```
import java.util.*;

/**
 * Illustrate two algorithms for the 'find' problem, finding the k-th largest el
ement in a list.
 * @author ola
 *
 */

public class FindExample
{
    public final static int SIZE = 100000;

    /**
     * The Finder interface allows us to test/stress algorithms as objects
     * for the find problem.
     */
    public interface Finder{
        /**
         * Return k-th largest item in a list.
         * @param list is collection of mutually-comparable items
         * @param kindex is in range [0..list.size())
         * @return kindex-th largest element where 0 is smallest
         */
        public Comparable find(ArrayList<Comparable> list, int kindex);
    }

    /**
     * Simple implementation of Finder that sorts to find.
     * Complexity is O(n log n) for an n-element list
     */
    public static class SortFinder implements Finder {
        public Comparable find(ArrayList<Comparable> list, int kindex){
            Collections.sort(list);
            return list.get(kindex);
        }
    }

    /**
     * Average case O(n) algorithm for finding k-th largest using partition
     * algorithm (e.g., as used in quicksort)
     */
    public static class FastFinder implements Finder {
        public Comparable find(ArrayList<Comparable> list, int kindex){
            return findHelper(list,0,list.size()-1,kindex);
        }
        private Comparable findHelper(ArrayList<Comparable> list,
                                      int first, int last, int kindex){

            int lastIndex = first;
            Comparable pivot = list.get(first);

            // invariant: elements in [first..lastIndex] are <= first
            //            elements in (lastIndex..k) are > first

            for(int k=first+1; k <= last; k++){
                Comparable ko = list.get(k);
                if (ko.compareTo(pivot) <= 0){
                    lastIndex++;
                    Collections.swap(list,lastIndex,k);
                }
            }
            Collections.swap(list,lastIndex,first);
            if (lastIndex == kindex) {
                return list.get(lastIndex);
            }
            if (kindex < lastIndex) {
                return findHelper(list,first,lastIndex-1,kindex);
            }
            else {
                return findHelper(list,lastIndex+1,last,kindex);
            }
        }
```



```
    }

    /**
     * Use a priority-queue which can be constructed in linear time
     * then call remove k times for "find-k". Should result in
     * O(n) + O(k log n) complexity.
     */
    public static class PQFinder implements Finder {

        public Comparable find(ArrayList<Comparable> list, int kindex) {
            PriorityQueue<Comparable> pq = new PriorityQueue<Comparable>(list);
            Comparable object=null;
            for(int k=0; k <= kindex; k++){
                object = pq.remove();
            }
            return object;
        }

    }

    public static double stress(int trials, ArrayList<Integer> list, Finder find
er){

        double total = 0.0;
        ArrayList<Comparable> copy = new ArrayList<Comparable>();
        Random rando = new Random();
        for(int k=0; k < trials; k++){
            copy.clear();
            copy.addAll(list);
            int index = rando.nextInt(list.size());
            double start = System.currentTimeMillis();
            Integer i = (Integer) finder.find(copy,index);
            double end = System.currentTimeMillis();
            total += (end-start)/1000;
            if (i.intValue() != index){
                System.out.println("fail on "+index+" with "+i);
            }
        }

        return total;
    }

    public static void main(String[] args){
        ArrayList<Integer> list = new ArrayList<Integer>();
        for(int k=0; k < SIZE; k++){
            list.add(k);
        }
        Collections.shuffle(list);

        Finder sorter = new SortFinder();
        Finder fast = new FastFinder();
        Finder pqfind = new PQFinder();

        double stime = stress(10,list,sorter);
        System.out.println("sort time: "+stime);


        double ftime = stress(10,list,fast);
        System.out.println("fast time: "+ftime);

        double ptime = stress(10,list,pqfind);
        System.out.println("pqueue time: "+ptime);

    }

}
```