To implement the shuffle(A) method that rearranges the elements of array A in such a way that each possible ordering is equally likely, we can use **Fisher-Yates shuffle** (also known as **Knuth shuffle**). This is an efficient algorithm that ensures each permutation of the array is equally likely and runs in **O(n)** time.

**Algorithm Overview (Fisher-Yates Shuffle)**

1. Start from the last element of the array and move towards the first.
2. For each element i, generate a random index j from the range [0, i].
3. Swap the elements at index i and index j.

This ensures that each element has an equal chance of appearing in any position in the array.

**Java Code Implementation:**

import java.util.Random;

public class ShuffleArray {

// Method to shuffle the array A

public static void shuffle(int[] A) {

Random rand = new Random(); // Create a Random object for generating random indices

// Traverse the array from the last element to the second element

for (int i = A.length - 1; i > 0; i--) {

// Generate a random index from 0 to i

int j = rand.nextInt(i + 1); // nextInt(i + 1) gives a random number between 0 and i

// Swap the elements at index i and index j

int temp = A[i];

A[i] = A[j];

A[j] = temp;

}

}

// Method to print the array (for testing purposes)

public static void printArray(int[] A) {

for (int i : A) {

System.out.print(i + " ");

}

System.out.println();

}

// Test the shuffle method

public static void main(String[] args) {

int[] A = {1, 2, 3, 4, 5};

System.out.println("Original Array:");

printArray(A);

shuffle(A); // Shuffle the array

System.out.println("Shuffled Array:");

printArray(A);

}

}

**Explanation of Code:**

1. **Random Object:**
   * We create a Random object rand to generate random integers.
2. **Loop through Array:**
   * The loop starts from the last element (A.length - 1) and moves backwards towards the first element.
3. **Generate Random Index:**
   * For each index i, we generate a random index j in the range [0, i] using rand.nextInt(i + 1). This ensures that all elements have an equal chance of being swapped.
4. **Swap Elements:**
   * We swap the elements at indices i and j.
5. **Testing the Method:**
   * In the main method, we create a sample array and call shuffle(A) to rearrange it. The shuffled array is printed to verify the result.

**Example Output:**

javascript

نسختحرير

Original Array:

1 2 3 4 5

Shuffled Array:

3 1 5 2 4

Since the Fisher-Yates shuffle is unbiased, every permutation of the array has an equal probability of occurring.

**Time Complexity:**

* The algorithm runs in **O(n)** time, where n is the length of the array, since it involves a single pass through the array and each swap is done in constant time.

**Space Complexity:**

* The space complexity is **O(1)**, as the shuffle is done in-place and does not require additional space proportional to the size of the array.

This implementation ensures that the elements of the array are rearranged in a truly random order, with every possible permutation being equally likely.