Contents

[**1.** SolveMeFirst 5](#_Toc520063706)

[2. Simple Array Sum 5](#_Toc520063707)

[3. Compare the Triplets 6](#_Toc520063708)

[4. A Very Big Sum 6](#_Toc520063709)

[5. Diagonal Difference 7](#_Toc520063710)

[6. Plus Minus 7](#_Toc520063711)

[7. Staircase 7](#_Toc520063712)

[8. Mini-Max Sum 8](#_Toc520063713)

[9. Birthday Cake Candles 8](#_Toc520063714)

[10. Time Conversion 8](#_Toc520063715)

[11. Grading Students 9](#_Toc520063716)

[12. Apple and Orange 9](#_Toc520063717)

[13. Kangaroo 10](#_Toc520063718)

[14. Between Two Sets 10](#_Toc520063719)

[15. Breaking the Records 11](#_Toc520063720)

[16. Birthday Chocolate 12](#_Toc520063721)

[17. Divisible Sum Pairs 13](#_Toc520063722)

[18. Migratory Birds 13](#_Toc520063723)

[19. Day of the Programmer 14](#_Toc520063724)

[20. Bon Appétit 14](#_Toc520063725)

[21. Sock Merchant 15](#_Toc520063726)

[22. Drawing Book 15](#_Toc520063727)

[23. Counting Valleys 15](#_Toc520063728)

[24. Electronics Shop 16](#_Toc520063729)

[25. Cats and a Mouse 17](#_Toc520063730)

[26. Forming a Magic Square 17](#_Toc520063731)

[27. Picking Numbers 18](#_Toc520063732)

[28. Climbing the Leaderboard 18](#_Toc520063733)

[29. The Hurdle Race 19](#_Toc520063734)

[30. Designer PDF Viewer 19](#_Toc520063735)

[31. Utopian Tree 20](#_Toc520063736)

[32. Angry Professor 20](#_Toc520063737)

[33. Beautiful Days at the Movies 20](#_Toc520063738)

[34. Viral Advertising 21](#_Toc520063739)

[35. Save the Prisoner! 21](#_Toc520063740)

[36. Circular Array Rotation 21](#_Toc520063741)

[37. Sequence Equation 22](#_Toc520063742)

[38. Jumping on the Clouds: Revisited 22](#_Toc520063743)

[39. Find Digits 23](#_Toc520063744)

[40. Extra Long Factorials 23](#_Toc520063745)

[41. Append and Delete 23](#_Toc520063746)

[42. Sherlock and Squares 24](#_Toc520063747)

[43. Library Fine 24](#_Toc520063748)

[44. Cut the sticks 25](#_Toc520063749)

[45. Non-Divisible Subset 25](#_Toc520063750)

[46. Repeated String 26](#_Toc520063751)

[47. Jumping on the Clouds 26](#_Toc520063752)

[48. Equalize the Array 26](#_Toc520063753)

[49. Queen's Attack II 27](#_Toc520063754)

[50. ACM ICPC Team 28](#_Toc520063755)

[51. Taum and B'day 29](#_Toc520063756)

[52. Organizing Containers of Balls 29](#_Toc520063757)

[53. Encryption 30](#_Toc520063758)

[54. Bigger is Greater 31](#_Toc520063759)

[55. Modified Kaprekar Numbers 32](#_Toc520063760)

[56. Beautiful Triplets 32](#_Toc520063761)

[57. Minimum Distances 33](#_Toc520063762)

[58. Halloween Sale 33](#_Toc520063763)

[59. The Time in Words 34](#_Toc520063764)

[60. Chocolate Feast 35](#_Toc520063765)

[61. Service Lane 35](#_Toc520063766)

[62. Lisa's Workbook 35](#_Toc520063767)

[63. Flatland Space Stations 36](#_Toc520063768)

[64. Fair Rations 37](#_Toc520063769)

[65. Cavity Map 37](#_Toc520063770)

[66. Manasa and Stones 38](#_Toc520063771)

[67. The Grid Search 38](#_Toc520063772)

[68. Happy Ladybugs 40](#_Toc520063773)

[69. Strange Counter 41](#_Toc520063774)

[70. 3D Surface Area 41](#_Toc520063775)

[71. Absolute Permutation 42](#_Toc520063776)

[72. The Bomberman Game 42](#_Toc520063777)

[73. Ema's Supercomputer 44](#_Toc520063778)

[74. Larry's Array 45](#_Toc520063779)

[75. Almost Sorted 46](#_Toc520063780)

[76. Matrix Layer Rotation 47](#_Toc520063781)

[77. Big Sorting 48](#_Toc520063782)

[78. Super Reduced String 48](#_Toc520063783)

[79. Intro to Tutorial Challenges 49](#_Toc520063784)

[80. CamelCase 49](#_Toc520063785)

[81. Insertion Sort - Part 1 49](#_Toc520063786)

[82. Strong Password 50](#_Toc520063787)

[83. Two Characters 51](#_Toc520063788)

[84. Insertion Sort - Part 2 51](#_Toc520063789)

[85. Correctness and the Loop Invariant 52](#_Toc520063790)

[86. Caesar Cipher 53](#_Toc520063791)

[87. Mars Exploration 53](#_Toc520063792)

[88. Running Time of Algorithms 54](#_Toc520063793)

[89. HackerRank in a String! 54](#_Toc520063794)

[90. Quicksort 1 - Partition 55](#_Toc520063795)

[91. Pangrams 55](#_Toc520063796)

[92. Weighted Uniform Strings 56](#_Toc520063797)

[93. Separate the Numbers 56](#_Toc520063798)

[94. Funny String 57](#_Toc520063799)

[95. Counting Sort 1 58](#_Toc520063800)

[96. Counting Sort 2 58](#_Toc520063801)

[97. Gemstones 58](#_Toc520063802)

[98. Alternating Characters 59](#_Toc520063803)

[99. The Full Counting Sort 59](#_Toc520063804)

[100. Beautiful Binary String 60](#_Toc520063805)

[101. Closest Numbers 60](#_Toc520063806)

[102. The Love-Letter Mystery 61](#_Toc520063807)

[103. Find the Median 61](#_Toc520063808)

[104. Palindrome Index 61](#_Toc520063809)

[105. Fraudulent Activity Notifications 62](#_Toc520063810)

[106. Anagram 63](#_Toc520063811)

[107. Lily's Homework 64](#_Toc520063812)

[108. Making Anagrams 65](#_Toc520063813)

[109. Game of Thrones - I 65](#_Toc520063814)

[110. Two Strings 65](#_Toc520063815)

[111. String Construction 66](#_Toc520063816)

[112. Sherlock and the Valid String 66](#_Toc520063817)

[113. Highest Value Palindrome 67](#_Toc520063818)

[114. Sherlock and Anagrams 68](#_Toc520063819)

[115. Common Child 68](#_Toc520063820)

[116. Bear and Steady Gene 69](#_Toc520063821)

[117. Morgan and a String 71](#_Toc520063822)

[118. Count Strings 72](#_Toc520063823)

[119. String Function Calculation 77](#_Toc520063824)

[120. Ice Cream Parlor 80](#_Toc520063825)

[121. Minimum Loss 81](#_Toc520063826)

[122. Missing Numbers 81](#_Toc520063827)

[123. Pairs 82](#_Toc520063828)

[124. Sherlock and Array 82](#_Toc520063829)

[125. Connected Cells in a Grid 83](#_Toc520063830)

[126. Short Palindrome 85](#_Toc520063831)

[127. Count Luck 85](#_Toc520063832)

[128. Cut the Tree 86](#_Toc520063833)

[129. The Coin Change Problem 88](#_Toc520063834)

[130. Equal 89](#_Toc520063835)

[131. Sam and substrings 89](#_Toc520063836)

[132. Fibonacci Modified 90](#_Toc520063837)

[133. Lonely Integer 90](#_Toc520063838)

[134. Maximizing XOR 91](#_Toc520063839)

[135. Sum vs XOR 91](#_Toc520063840)

[136. Flipping bits 91](#_Toc520063841)

[137. Recursive Digit Sum 91](#_Toc520063842)

[138. Game of Stones 92](#_Toc520063843)

[139. Tower Breakers 92](#_Toc520063844)

[140. A Chessboard Game 93](#_Toc520063845)

[141. Introduction to Nim Game 93](#_Toc520063846)

[142. Walking the Approximate Longest Path 94](#_Toc520063847)

[143. Pseudo-Isomorphic Substrings 95](#_Toc520063848)

[144. Insertion Sort Advanced Analysis 102](#_Toc520063849)

[145. Hackerland Radio Transmitters 103](#_Toc520063850)

[146. Gridland Metro 103](#_Toc520063851)

[147. Maximum Subarray Sum 104](#_Toc520063852)

1. **Solve Me First**

|  |
| --- |
| public class Solution {  static int solveMeFirst(int a, int b) {  return a+b;  }  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int a;  a = in.nextInt();  int b;  b = in.nextInt();  int sum;  sum = solveMeFirst(a, b);  System.out.println(sum);  }  } |

# Simple Array Sum

|  |
| --- |
| public class Solution {  static int simpleArraySum(int n, int[] ar) {  int sum =0;  for (n=0; n<ar.length; n++){  sum += ar[n];  }  return sum;  }  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int n = in.nextInt();  int[] ar = new int[n];  for(int ar\_i = 0; ar\_i < n; ar\_i++){  ar[ar\_i] = in.nextInt();  }  int result = simpleArraySum(n, ar);  System.out.println(result);  }  } |

# Compare the Triplets

|  |
| --- |
| public class Solution {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int a0 = in.nextInt();  int a1 = in.nextInt();  int a2 = in.nextInt();  int b0 = in.nextInt();  int b1 = in.nextInt();  int b2 = in.nextInt();  int aliceScore = 0;  int bobScore = 0;  if(a0 > b0) {  aliceScore++;  }  if(a1 > b1) {  aliceScore++;  }  if(a2 > b2) {  aliceScore++;  }  if(a0 < b0) {  bobScore++;  }  if(a1 < b1) {  bobScore++;  }  if(a2 < b2) {  bobScore++;  }  System.out.println(aliceScore + " " + bobScore);  in.close();  }  } |

# A Very Big Sum

|  |
| --- |
| public class Solution {  public static void main(String[] args) {  Scanner stdin = new Scanner(System.in);  int numbers = Integer.parseInt(stdin.nextLine());  String[] integers = stdin.nextLine().split(" ");  long sum = 0;  for(int i = 0; i < numbers; i++) {  sum += Long.parseLong(integers[i]);  }  System.out.println(sum);  stdin.close();  }  } |

# Diagonal Difference

|  |
| --- |
| public class Solution {  public static void main(String[] args) {  Scanner stdin = new Scanner(System.in);  int n = Integer.parseInt(stdin.nextLine());  long sumLeftRight = 0;  long sumRightLeft = 0;  for(int i = 0; i < n; i++){  String[] elements = stdin.nextLine().split(" ");  sumLeftRight += Long.parseLong(elements[i]);  sumRightLeft += Long.parseLong(elements[n - i - 1]);  }  System.out.println(Math.abs(sumLeftRight - sumRightLeft));  stdin.close();  }  } |

# Plus Minus

|  |
| --- |
| public class Solution {  public static void main(String[] args) {  Scanner stdin = new Scanner(System.in);  int n = Integer.parseInt(stdin.nextLine());  String[] numbers = stdin.nextLine().split(" ");  int positives = 0;  int negatives = 0;  int zeroes = 0;  for(int i = 0; i < n; i++){  int num = Integer.parseInt(numbers[i]);  if(num > 0){  positives++;  }else if(num < 0){  negatives++;  }else{  zeroes++;  }  }  System.out.println(String.format(Locale.ENGLISH, "%.3f", (double) positives / n));  System.out.println(String.format(Locale.ENGLISH, "%.3f", (double) negatives / n));  System.out.println(String.format(Locale.ENGLISH, "%.3f", (double) zeroes / n));  stdin.close();  }  } |

# Staircase

|  |
| --- |
| public class Solution {  public static void main(String[] args) {  Scanner stdin = new Scanner(System.in);  int height = Integer.parseInt(stdin.nextLine());  for(int i = 0; i < height; i++){  for(int j = 0; j < height; j++){  if(height - i - 2 < j){  System.out.print("#");  }else{  System.out.print(" ");  }  }  System.out.println();  }  stdin.close();  }  } |

# Mini-Max Sum

|  |
| --- |
| public class MiniMaxSum {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int[] numbers = new int[5];  for (int i = 0; i < numbers.length; i++) {  numbers[i] = sc.nextInt();  }  long sum = Arrays.stream(numbers).mapToLong(x -> x).sum();  int min = Arrays.stream(numbers).min().getAsInt();  int max = Arrays.stream(numbers).max().getAsInt();  System.out.println(String.format("%d %d", sum - max, sum - min));  sc.close();  }  } |

# Birthday Cake Candles

|  |
| --- |
| public class BirthdayCakeCandles {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] heights = new int[n];  for (int i = 0; i < n; i++) {  heights[i] = sc.nextInt();  }  System.out.println(solve(heights));  sc.close();  }  static int solve(int[] heights) {  int maxHeight = Arrays.stream(heights).max().getAsInt();  return (int) Arrays.stream(heights).filter(height -> height == maxHeight).count();  }  } |

# Time Conversion

|  |
| --- |
| public class TimeConversion {  public static void main(String[] args) throws IOException {  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  //Get time  String time = br.readLine();  //Get hour  byte hour = Byte.parseByte(time.substring(0, 2));  //If noon or midnight  if(hour == 12){  hour = 0;  }  //Add 12 hours to afternoon (PM)  if("P".equals(time.substring(8, 9))){  hour += 12;  }  //Print time in military format  System.out.print(String.format("%02d", hour) + time.substring(2, 8));  }  } |

# Grading Students

|  |
| --- |
| public class GradingStudents {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  for (int i = 0; i < n; i++) {  int grade = sc.nextInt();  System.out.println(grade < 38 ? grade : round(grade));  }  sc.close();  }  static int round(int grade) {  int nextFive = (grade / 5 + 1) \* 5;  int distanceToNextFive = nextFive - grade;  return (distanceToNextFive < 3) ? nextFive : grade;  }  } |

# Apple and Orange

|  |
| --- |
| public class AppleandOrange {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int s = sc.nextInt();  int t = sc.nextInt();  int a = sc.nextInt();  int b = sc.nextInt();  int m = sc.nextInt();  int n = sc.nextInt();  int[] apples = readArray(sc, m);  int[] oranges = readArray(sc, n);  int[] fallOns = solve(s, t, a, b, apples, oranges);  System.out.println(fallOns[0]);  System.out.println(fallOns[1]);  sc.close();  }  static int[] readArray(Scanner sc, int size) {  int[] result = new int[size];  for (int i = 0; i < result.length; i++) {  result[i] = sc.nextInt();  }  return result;  }  static int[] solve(int s, int t, int a, int b, int[] apples, int[] oranges) {  return new int[] { countFallOn(Arrays.stream(apples).map(apple -> a + apple), s, t),  countFallOn(Arrays.stream(oranges).map(orange -> b + orange), s, t) };  }  static int countFallOn(IntStream locations, int s, int t) {  return (int) locations.filter(location -> location >= s && location <= t).count();  }  } |

# Kangaroo

|  |
| --- |
| public class Kangaroo {  public static void main(String[] arsg) {  Scanner sc = new Scanner(System.in);  int x1 = sc.nextInt();  int v1 = sc.nextInt();  int x2 = sc.nextInt();  int v2 = sc.nextInt();  System.out.println ((v1 > v2 && (x2 - x1) % (v1 - v2) == 0) ? "YES" : "NO");  sc.close();  }  } |

# Between Two Sets

|  |
| --- |
| public class BetweenTwoSets {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int m = sc.nextInt();  int[] A = readArray(sc, n);  int[] B = readArray(sc, m);  System.out.println(solve(A, B));  sc.close();  }  static int[] readArray(Scanner sc, int size) {  int[] result = new int[size];  for (int i = 0; i < result.length; i++) {  result[i] = sc.nextInt();  }  return result;  }  static int solve(int[] A, int[] B) {  int g = gcd(B);  if (!Arrays.stream(A).allMatch(a -> g % a == 0)) {  return 0;  }  return computeDivisorNum(g / lcm(A));  }  static int gcd(int[] x) {  int g = x[0];  for (int i = 1; i < x.length; i++) {  g = gcd(g, x[i]);  }  return g;  }  static int gcd(int x, int y) {  return (y == 0) ? x : gcd(y, x % y);  }  static int lcm(int[] x) {  int l = x[0];  for (int i = 1; i < x.length; i++) {  l = lcm(l, x[i]);  }  return l;  }  static int lcm(int x, int y) {  return x \* y / gcd(x, y);  }  static int computeDivisorNum(int x) {  int divisorNum = 1;  int prime = 2;  while (x != 1) {  while (!isPrime(prime)) {  prime++;  }  int exponent = 0;  while (x % prime == 0) {  x /= prime;  exponent++;  }  divisorNum \*= exponent + 1;  prime++;  }  return divisorNum;  }  static boolean isPrime(int x) {  for (int i = 2; i \* i <= x; i++) {  if (x % i == 0) {  return false;  }  }  return true;  }  } |

# Breaking the Records

|  |
| --- |
| public class BreakingRecords {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] s = new int[n];  for (int i = 0; i < s.length; i++) {  s[i] = sc.nextInt();  }  Result result = solve(s);  System.out.println(result.maxBreakNum + " " + result.minBreakNum);  sc.close();  }  static Result solve(int[] s) {  int max = s[0];  int min = s[0];  int maxBreakNum = 0;  int minBreakNum = 0;  for (int i = 1; i < s.length; i++) {  if (s[i] > max) {  max = s[i];  maxBreakNum++;  } else if (s[i] < min) {  min = s[i];  minBreakNum++;  }  }  return new Result(maxBreakNum, minBreakNum);  }  }  class Result {  int maxBreakNum;  int minBreakNum;  Result(int maxBreakNum, int minBreakNum) {  this.maxBreakNum = maxBreakNum;  this.minBreakNum = minBreakNum;  }  } |

# Birthday Chocolate

|  |
| --- |
| public class BirthdayChocolate {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] s = new int[n];  for (int i = 0; i < s.length; i++) {  s[i] = sc.nextInt();  }  int d = sc.nextInt();  int m = sc.nextInt();  System.out.println(solve(s, d, m));  sc.close();  }  static int solve(int[] s, int d, int m) {  if (m > s.length) {  return 0;  }  int sum = IntStream.range(0, m).map(i -> s[i]).sum();  int result = 0;  for (int i = 0; i + m <= s.length; i++) {  if (sum == d) {  result++;  }  if (i + m < s.length) {  sum += s[i + m] - s[i];  }  }  return result;  }  } |

# Divisible Sum Pairs

|  |
| --- |
| public class DivisibleSumPairs {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int k = sc.nextInt();  int[] a = new int[n];  for (int i = 0; i < a.length; i++) {  a[i] = sc.nextInt();  }  int pairNum = 0;  for (int i = 0; i < a.length; i++) {  for (int j = i + 1; j < a.length; j++) {  if ((a[i] + a[j]) % k == 0) {  pairNum++;  }  }  }  System.out.println(pairNum);  sc.close();  }  } |

# Migratory Birds

|  |
| --- |
| public class MigratoryBirds {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] types = new int[n];  for (int i = 0; i < types.length; i++) {  types[i] = sc.nextInt();  }  System.out.println(solve(types));  sc.close();  }  static int solve(int[] types) {  int[] counts = new int[6];  for (int type : types) {  counts[type]++;  }  int maxCount = IntStream.range(1, counts.length).map(i -> counts[i]).max().getAsInt();  for (int i = 1;; i++) {  if (counts[i] == maxCount) {  return i;  }  }  }  } |

# Day of the Programmer

|  |
| --- |
| public class DayofProgrammer {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int y = sc.nextInt();  System.out.println(solve(y));  sc.close();  }  static String solve(int year) {  int day = 256;  int month = 1;  while (true) {  int daysInMonth = computeDaysInMonth(year, month);  if (day <= daysInMonth) {  break;  }  day -= daysInMonth;  month++;  }  return String.format("%02d.%02d.%d", day, month, year);  }  static int computeDaysInMonth(int year, int month) {  if (month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month == 10 || month == 12) {  return 31;  } else if (month == 4 || month == 6 || month == 9 || month == 11) {  return 30;  } else if (year == 1918) {  return 15;  } else if (isLeapYear(year)) {  return 29;  } else {  return 28;  }  }  static boolean isLeapYear(int year) {  return (year <= 1917 && year % 4 == 0)  || (year >= 1919 && (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)));  }  } |

# Bon Appétit

|  |
| --- |
| public class BonAppetit {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int k = sc.nextInt();  int[] c = new int[n];  for (int i = 0; i < c.length; i++) {  c[i] = sc.nextInt();  }  int b = sc.nextInt();  int refund = solve(c, k, b);  System.out.println(refund == 0 ? "Bon Appetit" : refund);  sc.close();  }  static int solve(int[] c, int k, int b) {  return b - IntStream.range(0, c.length).filter(i -> i != k).map(i -> c[i]).sum() / 2;  }  } |

# Sock Merchant

|  |
| --- |
| public class SockMerchant {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] c = new int[n];  for (int i = 0; i < c.length; i++) {  c[i] = sc.nextInt();  }  System.out.println(solve(c));  sc.close();  }  static int solve(int[] c) {  Map<Integer, Integer> color2count = new HashMap<Integer, Integer>();  for (int color : c) {  if (!color2count.containsKey(color)) {  color2count.put(color, 0);  }  color2count.put(color, color2count.get(color) + 1);  }  return color2count.values().stream().mapToInt(count -> count / 2).sum();  }  } |

# Drawing Book

|  |
| --- |
| public class DrawingBook {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int p = sc.nextInt();  System.out.println(solve(n, p));  sc.close();  }  static int solve(int n, int p) {  return Math.min(p / 2, (n / 2 \* 2 + 1 - p) / 2);  }  } |

# Counting Valleys

|  |
| --- |
| public class CountingValleys {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  sc.nextInt();  String steps = sc.next();  System.out.println(solve(steps));  sc.close();  }  static int solve(String steps) {  int valleyNum = 0;  int level = 0;  for (char step : steps.toCharArray()) {  if (level == -1 && step == 'U') {  valleyNum++;  }  if (step == 'U') {  level++;  } else {  level--;  }  }  return valleyNum;  }  } |

# Electronics Shop

|  |
| --- |
| public class ElectronicsShop {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int s = sc.nextInt();  int n = sc.nextInt();  int m = sc.nextInt();  int[] keyboards = readArray(sc, n);  int[] drives = readArray(sc, m);  System.out.println(solve(s, keyboards, drives));  sc.close();  }  static int[] readArray(Scanner sc, int size) {  int[] result = new int[size];  for (int i = 0; i < result.length; i++) {  result[i] = sc.nextInt();  }  return result;  }  static int solve(int s, int[] keyboards, int[] drives) {  int result = -1;  Arrays.sort(drives);  for (int keyboard : keyboards) {  int index = Arrays.binarySearch(drives, s - keyboard);  if (index < 0) {  index = -1 - index - 1;  }  if (index >= 0) {  result = Math.max(result, keyboard + drives[index]);  }  }  return result;  }  } |

# Cats and a Mouse

|  |
| --- |
| public class CatsandMouse {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int q = sc.nextInt();  for (int tc = 0; tc < q; tc++) {  int x = sc.nextInt();  int y = sc.nextInt();  int z = sc.nextInt();  System.out.println(solve(x, y, z));  }  sc.close();  }  static String solve(int x, int y, int z) {  int distanceA = Math.abs(x - z);  int distanceB = Math.abs(y - z);  if (distanceA < distanceB) {  return "Cat A";  } else if (distanceA > distanceB) {  return "Cat B";  } else {  return "Mouse C";  }  }  } |

# Forming a Magic Square

|  |
| --- |
| public class FormingMagicSquare {  public static void main(String[] args) {  Scanner input = new Scanner(System.in);  int[][][] possiblePermutations = {  {{8, 1, 6}, {3, 5, 7}, {4, 9, 2}},// 1  {{6, 1, 8}, {7, 5, 3}, {2, 9, 4}},// 2  {{4, 9, 2}, {3, 5, 7}, {8, 1, 6}},// 3  {{2, 9, 4}, {7, 5, 3}, {6, 1, 8}},// 4  {{8, 3, 4}, {1, 5, 9}, {6, 7, 2}},// 5  {{4, 3, 8}, {9, 5, 1}, {2, 7, 6}},// 6  {{6, 7, 2}, {1, 5, 9}, {8, 3, 4}},// 7  {{2, 7, 6}, {9, 5, 1}, {4, 3, 8}},// 8  };  int[][] given = new int[3][3];  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++)  given[i][j] = input.nextInt();  }  int minCost = Integer.MAX\_VALUE;  for (int permutation = 0; permutation < 8; permutation++) {  int permutationCost = 0;  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++)  permutationCost += Math.abs(given[i][j] - possiblePermutations[permutation][i][j]);  }  minCost = Math.min(minCost, permutationCost);  }  System.out.println(minCost);  }  } |

# Picking Numbers

|  |
| --- |
| public class PickingNumbers {  static final int LIMIT = 100;  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] a = new int[n];  for (int i = 0; i < a.length; i++) {  a[i] = sc.nextInt();  }  System.out.println(solve(a));  sc.close();  }  static int solve(int[] a) {  int[] counts = new int[LIMIT];  for (int number : a) {  counts[number]++;  }  int result = Arrays.stream(counts).max().getAsInt();  for (int i = 0; i + 1 < counts.length; i++) {  result = Math.max(result, counts[i] + counts[i + 1]);  }  return result;  }  } |

# Climbing the Leaderboard

|  |
| --- |
| public class ClimbingLeaderboard {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] scores = new int[n];  for (int i = 0; i < scores.length; i++) {  scores[i] = sc.nextInt();  }  int[] ranks = buildRanks(scores);  int m = sc.nextInt();  for (int i = 0; i < m; i++) {  int alice = sc.nextInt();  System.out.println(solve(scores, ranks, alice));  }  sc.close();  }  static int[] buildRanks(int[] scores) {  int[] ranks = new int[scores.length];  int rank = 1;  for (int i = 0; i < ranks.length; i++) {  if (i > 0 && scores[i] != scores[i - 1]) {  rank++;  }  ranks[i] = rank;  }  return ranks;  }  static int solve(int[] scores, int[] ranks, int alice) {  int lower = 0;  int upper = scores.length - 1;  int aliceRank = 1;  while (lower <= upper) {  int middle = (lower + upper) / 2;  if (alice == scores[middle]) {  aliceRank = ranks[middle];  break;  } else if (alice < scores[middle]) {  aliceRank = ranks[middle] + 1;  lower = middle + 1;  } else {  upper = middle - 1;  }  }  return aliceRank;  }  } |

# The Hurdle Race

|  |
| --- |
| public class HurdleRace {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int k = sc.nextInt();  int[] heights = new int[n];  for (int i = 0; i < heights.length; i++) {  heights[i] = sc.nextInt();  }  System.out.println(solve(k, heights));  sc.close();  }  static int solve(int k, int[] heights) {  return Math.max(0, Arrays.stream(heights).max().getAsInt() - k);  }  } |

# Designer PDF Viewer

|  |
| --- |
| public class DesignerPDFViewer {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int[] h = new int[26];  for (int i = 0; i < h.length; i++) {  h[i] = sc.nextInt();  }  String word = sc.next();  System.out.println(solve(h, word));  sc.close();  }  static int solve(int[] h, String word) {  return word.length() \* word.chars().map(c -> h[c - 'a']).max().getAsInt();  }  } |

# Utopian Tree

|  |
| --- |
| public class UtopianTree {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int T = in.nextInt();  for (int tc = 0; tc < T; tc++) {  int N = in.nextInt();  int height = 1;  for (int i = 0; i < N; i++) {  if (i % 2 == 0) {  height \*= 2;  } else {  height++;  }  }  System.out.println(height);  }  in.close();  }  } |

# Angry Professor

|  |
| --- |
| public class AngryProfessor {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int T = in.nextInt();  for (int tc = 0; tc < T; tc++) {  int N = in.nextInt();  int K = in.nextInt();  int presentCount = 0;  for (int i = 0; i < N; i++) {  int time = in.nextInt();  if (time <= 0) {  presentCount++;  }  }  System.out.println(presentCount < K ? "YES" : "NO");  }  in.close();  }  } |

# Beautiful Days at the Movies

|  |
| --- |
| public class BeautifulDaysMovies {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int i = sc.nextInt();  int j = sc.nextInt();  int k = sc.nextInt();  System.out.println(solve(i, j, k));  sc.close();  }  static int solve(int i, int j, int k) {  return (int) IntStream.rangeClosed(i, j).filter(x -> Math.abs(x - reverse(x)) % k == 0).count();  }  static int reverse(int x) {  return Integer.parseInt(new StringBuilder(String.valueOf(x)).reverse().toString());  }  } |

# Viral Advertising

|  |
| --- |
| public class ViralAdvertising {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  System.out.println(solve(n));  sc.close();  }  static int solve(int n) {  int receive = 5;  int likeTotal = 0;  for (int i = 0; i < n; i++) {  int like = receive / 2;  likeTotal += like;  receive = like \* 3;  }  return likeTotal;  }  } |

# Save the Prisoner!

|  |
| --- |
| public class SavePrisoner {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int T = sc.nextInt();  for (int tc = 0; tc < T; tc++) {  int N = sc.nextInt();  int M = sc.nextInt();  int S = sc.nextInt();  System.out.println((S + M - 2) % N + 1);  }  sc.close();  }  } |

# Circular Array Rotation

|  |
| --- |
| public class CircularArrayRotation {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int k = sc.nextInt();  int q = sc.nextInt();  int[] a = new int[n];  for (int i = 0; i < a.length; i++) {  a[i] = sc.nextInt();  }  for (int i = 0; i < q; i++) {  int m = sc.nextInt();  System.out.println(a[((m - k) % n + n) % n]);  }  sc.close();  }  } |

# Sequence Equation

|  |
| --- |
| public class SequenceEquation {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] p = new int[n];  for (int i = 0; i < p.length; i++) {  p[i] = sc.nextInt() - 1;  }  Arrays.stream(solve(p)).forEach(System.out::println);  sc.close();  }  static int[] solve(int[] p) {  int[] result = new int[p.length];  for (int i = 0; i < result.length; i++) {  for (int j = 0;; j++) {  if (p[p[j]] == i) {  result[i] = j + 1;  break;  }  }  }  return result;  }  } |

# Jumping on the Clouds: Revisited

|  |
| --- |
| public class JumpingonCloudsRevisited {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int k = sc.nextInt();  int[] c = new int[n];  for (int i = 0; i < c.length; i++) {  c[i] = sc.nextInt();  }  System.out.println(solve(c, k));  sc.close();  }  static int solve(int[] c, int k) {  int E = 100;  int index = 0;  do {  index = (index + k) % c.length;  E--;  if (c[index] == 1) {  E -= 2;  }  } while (index != 0);  return E;  }  } |

# Find Digits

|  |
| --- |
| public class FindDigits {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int T = in.nextInt();  for (int tc = 0; tc < T; tc++) {  long N = in.nextLong();  System.out.println((N + "").chars()  .filter(digit -> digit != '0' && N % (digit - '0') == 0)  .count());  }  in.close();  }  } |

# Extra Long Factorials

|  |
| --- |
| public class ExtraLongFactorials {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int N = in.nextInt();  BigInteger factorial = BigInteger.ONE;  for (int i = 1; i <= N; i++) {  factorial = factorial.multiply(new BigInteger(i + ""));  }  System.out.println(factorial);  in.close();  }  } |

# Append and Delete

|  |
| --- |
| public class AppendandDelete {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  String s = sc.next();  String t = sc.next();  int k = sc.nextInt();  System.out.println(solve(s, t, k) ? "Yes" : "No");  sc.close();  }  static boolean solve(String s, String t, int k) {  if (k >= s.length() + t.length()) {  return true;  }  for (int i = s.length();; i--) {  String prefixS = s.substring(0, i);  if (t.startsWith(prefixS)) {  int diff = (s.length() - i) + (t.length() - i);  return diff <= k && (k - diff) % 2 == 0;  }  }  }  } |

# Sherlock and Squares

|  |
| --- |
| public class SherlockandSquares {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int T = in.nextInt();  for (int tc = 0; tc < T; tc++) {  int A = in.nextInt();  int B = in.nextInt();  System.out.println(Math.max(0, sqrtToFloor(B) - sqrtToCeil(A) + 1));  }  in.close();  }  static int sqrtToNearest(int number) {  return (int) Math.round(Math.sqrt(number));  }  static int sqrtToFloor(int number) {  int result = sqrtToNearest(number);  if (result \* result > number) {  result--;  }  return result;  }  static int sqrtToCeil(int number) {  int result = sqrtToNearest(number);  if (result \* result < number) {  result++;  }  return result;  }  } |

# Library Fine

|  |
| --- |
| public class LibraryFine {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int actualD = in.nextInt();  int actualM = in.nextInt();  int actualY = in.nextInt();  int expectedD = in.nextInt();  int expectedM = in.nextInt();  int expectedY = in.nextInt();  int fine = 0;  if (actualY > expectedY) {  fine = 10000;  } else if (actualY == expectedY) {  if (actualM > expectedM) {  fine = 500 \* (actualM - expectedM);  } else if (actualM == expectedM) {  if (actualD > expectedD) {  fine = 15 \* (actualD - expectedD);  }  }  }  System.out.println(fine);  in.close();  }  } |

# Cut the sticks

|  |
| --- |
| public class CuttheSticks {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  SortedMap<Integer, Integer> length2count = new TreeMap<Integer, Integer>();  int N = in.nextInt();  for (int i = 0; i < N; i++) {  int length = in.nextInt();  if (!length2count.containsKey(length)) {  length2count.put(length, 0);  }  length2count.put(length, length2count.get(length) + 1);  }  int leftNum = N;  for (int count : length2count.values()) {  System.out.println(leftNum);  leftNum -= count;  }  in.close();  }  } |

# Non-Divisible Subset

|  |
| --- |
| public class NonDivisibleSubset {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int k = sc.nextInt();  int[] remainders = new int[k];  for (int i = 0; i < n; i++) {  int ai = sc.nextInt();  remainders[ai % k]++;  }  int result = 0;  for (int i = 0; i \* 2 <= k; i++) {  int opposite = (k - i) % k;  if (i == opposite) {  result += Math.min(remainders[i], 1);  } else {  result += Math.max(remainders[i], remainders[opposite]);  }  }  System.out.println(result);  sc.close();  }  } |

# Repeated String

|  |
| --- |
| public class RepeatedString {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  String s = sc.next();  long n = sc.nextLong();  System.out.println(solve(s, n));  sc.close();  }  static long solve(String s, long n) {  return n / s.length() \* count(s, 'a') + count(s.substring(0, (int) (n % s.length())), 'a');  }  static int count(String s, char target) {  return (int) s.chars().filter(c -> c == target).count();  }  } |

# Jumping on the Clouds

|  |
| --- |
| public class JumpingonClouds {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] c = new int[n];  for (int i = 0; i < c.length; i++) {  c[i] = sc.nextInt();  }  int[] minSteps = new int[c.length];  Arrays.fill(minSteps, 1, minSteps.length, -1);  for (int i = 0; i < minSteps.length; i++) {  if (c[i] == 1) {  continue;  }  if (i >= 1 && c[i - 1] == 0 && minSteps[i - 1] >= 0) {  minSteps[i] = Math.min((minSteps[i] < 0) ? Integer.MAX\_VALUE : minSteps[i], minSteps[i - 1] + 1);  }  if (i >= 2 && c[i - 2] == 0 && minSteps[i - 2] >= 0) {  minSteps[i] = Math.min((minSteps[i] < 0) ? Integer.MAX\_VALUE : minSteps[i], minSteps[i - 2] + 1);  }  }  System.out.println(minSteps[minSteps.length - 1]);  sc.close();  }  } |

# Equalize the Array

|  |
| --- |
| public class EqualizeArray {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] a = new int[n];  for (int i = 0; i < a.length; i++) {  a[i] = sc.nextInt();  }  System.out.println(solve(a));  sc.close();  }  static int solve(int[] a) {  Map<Integer, Integer> value2count = new HashMap<Integer, Integer>();  for (int value : a) {  if (!value2count.containsKey(value)) {  value2count.put(value, 0);  }  value2count.put(value, value2count.get(value) + 1);  }  return a.length - value2count.values().stream().mapToInt(x -> x).max().getAsInt();  }  } |

# Queen's Attack II

|  |
| --- |
| public class QueenAttackII {  static final int[] R\_OFFSETS = { -1, 0, 1, 0, -1, -1, 1, 1 };  static final int[] C\_OFFSETS = { 0, 1, 0, -1, -1, 1, -1, 1 };  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int k = sc.nextInt();  int rq = sc.nextInt();  int cq = sc.nextInt();  Set<Position> obstacles = new HashSet<Position>();  for (int i = 0; i < k; i++) {  int r = sc.nextInt();  int c = sc.nextInt();  obstacles.add(new Position(r, c));  }  System.out.println(solve(n, rq, cq, obstacles));  sc.close();  }  static int solve(int n, int rq, int cq, Set<Position> obstacles) {  return IntStream.range(0, R\_OFFSETS.length).map(i -> attack(n, rq, cq, R\_OFFSETS[i], C\_OFFSETS[i], obstacles))  .sum();  }  static int attack(int n, int rq, int cq, int rOffset, int cOffset, Set<Position> obstacles) {  int result = 0;  while (true) {  rq += rOffset;  cq += cOffset;  if (!(rq >= 1 && rq <= n && cq >= 1 && cq <= n) || obstacles.contains(new Position(rq, cq))) {  break;  }  result++;  }  return result;  }  }  class Position {  int r;  int c;  Position(int r, int c) {  this.r = r;  this.c = c;  }  @Override  public int hashCode() {  return r \* c;  }  @Override  public boolean equals(Object obj) {  Position other = (Position) obj;  return r == other.r && c == other.c;  }  } |

# ACM ICPC Team

|  |
| --- |
| public class ACMICPCTeam {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int N = in.nextInt();  in.nextInt();  String[] lines = new String[N];  for (int i = 0; i < lines.length; i++) {  lines[i] = in.next();  }  int maxTopic = 0;  int count = 0;  for (int i = 0; i < lines.length; i++) {  for (int j = i + 1; j < lines.length; j++) {  String merged = merge(lines[i], lines[j]);  int topic = (int) merged.chars().filter(x -> x == '1').count();  if (topic > maxTopic) {  maxTopic = topic;  count = 1;  } else if (topic == maxTopic) {  count++;  }  }  }  System.out.println(maxTopic);  System.out.println(count);  in.close();  }  static String merge(String line1, String line2) {  StringBuilder merged = new StringBuilder();  for (int i = 0; i < line1.length(); i++) {  if (line1.charAt(i) == '1' || line2.charAt(i) == '1') {  merged.append('1');  } else {  merged.append('0');  }  }  return merged.toString();  }  } |

# Taum and B'day

|  |
| --- |
| public class TaumandBday {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int T = in.nextInt();  for (int tc = 0; tc < T; tc++) {  int B = in.nextInt();  int W = in.nextInt();  int X = in.nextInt();  int Y = in.nextInt();  int Z = in.nextInt();  long cost;  if (X + Z < Y) {  cost = (long) X \* B + (long) (X + Z) \* W;  } else if (Y + Z < X) {  cost = (long) (Y + Z) \* B + (long) Y \* W;  } else {  cost = (long) X \* B + (long) Y \* W;  }  System.out.println(cost);  }  in.close();  }  } |

# Organizing Containers of Balls

|  |
| --- |
| public class OrganizingContainersBalls {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int q = sc.nextInt();  for (int tc = 0; tc < q; tc++) {  int n = sc.nextInt();  int[][] M = new int[n][n];  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  M[i][j] = sc.nextInt();  }  }  System.out.println(solve(M) ? "Possible" : "Impossible");  }  sc.close();  }  static boolean solve(int[][] M) {  int n = M.length;  long[] rowSums = new long[n];  for (int r = 0; r < n; r++) {  for (int c = 0; c < n; c++) {  rowSums[r] += M[r][c];  }  }  long[] colSums = new long[n];  for (int c = 0; c < n; c++) {  for (int r = 0; r < n; r++) {  colSums[c] += M[r][c];  }  }  return isSame(rowSums, colSums);  }  static boolean isSame(long[] a, long[] b) {  Arrays.sort(a);  Arrays.sort(b);  for (int i = 0; i < a.length; i++) {  if (a[i] != b[i]) {  return false;  }  }  return true;  }  } |

# Encryption

|  |
| --- |
| public class Encryption {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  String message = in.next();  int length = message.length();  int row = (int) Math.round(Math.sqrt(length));  if (row \* row > length) {  row--;  }  int col;  if (row \* row == length) {  col = row;  } else if (row \* (row + 1) >= length) {  col = row + 1;  } else {  row++;  col = row;  }  char[][] grid = new char[row][col];  int r = 0;  int c = 0;  for (int i = 0; i < length; i++) {  grid[r][c] = message.charAt(i);  c++;  if (c == col) {  r++;  c = 0;  }  }  StringBuilder encoded = new StringBuilder();  for (int i = 0; i < col; i++) {  for (int j = 0; j < row; j++) {  if (grid[j][i] != 0) {  encoded.append(grid[j][i]);  }  }  encoded.append(" ");  }  System.out.println(encoded);  in.close();  }  } |

# Bigger is Greater

|  |
| --- |
| public class BiggerisGreater {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int t = in.nextInt();  for (int tc = 0; tc < t; tc++) {  String w = in.next();  String next = findNext(w);  System.out.println((next == null) ? "no answer" : next);  }  in.close();  }  static String findNext(String word) {  int index = word.length() - 2;  while (index >= 0 && word.charAt(index) >= word.charAt(index + 1)) {  index--;  }  if (index < 0) {  return null;  }  int biggerIndex = index + 1;  while (biggerIndex + 1 < word.length() && word.charAt(biggerIndex + 1) > word.charAt(index)) {  biggerIndex++;  }  StringBuilder next = new StringBuilder(word);  swap(next, index, biggerIndex);  reverse(next, index + 1, next.length() - 1);  return next.toString();  }  static void swap(StringBuilder sb, int index1, int index2) {  char temp = sb.charAt(index1);  sb.setCharAt(index1, sb.charAt(index2));  sb.setCharAt(index2, temp);  }  static void reverse(StringBuilder sb, int beginIndex, int endIndex) {  for (int i = beginIndex, j = endIndex; i < j; i++, j--) {  swap(sb, i, j);  }  }  } |

# Modified Kaprekar Numbers

|  |
| --- |
| public class ModifiedKaprekarNumbers {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int p = in.nextInt();  int q = in.nextInt();  List<Integer> result = IntStream  .rangeClosed(p, q)  .filter(ModifiedKaprekarNumbers::isKaprekar)  .collect(ArrayList<Integer>::new, List<Integer>::add,  List<Integer>::addAll);  if (result.isEmpty()) {  System.out.println("INVALID RANGE");  } else {  System.out.println(String.join(" ", result.stream()  .map(x -> x + "").collect(Collectors.toList())));  }  in.close();  }  static boolean isKaprekar(int number) {  long square = (long) number \* number;  String squareStr = square + "";  int leftLength = squareStr.length() / 2;  return Integer.parseInt("0" + squareStr.substring(0, leftLength))  + Integer.parseInt(squareStr.substring(leftLength)) == number;  }  } |

# Beautiful Triplets

|  |
| --- |
| public class BeautifulTriplets {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int d = sc.nextInt();  int[] a = new int[n];  for (int i = 0; i < a.length; i++) {  a[i] = sc.nextInt();  }  System.out.println(solve(a, d));  sc.close();  }  static int solve(int[] a, int d) {  Set<Integer> numbers = Arrays.stream(a).collect(HashSet<Integer>::new, Set<Integer>::add, Set<Integer>::addAll);  return (int) Arrays.stream(a).filter(x -> numbers.contains(x + d) && numbers.contains(x + d \* 2)).count();  }  } |

# Minimum Distances

|  |
| --- |
| public class MinimumDistances {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] a = new int[n];  for (int i = 0; i < a.length; i++) {  a[i] = sc.nextInt();  }  System.out.println(solve(a));  sc.close();  }  static int solve(int[] a) {  int minDistance = Integer.MAX\_VALUE;  Map<Integer, Integer> value2lastIndex = new HashMap<Integer, Integer>();  for (int i = 0; i < a.length; i++) {  if (value2lastIndex.containsKey(a[i])) {  minDistance = Math.min(minDistance, i - value2lastIndex.get(a[i]));  }  value2lastIndex.put(a[i], i);  }  return (minDistance == Integer.MAX\_VALUE) ? -1 : minDistance;  }  } |

# Halloween Sale

|  |
| --- |
| public class Solution {  // Complete the howManyGames function below.  static int howManyGames(int p, int d, int m, int s) {  // Return the number of games you can buy  int counter = 0, currentPrice=p;  while(s >= currentPrice){  counter++;  s -= currentPrice;  currentPrice = (currentPrice - d) >= m ? currentPrice - d : m;  }  return counter;  }  private static final Scanner scanner = new Scanner(System.in);  public static void main(String[] args) throws IOException {  BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));  String[] pdms = scanner.nextLine().split(" ");  int p = Integer.parseInt(pdms[0]);  int d = Integer.parseInt(pdms[1]);  int m = Integer.parseInt(pdms[2]);  int s = Integer.parseInt(pdms[3]);  int answer = howManyGames(p, d, m, s);  bufferedWriter.write(String.valueOf(answer));  bufferedWriter.newLine();  bufferedWriter.close();  scanner.close();  }  } |

# The Time in Words

|  |
| --- |
| public class TimeinWords {  static final String[] DIGIT\_WORDS = { "zero", "one", "two", "three",  "four", "five", "six", "seven", "eight", "nine" };  static final String[] TEEN\_WORDS = { "ten", "eleven", "twelve", "thirteen",  "fourteen", "fifteen", "sixteen", "seventeen", "eighteen",  "nineteen" };  static final String[] TEN\_WORDS = { null, "ten", "twenty", "thirty",  "forty", "fifty", "sixty", "seventy", "eighty", "ninety" };  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int H = in.nextInt();  int M = in.nextInt();  String time;  if (M == 0) {  time = toWords(H) + " o' clock";  } else if (M == 1) {  time = "one minute past " + toWords(H);  } else if (M == 15) {  time = "quarter past " + toWords(H);  } else if (M < 30) {  time = toWords(M) + " minutes past " + toWords(H);  } else if (M == 30) {  time = "half past " + toWords(H);  } else if (M == 45) {  time = "quarter to " + toWords(H + 1);  } else if (M == 59) {  time = "one minute to " + toWords(H + 1);  } else {  time = toWords(60 - M) + " minutes to " + toWords(H + 1);  }  System.out.println(time);  in.close();  }  static String toWords(int number) {  if (number >= 10 && number < 20) {  return TEEN\_WORDS[number - 10];  } else if (number < 10) {  return DIGIT\_WORDS[number];  } else if (number % 10 == 0) {  return TEN\_WORDS[number / 10];  } else {  return TEN\_WORDS[number / 10] + " " + DIGIT\_WORDS[number % 10];  }  }  } |

# Chocolate Feast

|  |
| --- |
| public class ChocolateFeast {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int T = in.nextInt();  for (int tc = 0; tc < T; tc++) {  int N = in.nextInt();  int C = in.nextInt();  int M = in.nextInt();  int count = N / C;  int wrapperNum = count;  while (wrapperNum >= M) {  int exchangeNum = wrapperNum / M;  count += exchangeNum;  wrapperNum = exchangeNum + wrapperNum % M;  }  System.out.println(count);  }  in.close();  }  } |

# Service Lane

|  |
| --- |
| public class ServiceLane {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int N = in.nextInt();  int T = in.nextInt();  int[] width = new int[N];  for (int i = 0; i < width.length; i++) {  width[i] = in.nextInt();  }  for (int tc = 0; tc < T; tc++) {  int i = in.nextInt();  int j = in.nextInt();  System.out.println(min(width, i, j));  }  in.close();  }  static int min(int[] a, int beginIndex, int endIndex) {  int result = Integer.MAX\_VALUE;  for (int i = beginIndex; i <= endIndex; i++) {  result = Math.min(result, a[i]);  }  return result;  }  } |

# Lisa's Workbook

|  |
| --- |
| public class LisaWorkbook {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int k = sc.nextInt();  int[] t = new int[n];  for (int i = 0; i < t.length; i++) {  t[i] = sc.nextInt();  }  System.out.println(solve(t, k));  sc.close();  }  static int solve(int[] t, int k) {  int special = 0;  int page = 1;  int index = 0;  for (int problemNum : t) {  for (int i = 1; i <= problemNum; i++) {  if (i == page) {  special++;  }  index++;  if (index == k) {  page++;  index = 0;  }  }  if (index != 0) {  page++;  index = 0;  }  }  return special;  }  } |

# Flatland Space Stations

|  |
| --- |
| public class FlatlandSpaceStations {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int m = sc.nextInt();  int[] stations = new int[m];  for (int i = 0; i < stations.length; i++) {  stations[i] = sc.nextInt();  }  System.out.println(solve(n, stations));  sc.close();  }  static int solve(int n, int[] stations) {  Arrays.sort(stations);  int maxDistance = Math.max(stations[0], n - 1 - stations[stations.length - 1]);  for (int i = 0; i + 1 < stations.length; i++) {  maxDistance = Math.max(maxDistance, (stations[i + 1] - stations[i]) / 2);  }  return maxDistance;  }  } |

# Fair Rations

|  |
| --- |
| public class FairRations {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int N = sc.nextInt();  int[] B = new int[N];  for (int i = 0; i < B.length; i++) {  B[i] = sc.nextInt();  }  int result = solve(B);  System.out.println(solve(B) == -1 ? "NO" : result);  sc.close();  }  static int solve(int[] B) {  int result = 0;  boolean[] evens = new boolean[B.length];  for (int i = 0; i < evens.length; i++) {  evens[i] = B[i] % 2 == 0;  }  for (int i = 0; i + 1 < evens.length; i++) {  if (!evens[i]) {  evens[i] = !evens[i];  evens[i + 1] = !evens[i + 1];  result += 2;  }  }  return evens[evens.length - 1] ? result : -1;  }  } |

# Cavity Map

|  |
| --- |
| public class CavityMap {  static final int[] OFFSET\_R = { -1, 0, 1, 0 };  static final int[] OFFSET\_C = { 0, 1, 0, -1 };  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int n = in.nextInt();  int[][] map = new int[n][n];  for (int i = 0; i < n; i++) {  String line = in.next();  for (int j = 0; j < n; j++) {  map[i][j] = line.charAt(j) - '0';  }  }  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  System.out.print(isCavity(map, i, j) ? 'X' : (map[i][j] + ""));  }  System.out.println();  }  in.close();  }  static boolean isCavity(int[][] map, int r, int c) {  int size = map.length;  for (int i = 0; i < OFFSET\_R.length; i++) {  int adjR = r + OFFSET\_R[i];  int adjC = c + OFFSET\_C[i];  if (!(adjR >= 0 && adjR < size && adjC >= 0 && adjC < size && map[adjR][adjC] < map[r][c])) {  return false;  }  }  return true;  }  } |

# Manasa and Stones

|  |
| --- |
| public class ManasaAndStones {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int T = in.nextInt();  for (int tc = 0; tc < T; tc++) {  int n = in.nextInt();  int a = in.nextInt();  int b = in.nextInt();  Set<Integer> numbers = new HashSet<Integer>();  numbers.add(0);  for (int i = 0; i < n - 1; i++) {  numbers = union(shift(numbers, a), shift(numbers, b));  }  Integer[] numberArray = numbers.toArray(new Integer[0]);  Arrays.sort(numberArray);  for (int i = 0; i < numberArray.length; i++) {  if (i != 0) {  System.out.print(" ");  }  System.out.print(numberArray[i]);  }  System.out.println();  }  in.close();  }  static Set<Integer> shift(Set<Integer> numbers, int offset) {  return numbers.stream().map(number -> number + offset)  .collect(Collectors.toSet());  }  static Set<Integer> union(Set<Integer> a, Set<Integer> b) {  Set<Integer> result = new HashSet<Integer>();  result.addAll(a);  result.addAll(b);  return result;  }  } |

# The Grid Search

|  |
| --- |
| public class GridSearch {  public static void main(String[] args) throws Throwable {  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));  String line = in.readLine();  StringTokenizer st = new StringTokenizer(line);  int T = Integer.parseInt(st.nextToken());  for (int tc = 0; tc < T; tc++) {  line = in.readLine();  st = new StringTokenizer(line);  int R = Integer.parseInt(st.nextToken());  String[] G = new String[R];  for (int i = 0; i < G.length; i++) {  G[i] = in.readLine();  }  line = in.readLine();  st = new StringTokenizer(line);  int r = Integer.parseInt(st.nextToken());  String[] P = new String[r];  for (int i = 0; i < P.length; i++) {  P[i] = in.readLine();  }  System.out.println(hasPattern(G, P) ? "YES" : "NO");  }  in.close();  }  static boolean hasPattern(String[] grid, String[] pattern) {  int patternRowLength = pattern[0].length();  Set<String> patternSet = new HashSet<String>();  for (String onePattern : pattern) {  patternSet.add(onePattern);  }  @SuppressWarnings("unchecked")  Map<String, Set<Integer>>[] str2indicesArray = new Map[grid.length];  for (int i = 0; i < str2indicesArray.length; i++) {  str2indicesArray[i] = new HashMap<String, Set<Integer>>();  for (int j = 0; j + patternRowLength <= grid[i].length(); j++) {  String str = grid[i].substring(j, j + patternRowLength);  if (patternSet.contains(str)) {  if (!str2indicesArray[i].containsKey(str)) {  str2indicesArray[i].put(str, new HashSet<Integer>());  }  str2indicesArray[i].get(str).add(j);  }  }  }  for (int i = 0; i + pattern.length <= str2indicesArray.length; i++) {  if (!merge(str2indicesArray, pattern, i).isEmpty()) {  return true;  }  }  return false;  }  static Set<Integer> merge(Map<String, Set<Integer>>[] str2indicesArray,  String[] pattern, int beginRow) {  Set<Integer> result = null;  for (int i = beginRow; i < beginRow + pattern.length; i++) {  if (!str2indicesArray[i].containsKey(pattern[i - beginRow])) {  return Collections.emptySet();  }  Set<Integer> indices = str2indicesArray[i]  .get(pattern[i - beginRow]);  result = (result == null) ? indices : intersect(result, indices);  if (result.isEmpty()) {  break;  }  }  return result;  }  static Set<Integer> intersect(Set<Integer> a, Set<Integer> b) {  return a.stream().filter(elemA -> b.contains(elemA))  .collect(Collectors.toSet());  }  } |

# Happy Ladybugs

|  |
| --- |
| public class HappyLadybugs {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int g = sc.nextInt();  for (int tc = 0; tc < g; tc++) {  sc.nextInt();  String b = sc.next();  System.out.println(solve(b) ? "YES" : "NO");  }  sc.close();  }  static boolean solve(String b) {  boolean hasEmpty = false;  boolean hasUnhappy = false;  Map<Character, Integer> color2count = new HashMap<Character, Integer>();  for (int i = 0; i < b.length(); i++) {  char cell = b.charAt(i);  if (cell == '\_') {  hasEmpty = true;  } else {  if ((i == 0 || b.charAt(i - 1) != cell) && (i == b.length() - 1 || b.charAt(i + 1) != cell)) {  hasUnhappy = true;  }  if (!color2count.containsKey(cell)) {  color2count.put(cell, 0);  }  color2count.put(cell, color2count.get(cell) + 1);  }  }  return !hasUnhappy || (hasEmpty && color2count.values().stream().mapToInt(x -> x).min().getAsInt() >= 2);  }  } |

# Strange Counter

|  |
| --- |
| public class StrangeCounter {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  long t = sc.nextLong();  System.out.println(solve(t));  sc.close();  }  static long solve(long t) {  long limit = 3;  while (t > limit) {  t -= limit;  limit \*= 2;  }  return limit + 1 - t;  }  } |

# 3D Surface Area

|  |
| --- |
| public class Solution {  static final int[] R\_OFFSETS = { -1, 0, 1, 0 };  static final int[] C\_OFFSETS = { 0, 1, 0, -1 };  // Complete the surfaceArea function below.  static int surfaceArea(int[][] heights) {  int row = heights.length;  int col = heights[0].length;  int surface = row \* col \* 2;  for (int r = 0; r < row; r++) {  for (int c = 0; c < col; c++) {  for (int i = 0; i < R\_OFFSETS.length; i++) {  int adjR = r + R\_OFFSETS[i];  int adjC = c + C\_OFFSETS[i];  int adjHeight = (adjR >= 0 && adjR < row && adjC >= 0 && adjC < col) ? heights[adjR][adjC] : 0;  surface += Math.max(0, heights[r][c] - adjHeight);  }  }  }  return surface;  }  private static final Scanner scanner = new Scanner(System.in);  public static void main(String[] args) throws IOException {  BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));  String[] HW = scanner.nextLine().split(" ");  int H = Integer.parseInt(HW[0]);  int W = Integer.parseInt(HW[1]);  int[][] A = new int[H][W];  for (int i = 0; i < H; i++) {  String[] ARowItems = scanner.nextLine().split(" ");  scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");  for (int j = 0; j < W; j++) {  int AItem = Integer.parseInt(ARowItems[j]);  A[i][j] = AItem;  }  }  int result = surfaceArea(A);  bufferedWriter.write(String.valueOf(result));  bufferedWriter.newLine();  bufferedWriter.close();  scanner.close();  }  } |

# Absolute Permutation

|  |
| --- |
| public class AbsolutePermutation {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int T = sc.nextInt();  for (int tc = 0; tc < T; tc++) {  int N = sc.nextInt();  int K = sc.nextInt();  int[] result = solve(N, K);  System.out.println(result == null ? -1  : String.join(" ", Arrays.stream(result).mapToObj(String::valueOf).collect(Collectors.toList())));  }  sc.close();  }  static int[] solve(int N, int K) {  if (K > 0 && N % (2 \* K) != 0) {  return null;  }  int[] original = IntStream.range(1, N + 1).toArray();  boolean[] used = new boolean[original.length];  int[] result = new int[original.length];  for (int i = 0; i < original.length; i++) {  if (!used[i]) {  result[i] = original[i + K];  result[i + K] = original[i];  used[i] = true;  used[i + K] = true;  }  }  return result;  }  } |

# The Bomberman Game

|  |
| --- |
| public class BombermanGame {  static final int[] R\_OFFSETS = { -1, 0, 1, 0 };  static final int[] C\_OFFSETS = { 0, 1, 0, -1 };  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int R = sc.nextInt();  int C = sc.nextInt();  int N = sc.nextInt();  int[][] grid = new int[R][C];  for (int r = 0; r < R; r++) {  String line = sc.next();  for (int c = 0; c < C; c++) {  grid[r][c] = line.charAt(c) == 'O' ? 3 : 0;  }  }  grid = solve(grid, N);  for (int r = 0; r < R; r++) {  for (int c = 0; c < C; c++) {  System.out.print(grid[r][c] == 0 ? '.' : 'O');  }  System.out.println();  }  sc.close();  }  static int[][] solve(int[][] grid, int N) {  int R = grid.length;  int C = grid[0].length;  if (N >= 2) {  N = (N - 2) % 4 + 2;  }  boolean plant = false;  for (int i = 0; i < N; i++) {  int[][] nextGrid = new int[R][C];  for (int r = 0; r < R; r++) {  for (int c = 0; c < C; c++) {  if (plant) {  if (grid[r][c] > 0) {  nextGrid[r][c] = grid[r][c] - 1;  } else {  nextGrid[r][c] = 3;  }  } else {  if (isDetonate(grid, r, c)) {  nextGrid[r][c] = 0;  } else {  nextGrid[r][c] = Math.max(0, grid[r][c] - 1);  }  }  }  }  plant = !plant;  grid = nextGrid;  }  return grid;  }  static boolean isDetonate(int[][] grid, int r, int c) {  int R = grid.length;  int C = grid[0].length;  if (grid[r][c] == 1) {  return true;  }  for (int i = 0; i < R\_OFFSETS.length; i++) {  int adjR = r + R\_OFFSETS[i];  int adjC = c + C\_OFFSETS[i];  if (adjR >= 0 && adjR < R && adjC >= 0 && adjC < C && grid[adjR][adjC] == 1) {  return true;  }  }  return false;  }  } |

# Ema's Supercomputer

|  |
| --- |
| public class EmaSupercomputer {  static final int[] R\_OFFSETS = { -1, 0, 1, 0 };  static final int[] C\_OFFSETS = { 0, 1, 0, -1 };  static int maxProduct;  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int N = sc.nextInt();  int M = sc.nextInt();  boolean[][] cells = new boolean[N][M];  for (int i = 0; i < N; i++) {  String line = sc.next();  for (int j = 0; j < M; j++) {  cells[i][j] = line.charAt(j) == 'G';  }  }  System.out.println(solve(cells));  sc.close();  }  static int solve(boolean[][] cells) {  maxProduct = 0;  search(cells, new int[2], 0);  return maxProduct;  }  static void search(boolean[][] cells, int[] sizes, int index) {  if (index == sizes.length) {  maxProduct = Math.max(maxProduct, (4 \* sizes[0] + 1) \* (4 \* sizes[1] + 1));  return;  }  int row = cells.length;  int col = cells[0].length;  for (int size = (index == 0) ? Math.min((cells.length - 1) / 2, (cells[0].length - 1) / 2)  : sizes[index - 1]; size >= 0; size--) {  for (int r = 0; r < row; r++) {  for (int c = 0; c < col; c++) {  if (canFill(cells, r, c, size)) {  fill(cells, r, c, size, false);  sizes[index] = size;  search(cells, sizes, index + 1);  fill(cells, r, c, size, true);  }  }  }  }  }  static void fill(boolean[][] cells, int r, int c, int size, boolean value) {  cells[r][c] = value;  for (int i = 0; i < R\_OFFSETS.length; i++) {  for (int j = 1; j <= size; j++) {  cells[r + R\_OFFSETS[i] \* j][c + C\_OFFSETS[i] \* j] = value;  }  }  }  static boolean canFill(boolean[][] cells, int r, int c, int size) {  if (!cells[r][c]) {  return false;  }  int row = cells.length;  int col = cells[0].length;  for (int i = 0; i < R\_OFFSETS.length; i++) {  int endR = r + R\_OFFSETS[i] \* size;  int endC = c + C\_OFFSETS[i] \* size;  if (!(endR >= 0 && endR < row && endC >= 0 && endC < col)) {  return false;  }  for (int j = 1; j <= size; j++) {  if (!cells[r + R\_OFFSETS[i] \* j][c + C\_OFFSETS[i] \* j]) {  return false;  }  }  }  return true;  }  } |

# Larry's Array

|  |
| --- |
| public class LarryArray {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int T = sc.nextInt();  for (int tc = 0; tc < T; tc++) {  int N = sc.nextInt();  int[] A = new int[N];  for (int i = 0; i < A.length; i++) {  A[i] = sc.nextInt();  }  System.out.println(solve(A) ? "YES" : "NO");  }  sc.close();  }  static boolean solve(int[] A) {  for (int i = 0; i < A.length; i++) {  int index = find(A, i + 1);  while (index >= i + 2) {  A[index] = A[index - 1];  A[index - 1] = A[index - 2];  A[index - 2] = i + 1;  index -= 2;  }  if (index == i + 1) {  if (index == A.length - 1) {  return false;  }  A[index] = A[index + 1];  A[index + 1] = A[index - 1];  A[index - 1] = i + 1;  }  }  return true;  }  static int find(int[] a, int target) {  for (int i = 0;; i++) {  if (a[i] == target) {  return i;  }  }  }  } |

# Almost Sorted

|  |
| --- |
| public class AlmostSorted {  public static void main(String[] args) throws IOException {  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  //Get input  final int N = Integer.parseInt(br.readLine());  final int[] arr = new int[N];  String[] line = br.readLine().split(" ");  for(int i = 0; i < N; ++i){  arr[i] = Integer.parseInt(line[i]);  }  //Print output  System.out.print(solve(arr, N));  }  private static String solve(final int[] A, final int N){  int l = 0;  int r = N - 1;  //Check for out of place index from the left  while(l < r && A[l] <= A[l+1]){  ++l;  }  //Check if array already sorted  if(l == r){  return "yes";  }  //Check for out of place index from the right  while(r > l && A[r] >= A[r-1]){  --r;  }  //Check if swapping or reversing would NOT sort the array  if((l > 0 && A[r] < A[l-1]) || (r < N-1 && A[l] > A[r+1])){  return "no";  }  //Check if we're dealing with a reversal  int m;  for(m = l+1; m < r && A[m] >= A[m+1]; ++m){}  if(m == r){  return "yes\n" + ((r-l < 2) ? "swap " : "reverse ") + (l+1) + " " + (r+1);  }  //Check if we're NOT dealing with a swap  if(m-l > 1 || A[l] < A[r-1] || A[r] > A[l+1]){  return "no";  }  //Check if we're dealing with a swap  for(int k = r-1; m < k && A[m] <= A[m+1]; ++m){}  return (r-m > 1) ? "no" : "yes\nswap " + (l+1) + " " + (r+1);  }  } |

# Matrix Layer Rotation

|  |
| --- |
| public class MatrixLayerRotation {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int m = in.nextInt();  int n = in.nextInt();  int r = in.nextInt();  int[][] a = new int[m][n];  for(int i = 0; i < m; ++ i){  for(int j = 0; j < n;j++){  a[i][j] = in.nextInt();  }  }  int layers = Math.min(m, n) / 2;  for(int layer = 0; layer < layers; ++ layer){  for(int x = 0; x < r % (2 \* (m + n - 2 - 4 \* layer)); x++){  int i = layer, j = layer;  int temp = a[layer][layer];  while(i < m - 1 - layer){  int temp2 = a[i + 1][j];  a[i + 1][j] = temp;  temp = temp2;  i++;  }  while(j < n - 1 - layer){  int temp2 = a[i][j + 1];  a[i][j + 1] = temp;  temp = temp2;  j++;  }  while(i > layer){  int temp2 = a[i-1][j];  a[i-1][j] = temp;  temp = temp2;  i--;  }  while(j > layer){  int temp2 = a[i][j-1];  a[i][j-1] = temp;  temp = temp2;  j--;  }  }  }  for(int x = 0; x < a.length; x++ ){  for(int y = 0; y < a[x].length; y++ ){  System.out.print(a[x][y] + " ");  }  System.out.println();  }  }  } |

# Big Sorting

|  |
| --- |
| public class BigSorting {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  String[] numbers = new String[n];  for (int i = 0; i < numbers.length; i++) {  numbers[i] = sc.next();  }  Arrays.sort(numbers, (n1, n2) -> {  if (n1.length() != n2.length()) {  return Integer.compare(n1.length(), n2.length());  } else {  for (int i = 0; i < n1.length(); i++) {  int digit1 = n1.charAt(i) - '0';  int digit2 = n2.charAt(i) - '0';  if (digit1 != digit2) {  return Integer.compare(digit1, digit2);  }  }  return 0;  }  });  Arrays.stream(numbers).forEach(System.out::println);  sc.close();  }  } |

# Super Reduced String

|  |
| --- |
| public class Super\_Reduced\_String {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  String s = in.nextLine();  String acc = "";  for(int i =0 ; i<s.length(); i++){  char c = s.charAt(i);  if(acc.length()>0 && acc.charAt(acc.length()-1)==c){  acc = acc.substring(0, acc.length()-1);  }else{  acc += c;  }  }  System.out.println(acc.isEmpty() ? "Empty String" :acc);  }  } |

# Intro to Tutorial Challenges

|  |
| --- |
| public class IntroTutorialChallenges {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int V = in.nextInt();  int n = in.nextInt();  int[] numbers = new int[n];  for (int i = 0; i < numbers.length; i++) {  numbers[i] = in.nextInt();  }  for (int i = 0;; i++) {  if (numbers[i] == V) {  System.out.println(i);  break;  }  }  in.close();  }  } |

# CamelCase

|  |
| --- |
| public class CamelCase {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  String s = in.next();  String words [] = s.split("[A-Z]");  System.out.println(words.length);  }  } |

# Insertion Sort - Part 1

|  |
| --- |
| public class InsertionSortPart1 {  // Fill up this function  public static void insertIntoSorted(int[] ar) {  int V = ar[ar.length - 1];  int index = ar.length - 2;  while (index >= 0 && ar[index] > V) {  ar[index + 1] = ar[index];  printArray(ar);  index--;  }  ar[index + 1] = V;  printArray(ar);  }  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int s = in.nextInt();  int[] ar = new int[s];  for (int i = 0; i < s; i++) {  ar[i] = in.nextInt();  }  insertIntoSorted(ar);  in.close();  }  private static void printArray(int[] ar) {  for (int n : ar) {  System.out.print(n + " ");  }  System.out.println();  }  } |

# Strong Password

|  |
| --- |
| public class StrongPassword {  static int minimumNumber(int n, String password) {  int digitCount = 0;  int lowerCount = 0;  int upperCount = 0;  int specialCount = 0;  int addCount = 0;  for(char c : password.toCharArray()){  if(Character.isDigit(c)){  digitCount++;  }else if(Character.isLowerCase(c)){  lowerCount++;  }else if(Character.isUpperCase(c)){  upperCount++;  }else{  specialCount++;  }  }  if(digitCount == 0){  addCount++;  digitCount++;  }  if(lowerCount == 0){  addCount++;  lowerCount++;  }  if(upperCount == 0){  addCount++;  upperCount++;  }  if(specialCount == 0){  addCount++;  specialCount++;  }  int total = digitCount + lowerCount + upperCount + specialCount;  if(total - 6 < 0){  addCount += 6 - total;  }  return addCount;  }  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int n = in.nextInt();  String password = in.next();  int answer = minimumNumber(n, password);  System.out.println(answer);  in.close();  }  } |

# Two Characters

|  |
| --- |
| public class TwoCharaters {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int len = in.nextInt();  String s = in.next();  Set<Character> dist = new HashSet<>();  for(int i =0 ; i<len ; i++){  dist.add(s.charAt(i));  }  List<Character> distList = new ArrayList<>(dist);  int max =0;  for(int i =0; i<dist.size()-1; i++){  for(int j=i+1; j<dist.size();j++){  char c1 = distList.get(i);  char c2 = distList.get(j);  String subset = s.replaceAll("[^"+c1+""+c2+"]","");  if(isAlternating(subset)){  int l = subset.length();  max = l >max ? l :max;  }  }  }  System.out.println(max);  }  private static boolean isAlternating(String s){  for(int i =0; i<s.length()-1; i++){  if(s.charAt(i) == s.charAt(i+1)){  return false;  }  }  return true;  }  } |

# Insertion Sort - Part 2

|  |
| --- |
| public class Solution {  public static void insertionSortPart2(int[] ar) {  // Fill up the code for the required logic here  // Manipulate the array as required  // The code for Input/Output is already provided  for (int i = 1; i < ar.length; i++) {  int number = ar[i];  int index = i - 1;  while (index >= 0 && ar[index] > number) {  ar[index + 1] = ar[index];  index--;  }  ar[index + 1] = number;  printArray(ar);  }  }  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int s = in.nextInt();  int[] ar = new int[s];  for (int i = 0; i < s; i++) {  ar[i] = in.nextInt();  }  insertionSortPart2(ar);  in.close();  }  private static void printArray(int[] ar) {  for (int n : ar) {  System.out.print(n + " ");  }  System.out.println("");  }  } |

# Correctness and the Loop Invariant

|  |
| --- |
| public class CorrectnessandLoopInvariant {  public static void insertionSort(int[] A) {  for (int i = 1; i < A.length; i++) {  int value = A[i];  int j = i - 1;  while (j >= 0 && A[j] > value) {  A[j + 1] = A[j];  j = j - 1;  }  A[j + 1] = value;  }  printArray(A);  }  static void printArray(int[] ar) {  for (int n : ar) {  System.out.print(n + " ");  }  }  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int n = in.nextInt();  int[] ar = new int[n];  for (int i = 0; i < n; i++) {  ar[i] = in.nextInt();  }  insertionSort(ar);  in.close();  }  } |

# Caesar Cipher

|  |
| --- |
| public class CaesarCipher {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int n = in.nextInt();  String s = in.next();  int k = in.nextInt() % 26;  String str = "";  for (int i = 0; i < n; i++) {  str = str + getUpdatedAsciiChar(s.charAt(i), k);  }  System.out.println(str);  }  static char getUpdatedAsciiChar(char ch, int incremental) {  if (Character.isLetter(ch)) {  int i = (int) ch;  i = i + incremental;  if (Character.isUpperCase(ch)) {  if (i > 90) {  int num = (int) ch + incremental;  i = num - 26;  }  } else if (i > 122) {  int num = (int) ch + incremental;  i = num - 26;  }  return (char) i;  }  return ch;  }  } |

# Mars Exploration

|  |
| --- |
| public class MarsExploration {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  String msg = in.nextLine();  int result = 0;  for(int i =0; i<msg.length(); i+=3){  if(msg.charAt(i)!='S'){  result++;  }  if(msg.charAt(i+1)!= 'O'){  result++;  }  if(msg.charAt(i+2)!='S'){  result++;  }  }  System.out.println(result);  in.close();  }  } |

# Running Time of Algorithms

|  |
| --- |
| public class RunningTimeAlgorithms {  public static void insertionSort(int[] A) {  int shiftNum = 0;  for (int i = 1; i < A.length; i++) {  int value = A[i];  int j = i - 1;  while (j >= 0 && A[j] > value) {  A[j + 1] = A[j];  j = j - 1;  shiftNum++;  }  A[j + 1] = value;  }  System.out.println(shiftNum);  }  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int n = in.nextInt();  int[] ar = new int[n];  for (int i = 0; i < n; i++) {  ar[i] = in.nextInt();  }  insertionSort(ar);  in.close();  }  } |

# HackerRank in a String!

|  |
| --- |
| public class hackerrankInString {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int q = in.nextInt();  queries:  for(int a0 = 0; a0 < q; a0++){  String s = in.next();  char[] find = "hackerrank".toCharArray();  int findIndex = 0;  for(char c : s.toCharArray()){  if(find[findIndex] == c)  findIndex++;  if(findIndex == find.length){ //We ran out of characters to find  System.out.println("YES");  continue queries;  }  }  System.out.println("NO"); //We didn't find all characters  }  }  } |

# Quicksort 1 - Partition

|  |
| --- |
| public class Quicksort1Partition {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int n = in.nextInt();  int[] ar = new int[n];  for (int i = 0; i < ar.length; i++) {  ar[i] = in.nextInt();  }  partition(ar);  in.close();  }  static void partition(int[] ar) {  List<Integer> left = new ArrayList<Integer>();  List<Integer> right = new ArrayList<Integer>();  for (int i = 1; i < ar.length; i++) {  if (ar[i] < ar[0]) {  left.add(ar[i]);  } else {  right.add(ar[i]);  }  }  List<Integer> partitioned = new ArrayList<Integer>();  partitioned.addAll(left);  partitioned.add(ar[0]);  partitioned.addAll(right);  printNumbers(partitioned);  }  static void printNumbers(List<Integer> numbers) {  System.out.println(String.join(" ", numbers.stream().map(String::valueOf).collect(Collectors.toList())));  }  } |

# Pangrams

|  |
| --- |
| public class Pangrams {  public static final int n = 26;  public int check(String arr){  // int count = 0;  if(arr.length() < n){  return -1;  }  for(char c = 'A'; c <= 'Z' ; c++){  if((arr.indexOf(c) < 0) && (arr.indexOf((char)(c + 32)) < 0)){  return -1;  }  }  return 1;  }  public static void main(String[] args) {  Scanner s1 = new Scanner(System.in);  String s = s1.nextLine();  Pangrams obj = new Pangrams();  int d = obj.check(s);  if(d == -1)  System.out.print("not pangram");  else  System.out.print("pangram");  }  } |

# Weighted Uniform Strings

|  |
| --- |
| public class WeightedUniformStrings {  private static HashSet<Integer> getWeights(String str) {  HashSet<Integer> weights = new HashSet<>();  int weight = 0;  char prev = ' '; // so it doesn't match 1st character  for (int i = 0; i < str.length(); i++) {  char curr = str.charAt(i);  if (curr != prev) {  weight = 0;  }  weight += curr - 'a' + 1;  weights.add(weight);  prev = curr;  }  return weights;  }  public static void main(String[] args) {  Scanner scan = new Scanner(System.in);  String str = scan.next();  int n = scan.nextInt();  HashSet<Integer> weights = getWeights(str);  while (n-- > 0) {  int x = scan.nextInt();  System.out.println(weights.contains(x) ? "Yes" : "No");  }  scan.close();  }  } |

# Separate the Numbers

|  |
| --- |
| public class SeparateNumbers {  public static void main(String[] args) {  Scanner stdin = new Scanner(System.in);  int tests = Integer.parseInt(stdin.nextLine());  t: for(int i = 0; i < tests; i++) {  String s = stdin.nextLine();  for(int j = 1; j <= s.length() / 2; j++) {  String head = s.substring(0, j);  long headVal = Long.parseLong(head);  long next = headVal + 1;  String rest = s.substring(j);  if(isContinous(rest, next)) {  System.out.println("YES " + head);  continue t;  }  }  System.out.println("NO");  }  stdin.close();  }  private static boolean isContinous(String rest, long next) {  String nextS = String.valueOf(next);  int i = nextS.length();  while(i <= rest.length()) {  if(!rest.startsWith(nextS)) {  return false;  } else {  next = next + 1;  rest = rest.substring(i);  nextS = String.valueOf(next);  i = nextS.length();  }  }  if(!rest.isEmpty()) {  return false;  }  return true;  }  } |

# Funny String

|  |
| --- |
| public class FunnyString {  public static void main(String[] args) throws IOException{  StringBuffer sb = new StringBuffer();  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  //For each test case  for(byte T = Byte.parseByte(br.readLine()); T > 0; --T){  //Check if input is funny  sb.append(isFunny(br.readLine().toCharArray()) ? "Funny\n": "Not Funny\n");  }  System.out.print(sb);  }  private static boolean isFunny(final char[] S){  for(int i = 0, n = S.length - 2; i < n; ++i, --n){  if(Math.abs(S[i] - S[i+1]) != Math.abs(S[n] - S[n+1])){  return false;  }  }  return true;  }  } |

# Counting Sort 1

|  |
| --- |
| public class CountingSort1 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] counts = new int[100];  for (int i = 0; i < n; i++) {  int number = sc.nextInt();  counts[number]++;  }  System.out.println(  String.join(" ", Arrays.stream(counts).mapToObj(Integer::toString).collect(Collectors.toList())));  sc.close();  }  } |

# Counting Sort 2

|  |
| --- |
| public class CountingSort2 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] counts = new int[100];  for (int i = 0; i < n; i++) {  int number = sc.nextInt();  counts[number]++;  }  boolean first = true;  for (int i = 0; i < counts.length; i++) {  for (int j = 0; j < counts[i]; j++) {  if (first) {  first = false;  } else {  System.out.print(" ");  }  System.out.print(i);  }  }  sc.close();  }  } |

# Gemstones

|  |
| --- |
| public class GemStones {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int rocks = in.nextInt();  char [] first = in.next().toCharArray();  Set<Character> result = new HashSet<>();  for(char c: first){  result.add(c);  }  for(int i = 1; i<rocks; i++){  char [] composition = in.next().toCharArray();  Set<Character> set = new HashSet<>();  for(char d : result){  for(int j = 0; j<composition.length; j++){  if(d == composition[j]){  set.add(composition[j]);  }  }  }  result = set;  }  System.out.println(result.size());  in.close();  }  } |

# Alternating Characters

|  |
| --- |
| public class AlternatingCharacters {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int tests = Integer.parseInt(in.nextLine());  for(int i = 0; i<tests ; i++){  String string = in.nextLine();  int deletions = 0;  char c = string.charAt(0);  for(int j = 1 ; j<string.length(); j++){  if(c == string.charAt(j)){  deletions++;  }else{  c = string.charAt(j);  }  }  System.out.println(deletions);  }  in.close();  }  } |

# The Full Counting Sort

|  |
| --- |
| public class FullCountingSort {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  @SuppressWarnings("unchecked")  List<String>[] sLists = new List[100];  for (int i = 0; i < sLists.length; i++) {  sLists[i] = new ArrayList<String>();  }  int n = sc.nextInt();  for (int i = 0; i < n; i++) {  int x = sc.nextInt();  String s = sc.next();  sLists[x].add((i < n / 2) ? "-" : s);  }  StringBuilder result = new StringBuilder();  for (int i = 0; i < sLists.length; i++) {  for (String s : sLists[i]) {  if (result.length() != 0) {  result.append(" ");  }  result.append(s);  }  }  System.out.println(result);  sc.close();  }  } |

# Beautiful Binary String

|  |
| --- |
| public class BeautifulBinaryString {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int l = Integer.parseInt(in.nextLine());  String s = in.nextLine();  String beautiful = s. replace("010" , "b");  int changes = beautiful.replaceAll("[01]","").length();  System.out.println(changes);  in.close();  }  } |

# Closest Numbers

|  |
| --- |
| public class ClosestNumbers {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int N = sc.nextInt();  int[] a = new int[N];  for (int i = 0; i < a.length; i++) {  a[i] = sc.nextInt();  }  Arrays.sort(a);  int minDiff = Integer.MAX\_VALUE;  List<Integer> indices = new ArrayList<Integer>();  for (int i = 0; i < a.length - 1; i++) {  int diff = a[i + 1] - a[i];  if (diff < minDiff) {  minDiff = diff;  indices.clear();  indices.add(i);  } else if (diff == minDiff) {  indices.add(i);  }  }  for (int i = 0; i < indices.size(); i++) {  if (i != 0) {  System.out.print(" ");  }  System.out.print(a[indices.get(i)] + " " + a[indices.get(i) + 1]);  }  sc.close();  }  } |

# The Love-Letter Mystery

|  |
| --- |
| public class Solution {  public static void main(String[] args) {  Scanner stdin = new Scanner(System.in);  int tests = stdin.nextInt();  for(int i = 0; i < tests; i++) {  String s = stdin.next();  int diff = 0;  for(int j = 0; j < s.length() / 2; j++) {  diff += Math.abs(Character.codePointAt(s, j) - Character.codePointAt(s, s.length() - j - 1));  }  System.out.println(diff);  }  stdin.close();  }  } |

# Find the Median

|  |
| --- |
| public class FindMedian {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] ar = new int[n];  for (int i = 0; i < ar.length; i++) {  ar[i] = sc.nextInt();  }  Arrays.sort(ar);  System.out.println(ar[ar.length / 2]);  sc.close();  }  } |

# Palindrome Index

|  |
| --- |
| public class PalindromeIndex {  public static void main(String[] args) {  Scanner input = new Scanner(System.in);  int n = input.nextInt();  input.nextLine();  tests: for (int t = 0; t < n; t++) {  String s = input.nextLine();  int outputIndex = -1;  boolean removal = false;  for (int i = 0, j = s.length() - 1; i < j; i++, j--) {  if (s.charAt(i) != s.charAt(j)) {  if (removal) {  removal = false;  outputIndex = -1;  break;  }  if (s.charAt(i + 1) == s.charAt(j)) {  removal = true;  outputIndex = i;  i++;  } else if (s.charAt(i) == s.charAt(j - 1)) {  removal = true;  outputIndex = j;  j--;  } else {  removal = false;  outputIndex = -1;  break;  }  }  }  if (outputIndex != -1) {  System.out.println(outputIndex);  continue tests;  }  for (int i = 0, j = s.length() - 1; i < j; i++, j--) {  if (s.charAt(i) != s.charAt(j)) {  if (removal) {  System.out.println(-1);  continue tests;  }  if (s.charAt(i) == s.charAt(j - 1)) {  removal = true;  outputIndex = j;  j--;  } else if (s.charAt(i + 1) == s.charAt(j)) {  removal = true;  outputIndex = i;  i++;  } else {  System.out.println(-1);  continue tests;  }  }  }  System.out.println(outputIndex);  }  }  } |

# Fraudulent Activity Notifications

|  |
| --- |
| public class FraudulentActivityNotifications {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int d = sc.nextInt();  int[] expenditures = new int[n];  for (int i = 0; i < expenditures.length; i++) {  expenditures[i] = sc.nextInt();  }  System.out.println(solve(expenditures, d));  sc.close();  }  static int solve(int[] expenditures, int d) {  int[] counts = new int[201];  for (int i = 0; i < d; i++) {  counts[expenditures[i]]++;  }  int result = 0;  for (int i = d; i < expenditures.length; i++) {  int lower = 0;  int leftNum = 0;  while ((leftNum + counts[lower]) \* 2 <= d) {  leftNum += counts[lower];  lower++;  }  int upper = counts.length - 1;  int rightNum = 0;  while ((rightNum + counts[upper]) \* 2 <= d) {  rightNum += counts[upper];  upper--;  }  if (expenditures[i] >= lower + upper) {  result++;  }  counts[expenditures[i - d]]--;  counts[expenditures[i]]++;  }  return result;  }  } |

# Anagram

|  |
| --- |
| public class Anagram {  public static void main(String[] args) {  Scanner stdin = new Scanner(System.in);  int tests = stdin.nextInt();  for(int i = 0; i < tests; i++) {  String s = stdin.next();  System.out.println(deletions(s));  }  stdin.close();  }  private static int deletions(String s) {  if(s.length() % 2 != 0) {  return -1;  }  String half1 = s.substring(0, s.length() / 2);  String half2 = s.substring(s.length() / 2);  for(int i = 0; i < half1.length(); i++) {  if(half2.contains(half1.substring(i, i + 1))) {  half2 = half2.replaceFirst(half1.substring(i, i + 1), "");  }  }  return half2.length();  }  } |

# Lily's Homework

|  |
| --- |
| public class LilyHomework {  public static void main(String[] args) {  Scanner input = new Scanner(System.in);  int n = input.nextInt();  int sortedSwaps = 0;  int[] homework = new int[n];  Integer[] homeworkSorted = new Integer[n];  Map<Integer,Integer> original = new HashMap<>();  int sortedReverseSwaps = 0;  int[] homework2ndCopy = new int[n];  Map<Integer,Integer> original2ndCopy = new HashMap<>();  //Initialize our arrays and maps  for(int i = 0; i < n; i++){  homeworkSorted[i] = input.nextInt();  homework[i] = homeworkSorted[i];  homework2ndCopy[i] = homeworkSorted[i];  original.put(homework[i],i);  original2ndCopy.put(homework2ndCopy[i],i);  }  Arrays.sort(homeworkSorted);//Sort the input ascending  for(int i = 0; i < n; i++){  if(homework[i] != homeworkSorted[i]){  //swap the element from homework to the right position  int tmp = homework[i];  homework[i] = homework[original.get(homeworkSorted[i])];  homework[original.get(homeworkSorted[i])] = tmp;  //Update index after swap  original.put(tmp,original.get(homeworkSorted[i]));  sortedSwaps++;  }  }  Arrays.sort(homeworkSorted, Collections.reverseOrder());//Sort the input descending  for(int i = 0; i < n; i++){  if(homework2ndCopy[i] != homeworkSorted[i]){  //swap the element from homework to the right position  int tmp = homework2ndCopy[i];  homework2ndCopy[i] = homework2ndCopy[original.get(homeworkSorted[i])];  homework2ndCopy[original2ndCopy.get(homeworkSorted[i])] = tmp;  //Update index after swap  original2ndCopy.put(tmp, original2ndCopy.get(homeworkSorted[i]));  sortedReverseSwaps++;  }  }  System.out.println(Math.min(sortedSwaps,sortedReverseSwaps));//Choose the smallest of the two possible smallest  }  } |

# Making Anagrams

|  |
| --- |
| public class Solution {  static int makingAnagrams(String s1, String s2){  int [] c1 = new int [256];  int [] c2 = new int [256];  for(int i =0; i<s1.length(); i++){  int index = (int)(s1.charAt(i));  c1[index] +=1;  }  for(int i =0; i<s2.length(); i++){  int index = (int)(s2.charAt(i));  c2[index] +=1;  }  int ans = 0;  for(int i = 0 ; i<256; i++){  ans += Math.abs(c1[i] - c2[i]);  }  return ans;  }  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  String s1 = in.next();  String s2 = in.next();  int result = makingAnagrams(s1, s2);  System.out.println(result);  }  } |

# Game of Thrones - I

|  |
| --- |
| public class Solution {  public static void main(String[] args) throws IOException{  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  System.out.print(isPalindromePermutation(br.readLine()) ? "YES" : "NO");  br.close();  }  private static boolean isPalindromePermutation(String str){  char[] chars = str.toCharArray();  int bits = 0;  int strLen = chars.length;  for(int i = 0; i < strLen; bits = bits ^ (1 << (chars[i++] - 'a'))){}  return (bits & (bits - (strLen & 1))) == 0;  }  } |

# Two Strings

|  |
| --- |
| public class TwoStrings {  public static void main(String[] args) {  /\* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. \*/  Scanner input = new Scanner(System.in);  int pairs = input.nextInt(); input.nextLine();  tests:  for(int t = 0; t < pairs; t++){  String a = input.nextLine();  String b = input.nextLine();  Set<Character> aLetterSet = new HashSet<>();  Set<Character> bLetterSet = new HashSet<>();  //Populate the sets  for(int i = 0; i < a.length(); i++)  aLetterSet.add(a.charAt(i));  for(int i = 0; i < b.length(); i++)  bLetterSet.add(b.charAt(i));  //Perform the intersection of the two sets  aLetterSet.retainAll(bLetterSet);  if(aLetterSet.size() > 0)  System.out.println("YES");  else  System.out.println("NO");  }  }  } |

# String Construction

|  |
| --- |
| public class StringConstruction {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int n = in.nextInt();  for(int a0 = 0; a0 < n; a0++){  String s = in.next();  Set<Character> uniqueChars = new HashSet<>();  for(char c : s.toCharArray()){  uniqueChars.add(c);  }  System.out.println(uniqueChars.size());  }  }  } |

# Sherlock and the Valid String

|  |
| --- |
| public class SherlockAndValidString {  public static void main(String[] args) throws IOException {  final char[] S = (new BufferedReader(new InputStreamReader(System.in))).readLine().toCharArray();  System.out.print(isValid(S) ? "YES" : "NO");  }  private static boolean isValid(final char[] S){  if(S.length < 2){  return true;  }  final int[] L = new int[26];  for(int i = 0, N = S.length; i < N; ++L[S[i++] - 'a']){}  Arrays.sort(L);  int i;  for(i = 0; i < 26 && L[i] == 0; ++i){}  if(L[i] == L[25]){  return true;  }  if(L[25] > L[24]){  return L[i] == L[24] && L[25] - L[24] == 1;  }  return L[i] < L[i+1] && L[i+1] == L[25] && L[i] == 1;  }  } |

# Highest Value Palindrome

|  |
| --- |
| public class Solution {  private static String largestPalindrome(String number, int k) {  char[] chars = number.toCharArray();  int minChange = 0;  for (int i = 0, j = chars.length - 1; i < j; i++, j--) {  if (chars[i] != chars[j]) {  minChange++;  }  }  if (minChange > k) {  return "-1";  }  int changeBoth = k - minChange;  int i = 0;  int j = chars.length - 1;  for (; i <= j; i++, j--) {  if (chars[i] != chars[j]) {  char maxChar = (char) Math.max(chars[i], chars[j]);  if (maxChar != '9' && changeBoth - 1 >= 0) {  chars[i] = '9';  chars[j] = '9';  changeBoth--;  } else {  chars[i] = maxChar;  chars[j] = maxChar;  minChange--;  }  } else {  char maxChar = (char) Math.max(chars[i], chars[j]);  if (maxChar != '9' && changeBoth - 2 >= 0) {  chars[i] = '9';  chars[j] = '9';  changeBoth -= 2;  }  }  }  if (changeBoth != 0 && i - 1 == j + 1) {  chars[i - 1] = '9';  }  String palindrome = new String(chars);  return palindrome;  }  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int n = in.nextInt();  int k = in.nextInt();  String number = in.next();  System.out.println(largestPalindrome(number, k));  }  } |

# Sherlock and Anagrams

|  |
| --- |
| public class SherlockandAnagrams {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int T = in.nextInt();  for (int tc = 0; tc < T; tc++) {  String S = in.next();  System.out.println(findUnorderedAnagramPairNum(S));  }  in.close();  }  static int findUnorderedAnagramPairNum(String str) {  Map<String, Integer> key2count = new HashMap<String, Integer>();  for (int i = 0; i < str.length(); i++) {  for (int j = i + 1; j <= str.length(); j++) {  String key = generateKey(str.substring(i, j));  if (!key2count.containsKey(key)) {  key2count.put(key, 0);  }  key2count.put(key, key2count.get(key) + 1);  }  }  return key2count.values().stream().mapToInt(count -> count \* (count - 1) / 2).sum();  }  static String generateKey(String str) {  return str.chars().sorted().mapToObj(letter -> (char) letter)  .collect(StringBuilder::new, StringBuilder::append, StringBuilder::append).toString();  }  } |

# Common Child

|  |
| --- |
| public class CommonChild {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  String a = in.next();  String b = in.next();  System.out.println(findMaxCommonLength(a, b));  in.close();  }  static int findMaxCommonLength(String s1, String s2) {  int length1 = s1.length();  int length2 = s2.length();  int[][] commonLengths = new int[length1 + 1][length2 + 1];  for (int i = 1; i <= length1; i++) {  for (int j = 1; j <= length2; j++) {  if (s1.charAt(i - 1) == s2.charAt(j - 1)) {  commonLengths[i][j] = commonLengths[i - 1][j - 1] + 1;  } else {  commonLengths[i][j] = Math.max(commonLengths[i - 1][j], commonLengths[i][j - 1]);  }  }  }  return commonLengths[length1][length2];  }  } |

# Bear and Steady Gene

|  |
| --- |
| public class BearandSteadyGene {  private static int trimTail (int tailIndex, int headIndex, HashMap<Character, Integer> tokens, char[] arr) {  while (true) {  char tailCh = arr[tailIndex];  if (tokens.get((Character)tailCh) == null) {  tailIndex++;  continue;  } else if ((tokens.get((Character)tailCh) < 0)) {  int incrementMe = tokens.get((Character)tailCh) + 1;  tokens.put((Character)tailCh, incrementMe);  tailIndex++;  continue;  } else if ((tokens.get((Character)tailCh) >= 0)) {  return tailIndex;  }  }  }  private static boolean subStringSatisfied (HashMap<Character, Integer> charMap) {  for (int count : charMap.values()) {  if (count > 0)  return false;  }  return true;  }  private static int findFirstHead(char[] arr, HashMap<Character, Integer> charMap, int tailIndex) {  int headIndex = tailIndex;  while (headIndex < arr.length) {  char tokenIn = arr[headIndex];  if (charMap.get(tokenIn) != null) {  int newCount = charMap.get(tokenIn) - 1;  charMap.put(tokenIn, newCount);  if (newCount == 0) {  if (subStringSatisfied(charMap)) {  return headIndex;  }  }  }  headIndex++;  }  return headIndex;  }  public static void main(String[] args) throws FileNotFoundException {  Scanner in = new Scanner(System.in);  int arrSize = in.nextInt();  String sequence = in.next();  int amountA = 0;  int amountC = 0;  int amountT = 0;  int amountG = 0;  char[] arr = new char[arrSize];  for (int i = 0; i < arr.length; i++) {  arr[i] = sequence.charAt(i);  switch (arr[i]) {  case 'A': amountA++;  break;  case 'C': amountC++;  break;  case 'T': amountT++;  break;  case 'G': amountG++;  break;  }  }  int oneFourth = arrSize/4;  if (amountA == oneFourth && amountC == oneFourth && amountT == oneFourth) {  System.out.println(0);  return;  }  HashMap<Character, Integer> tokens = new HashMap<Character, Integer>();  if (amountA - oneFourth > 0)  tokens.put('A', amountA-oneFourth);  if (amountC - oneFourth > 0)  tokens.put('C', amountC-oneFourth);  if (amountT - oneFourth > 0)  tokens.put('T', amountT-oneFourth);  if (amountG - oneFourth > 0)  tokens.put('G', amountG-oneFourth);  int minSize = 0;  for (int i : tokens.values())  minSize += i;  int tailIndex = 0;  while (!tokens.keySet().contains((Character)arr[tailIndex])) {  tailIndex++;  }  int headIndex = findFirstHead(arr, tokens, tailIndex);  tailIndex = trimTail(tailIndex, headIndex, tokens, arr);  int bestSizeFound = headIndex + 1 - tailIndex;  int reqTokens = 0;  while (headIndex + 1 + reqTokens < arr.length) {  headIndex++;  tailIndex++;  if (tokens.get((Character)arr[tailIndex - 1]) != null) {  tokens.put((Character)arr[tailIndex - 1], tokens.get((Character)arr[tailIndex - 1]) + 1);  }  if (tokens.get((Character)arr[headIndex]) != null) {  int deIncrementMe = tokens.get((Character)arr[headIndex]) - 1;  tokens.put((Character)arr[headIndex], deIncrementMe);  if (subStringSatisfied(tokens))  tailIndex = trimTail(tailIndex, headIndex, tokens, arr);  }  }  System.out.println(headIndex+1-tailIndex);  }  } |

# Morgan and a String

|  |
| --- |
| public class MorganAndString {  public static void main(String[] args) {  Scanner input = new Scanner(System.in);  int t = input.nextInt();  for (int a0 = 0; a0 < t; a0++) {  StringBuilder s1 = new StringBuilder(input.next());  s1.append("z");  StringBuilder s2 = new StringBuilder(input.next());  s2.append("z");  StringBuilder output = new StringBuilder("");  int i = 0, j = 0;  while (i < s1.length() && j < s2.length()) {  if (s1.charAt(i) < s2.charAt(j)) {  output.append(s1.charAt(i));  i++;  } else if (s1.charAt(i) > s2.charAt(j)) {  output.append(s2.charAt(j));  j++;  } else {  if (s1.charAt(i) == 'z') {  i++;  j++;  continue;  }  int startingI = i;  int startingJ = j;  while (s1.charAt(i) == s2.charAt(j)) {  i++;  j++;  if (i >= s1.length() && j >= s2.length()) {  i = startingI;  j = startingJ;  break;  } else if (i >= s1.length()) {  char prev = s2.charAt(startingJ);  while (s2.charAt(startingJ) <= prev) {  output.append(s2.charAt(startingJ));  prev = s2.charAt(startingJ);  startingI++;  }  i = startingI;  j = startingJ;  } else if (j >= s2.length()) {  char prev = s1.charAt(startingI);  while (s1.charAt(startingI) <= prev) {  output.append(s1.charAt(startingI));  prev = s1.charAt(startingI);  startingI++;  }  i = startingI;  j = startingJ;  }  }  if (s1.charAt(i) <= s2.charAt(j)) {  char prev = s1.charAt(startingI);  while (s1.charAt(startingI) <= prev) {  output.append(s1.charAt(startingI));  prev = s1.charAt(startingI);  startingI++;  }  i = startingI;  j = startingJ;  }  if (s1.charAt(i) > s2.charAt(j)) {  char prev = s2.charAt(startingJ);  while (s2.charAt(startingJ) <= prev) {  output.append(s2.charAt(startingJ));  prev = s2.charAt(startingJ);  startingJ++;  }  i = startingI;  j = startingJ;  }  }  }  while (i < s1.length()) {  output.append(s1.charAt(i));  i++;  }  while (j < s2.length()) {  output.append(s2.charAt(j));  j++;  }  System.out.println(output);  }  }  } |

# Count Strings

|  |
| --- |
| public class CountStrings {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  for (int n = sc.nextInt(); n > 0; --n)  System.out.println(count(sc.next(), sc.nextInt()));  }  static long count(String regex, int len) {  return len == 0 ? 0 : new RegexCounter(new Regex(regex)).count(len);  }  }  class RegexCounter {  private final long[][] adjacencyMatrix;  private final ArrayList<Integer> finalStates;  RegexCounter(Regex regex) {  Map<Regex.Node, Integer> indexes = new HashMap<>();  this.adjacencyMatrix = new long[regex.nodes.length][regex.nodes.length];  this.finalStates = new ArrayList<>();  for (Regex.Node node : regex.nodes) {  int index = getIndex(indexes, node);  for (Regex.Edge edge : node.edges) {  int next = getIndex(indexes, edge.node);  adjacencyMatrix[index][next] = 1;  }  }  }  private int getIndex(Map<Regex.Node, Integer> indexes, Regex.Node node) {  Integer index = indexes.get(node);  if (index == null) {  indexes.put(node, index = indexes.size());  if (node.isFinal)  finalStates.add(index);  }  return index;  }  long count(int len) {  long[][] m = new MatrixPower(adjacencyMatrix).power(len);  long count = 0;  for (int finalState : finalStates)  count = (count + m[0][finalState]) % 1000000007L;  return count;  }  }  class Regex {  static char e = '\0';  final Node root;  final Node[] nodes;  Regex(String regex) {  Parser parser = new Parser(regex);  parser.parse();  SubsetConstruction subset = new SubsetConstruction(parser.expr.start, parser.expr.end);  this.nodes = subset.dfa();  this.root = nodes[0];  }  static class SubsetConstruction {  private final Node nfaEnd;  private final Queue<State> queue;  private final Map<Set<Node>, Node> dfaMap;  private final Node dfaRoot;  SubsetConstruction(Node nfaRoot, Node nfaEnd) {  this.nfaEnd = nfaEnd;  this.queue = new ArrayDeque<>();  this.dfaMap = new HashMap<>();  this.dfaRoot = addState(eClosure(nfaRoot)).dfa;  }  Node[] dfa() {  while (!queue.isEmpty()) {  State state = queue.poll();  for (char c : new char[]{'a', 'b'}) {  Set<Node> nfaNext = eClosure(next(state.nfa, c));  if (nfaNext.isEmpty())  continue;  Node dfaNext = dfaMap.get(nfaNext);  if (dfaNext == null)  dfaNext = addState(nfaNext).dfa;  state.dfa.edges.add(new Edge(c, dfaNext));  }  }  return getNodes();  }  private Node[] getNodes() {  ArrayList<Node> nodes = new ArrayList<>();  nodes.add(dfaRoot);  for (Node node : dfaMap.values())  if (node != dfaRoot) nodes.add(node);  return nodes.toArray(new Node[nodes.size()]);  }  private State addState(Set<Node> nfa) {  State state = new State(nfa);  state.dfa.isFinal = nfa.contains(nfaEnd);  dfaMap.put(state.nfa, state.dfa);  queue.add(state);  return state;  }  static class State {  final Set<Node> nfa;  final Node dfa = new Node();  State(Set<Node> nfa) { this.nfa = nfa; }  }  private Set<Node> eClosure(Node node) {  return eClosure(Collections.singletonList(node));  }  private static Set<Node> eClosure(Collection<Node> nodes) {  Set<Node> closure = new HashSet<>();  Stack<Node> stack = new Stack<>();  stack.addAll(nodes);  while (!stack.isEmpty()) {  Node node = stack.pop();  if (closure.add(node))  stack.addAll(next(node, e));  }  return closure;  }  private static Collection<Node> next(Collection<Node> nodes, char c) {  Collection<Node> list = new ArrayList<>();  for (Node node : nodes)  list.addAll(next(node, c));  return list;  }  private static Collection<Node> next(Node node, char c) {  Collection<Node> list = new ArrayList<>();  for (Edge edge : node.edges)  if (edge.value == c) list.add(edge.node);  return list;  }  }  static class Parser {  private final String regex;  private int pos;  Expression expr;  Parser(String regex) {  this.regex = regex;  this.pos = 0;  }  Node parse() {  return (expr = sequence()).start;  }  private static class Expression {  Node start = new Node();  Node end = start;  }  private Expression sequence() {  Expression expr = new Expression();  for (; ; ) {  char c = take();  switch (c) {  case 'a':  case 'b': literal(expr, c); break;  case '(': expr = parenthesis(expr); break;  case '|': expr = pipe(expr); break;  case '\*': expr = star(expr); break;  default: putback(); return expr;  }  }  }  private void literal(Expression expr, char c) {  expr.end.edges.add(new Edge(c, expr.end = new Node()));  }  private Expression parenthesis(Expression expr) {  Expression nested = sequence();  if (take() != ')')  throw new IllegalStateException("syntax error: " + ") expected");  if (expr.start == expr.end) return nested;  expr.end.edges.add(new Edge(e, nested.start));  expr.end = nested.end;  return expr;  }  private Expression pipe(Expression first) {  Expression second = sequence();  Expression expr = new Expression();  expr.start.edges.add(new Edge(e, first.start));  expr.start.edges.add(new Edge(e, second.start));  first.end.edges.add(new Edge(e, expr.end = new Node()));  second.end.edges.add(new Edge(e, expr.end));  return expr;  }  private Expression star(Expression inner) {  Expression expr = new Expression();  expr.start.edges.add(new Edge(e, inner.start));  inner.end.edges.add(new Edge(e, inner.start));  inner.end.edges.add(new Edge(e, expr.end = new Node()));  expr.start.edges.add(new Edge(e, expr.end));  return expr;  }  private char take() {  return skipWs() ? regex.charAt(pos++) : e;  }  private void putback() {  if (pos >= 0) --pos;  }  private boolean skipWs() {  while (pos < regex.length() && Character.isWhitespace(regex.charAt(pos))) ++pos;  return pos < regex.length();  }  }  static class Node {  final ArrayList<Edge> edges = new ArrayList<>();  boolean isFinal;  }  static class Edge {  final char value;  final Node node;  Edge(char value, Node node) {  this.value = value;  this.node = node;  }  }  }  class MatrixPower {  private final long[][] matrix;  private final Map<Integer, long[][]> powers;  MatrixPower(long[][] matrix) {  this.matrix = matrix;  this.powers = new HashMap<>();  }  long[][] power(int p) {  if (p == 1) return matrix;  long[][] result = powers.get(p);  if (result != null)  return result;  result = power(p / 2);  powers.put(p / 2 \* 2, result = multiply(result, result));  if (p % 2 > 0)  powers.put(p, result = multiply(result, power(p % 2)));  return result;  }  private long[][] multiply(long[][] a, long[][] b) {  long[][] m = new long[a.length][a.length];  for (int i = 0; i < a.length; ++i) {  for (int j = 0; j < a.length; ++j) {  long sum = 0;  for (int k = 0; k < a.length; ++k)  sum = (sum + a[i][k] \* b[k][j]) % 1000000007L;  m[i][j] = sum;  }  }  return m;  }  } |

# String Function Calculation

|  |
| --- |
| public class StringFunctionCalculation {  public static long solve(String str) {  SuffixAutomata a = new SuffixAutomata(str);  return a.root.dp();  }  public static void main(String[] args) throws Exception {  FastScanner sc = new FastScanner(System.in);  String str = sc.next();  System.out.println(solve(str));  }  private static class SuffixAutomata {  private Vertex root;  private Vertex last;  private class Vertex {  Vertex suffixLink = null;  Vertex[] edges;  int log = 0;  boolean visited = false;  int terminals = 0;  public Vertex(Vertex o, int log) {  edges = o.edges.clone();  this.log = log;  }  public Vertex(int log) {  edges = new Vertex[26];  this.log = log;  }  long dp() {  if (visited) {  return 0;  }  visited = true;  long r = 0;  for (Vertex v : edges) {  if (v != null) {  r = Math.max(r, v.dp());  terminals += v.terminals;  }  }  return Math.max(r, 1L \* log \* terminals);  }  }  public SuffixAutomata(String str) {  last = root = new Vertex(0);  for (int i = 0; i < str.length(); i++) {  addChar(str.charAt(i));  }  addTerminal();  }  private void addChar(char c) {  Vertex cur = last;  last = new Vertex(cur.log + 1);  while (cur != null && cur.edges[c - 'a'] == null) {  cur.edges[c - 'a'] = last;  cur = cur.suffixLink;  }  if (cur != null) {  Vertex q = cur.edges[c - 'a'];  if (q.log == cur.log + 1) {  last.suffixLink = q;  } else {  Vertex r = new Vertex(q, cur.log + 1);  r.suffixLink = q.suffixLink;  q.suffixLink = r;  last.suffixLink = r;  while (cur != null) {  if (cur.edges[c - 'a'] == q) {  cur.edges[c - 'a'] = r;  } else {  break;  }  cur = cur.suffixLink;  }  }  } else {  last.suffixLink = root;  }  }  private void addTerminal() {  Vertex cur = last;  while (cur != null) {  cur.terminals++;  cur = cur.suffixLink;  }  }  }  static class FastScanner {  private static BufferedReader reader;  private static StringTokenizer tokenizer;  public FastScanner(InputStream in) throws Exception {  reader = new BufferedReader(new InputStreamReader(in));  tokenizer = new StringTokenizer(reader.readLine().trim());  }  public int numTokens() throws Exception {  if (!tokenizer.hasMoreTokens()) {  tokenizer = new StringTokenizer(reader.readLine().trim());  return numTokens();  }  return tokenizer.countTokens();  }  public boolean hasNext() throws Exception {  if (!tokenizer.hasMoreTokens()) {  tokenizer = new StringTokenizer(reader.readLine().trim());  return hasNext();  }  return true;  }  public String next() throws Exception {  if (!tokenizer.hasMoreTokens()) {  tokenizer = new StringTokenizer(reader.readLine().trim());  return next();  }  return tokenizer.nextToken();  }  public double nextDouble() throws Exception {  return Double.parseDouble(next());  }  public float nextFloat() throws Exception {  return Float.parseFloat(next());  }  public long nextLong() throws Exception {  return Long.parseLong(next());  }  public int nextInt() throws Exception {  return Integer.parseInt(next());  }  public int[] nextIntArray() throws Exception {  String[] line = reader.readLine().trim().split(" ");  int[] out = new int[line.length];  for (int i = 0; i < line.length; i++) {  out[i] = Integer.valueOf(line[i]);  }  return out;  }  public double[] nextDoubleArray() throws Exception {  String[] line = reader.readLine().trim().split(" ");  double[] out = new double[line.length];  for (int i = 0; i < line.length; i++) {  out[i] = Double.valueOf(line[i]);  }  return out;  }  public Integer[] nextIntegerArray() throws Exception {  String[] line = reader.readLine().trim().split(" ");  Integer[] out = new Integer[line.length];  for (int i = 0; i < line.length; i++) {  out[i] = Integer.valueOf(line[i]);  }  return out;  }  public BigInteger[] nextBigIngtegerArray() throws Exception {  String[] line = reader.readLine().trim().split(" ");  BigInteger[] out = new BigInteger[line.length];  for (int i = 0; i < line.length; i++) {  out[i] = new BigInteger(line[i]);  }  return out;  }  public BigInteger nextBigInteger() throws Exception {  return new BigInteger(next());  }  public String nextLine() throws Exception {  return reader.readLine().trim();  }  public long[] nextLongArray() throws Exception {  String[] line = reader.readLine().trim().split(" ");  long[] out = new long[line.length];  for (int i = 0; i < line.length; i++) {  out[i] = Long.valueOf(line[i]);  }  return out;  }  }  } |

# Ice Cream Parlor

|  |
| --- |
| public class IceCreamParlor{  public static void main(String[] args) {  Scanner in=new Scanner(System.in);  int tt=in.nextInt();  for(int i=0;i<tt;i++){  int amnt=in.nextInt();  int flav=in.nextInt();  int[] a=new int[flav];  for(int j=0;j<a.length;j++){  a[j]=in.nextInt();  }  outloop:  for(int j=0;j<a.length;j++){  for(int k=0;k<a.length;k++){  if(k!=j && (a[k]+a[j])==amnt){  System.out.println((j+1)+" "+(k+1));  break outloop;  }  }  }  }  }  } |

# Minimum Loss

|  |
| --- |
| public class MinimumLoss {  public static void main(String[] args) {  /\* Enter your code here. Read input from STDIN. Print output to STDOUT. Your class should be named Solution. \*/  Scanner input = new Scanner(System.in);  int n = input.nextInt();  long[] prices = new long[n];  HashMap<Long,Integer> indices = new HashMap<>();  //Populate prices array with the input  for(int i = 0; i < n; i++){  prices[i] = input.nextLong();  indices.put(prices[i],i);  }  Arrays.sort(prices);//Performs a double pivot quicksort and sorts ascending  Long minimumLoss = Long.MAX\_VALUE;  //Iterate from end to start  for(int i = n-1; i >0; i--){  //Make sure it is a smaller loss  if(prices[i]-prices[i-1] < minimumLoss){  //Verify if the pair is a valid transaction  if(indices.get(prices[i]) < indices.get(prices[i-1]))  minimumLoss = prices[i]-prices[i-1];  }  }  System.out.println(minimumLoss);  }  } |

# Missing Numbers

|  |
| --- |
| public class MissingNumbers {  private static int RADIUS = 100;  public static void main(String[] args) throws IOException {  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  int[] C = new int[2\*RADIUS];  //First list  int N = Integer.parseInt(br.readLine());  String[] temp = br.readLine().split(" ");  int offset = Integer.parseInt(temp[0]) - RADIUS;  for(int i = 0; i < N; i++){  short v = Short.parseShort(temp[i]);  C[v - offset]--;  }  //Second list  int M = Integer.parseInt(br.readLine());  temp = br.readLine().split(" ");  for(int i = 0; i < M; i++){  short v = Short.parseShort(temp[i]);  C[v - offset]++;  }  //Print  StringBuffer str = new StringBuffer();  for(int i = 0; i < C.length; i++){  if (C[i] != 0){  str.append((i + offset) + " ");  }  }  System.out.print(str);  }  } |

# Pairs

|  |
| --- |
| public class Pairs {  public static void main(String[] args) throws IOException {  BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));  //Get and parse the first line  String[] line = reader.readLine().split(" ");  int N = Integer.parseInt(line[0]), K = Integer.parseInt(line[1]);  //Get and parse the set of numbers  line = reader.readLine().split(" ");  int[] nums = new int[N];  for(int i = 0; i < N; i++) {  nums[i] = Integer.parseInt(line[i]);  }  line = null;  //Find the amount of valid pairs  int count = 0;  Arrays.sort(nums);  for(int i = 0; i < nums.length; i++) {  count += Arrays.binarySearch(nums, i+1, nums.length, nums[i] + K) < 0 ? 0 : 1;  }  //Print answer  System.out.print(count);  }  } |

# Sherlock and Array

|  |
| --- |
| public class SherlockandArray{  public static void main(String[] args) throws IOException {  StringBuffer sb = new StringBuffer();  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  //For each test  for(byte T = Byte.parseByte(br.readLine()); T > 0; --T){  //Get input  int N = Integer.parseInt(br.readLine());  short[] A = new short[N];  int i = 0;  for(String s : br.readLine().split(" ")){  A[i++] = Short.parseShort(s);  }  //Solve  int sum;  int[] S = new int[N];  //Get left to right sums  sum = 0;  for(i = 0; i < N; ++i){  S[i] = sum += A[i];  }  //Subtract right to left sums  sum = 0;  for(i = N - 1; i >= 0; --i){  S[i] -= sum += A[i];  }  //When equal, left-to-right - right-to-left = 0  sb.append((Arrays.binarySearch(S, 0) < 0) ? "NO\n" : "YES\n");  }  //Print  System.out.print(sb);  }  } |

# Connected Cells in a Grid

|  |
| --- |
| public class ConnectedCellsinGrid {  public static void main(String[] args) throws IOException {  //Get input  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  final byte Y = Byte.parseByte(br.readLine());  final byte X = Byte.parseByte(br.readLine());  final byte[][] M = new byte[Y][X];  for(byte y = 0; y < Y; ++y){  final String[] line = br.readLine().split(" ");  for(byte x = 0; x < X; ++x){  M[y][x] = Byte.parseByte(line[x]);  }  }  br.close();  //Go through the matrix  byte maxRegionSize = 0;  for(byte y = 0; y < Y; ++y){  for(byte x = 0; x < X; ++x){  //Find regions and keep track of largest size  if(M[y][x] > 0){  maxRegionSize = (byte)Math.max(maxRegionSize, getRegionSize(M, y, x, Y, X));  }  }  }  //Print output  System.out.print(maxRegionSize);  }  private static byte getRegionSize(final byte[][] M, final byte y, final byte x, final byte Y, final byte X){  byte count = 0;  M[y][x] = 0;  final Queue<Point> Q = new LinkedList<Point>();  Q.add(new Point(y, x));  do{  ++count;  Point p = Q.poll();  if(p.y > 0){  if(p.x > 0 && M[p.y-1][p.x-1] > 0){  Q.add(new Point((byte)(p.y-1), (byte)(p.x-1)));  M[p.y-1][p.x-1] = 0;  }  if(M[p.y-1][p.x] > 0){  Q.add(new Point((byte)(p.y-1), (byte)(p.x)));  M[p.y-1][p.x] = 0;  }  if(p.x+1 < X && M[p.y-1][p.x+1] > 0){  Q.add(new Point((byte)(p.y-1), (byte)(p.x+1)));  M[p.y-1][p.x+1] = 0;  }  }  if(p.x > 0 && M[p.y][p.x-1] > 0){  Q.add(new Point((byte)(p.y), (byte)(p.x-1)));  M[p.y][p.x-1] = 0;  }  if(p.x+1 < X && M[p.y][p.x+1] > 0){  Q.add(new Point((byte)(p.y), (byte)(p.x+1)));  M[p.y][p.x+1] = 0;  }  if(p.y+1 < Y){  if(p.x > 0 && M[p.y+1][p.x-1] > 0){  Q.add(new Point((byte)(p.y+1), (byte)(p.x-1)));  M[p.y+1][p.x-1] = 0;  }  if(M[p.y+1][p.x] > 0){  Q.add(new Point((byte)(p.y+1), (byte)(p.x)));  M[p.y+1][p.x] = 0;  }  if(p.x+1 < X && M[p.y+1][p.x+1] > 0){  Q.add(new Point((byte)(p.y+1), (byte)(p.x+1)));  M[p.y+1][p.x+1] = 0;  }  }  } while (!Q.isEmpty());  return count;  }  private static class Point{  public final byte y;  public final byte x;  public Point(final byte y, final byte x){  this.y = y;  this.x = x;  }  }  } |

# Short Palindrome

|  |
| --- |
| public class ShortPalindrome {  public static void main(String[] args) {  Scanner input = new Scanner(System.in);  String s = input.next();  int[] freq = new int[26];  int[][] pairfreq = new int[26][26];  int[] tripfreq = new int[26];  final int CONSTANT = 1000000000+7;  int ans = 0;  for(char c:s.toCharArray()){  ans = (ans+tripfreq[c-'a'])%CONSTANT;  for(int i=0; i<26; i++){  tripfreq[i] = (tripfreq[i] + pairfreq[i][c-'a'])%CONSTANT;  }  for(int i=0; i<26; i++){  pairfreq[i][c-'a'] = (pairfreq[i][c-'a'] + freq[i])%CONSTANT;  }  freq[c-'a']++;  }  System.out.println(ans);  }  } |

# Count Luck

|  |
| --- |
| public class CountLuck {  public static void main(String[] args) {  Scanner input = new Scanner(System.in);  int T = input.nextInt();  for(int t=0; t<T; t++){  int m = input.nextInt();  int n = input.nextInt();  int[][] A = new int[m][n];  int starti = 0,startj = 0;  for(int i=0; i<m; i++){  String s = input.next();  for(int j=0; j<n; j++){  if(s.charAt(j)=='X'){  A[i][j] = 1;  }else if(s.charAt(j)=='M'){  A[i][j] = 1;  starti = i; startj = j;  }else if(s.charAt(j)=='\*'){  A[i][j] = 2;  }  }  }  int k = input.nextInt();  int ans = start(A,starti,startj, m, n, 0);  if(k!=ans){  System.out.println("Oops!");  }else{  System.out.println("Impressed");  }  }  }  public static int start(int[][] A, int i, int j, int m, int n, int kSoFar){  if(A[i][j] == 2)return kSoFar;  A[i][j] = 1;  boolean left=false, right=false, up=false, down=false;  if(i-1>=0 && (A[i-1][j]==0 || A[i-1][j]==2))up = true;  if(i+1<m && (A[i+1][j]==0 || A[i+1][j]==2))down = true;  if(j-1>=0 && (A[i][j-1]==0 || A[i][j-1]==2))left = true;  if(j+1<n && (A[i][j+1]==0 || A[i][j+1]==2))right = true;  if(!up && !down && !left && !right)return 0;  if(!up && !down && !left && right){  return start(A,i,j+1,m,n,kSoFar);  }  if(!up && !down && left && !right){  return start(A,i,j-1,m,n,kSoFar);  }  if(!up && down && !left && !right){  return start(A,i+1,j,m,n,kSoFar);  }  if(up && !down && !left && !right){  return start(A,i-1,j,m,n,kSoFar);  }  int returnValue = 0;  if(right) returnValue += start(A,i,j+1,m,n,kSoFar+1);  if(left) returnValue += start(A,i,j-1,m,n,kSoFar+1);  if(up) returnValue += start(A,i-1,j,m,n,kSoFar+1);  if(down) returnValue += start(A,i+1,j,m,n,kSoFar+1);  return returnValue;  }  } |

# Cut the Tree

|  |
| --- |
| public class CutTree{  public static void main(String[] args) throws IOException {  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  int N = Integer.parseInt(br.readLine());  Vertex[] verts = new Vertex[N];  N = 0;  int sum = 0;  for(String s : br.readLine().split(" ")){  short v = Short.parseShort(s);  sum += v;  verts[N++] = new Vertex(v);  }  while (--N > 0){  String[] temp = br.readLine().split(" ");  int i1 = Integer.parseInt(temp[0]) - 1;  int i2 = Integer.parseInt(temp[1]) - 1;  verts[i1].addEdge(verts[i2], sum);  verts[i2].addEdge(verts[i1], sum);  }  Vertex.init(verts[0]);  int min = Integer.MAX\_VALUE;  for(Vertex vertex : verts){  int curMin = vertex.minDiff(sum);  min = (curMin < min) ? curMin : min;  }  System.out.print(min);  }  private static class Vertex{  private int minDiff;  private short value;  private boolean isInit;  private List<Edge> edges;  public Vertex(short value){  this.value = value;  this.isInit = false;  this.edges = new ArrayList<Edge>();  }  public void addEdge(Vertex vertex, int value){  edges.add(new Edge(vertex, value));  }  public int minDiff(int sum){  int min = sum;  for(Edge edge : edges){  int curMin = sum - 2\*edge.value;  curMin = (curMin < 0) ? -curMin : curMin;  min = (curMin < min) ? curMin : min;  }  return min;  }  public static void init(Vertex v){  if (!v.isInit){  v.init();  }  }  private int init(){  this.isInit = true;  Edge caller = null;  int sum = this.value;  for(Edge edge : edges){  if (edge.vertex.isInit){  caller = edge;  } else {  sum += edge.value = edge.vertex.init();  }  }  if (caller != null){  caller.value -= sum;  }  return sum;  }  private class Edge{  public int value;  public Vertex vertex;  public Edge(Vertex vertex, int value){  this.value = value;  this.vertex = vertex;  }  }  }  } |

# The Coin Change Problem

|  |
| --- |
| public class CoinChangeProblem {  public static void main(String[] args) {  /\* Save input \*/  Scanner scan = new Scanner(System.in);  int n = scan.nextInt();  int m = scan.nextInt();  int [] coins = new int[m];  for (int i = 0; i < m; i++) {  coins[i] = scan.nextInt();  }  scan.close();  System.out.println(numWays(n, coins));  }  public static long numWays(int n, int [] coins) {  if (n < 0) {  return 0;  }  return numWays(n, coins, 0, new HashMap<String, Long>());  }  public static long numWays(int n, int [] coins, int coinNumber, HashMap<String, Long> cache) {  /\* Check our cache \*/  String key = n + "," + coinNumber;  if (cache.containsKey(key)) {  return cache.get(key);  }  /\* Base case \*/  if (coinNumber == coins.length - 1) {  if (n % coins[coinNumber] == 0) {  cache.put(key, 1L);  return 1;  } else {  cache.put(key, 0L);  return 0;  }  }  /\* Recursive case \*/  long ways = 0;  for (int i = 0; i <= n; i += coins[coinNumber]) {  ways += numWays(n - i, coins, coinNumber + 1, cache);  }  /\* Cache and return solution \*/  cache.put(key, ways);  return ways;  }  } |

# Equal

|  |
| --- |
| public class Equal {  public static void main(String[] args) throws IOException {  StringBuffer sb = new StringBuffer();  //INPUT  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  for(byte T = Byte.parseByte(br.readLine()); T > 0; --T){  final short N = Short.parseShort(br.readLine());  final short[] A = new short[N];  String[] input = br.readLine().split(" ");  for(short i = 0; i < N; ++i){  A[i] = Short.parseShort(input[i]);  }  input = null;  //SOLVE  if(N < 2){  sb.append("0\n");  continue;  }  int minVal = A[0];  for(int i = 1; i < N; ++i){  if (A[i] < minVal){  minVal = A[i];  }  }  int minCount = Integer.MAX\_VALUE;  for(int i = 0; i <= 5; ++i){  int count = 0;  for(short j = 0; j < N; ++j){  short V = (short)(A[j] - (minVal - i));  count += V/5 + (V %= 5)/2 + (V & 1);  }  if (count < minCount){  minCount = count;  }  }  //OUTPUT  sb.append(minCount+ "\n");  }  System.out.print(sb);  }  } |

# Sam and substrings

|  |
| --- |
| public class SamAndSubstrings {  private final static int MOD = 1000000007;  public static void main(String[] args) throws IOException {  int[] balls = strNumToArr((new BufferedReader(new InputStreamReader(System.in))).readLine());  int n = balls.length;  for(int i = n - 2; i >= 0; --i){  balls[i] = (int)((balls[i+1] + (((long)balls[i])\*(n - i))%MOD)%MOD);  }  int pow = 1;  int total = 0;  for(int i = 0; i < n; ++i){  total = (int)((total + (((long)balls[i])\*pow)%MOD)%MOD);  pow = (int)((pow\*10L)%MOD);  }  System.out.print(total);  }  private static int[] strNumToArr(String str){  int n = str.length();  int[] ar = new int[n];  for(char c : str.toCharArray()){  ar[--n] = c - '0';  }  return ar;  }  } |

# Fibonacci Modified

|  |
| --- |
| public class FibonacciModified {  public static void main(String[] args) throws IOException{  //Get input  String[] line = (new BufferedReader(new InputStreamReader(System.in))).readLine().split(" ");  BigInteger A = BigInteger.valueOf(Byte.parseByte(line[0]));  BigInteger B = BigInteger.valueOf(Byte.parseByte(line[1]));  //Solve  for(byte n = (byte)(Byte.parseByte(line[2]) - 2); n > 0; --n){  BigInteger C = B.multiply(B).add(A);  A = B;  B = C;  }  //Print output  System.out.print(B);  }  } |

# Lonely Integer

|  |
| --- |
| public class Solution {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int[] a = new int[n];  for (int i = 0; i < a.length; i++) {  a[i] = sc.nextInt();  }  System.out.println(solve(a));  sc.close();  }  static int solve(int[] a) {  return Arrays.stream(a).reduce((x, y) -> x ^ y).getAsInt();  }  } |

# Maximizing XOR

|  |
| --- |
| public class Solution {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int l = sc.nextInt();  int r = sc.nextInt();  System.out.println(solve(l, r));  sc.close();  }  static int solve(int l, int r) {  return IntStream.rangeClosed(l, r).flatMap(a -> IntStream.rangeClosed(a, r).map(b -> a ^ b)).max().getAsInt();  }  } |

# Sum vs XOR

|  |
| --- |
| public class Solution {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  long n = sc.nextLong();  System.out.println(solve(n));  sc.close();  }  static long solve(long n) {  if (n == 0) {  return 1;  }  return 1L << Long.toBinaryString(n).chars().filter(ch -> ch == '0').count();  }  } |

# Flipping bits

|  |
| --- |
| public class Solution {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int T = sc.nextInt();  for (int tc = 0; tc < T; tc++) {  long n = sc.nextLong();  System.out.println(solve(n));  }  sc.close();  }  static long solve(long n) {  return ~n & ((1L << 32) - 1);  }  } |

# Recursive Digit Sum

|  |
| --- |
| public class Solution {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  String n = sc.next();  int k = sc.nextInt();  System.out.println(solve(n, k));  sc.close();  }  static int solve(String n, int k) {  long result = (long) computeDigitSum(n) \* k;  while (result >= 10) {  result = computeDigitSum(String.valueOf(result));  }  return (int) result;  }  static int computeDigitSum(String s) {  return s.chars().map(ch -> ch - '0').sum();  }  } |

# Game of Stones

|  |
| --- |
| public class Solution {  static final int LIMIT = 100;  static final int[] STONES\_IN\_ONE\_MOVE = { 2, 3, 5 };  static boolean[] firstWins;  public static void main(String[] args) {  buildFirstWins();  Scanner sc = new Scanner(System.in);  int T = sc.nextInt();  for (int tc = 0; tc < T; tc++) {  int n = sc.nextInt();  System.out.println(firstWins[n] ? "First" : "Second");  }  sc.close();  }  static void buildFirstWins() {  firstWins = new boolean[LIMIT + 1];  for (int i = 0; i < firstWins.length; i++) {  for (int stone : STONES\_IN\_ONE\_MOVE) {  if (i >= stone && !firstWins[i - stone]) {  firstWins[i] = true;  break;  }  }  }  }  } |

# Tower Breakers

|  |
| --- |
| public class Solution {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int t = sc.nextInt();  for (int tc = 0; tc < t; tc++) {  int n = sc.nextInt();  int m = sc.nextInt();  System.out.println(solve(n, m));  }  sc.close();  }  static int solve(int n, int m) {  return (n % 2 == 1 && m != 1) ? 1 : 2;  }  } |

# A Chessboard Game

|  |
| --- |
| public class Solution {  static final int SIZE = 15;  static final int[] R\_OFFSETS = { -1, 1, 2, 2 };  static final int[] C\_OFFSETS = { 2, 2, 1, -1 };  static boolean[][] firstWins;  public static void main(String[] args) {  buildFirstWins();  Scanner sc = new Scanner(System.in);  int t = sc.nextInt();  for (int tc = 0; tc < t; tc++) {  int x = sc.nextInt();  int y = sc.nextInt();  System.out.println(firstWins[x - 1][y - 1] ? "First" : "Second");  }  sc.close();  }  static void buildFirstWins() {  firstWins = new boolean[SIZE][SIZE];  for (int sum = 0; sum <= (SIZE - 1) \* 2; sum++) {  for (int r = 0; r <= sum; r++) {  int c = sum - r;  if (isInBoard(r, c) && !firstWins[r][c]) {  for (int i = 0; i < R\_OFFSETS.length; i++) {  int nextR = r + R\_OFFSETS[i];  int nextC = c + C\_OFFSETS[i];  if (isInBoard(nextR, nextC)) {  firstWins[nextR][nextC] = true;  }  }  }  }  }  }  static boolean isInBoard(int r, int c) {  return r >= 0 && r < SIZE && c >= 0 && c < SIZE;  }  } |

# Introduction to Nim Game

|  |
| --- |
| public class Solution {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int g = sc.nextInt();  for (int tc = 0; tc < g; tc++) {  int n = sc.nextInt();  int[] piles = new int[n];  for (int i = 0; i < piles.length; i++) {  piles[i] = sc.nextInt();  }  System.out.println(solve(piles));  }  sc.close();  }  static String solve(int[] piles) {  return Arrays.stream(piles).reduce((result, pile) -> result ^ pile).getAsInt() != 0 ? "First" : "Second";  }  } |

# Walking the Approximate Longest Path

|  |
| --- |
| public class Solution {  public static void main(String[] args) {  Scanner input = new Scanner(System.in);  int v = input.nextInt();  int e = input.nextInt();  Graph g = new Graph(v,e);  for(int i=0; i<e; i++){  g.addEdge(input.nextInt(),input.nextInt());  }  //System.out.println(g.toString());  ArrayList<Integer> path = new ArrayList<>();  int currentVertex = g.getMinVertex();  //System.out.println(currentVertex);  while(true){  path.add(currentVertex);  currentVertex = g.getNextVertex(currentVertex);  //System.out.println(currentVertex);  if(currentVertex == -1)break;  }  System.out.println(path.size());  for(int i=0; i<path.size(); i++){  System.out.print((path.get(i)+1)+" ");  }  }  public static class Graph{  int v;  int e;  int minV;  ArrayList<HashSet<Integer>> adjList;  boolean[] visited;  Graph(int v, int e){  this.v = v; this.e = e;  adjList = new ArrayList<>(v);  visited = new boolean[v];  for(int i=0; i<v; i++){  adjList.add(new HashSet<Integer>());  }  }  public void addEdge(int v1, int v2){  this.adjList.get(v1-1).add(v2-1);  this.adjList.get(v2-1).add(v1-1);  }  public int getMinVertex(){  int minVertex = 0;  int minValue = Integer.MAX\_VALUE;  for(int i=0; i<v;i++){  if(adjList.get(i).size() < minValue){  minValue = adjList.get(i).size();  minVertex = i;  }  }  return minVertex;  }  public int getNextVertex(int v){  this.visited[v] = true;  int minVertex = -1;  int minValue = Integer.MAX\_VALUE;  for(int i:this.adjList.get(v)){  if(!visited[i]){  if(this.adjList.get(i).size() < minValue){  minValue = this.adjList.get(i).size();  minVertex = i;  }  }  }  return minVertex;  }  public String toString(){  StringBuffer s = new StringBuffer("");  for(int i=0; i<v; i++){  s.append(i+": "+this.adjList.get(i)+"\n");  }  return s.toString();  }  }  } |

# Pseudo-Isomorphic Substrings

|  |
| --- |
| public class PseudoIsomorphicSubstrings {  public static class ST {  public static class Node {  private final int valuesFromRootCount;  private final Map<Integer, Edge> edges;  private Node slink;  public Node(final int valuesFromRootCount) {  this.valuesFromRootCount = valuesFromRootCount;  this.edges = new HashMap<>();  }  public int getValuesFromRootCount() {  return this.valuesFromRootCount;  }  public void setSlink(final Node slink) {  this.slink = slink;  }  public Map<Integer, Edge> getEdges() {  return this.edges;  }  public Node getSlink() {  return this.slink;  }  }  public static class Edge {  private final int suffixStartIndex;  private final int from;  private final int to;  private final Node next;  public Edge(final int suffixStartIndex, final int from, final int to, final Node next) {  this.suffixStartIndex = suffixStartIndex;  this.from = from;  this.to = to;  this.next = next;  }  public int getSuffixStartIndex() {  return this.suffixStartIndex;  }  public int getFrom() {  return this.from;  }  public int getTo() {  return this.to;  }  public Node getNext() {  return this.next;  }  public int getLength() {  return this.to - this.from;  }  }  public static class ActivePoint {  private final Node activeNode;  private final Integer activeEdgeFirstValue;  private final int activeLength;  public ActivePoint(final Node activeNode,  final Integer activeEdgeFirstValue,  final int activeLength) {  this.activeNode = activeNode;  this.activeEdgeFirstValue = activeEdgeFirstValue;  this.activeLength = activeLength;  }  private Edge getActiveEdge() {  return this.activeNode.getEdges().get(this.activeEdgeFirstValue);  }  private int getActiveValue(final int[] values) {  final int valueOnEdgeIndex = getActiveEdge().getFrom() + this.activeLength;  return getSuffixValue(getActiveEdge().getSuffixStartIndex(), values, valueOnEdgeIndex);  }  public int getSuffixValue(final int suffixStartIndex, final int[] values, final int valueIndex) {  return suffixStartIndex <= valueIndex - values[valueIndex] ? values[valueIndex] : 0;  }  public boolean pointsToActiveNode() {  return this.activeLength == 0;  }  public boolean activeNodeIs(final Node node) {  return this.activeNode == node;  }  public boolean activeNodeHasEdgeStartingWith(final int value) {  return this.activeNode.getEdges().containsKey(value);  }  public boolean activeNodeHasSlink() {  return this.activeNode.getSlink() != null;  }  public boolean pointsToOnActiveEdge(final int[] values, final int value) {  return getActiveValue(values) == value;  }  public boolean pointsToTheEndOfActiveEdge() {  return this.getActiveEdge().getLength() == this.activeLength;  }  public boolean pointsAfterTheEndOfActiveEdge() {  return this.getActiveEdge().getLength() < this.activeLength;  }  public ActivePoint moveToEdgeStartingWithAndByOne(final int value) {  return new ActivePoint(this.activeNode, value, 1);  }  public ActivePoint moveToNextNodeOfActiveEdge() {  return new ActivePoint(this.getActiveEdge().getNext(), null, 0);  }  public ActivePoint moveToSlink(final int[] values,  final int suffixStartIndex,  final int index) {  final int slinkIndex = suffixStartIndex + this.activeNode.getSlink().getValuesFromRootCount();  final int slinkValue = getSuffixValue(suffixStartIndex, values, slinkIndex);  final int slinkActiveLength = index - slinkIndex;  return new ActivePoint(this.activeNode.getSlink(),  slinkValue,  slinkActiveLength);  }  public ActivePoint moveTo(final Node node) {  return new ActivePoint(node, null, 0);  }  public ActivePoint moveTo(final Node node, final int activeLength) {  return new ActivePoint(node, 0, activeLength);  }  public ActivePoint moveByOneValue() {  return new ActivePoint(this.activeNode,  this.activeEdgeFirstValue,  this.activeLength + 1);  }  public ActivePoint moveToNextNodeOfActiveEdge(final int[] values,  final int suffixStartIndex) {  final int valueAtTheEndOfEdgeIndex = suffixStartIndex + this.activeNode.getValuesFromRootCount() + this.getActiveEdge()  .getLength();  final int valueAtTheEndOfEdge = getSuffixValue(suffixStartIndex, values, valueAtTheEndOfEdgeIndex);  return new ActivePoint(this.getActiveEdge().getNext(),  valueAtTheEndOfEdge,  this.activeLength - this.getActiveEdge().getLength());  }  public void addEdgeToActiveNode(final int value, final Edge edge) {  this.activeNode.getEdges().put(value, edge);  }  public Node splitActiveEdge(final int[] values,  final int suffixStartIndex,  final int index,  final int value) {  final Node newNode = new Node(this.activeNode.getValuesFromRootCount()  + this.activeLength);  final Edge activeEdgeToSplit = this.getActiveEdge();  final Edge splittedEdge = new Edge(activeEdgeToSplit.getSuffixStartIndex(), activeEdgeToSplit.getFrom(),  activeEdgeToSplit.getFrom() + this.activeLength,  newNode);  newNode.getEdges().put(getActiveValue(values),  new Edge(activeEdgeToSplit.getSuffixStartIndex(),  activeEdgeToSplit.getFrom() + this.activeLength,  activeEdgeToSplit.getTo(),  activeEdgeToSplit.getNext()));  newNode.getEdges().put(value, new Edge(suffixStartIndex, index, values.length, null));  this.activeNode.getEdges().put(this.activeEdgeFirstValue, splittedEdge);  return newNode;  }  public Node setSlinkTo(final Node previouslyAddedNodeOrAddedEdgeNode,  final Node node) {  if(previouslyAddedNodeOrAddedEdgeNode != null) {  previouslyAddedNodeOrAddedEdgeNode.setSlink(node);  }  return node;  }  public Node setSlinkToActiveNode(final Node previouslyAddedNodeOrAddedEdgeNode) {  return setSlinkTo(previouslyAddedNodeOrAddedEdgeNode, this.activeNode);  }  }  private final int[] values;  private final Node root;  private ActivePoint activePoint;  private int remainder;  private long totalLeavesCount;  private long leavesSum;  public ST(final int[] values) {  this.values = values;  this.root = new Node(0);  this.activePoint = new ActivePoint(this.root, null, 0);  this.remainder = 0;  this.totalLeavesCount = 0;  this.leavesSum = 0;  build();  }  private void build() {  for(int i = 0; i < this.values.length; i++) {  add(i);  }  }  private void add(final int index) {  this.remainder++;  boolean valueFoundInTheTree = false;  Node previouslyAddedNodeOrAddedEdgeNode = null;  while(!valueFoundInTheTree && this.remainder > 0) {  final int suffixStartIndex = index - this.remainder + 1;  final int valueToCheckExistanceInTree = this.activePoint.getSuffixValue(suffixStartIndex, this.values, index);  if(this.activePoint.pointsToActiveNode()) {  if(this.activePoint.activeNodeHasEdgeStartingWith(valueToCheckExistanceInTree)) {  activeNodeHasEdgeStartingWithValue(valueToCheckExistanceInTree,  previouslyAddedNodeOrAddedEdgeNode);  valueFoundInTheTree = true;  }  else {  if(this.activePoint.activeNodeIs(this.root)) {  rootNodeHasNotEdgeStartingWithValue(suffixStartIndex, index, valueToCheckExistanceInTree);  }  else {  previouslyAddedNodeOrAddedEdgeNode = internalNodeHasNotEdgeStartingWithValue(  suffixStartIndex,  index,  valueToCheckExistanceInTree,  previouslyAddedNodeOrAddedEdgeNode);  }  }  }  else {  if(this.activePoint.pointsToOnActiveEdge(this.values, valueToCheckExistanceInTree)) {  activeEdgeHasValue();  valueFoundInTheTree = true;  }  else {  if(this.activePoint.activeNodeIs(this.root)) {  previouslyAddedNodeOrAddedEdgeNode = edgeFromRootNodeHasNotValue(suffixStartIndex,  index,  valueToCheckExistanceInTree,  previouslyAddedNodeOrAddedEdgeNode);  }  else {  previouslyAddedNodeOrAddedEdgeNode = edgeFromInternalNodeHasNotValue(suffixStartIndex,  index,  valueToCheckExistanceInTree,  previouslyAddedNodeOrAddedEdgeNode);  }  }  }  }  this.leavesSum += this.totalLeavesCount;  System.out.println(this.leavesSum);  }  private void activeNodeHasEdgeStartingWithValue(final int value,  final Node previouslyAddedNodeOrAddedEdgeNode) {  this.activePoint.setSlinkToActiveNode(previouslyAddedNodeOrAddedEdgeNode);  this.activePoint = this.activePoint.moveToEdgeStartingWithAndByOne(value);  if(this.activePoint.pointsToTheEndOfActiveEdge()) {  this.activePoint = this.activePoint.moveToNextNodeOfActiveEdge();  }  }  private void rootNodeHasNotEdgeStartingWithValue(final int suffixStartIndex, final int index, final int value) {  this.activePoint.addEdgeToActiveNode(value, new Edge(suffixStartIndex, index, this.values.length, null));  this.totalLeavesCount++;  this.activePoint = this.activePoint.moveTo(this.root);  this.remainder--;  assert this.remainder == 0;  }  private Node internalNodeHasNotEdgeStartingWithValue(final int suffixStartIndex,  final int index,  final int value,  Node previouslyAddedNodeOrAddedEdgeNode) {  this.activePoint.addEdgeToActiveNode(value, new Edge(suffixStartIndex, index, this.values.length, null));  this.totalLeavesCount++;  previouslyAddedNodeOrAddedEdgeNode = this.activePoint.setSlinkToActiveNode(  previouslyAddedNodeOrAddedEdgeNode);  if(this.activePoint.activeNodeHasSlink()) {  this.activePoint = this.activePoint.moveToSlink(this.values, suffixStartIndex + 1, index);  }  else {  this.activePoint = this.activePoint.moveTo(this.root, this.remainder - 2);  }  this.activePoint = walkDown(suffixStartIndex + 1);  this.remainder--;  return previouslyAddedNodeOrAddedEdgeNode;  }  private void activeEdgeHasValue() {  this.activePoint = this.activePoint.moveByOneValue();  if(this.activePoint.pointsToTheEndOfActiveEdge()) {  this.activePoint = this.activePoint.moveToNextNodeOfActiveEdge();  }  }  private Node edgeFromRootNodeHasNotValue(final int suffixStartIndex,  final int index,  final int value,  Node previouslyAddedNodeOrAddedEdgeNode) {  final Node newNode = this.activePoint.splitActiveEdge(this.values,  suffixStartIndex,  index,  value);  this.totalLeavesCount++;  previouslyAddedNodeOrAddedEdgeNode = this.activePoint.setSlinkTo(previouslyAddedNodeOrAddedEdgeNode,  newNode);  this.activePoint = this.activePoint.moveTo(this.root, this.remainder - 2);  this.activePoint = walkDown(suffixStartIndex + 1);  this.remainder--;  return previouslyAddedNodeOrAddedEdgeNode;  }  private Node edgeFromInternalNodeHasNotValue(final int suffixStartIndex,  final int index,  final int value,  Node previouslyAddedNodeOrAddedEdgeNode) {  final Node newNode = this.activePoint.splitActiveEdge(this.values,  suffixStartIndex,  index,  value);  this.totalLeavesCount++;  previouslyAddedNodeOrAddedEdgeNode = this.activePoint.setSlinkTo(previouslyAddedNodeOrAddedEdgeNode,  newNode);  if(this.activePoint.activeNodeHasSlink()) {  this.activePoint = this.activePoint.moveToSlink(this.values, suffixStartIndex + 1, index);  }  else {  this.activePoint = this.activePoint.moveTo(this.root, this.remainder - 2);  }  this.activePoint = walkDown(suffixStartIndex + 1);  this.remainder--;  return previouslyAddedNodeOrAddedEdgeNode;  }  private ActivePoint walkDown(final int suffixStartIndex) {  while(!this.activePoint.pointsToActiveNode()  && (this.activePoint.pointsToTheEndOfActiveEdge() || this.activePoint.pointsAfterTheEndOfActiveEdge())) {  if(this.activePoint.pointsAfterTheEndOfActiveEdge()) {  this.activePoint = this.activePoint.moveToNextNodeOfActiveEdge(this.values, suffixStartIndex);  }  else {  this.activePoint = this.activePoint.moveToNextNodeOfActiveEdge();  }  }  return this.activePoint;  }  }  public static void main(final String[] args) {  final Scanner in = new Scanner(System.in);  final String line = in.next();  final char[] charsLine = line.toCharArray();  final int[] indexesMinusprevOccurenceIndexes = new int[line.length()];  final Map<Character, Integer> prev = new HashMap<>();  for(int i = 0; i < line.length(); i++) {  if(prev.containsKey(charsLine[i])) {  indexesMinusprevOccurenceIndexes[i] = i - prev.get(charsLine[i]);  }  prev.put(charsLine[i], i);  }  new ST(indexesMinusprevOccurenceIndexes);  }  } |

# Insertion Sort Advanced Analysis

|  |
| --- |
| **public** **class** InsertionSortAdvancedAnalysis {  **static** **int** getSum(**int** BITree[], **int** index) {  **int** sum = 0;  **while** (index > 0) {  sum = sum + BITree[index];  index = index - (index & (-index));  }  **return** sum;  }  **static** **void** updateBIT(**int** BITree[], **int** n, **int** index, **int** val) {  **while** (index <= n) {  BITree[index] = BITree[index] + val;  index = index + (index & (-index));  }  }  **static** **long** getInvCount(**int** arr[], **int** n) {  **long** invcount = 0;  **int** maxElement = 0;  **for** (**int** i = 0; i < n; i++) {  **if** (maxElement < arr[i])  maxElement = arr[i];  }  **int** BIT[] = **new** **int**[maxElement + 1];  **for** (**int** i = 1; i <= maxElement; i++) {  BIT[i] = 0;  }  **for** (**int** i = n - 1; i >= 0; i--) {  invcount = invcount + *getSum*(BIT, arr[i] - 1);  *updateBIT*(BIT, maxElement, arr[i], 1);  }  **return** invcount;  }  **public** **static** **void** main(String[] args) {  Scanner s = **new** Scanner(System.***in***);  **int** totalTime = s.nextInt();  **long** inversion[] = **new** **long**[totalTime];  **int** length = 0;  **for** (**int** i = 1; i <= totalTime; i++) {  **int** formArray = s.nextInt();  **int** data[] = **new** **int**[formArray];  length = data.length;  **for** (**int** j = 0; j < formArray; j++) {  data[j] = s.nextInt();  }  inversion[i - 1] = *getInvCount*(data, length);  } // outer loop  **for** (**long** k : inversion)  System.***out***.println(k);  }  } |

# Hackerland Radio Transmitters

|  |
| --- |
| **public** **class** HackerlandRadioTransmitters {  **public** **static** **void** main(String[] args) {  Scanner in = **new** Scanner(System.***in***);  **int** n = in.nextInt();  **int** k = in.nextInt();  **int**[] x = **new** **int**[n];  **for** (**int** x\_i = 0; x\_i < n; x\_i++) {  x[x\_i] = in.nextInt();  }  Arrays.*sort*(x);  **int** numOfTransmitters = 0;  **int** i = 0;  **while** (i < n) {  numOfTransmitters++;  **int** loc = x[i] + k;  **while** (i < n && x[i] <= loc)  i++;  loc = x[--i] + k;  **while** (i < n && x[i] <= loc)  i++;  }  System.***out***.println(numOfTransmitters);  }  } |

# Gridland Metro

|  |
| --- |
| **public** **class** GridlandMetro {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  **long** n=sc.nextLong();  **long** m= sc.nextLong();  **int** k=sc.nextInt();  HashMap<Integer,TreeMap<Integer,Integer>> tracks = **new** HashMap();  **for**(**int** i=0;i<k;i++){  **int** r= sc.nextInt();  **int** c1=sc.nextInt();  **int** c2=sc.nextInt();  **if**(!tracks.containsKey(r)){  TreeMap<Integer,Integer> temp = **new** TreeMap<Integer, Integer>();  temp.put(c1, c2);  tracks.put(r, temp);  }  **else**{  TreeMap<Integer,Integer> temp = tracks.get(r);  **if**(temp.containsKey(c1)){  temp.put(c1, Math.*max*(c2, temp.get(c1)));  } **else**  temp.put(c1, c2);  }  }  **long** OccupiedCells= 0L;  Set<Entry<Integer, TreeMap<Integer,Integer>>> parentSet = tracks.entrySet();  Iterator<Entry<Integer, TreeMap<Integer,Integer>>> itrParent = parentSet.iterator();  **while**(itrParent.hasNext()){  Entry<Integer, TreeMap<Integer,Integer>> parentEntry = itrParent.next();  TreeMap<Integer,Integer> temp = parentEntry.getValue();  Set<Entry<Integer, Integer>> set = temp.entrySet();  Iterator<Entry<Integer, Integer>> itr = set.iterator();  **int** CurrentLowerBound=0;  **int** CurrentUpperBound=0;  **while**(itr.hasNext()){  Entry<Integer, Integer> entry = itr.next();  **int** c1=entry.getKey();  **int** c2=entry.getValue();  **if**(CurrentLowerBound==0 && CurrentUpperBound==0){  CurrentLowerBound=c1;  CurrentUpperBound=c2;  } **else**{  **if**(c1>CurrentUpperBound){  OccupiedCells=OccupiedCells+CurrentUpperBound-CurrentLowerBound+1;  CurrentLowerBound=c1;  CurrentUpperBound=c2;  } **else** **if**(c1==CurrentUpperBound){  CurrentUpperBound=c2;  } **else**{  **if**(c2>CurrentUpperBound){    CurrentUpperBound=c2;  }  }  }  }  OccupiedCells=OccupiedCells+CurrentUpperBound-CurrentLowerBound+1;  }  **long** totalCels = n\*m;  System.***out***.println(totalCels-OccupiedCells);  }  } |

# Maximum Subarray Sum

|  |
| --- |
| **public** **class** MaximumSubarraySum {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  **int** q = sc.nextInt();  **long**[] res = **new** **long**[q];  **for** (**int** i = 0; i < q; i++) {  **int** n = sc.nextInt();  **long** m = sc.nextLong();  **long**[] arr = **new** **long**[n];  **for** (**int** j = 0; j < n; j++) {  arr[j] = sc.nextLong();  }  res[i] = *getMaxSum*(n, m, arr);  }  **for** (**int** i = 0; i < q; i++) {  System.***out***.println(res[i]);  }  }  **private** **static** **long** getMaxSum(**int** n, **long** m, **long**[] arr) {  **long** maxSum = 0;  TreeSet<Long> prefix = **new** TreeSet<Long>();  **long** currentSum = 0;  **for** (**int** i = 0; i < n; i++) {  currentSum = (currentSum + arr[i] % m) % m;  SortedSet<Long> set = prefix.tailSet(currentSum + 1);  Iterator<Long> itr = set.iterator();  **if** (itr.hasNext()) {  maxSum = Math.*max*(maxSum, (currentSum - itr.next() + m) % m);  }  maxSum = Math.*max*(maxSum, currentSum);  prefix.add(currentSum);  }  **return** maxSum;  }  } |