Exercises and Homework

java.util Methods for Arrays

fill(A, x)

copyOf(A, n)

copyOfRange(A, s, t):

toString(A)

sort(A):

binarySearch(A, x)

|  |  |  |
| --- | --- | --- |
| 1 | R-3.1 | Give the next five pseudorandom numbers generated by the process described on page 113, with a = 12, b = 5, and n = 100, and 92 as the seed for cur.  See page 113   1. **9** 2. **13** 3. **61** 4. **37** 5. **49** |
| 2 | R-3.2 | Write a Java method that repeatedly selects and removes a random entry from an array until the array holds no more entries.  import java.util.Random;  import java.util.Arrays;  public class RandomArrayRemoval {  public static void main(String[] args) {  String[] countries = {"Brazil", "France", "Germany", "Canada", "Italy", "England"};  int size = countries.length;  // إنشاء مولد أرقام عشوائية  Random random = new Random();  // كرر العملية حتى تكون المصفوفة خالية من المزيد من العناصر  while (size > 0) {  // إنشاء فهرس عشوائي  int randomIndex = random.nextInt(size);  // قم بطباعة وإزالة العنصر العشوائي  System.out.println("تمت الإزالة: " + countries[randomIndex]);  countries[randomIndex] = countries[size - 1];  size--;  }  }  } |
| 3 | R-3.3 | Explain the changes that would have to be made to the program of Code Fragment 3.8 so that it could perform the Caesar cipher for messages that are written in an alphabet-based language other than English, such as Greek, Russian, or Hebrew.   1. **تعديل الأبجدية**: قم بتعديل الأبجدية المستخدمة في البرنامج لتتناسب مع اللغة المستهدفة. على سبيل المثال، إذا كنت ترغب في استخدام اللغة اليونانية، فعليك استبدال الأبجدية الإنجليزية بالأبجدية اليونانية. 2. **تعديل القيمة الثابتة للتحويل**: في شيفرة قيصر، يتم تحديد القيمة الثابتة للتحويل (عدد الحروف المنزلة) بواسطة معامل الانتقال. يجب تعديل هذه القيمة لتتناسب مع اللغة المستهدفة. على سبيل المثال، إذا كنت تستخدم اللغة الروسية، قد تحتاج إلى تحديد قيمة انتقال مختلفة. 3. **تعديل الحسابات الرياضية**: قد تحتاج إلى تعديل الحسابات الرياضية المستخدمة في البرنامج لتتناسب مع الأبجدية والقيمة الثابتة للتحويل. 4. **اختبار البرنامج**: بعد التعديلات، قم بتشغيل البرنامج باستخدام نصوص من اللغة المستهدفة للتحقق من صحة الشيفرة. |
| 4 | R-3.4 | The TicTacToe class of Code Fragments 3.9 and 3.10 has a flaw, in that it allows a player to place a mark even after the game has already been won by someone. Modify the class so that the putMark method throws an IllegalStateException in that case  public int findDuplicate(int[] nums) {  // The "tortoise and hare" step. We start at the end of the array and try  // to find an intersection point in the cycle.  int tortoise = nums[0];  int hare = nums[0];  do {  tortoise = nums[tortoise];  hare = nums[nums[hare]];  } while (tortoise != hare);  // Start up another pointer from the end of the array and march it forward  // until it hits the pointer inside the array.  int ptr1 = nums[0];  int ptr2 = tortoise;    while (ptr1 != ptr2) {  ptr1 = nums[ptr1];  ptr2 = nums[ptr2];  }  // Since there are multiple values pointing to the first node in the cycle,  // its position is a duplicate in our array.  return ptr1;  } |
| 5 | R-3.13 | What is the difference between a shallow equality test and a deep equality test between two Java arrays, A and B, if they are one-dimensional arrays of type int? What if the arrays are two-dimensional arrays of type int?  public class TicTacToe {  private char[][] board;  private char currentPlayerMark;  private boolean gameWon;  public TicTacToe() {  board = new char[3][3];  currentPlayerMark = 'X';  gameWon = false;  initializeBoard();  }  // Initialize the board with empty cells  private void initializeBoard() {  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++) {  board[i][j] = '-';  }  }  }  // Other methods for drawing the board, getting player moves, checking for win, etc.  public void putMark(int row, int col) {  if (gameWon) {  throw new IllegalStateException("تم الفوز باللعبة بالفعل. لا يمكن وضع علامة.");  }  if (board[row][col] != '-') {  throw new IllegalArgumentException("الخلية محجوزة بالفعل. اختر خلية فارغة.");  }  board[row][col] = currentPlayerMark;  togglePlayer();  checkForWin();  }  private void togglePlayer() {  currentPlayerMark = (currentPlayerMark == 'X') ? 'O' : 'X';  }  private void checkForWin() {  // تحقق من الفوز وتعيين gameWon إلى true إذا تم الفوز  // ...  }  } |
| 6 | R-3.14 | Give three different examples of a single Java statement that assigns variable, backup, to a new array with copies of all int entries of an existing array, original.  Using Arrays.copyOf:  int[] originalArray1 = {1, 2, 3, 4, 5};  int[] backup1 = Arrays.copyOf(originalArray1,  originalArray1.length);  Using clone():  int[] originalArray2 = {6, 7, 8, 9, 10};  int[] backup2 = originalArray2.clone();  Using IntStream:  int[] originalArray3 = {11, 12, 13, 14, 15};  int[] backup3 = IntStream.of(originalArray3).toArray(); |
| 7 | C-3.17 | Let A be an array of size n ≥ 2 containing integers from 1 to n−1 inclusive, one of which is repeated. Describe an algorithm for finding the integer in A that is repeated.  public int findDuplicate(int[] nums) {  // The "tortoise and hare" step. We start at the end of the array and try  // to find an intersection point in the cycle.  int tortoise = nums[0];  int hare = nums[0];  do {  tortoise = nums[tortoise];  hare = nums[nums[hare]];  } while (tortoise != hare);  // Start up another pointer from the end of the array and march it forward  // until it hits the pointer inside the array.  int ptr1 = nums[0];  int ptr2 = tortoise;    while (ptr1 != ptr2) {  ptr1 = nums[ptr1];  ptr2 = nums[ptr2];  }  // Since there are multiple values pointing to the first node in the cycle,  // its position is a duplicate in our array.  return ptr1;  }  // Example usage  int[] A = {1, 2, 3, 4, 4};  int repeatedInteger = findDuplicate(A);  System.out.println("The repeated integer in array A is: " + repeatedInteger); |
| 8 | C-3.18 | Let B be an array of size n ≥ 6 containing integers from 1 to n−5 inclusive, five of which are repeated. Describe an algorithm for finding the five integers in B that are repeated.  **Algorithm:** import java.util.ArrayList;  import java.util.List;  public class FindRepeatedIntegers {  public static List<Integer> findRepeatedIntegers(int[] B) {  List<Integer> result = new ArrayList<>();  int tortoise = B[0];  int hare = B[0];  // Move the hare twice as fast as the tortoise  while (true) {  tortoise = B[tortoise];  hare = B[B[hare]];  if (tortoise == hare) {  break;  }  }  // Reset the tortoise to the start  tortoise = B[0];  // Move both pointers at the same speed until they meet  while (tortoise != hare) {  tortoise = B[tortoise];  hare = B[hare];  }  // The meeting point is part of the cycle  int start = tortoise;  // Find the length of the cycle  int length = 0;  do {  tortoise = B[tortoise];  length++;  } while (tortoise != start);  // Move the hare ahead by the length of the cycle  hare = B[0];  for (int i = 0; i < length; i++) {  hare = B[hare];  }  // Move both pointers until they meet again (at the start of the cycle)  while (tortoise != hare) {  tortoise = B[tortoise];  hare = B[hare];  }  // Add the repeated integers to the result  for (int i = 0; i < 5; i++) {  result.add(tortoise);  tortoise = B[tortoise];  }  return result;  }  public static void main(String[] args) {  int[] B = {1, 2, 3, 4, 5, 2, 3, 4, 5};  List<Integer> repeatedIntegers = findRepeatedIntegers(B);  System.out.println("The repeated integers in array B are: " + repeatedIntegers);  }  }   1. Create a set S to store the distinct elements encountered so far. Initialize S to an empty set. 2. Iterate through the array B: a. For each element b in B: i. If b is not in S, add b to S. This indicates that the element b has been seen once. ii. If b is already in S, then b is a repeated element. Add b to a list of repeated elements. 3. Since there are five repeated elements, continue iterating through B until you find five distinct elements that are repeated. 4. The list of repeated elements contains the five repeated integers in B.   **Analysis:**  Time Complexity: O(n), where n is the size of the array B. This is because the algorithm iterates through the array B only once, and each operation takes constant time. |
| 9 | C-3.19 | Give Java code for performing add(e) and remove(i) methods for the Scoreboard class, as in Code Fragments 3.3 and 3.4, except this time, don’t maintain the game entries in order. Assume that we still need to keep n entries stored in indices 0 to n−1. You should be able to implement the methods without using any loops, so that the number of steps they perform does not depend on n.  import java.util.Arrays;  class GameEntry {  private String name;  private int score;  public GameEntry(String name, int score) {  this.name = name;  this.score = score;  }  public String getName() {  return name;  }  public int getScore() {  return score;  }  @Override  public String toString() {  return "(" + name + ", " + score + ")";  }  }  class Scoreboard {  private GameEntry[] board;  private int n;  public Scoreboard(int capacity) {  board = new GameEntry[capacity];  n = 0;  }  public void add(GameEntry entry) {  if (n < board.length) {  board[n] = entry;  n++;  } else {  int minScoreIndex = 0;  for (int i = 1; i < board.length; i++) {  if (board[i].getScore() < board[minScoreIndex].getScore()) {  minScoreIndex = i;  }  }  if (entry.getScore() > board[minScoreIndex].getScore()) {  board[minScoreIndex] = entry;  }  }  Arrays.sort(board, 0, n, (a, b) -> Integer.compare(b.getScore(), a.getScore()));  }  public void remove(int i) {  if (i >= 0 && i < n) {  board[i] = null;  n--;  Arrays.sort(board, 0, n, (a, b) -> Integer.compare(b.getScore(), a.getScore()));  }  }  @Override  public String toString() {  StringBuilder sb = new StringBuilder();  for (int i = 0; i < n; i++) {  if (board[i] != null) {  sb.append(board[i]).append("\n");  }  }  return sb.toString();  }  public static void main(String[] args) {  Scoreboard scoreboard = new Scoreboard(5);  scoreboard.add(new GameEntry("Alice", 100));  scoreboard.add(new GameEntry("Bob", 80));  scoreboard.add(new GameEntry("Charlie", 120));  scoreboard.add(new GameEntry("David", 90));  scoreboard.add(new GameEntry("Eve", 110));  System.out.println(scoreboard);  scoreboard.remove(2);  System.out.println("\nAfter removing entry at index 2:");  System.out.println(scoreboard);  }  } |
| 10 | C-3.20 | Give examples of values for a and b in the pseudorandom generator given on page 113 of this chapter such that the result is not very random looking, for n = 1000.   1. Linear Congruential Generator (LCG):    * a = 1, b = 0    * a = 1, b = 1 2. Multiplicative Congruential Generator (MCG):    * a = 1, b = 0    * a = 1, b = 1 3. Mixed Congruential Generator:    * a = 1, b = 0    * a = 1, b = 1 4. Other Pseudorandom Generators:    * a = 1, b = 0    * a = 1, b = 1    * a = 1, b = 2    * a = 1, b = 3    * a = 1, b = 4    * a = 1, b = 5    * a = 1, b = 6    * a = 1, b = 7    * a = 1, b = 8    * a = 1, b = 9    * a = 1, b = 10    * a = 1, b = 11    * a = 1, b = 12    * a = 1, b = 13 |
| 11 | C-3.21 | Suppose you are given an array, A, containing 100 integers that were generated using the method r.nextInt(10), where r is an object of type java.util.Random. Let x denote the product of the integers in A. There is a single number that x will equal with probability at least 0.99. What is that number and what is a formula describing the probability that x is equal to that number? |
| 12 | C-3.22 | Write a method, shuffle(A), that rearranges the elements of array A so that every possible ordering is equally likely. You may rely on the nextInt(n) method of the java.util.Random  class, which returns a random number between 0 and n−1 inclusive.  import java.util.Random;  public class ShuffleArray {  public static void shuffle(int[] A) {  Random random = new Random();  for (int i = A.length - 1; i > 0; i--) {  int j = random.nextInt(i + 1);  swap(A, i, j);  }  }  private static void swap(int[] A, int i, int j) {  int temp = A[i];  A[i] = A[j];  A[j] = temp;  }  public static void main(String[] args) {  int[] A = {1, 2, 3, 4, 5};  shuffle(A);  System.out.println("Shuffled array:");  for (int num : A) {  System.out.print(num + " ");  }  }  } |
| 13 | C-3.23 | Suppose you are designing a multiplayer game that has n ≥ 1000 players, numbered 1 to n, interacting in an enchanted forest. The winner of this game is the first player who can meet all the other players at least once (ties are allowed). Assuming that there is a method meet(i, j), which is called each time a player i meets a player j (with i 6= j), describe a way to keep track of the pairs of meeting players and who is the winner.   1. **توليد الأعداد الصحيحة**:    * نعطى مصفوفة A تحتوي على 100 عدد صحيح تم توليده باستخدام الطريقة r.nextInt(10)، حيث r هو كائن من نوع java.util.Random.    * الطريقة r.nextInt(10) تولد أعداد صحيحة من 0 إلى 9 (شاملاً). 2. **منتج الأعداد الصحيحة**:    * لنعتبر x منتج جميع الأعداد في المصفوفة A. 3. **احتمال أن**x**يكون مساويًا لرقم معين**:    * نريد أن نجد رقمًا واحدًا حيث يكون x مساويًا له بمعدل لا يقل عن 0.99. 4. **التحليل**:    * نظرًا لأن الطريقة تولد أعدادًا من 0 إلى 9، الطريقة الوحيدة ليكون المنتج x يساوي صفر هي إذا كان أحد الأعداد في A يساوي صفر.    * لذلك، الرقم الذي نبحث عنه هو 0. 5. **الصيغة لاحتمالية الحدوث**:    * الاحتمال المكمل أن لا يكون أي من الأعداد في A يساوي صفر هو: [ P(\text{على الأقل واحد صفر}) = 1 - P(\text{لا يوجد أصفار}) ]    * نظرًا لأن كل عدد لديه فرصة (\frac{1}{10}) ليكون صفرًا: [ P(\text{لا يوجد أصفار}) = \left(\frac{9}{10}\right)^{100} ]    * لذلك: [ P(x = 0) = 1 - P(\text{لا يوجد أصفار}) = 1 - \left(\frac{9}{10}\right)^{100} ] 6. **النتيجة**:    * الاحتمال أن x يكون مساويًا لـ 0 تقريبًا 0.999973.   لذا، الرقم الذي نبحث عنه هو 0، والصيغة التي تصف احتمالية أن x يكون مساويًا لـ 0 هي (1 - \left(\frac{9}{10}\right)^{100}). 😊  Here's a strategy to track pairs of meeting players and determine the  4. Handling Ties: |
| 14 | C-3.24 | Write a Java method that takes two three-dimensional integer arrays and adds them componentwise.  public int[][][] add3DArrays(int[][][] array1, int[][][] array2) {  if (array1.length != array2.length ||  array1[0].length != array2[0].length ||  array1[0][0].length != array2[0][0].length) {  throw new IllegalArgumentException("يجب أن تكون الأبعاد متساوية لكلا المصفوفتين");  }  int[][][] result = new int[array1.length][array1[0].length][array1[0][0].length];  for (int i = 0; i < array1.length; i++) {  for (int j = 0; j < array1[i].length; j++) {  for (int k = 0; k < array1[i][j].length; k++) {  result[i][j][k] = array1[i][j][k] + array2[i][j][k];  }  }  }  return result;  }  } |