_\_| | | | | | |  \_\_\_|\_\_\_|\_\_\_|\_\_\_|\_\_\_|\_\_\_|\_\_\_|\_\_\_|\_\_\_|\_\_\_|\_\_\_|\_\_\_  { 0, 1, 3, 0, 1, 2, 0, 4, 2, 0, 3, 0 }  \_\_\_  \_\_\_ | | \_\_\_  | | \* \* \_\*\_ \* | |\_\*\_ \* | |  \* \_\_\_| | \* \_\*\_| | \* | | | \* | |  \_\_\_|\_\_\_|\_\_\_|\_\*\_|\_\_\_|\_\_\_|\_\*\_|\_\_\_|\_\_\_|\_\*\_|\_\_\_|\_\_\_  { 0, 1, 3, 0, 1, 2, 0, 4, 2, 0, 3, 0 }  Solution: In this example 13 units of snow (\*) could be captured.  Signature: public static Integer computeSnowpack(Integer[] arr) {  }  Test Cases:  Input :{0, 1, 3, 0, 1, 2, 0, 4, 2, 0, 3, 0}  Output: 13  Input :{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1}  Output:10 |