```java
import java.util.*;

public class AlgorithmService {
    /**
     *
     * @param arr
     * @return
     */
    public int[] fillArray(int[] arr) {
        Random random = new Random();
        for (int i = 0; i < arr.length; i++) {
            arr[i] = random.nextInt(2000);
        }
        return arr;
    }

    public int kthSmallest(int arr[], int l, int r, int k) {
        // If k is smaller than
        // number of elements in array
        if (k > 0 && k <= r - l + 1) {
            int n = r - l + 1; // Number of elements in arr[l..r]

            // Divide arr[] in groups of size 5,
            // calculate median of every group
            // and store it in median[] array.
            int i;

            // There will be floor((n+4)/5) groups;
            int[] median = new int[(n + 4) / 5];
            for (i = 0; i < n / 5; i++)
                median[i] = findMedian(arr, l + i * 5, l + i * 5 + 5);

            // For last group with less than 5 elements
            if (i * 5 < n) {
                median[i] = findMedian(arr, l + i * 5, l + i * 5 + n % 5);
                i++;
            }

            // Find median of all medians using recursive call.
            // If median[] has only one element, then no need
            // of recursive call
            int medOfMed = (i == 1) ? median[i - 1] : kthSmallest(median, 0, i - 1, i
/ 2);

            // Partition the array around a random element and
            // get position of pivot element in sorted array
            int pos = partition(arr, l, r, medOfMed);

            // If position is same as k
            if (pos - l == k - 1)
                return arr[pos];
            if (pos - l > k - 1) // If position is more, recur for left
                return kthSmallest(arr, l, pos - 1, k);

            // Else recur for right subarray
            return kthSmallest(arr, pos + 1, r, k - pos + l - 1);
        }

        // If k is more than number of elements in array
```

```
59          return Integer.MAX_VALUE;
60      }
61
62      public int[] swap(int[] arr, int i, int j) {
63          int temp = arr[i];
64          arr[i] = arr[j];
65          arr[j] = temp;
66          return arr;
67      }
68
69      public int findMedian(int arr[], int i, int n) {
70          Arrays.sort(arr, i, n);
71          return arr[i + (n - i) / 2]; // sort the array and return middle element
72      }
73
74      public int partition(int arr[], int l,
75              int r, int x) {
76          // Search for x in arr[l..r] and move it to end
77          int i;
78          for (i = l; i < r; i++)
79              if (arr[i] == x)
80                  break;
81          swap(arr, i, r);
82
83          // Standard partition algorithm
84          i = l;
85          for (int j = l; j <= r - 1; j++) {
86              if (arr[j] <= x) {
87                  swap(arr, i, j);
88                  i++;
89              }
90          }
91          swap(arr, i, r);
92          return i;
93      }
94  }
95
```