```
import java.util.Random;

public class QuickSortAnalysis {
    // Deterministic QuickSort
    public static void deterministicQuickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            deterministicQuickSort(arr, low, pi - 1);
            deterministicQuickSort(arr, pi + 1, high);
        }
    }

    // Randomized QuickSort
    public static void randomizedQuickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = randomizedPartition(arr, low, high);
            randomizedQuickSort(arr, low, pi - 1);
            randomizedQuickSort(arr, pi + 1, high);
        }
    }

    // Partition method used in deterministic QuickSort
    private static int partition(int[] arr, int low, int high) {
        int pivot = arr[high]; // Pivot element
        int i = low - 1; // Index of smaller element

        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                swap(arr, i, j);
            }
```

```
    }
    swap(arr, i + 1, high);
    return i + 1;
}

// Randomized partition for Randomized QuickSort
private static int randomizedPartition(int[] arr, int low, int high) {
    Random rand = new Random();
    int randomIndex = low + rand.nextInt(high - low + 1);
    swap(arr, randomIndex, high); // Move random element to the end
    return partition(arr, low, high);
}

// Utility method to swap elements in the array
private static void swap(int[] arr, int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}

// Utility method to generate a random array
public static int[] generateRandomArray(int size, int range) {
    Random rand = new Random();
    int[] arr = new int[size];
    for (int i = 0; i < size; i++) {
        arr[i] = rand.nextInt(range);
    }
    return arr;
}

// Method to copy an array (used to ensure same input for both variants)
```

```
    public static int[] copyArray(int[] arr) {
        int[] newArr = new int[arr.length];
        System.arraycopy(arr, 0, newArr, 0, arr.length);
        return newArr;
    }

    // Main method to analyze the performance
    public static void main(String[] args) {
        int size = 10000; // Size of the array
        int range = 10000; // Range of random numbers

        // Generate a random array for testing
        int[] originalArray = generateRandomArray(size, range);

        // Test Deterministic QuickSort
        int[] arr1 = copyArray(originalArray);
        long startTime = System.nanoTime();
        deterministicQuickSort(arr1, 0, arr1.length - 1);
        long endTime = System.nanoTime();
        System.out.println("Deterministic QuickSort time: " + (endTime - startTime) + " ns");

        // Test Randomized QuickSort
        int[] arr2 = copyArray(originalArray);
        startTime = System.nanoTime();
        randomizedQuickSort(arr2, 0, arr2.length - 1);
        endTime = System.nanoTime();
        System.out.println("Randomized QuickSort time: " + (endTime - startTime) + " ns");
    }
}
```