Ryan Gunawan
CS331-01
January 30, 2017

Programming Assignment #2 – Analysis of Divide and Conquer Techniques in Algorithms

Assignment Prompt:

Implement the two algorithms for finding the kth smallest integer in a set of integers using only one array that contains all the integers. Test your programs and compute the CPU time for different sets of integers generated by a random number generator.

Approach:

The assignment required finding the k^{th} smallest term in an unsorted array of integers. To do this, two methods were implemented which were discussed during lecture and outlined in the lecture notes. Both algorithms followed a sequence like that of QuickSort where it partitioned the array into three segments. Values less than the selected pivot point, values equal to the selected pivot point and values greater than the selected pivot point. The difference between the two algorithms were the selected point. The first algorithm utilized a random value within the array as a pivot point, whereas the second algorithm determined the median of medians of sequences of five integers of the array as the pivot point. The resulting computing run times are tabulated in Table 1. The results don't accurately reflect the theoretical numbers, where the second algorithm should have a linear run time compared to a worst-case complexity of n^2 for the first algorithm.

Table 1 – Results of CPU Time for Each Algorithm finding the kth smallest value

Data Set Size (Integers)	Algorithm 1 CPU Time (nanoseconds)	Algorithm 2 CPU Time (nanoseconds)
1,000	225911	784823
1,000	305837	1148607
1,000	237959	1333387
10,000	1339862	6376729
10,000	1309966	6871053
10,000	1662150	7342982
100,000	13624798	30976621
100,000	10489861	29566265
100,000	9803955	29271528
1,000,000	32595574	81642103
1,000,000	30163465	69558342
1,000,000	48082280	63106590

Ryan Gunawan
CS331-01
January 30, 2017

10,000,000	94171022	362475957
10,000,000	182277507	665679372
10,000,000	174326920	454466171
100,000,000	932316986	6207223253
100,000,000	1218837820	4303114204
100,000,000	1106967760	4475376508

Ryan Gunawan
CS331-01
January 30, 2017

Code:

```
import java.util.*;

public class PA2 {

	public static void main(String[] args) {
		System.out.println("Find the k'th smallest value in an array.");
		Random generator = new Random();
		long timer, alg1, timer2, alg2;
		int[] input = new int[10000000];
		int k = 404;
		for (int i = 0; i < input.length-1; i++){
			input[i] = generator.nextInt();
		}
		System.out.println("Array Size: " + input.length);
		System.out.println("K: " + k);

		timer = System.nanoTime();
		System.out.println("Algorithm1 Value: " + kth1(k, input));
		alg1 = System.nanoTime() - timer;
		System.out.println("Time: " + alg1);

		timer2 = System.nanoTime();
		System.out.println("Algorithm2 Value: " + kth2(k, input));
		alg2 = System.nanoTime() - timer2;
		System.out.println("Time: " + alg2);
	}

	public static int kth1 (int k, int[] set){
		int[] newSet = new int[set.length];
		// if |S| = 1, return single element
		if (set.length == 1) return set[0];
		else {
			// else, choose an element randomly from set
			int selected = set[set.length/2];
			// let s1 be less than, s2 equal to, s3 greater than m
			int s1 = 0, s2 = 0, s3 = 0;
			// fill in single partitioned array with tracked indices
			for (int a = 0; a < set.length; a++){
				if(set[a] < selected){
					newSet[s1] = set[a];
					s1++;
				} else if (set[a] > selected){
					newSet[(newSet.length-1)-s3] = set[a];
					s3++;
				} else {
					s2++;
				}
			}
			int sTemp = s1;
			if (s2 > 0){
				for (int b = 0; b < s2; b++){
					newSet[sTemp] = selected;
					sTemp++;
				}
```

Ryan Gunawan
CS331-01
January 30, 2017

```
                        }
                        // if s1 >= k, return SELECT(k, S1)
                        if (s1 >= k){
                                return kth1(k, Arrays.copyOfRange(newSet, 0, s1));
                        } else {
                                // else; if (|S1|+|S2| >= K), return m
                                if ((s1 + s2) >= k){
                                        return selected;
                                } else {
                                        // else, return SELECT(k-|S1|-|S2|, S3)
                                        return kth1(k - s1 - s2, Arrays.copyOfRange(newSet,
s1+s2, newSet.length));
                                }
                        }
                }

        }

        public static int kth2 (int k, int[] set){
                // if array is relatively small, use Java Sort instead
                if (set.length < 50){
                        Arrays.sort(set);
                        return set[k];
                } else {
                        int[] newSet = new int[set.length];
                        // divide into sequences of five elements and store medians into M
                        int[] M = new int[set.length/5];
                        for(int i = 0; i < set.length/5; i++){
                                int[] temp = Arrays.copyOfRange(set, i*5, (i+1)*5-1);
                                Arrays.sort(temp);
                                M[i] = temp[temp.length/2];
                        }
                        Arrays.sort(M);
                        // find median of medians as the pivot value
                        int selected = M[M.length/2];
                        int s1 = 0, s2 = 0, s3 = 0;
                        for (int a = 0; a < set.length; a++){
                                if(set[a] < selected){
                                        newSet[s1] = set[a];
                                        s1++;
                                } else if (set[a] > selected){
                                        newSet[(newSet.length-1)-s3] = set[a];
                                        s3++;
                                } else {
                                        s2++;
                                }
                        }
                        int sTemp = s1;
                        if (s2 > 0){
                                for (int b = 0; b < s2; b++){
                                        newSet[sTemp] = selected;
                                        sTemp++;
                                }
                        }
                        // if s1 >= k, return SELECT(k, S1)
```

Ryan Gunawan
CS331-01
January 30, 2017

```
                        if (s1 >= k){
                                return kth2(k, Arrays.copyOfRange(newSet, 0, s1));
                        } else {
                                if ((s1 + s2) >= k){
                                        // else; if (|S1|+|S2| >= K), return m
                                        return selected;
                                } else {
                                        // else, return SELECT(k-|S1|-|S2|, S3)
                                        return kth2(k - s1 - s2, Arrays.copyOfRange(newSet,
s1+s2, newSet.length));
                                }
                        }

                }



        }

}
```

Ryan Gunawan
CS331-01
January 30, 2017

Sample Console Output:

```
Find the k'th smallest value in an array.
Array Size: 1000
K: 404
Algorithm1 Value: -474051540
Time: 244047
Algorithm2 Value: -474051540
Time: 989354


Find the k'th smallest value in an array.
Array Size: 10000
K: 404
Algorithm1 Value: -1979632586
Time: 1737555
Algorithm2 Value: -1979632586
Time: 7358581


Find the k'th smallest value in an array.
Array Size: 100000
K: 404
Algorithm1 Value: -2129962024
Time: 15195789
Algorithm2 Value: -2129962024
Time: 55512150


Find the k'th smallest value in an array.
Array Size: 1000000
K: 404
Algorithm1 Value: -2145665618
Time: 40704322
Algorithm2 Value: -2145665618
Time: 169889748

Find the k'th smallest value in an array.
Array Size: 10000000
K: 404
Algorithm1 Value: -2147306066
Time: 157620159
Algorithm2 Value: -2147306066
Time: 1046437927


Find the k'th smallest value in an array.
Array Size: 100000000
K: 404
Algorithm1 Value: -2147465352
Time: 1141856467
Algorithm2 Value: -2147465352
Time: 10356591753
```