```java
/*
Alif Rahi
cs323 \ section 14 - 10:45am-12pm
Midterm problem set
*/



import java.util.Arrays;
import java.util.Random;

public class midterm {

    public static void main(String[] args) {
        method1(5);
        method2(5);
        method3(5);
    }

    public static void method1(int n) {
        int[] arr = new int[n];

        Random num = new Random();
        for (int i = 0; i < arr.length; i++) {
            arr[i] = num.nextInt(5) + 1;
            // creates random num in arr[i] from 1-5
            for (int k = 0; k < i; k++) {
                if (arr[i] == arr[k]) {
                    i--;
                }
            // We check for duplicates from 0-i
            // If duplicate is found, try again
            }
        }
        System.out.println(Arrays.toString(arr));

/*
The expected runtime for this method is O(n^2) because the best case and
the worst case are both O(n^2). You will always have to itterate through
the nested for loop even when you dont need to.
*/

    }
```

```
    public static void method2(int n) {
        int[] arr = new int[n];
        boolean[] used = new boolean[n];
        Random num = new Random();
        for (int i = 0; i < arr.length; i++) {
            arr[i] = num.nextInt(5) + 1;
            // creates random num in [arr[i]-1] from 1-5.
            //(-1 is to shift 0-4 -> 1-5 )
            while (used[arr[i] - 1]) {
                // We check if [arr[i]-1] was duplicated by the index
                arr[i] = num.nextInt(5) + 1;
                // if arr[i]==true that means generate new num
            }
            used[arr[i] - 1] = true;
            // After leaving while loop, set the index arr[i] to true
        }
        System.out.println(Arrays.toString(arr));
/*
The expected runtime for this code is harder to define because the best
case can be O(n) if you always create a random number thats not a
duplicate. In that case, you would never itterate through the while loop.
This is close to rare because usually you wont randomly generate an array
of all unique numbers. The worst case would be O(n^2).
*/

    }

    public static void method3(int n) {
        int[] arr = new int[n];
        Random num = new Random();

        for (int i = 0; i < arr.length; i++) { // --> O(n)
            arr[i] = i + 1; // start the array with values 1-10 in order
        }
        for (int i = 0; i < n; i++) { // --> O(n)
            int random = num.nextInt(n); // choose random index from 0-n
            // swap is constant O(1)
            int temp = arr[i];
            arr[i] = arr[random];
            arr[random] = temp;
        }
        System.out.println(Arrays.toString(arr));
/*
This method has an expected runtime of O(n). As you can see, we are not
nesting loops at all. This is the best method out of all three methods
with a worst case runtime of O(n) as well.
*/

    }

}
```