```java
import java.util.*;

public class mmRuleAlgorithm {
    public static void main(String[] args) {
        AlgorithmService alg = new AlgorithmService();
        int size = 100;
        long sum = 0;

        int k = size / 4;
        int k1 = size / 2;
        int k11 = 3 * size / 4;

        int arr[] = new int[size];
        alg.fillArray(arr);

        for (int i = 0; i < 20; i++) {
            long start = System.nanoTime();

            mmRule(arr, 0, arr.length - 1, 1);
            alg.kthSmallest(arr, arr[0], arr.length, k);

            long end = System.nanoTime();
            long total = end - start;
            sum += total;

            System.out.println(total);
        }
        System.out.println("\nThe average time is: " + sum / 15 + " nanoseconds");
    }

    // ----------------------------mm RULE-----------------------------
    /**
     *
     * @param arr
     * @param left
     * @param right
     * @param middle
     * @return
     */
    public static int mmRule(int arr[], int left, int right, int k) {
        AlgorithmService alg = new AlgorithmService();
        if (k > 0 && k <= right - left + 1) {
            int numOfElements = right - left + 1;
            int numOfGroups = (numOfElements + 6) / 7;
            int[] median = new int[numOfGroups];
            int count;

            for (count = 0; count < numOfElements / 7; count++) {
                median[count] = alg.findMedian(arr, left + count * 7, left + count *
7 + 7);
            }

            if (count * 7 < numOfElements) {
                median[count] = alg.findMedian(arr, left + count * 7, left + count *
7 + numOfElements % 7);
                count++;
            }

            int mm;
```

```
58              if (count == 1) {
59                  mm = median[count - 1];
60              } else {
61                  mm = mmRule(median, 0, count - 1, count / 2);
62              }
63
64              int location = alg.partition(arr, left, right, mm);
65              if (location - left == k - 1) {
66                  return arr[location];
67              } else if (location - left > k - 1) {
68                  return mmRule(arr, left, location - 1, k);
69              } else {
70                  return mmRule(arr, location + 1, right, k - location + left - 1);
71              }
72
73          } else {
74              int x = -1;
75              return x;
76          }
77      }
78      // ------------------------------END mm RULE-----------------------------
79  }
```