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  9  13  61  37  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.  public static void removeRandomEntries(int[] array) {  Random random = new Random();  while (array.length > 0) {  // Generate a random index within the array's current bounds  int index = random.nextInt(array.length);  // Remove the entry at the randomly selected index  int removedEntry = array[index];  // Shift the remaining elements to fill the gap created by the removal  for (int i = index; i < array.length - 1; i++) {  array[i] = array[i + 1];  }  // Decrement the array's effective length  array[array.length - 1] = 0;  array = Arrays.copyOf(array, array.length - 1);  }  } |
| 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.  **Character Set Support**:  **Modular Arithmetic Adjustment**:  **Case Handling**:  **Non-Alphabetic Characters**: |
| 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 void putMark(int i, int j) throws IllegalArgumentException {  if (winner() != 0) {  throw new IllegalStateException("Game has already been won. No further moves allowed.");  }  if ((i < 0) || (i > 2) || (j < 0) || (j > 2)) {  throw new IllegalArgumentException("Invalid board position");  }  if (board[i][j] != EMPTY) {  throw new IllegalArgumentException("Board position occupied");  }  board[i][j] = player; // place the mark for the current player  player = -player; // switch players (uses fact that O = -X)  } |
| 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?  **One-Dimensional Arrays (int[] A, B)**  **Shallow Equality (A == B)**   * This checks whether A and B refer to the same memory location (i.e., they are the same object).  ****Deep Equality (****Arrays.equals(A, B)****)****  * This checks whether A and B have the same length and corresponding elements are equal.  ****Two-Dimensional Arrays (****int[][] A, B****)********Shallow Equality (****A == B****)****  * Similar to one-dimensional arrays, this checks whether A and B refer to the same memory location.  ****Deep Equality (****Arrays.deepEquals(A, B)****)****  * Since A and B are arrays of arrays, Arrays.equals(A, B) only checks if their top-level references are equal, not their contents. * Arrays.deepEquals(A, B) is needed for actual content comparison. |
| 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.  backup = original.clone( );  int[ ] temp = Arrays.copyOf(original, n);  **public** **static** **void** arraycopy(      Object src\_array, **int** src\_Pos,Object dest\_array, **int** dest\_Pos, **int** length )  System.arraycopy(src\_array, 0, dest\_array, 0,19); |
| 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.  def find\_repeated\_element(B):  distinct\_elements = set()  for b in B:  if b in distinct\_elements:  return b  else:  distinct\_elements.add(b)  return None |
| 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.  def find\_five\_repeated\_elements(B):  seen = set()  repeated = []  for b in B:  if b in seen:  if b not in repeated:  repeated.append(b)  if len(repeated) == 5: # Stop once we find five repeated numbers  break  else:  seen.add(b)  return repeated  # Example usage  B = [1, 2, 3, 4, 5, 6, 7, 8, 4, 6, 2, 7, 3] # Example array  print(find\_five\_repeated\_elements(B)) # Output: [4, 6, 2, 7, 3]  **Algorithm:**   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.  Space Complexity: O(n-5), where n-5 is the size of the set S. This is because the set S stores a maximum of n-5 distinct elements. |
| 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.  public class Scoreboard {  private int numEntries = 0; // Current number of entries  private GameEntry[] board; // Array of game scores  public Scoreboard(int capacity) {  board = new GameEntry[capacity]; // Initialize array with given capacity  }  /\*\* Adds a game entry without maintaining order \*/  public void add(GameEntry e) {  if (numEntries < board.length) {  board[numEntries] = e; // Place at the next available index  numEntries++; // Increment count  } else {  board[numEntries - 1] = e; // Replace last entry if full  }  }  /\*\* Removes an entry at index i without shifting elements \*/  public GameEntry remove(int i) throws IndexOutOfBoundsException {  if (i < 0 || i >= numEntries) {  throw new IndexOutOfBoundsException("Invalid index: " + i);  }  GameEntry removed = board[i]; // Store the removed entry  board[i] = board[numEntries - 1]; // Replace with the last entry  board[numEntries - 1] = null; // Nullify last position  numEntries--; // Decrement count  return removed;  }  /\*\* Displays the scoreboard \*/  public void display() {  for (int i = 0; i < numEntries; i++) {  System.out.println(board[i]);  }  }  } |
| 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.  next=(a⋅cur+b)modn  **a=37,b=17** for n=1000n = 1000n=1000 |
| 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?   * The number that x equals with probability **at least 0.99** is **0**. * The probability formula is: P(x=0)=1−(910)100P(x = 0) = 1 - \left(\frac{9}{10}\right)^{100}P(x=0)=1−(109​)100 * Since this probability is **greater than 0.99**, x = 0 almost always. |
| 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 r = new Random(); // Random number generator    for (int i = A.length - 1; i > 0; i--) {  int j = r.nextInt(i + 1); // Random index from 0 to i  swap(A, i, j); // Swap elements  }  }  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, 6, 7, 8, 9, 10};    System.out.println("Before shuffle:");  for (int num : A) System.out.print(num + " ");    shuffle(A);    System.out.println("\nAfter shuffle:");  for (int num : A) System.out.print(num + " ");  }  }  public static void shuffle(int[] A) {  Random rnd = new Random();  for (int i = A.length - 1; i > 0; i--) {  // Swap the current element with a randomly chosen element from the remaining array  int j = rnd.nextInt(i + 1);  int temp = A[i];  A[i] = A[j];  A[j] = temp;  }  } |
| 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.  **An array of sets:**   * Maintain an array meetings, where meetings[i] is a **set** that stores all players that **player iii has met**. * **Data structure:**   Set<Integer>[] meetings = new HashSet[n + 1]; // Index 1 to n   *  **Why?**   + HashSet allows **O(1) insertion and lookup**.   + Easily check if a player has met all others (meetings[i].size() == n - 1).   Here's a strategy to track pairs of meeting players and determine the winner(s):  1. Data Structure for Tracking Meetings:   * Bit Array: Employ a 2D boolean array meetings of size n x n. Set meetings[i][j] = true when players i and j meet, indicating a meeting has occurred. * Alternative: For extremely large n, consider a Bit Set for memory efficiency.   2. Tracking Meetings within meet(i, j):   * When meet(i, j) is called:   + Set both meetings[i][j] and meetings[j][i] to true (meetings are bidirectional).   + Increment counters for both players:     - meetingCount[i]++     - meetingCount[j]++   3. Determining Winners:   * After each meet(i, j) call:   + Check if either player i or j has met all other players:     - If meetingCount[i] == n-1, player i has met everyone and is a winner.     - Similarly, if meetingCount[j] == n-1, player j is a winner.   4. Handling Ties:   * Maintain a Set<Integer> called winners to track distinct winners. * Whenever a potential winner is found, add their ID to winners.   5. Code Structure (Example in Java):  Java  // Data structures  boolean[][] meetings = new boolean[n][n];  int[] meetingCount = new int[n];  Set<Integer> winners = new HashSet<>();  void meet(int i, int j) {  meetings[i][j] = meetings[j][i] = true;  meetingCount[i]++;  meetingCount[j]++;  if (meetingCount[i] == n - 1) {  winners.add(i);  // Handle player i's victory (e.g., announce win, mark as finished)  }  if (meetingCount[j] == n - 1) {  winners.add(j);  // Handle player j's victory (e.g., announce win, mark as finished)  }  }  Use code with caution. [Learn more](https://bard.google.com/faq#coding)  content\_copy  Additional Considerations:   * Game Ending: Decide when the game ends (e.g., when a certain number of winners emerge or after a time limit). * Winner Announcement: Appropriately announce the winner(s) based on the game's design. * Memory Optimization: For extremely large n, consider memory-optimized data structures like Bit Sets for storing meeting information. |
| 14 | C-3.24 | Write a Java method that takes two three-dimensional integer arrays and adds them componentwise.  public class ArrayAddition {  public static int[][][] add3DArrays(int[][][] A, int[][][] B) {  int x = A.length; // Number of 2D slices  int y = A[0].length; // Number of rows  int z = A[0][0].length; // Number of columns    int[][][] result = new int[x][y][z]; // Result array    for (int i = 0; i < x; i++) {  for (int j = 0; j < y; j++) {  for (int k = 0; k < z; k++) {  result[i][j][k] = A[i][j][k] + B[i][j][k]; // Component-wise addition  }  }  }  return result;  }  // Example usage  public static void main(String[] args) {  int[][][] A = {  {{1, 2}, {3, 4}},  {{5, 6}, {7, 8}}  };    int[][][] B = {  {{9, 8}, {7, 6}},  {{5, 4}, {3, 2}}  };  int[][][] sum = add3DArrays(A, B);  // Print result  for (int[][] matrix : sum) {  for (int[] row : matrix) {  for (int val : row) {  System.out.print(val + " ");  }  System.out.println();  }  System.out.println();  }  }  }  public static int[][] addThreeDimensionalArrays(int[][][] array1, int[][][] array2) {  // Check if the arrays have the same dimensions  if (array1.length != array2.length || array1[0].length != array2[0].length || array1[0][0].length != array2[0][0].length) {  throw new IllegalArgumentException("Arrays must have the same dimensions");  }  // Create a new array to store the result  int[][][] result = new int[array1.length][array1[0].length][array1[0][0].length];  // Add the corresponding elements of the two arrays  for (int i = 0; i < result.length; i++) {  for (int j = 0; j < result[0].length; j++) {  for (int k = 0; k < result[0][0].length; k++) {  result[i][j][k] = array1[i][j][k] + array2[i][j][k];  }  }  }  return result;  } |