CS-146 Homework-5 Due: July-5-2020

1. Statisticians often are interested in the median value in a collection of data. In a collection, about the same number of values are greater than the median value as are less than the median value. One way to find the median is to sort the data and take the value that is at—or nearly at—the center of the collection. But sorting does more than necessary to find the median. You need to find only the kth smallest entry in the collection for an appropriate value of k. To find the median of n items, you would take k as n/2 rounded up—that is, the smallest integer greater than or equal to n/2, or [n/2]. You can use the partitioning strategy of quick sort to find the kth smallest entry in an array. After choosing a pivot and forming the subarrays Smaller and Larger, as shown in Figure 16-5, you can draw one of the following conclusions:

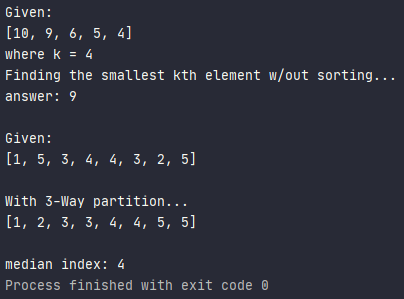
If Smaller contains k or more entries, it must contain the kth smallest entry.

If Smaller contains k−1 entries, the kth smallest entry is the pivot.

If Smaller contains fewer than k−1 entries, the kth smallest entry is in Larger.

You now can develop a recursive solution to finding the kth smallest entry. The first and last conclusions correspond to the recursive calls. The remaining one is the base case. **Implement a recursive method that finds the kth smallest entry in an unsorted array**. **Use your method to find the median in the array.**

/\*  
 \* Author: Ken Ho  
 \* Instructor: Prof. Faramarz Mortezaie  
 \* Class: CS146, SUM20  
 \* Topic: Finding the Kth smallest entry and implementing the 3-way partition  
 \*/  
  
import Exceptions.KthElementNotFound;  
import java.util.Arrays;  
  
public class Question1 {  
 public static void swap(int[] a, int i, int j) {  
 int temp = a[i];  
 a[i] = a[j];  
 a[j] = temp;  
 }  
  
 public static int partition(int[] a, int p, int r) {  
 int x = a[r-1];  
 int i = p - 1;  
 for (int j = p; j < r-1; j++) {  
 if (a[j] < x) {  
 i++;  
 **swap**(a, i, j);  
 }  
 }  
 **swap**(a, i+1, r-1);  
 return i+1;  
 }  
  
 public static int threeWayPartition(int[] a, int p, int r) {  
 //determine which pivot should stay in mid  
 if (a[p] > a[r-1]) {  
 if (a[p] > a[r/2]) {  
 **swap**(a, p, r-1);  
 }  
 else {  
 **swap**(a, p, r/2);  
 **swap**(a, p, r-1);  
 }  
 }  
 int mid = a[r/2];  
 int leftPtr = p+1;  
 //r-2 because r-1 is the last element, we already checked that so move one index  
 //down  
 int rightPtr = r-2;  
 while (leftPtr < rightPtr) {  
 //while leftPtr doesn't cross rightPtr  
 if (a[leftPtr] > mid) {  
 if (a[rightPtr] < mid) {  
 **swap**(a, leftPtr, rightPtr);  
 leftPtr++;  
 rightPtr--;  
 }  
 }  
 else {  
 //if a[leftPtr] <= mid  
 leftPtr++;  
 if (a[rightPtr] < mid) {  
 **swap**(a, leftPtr, rightPtr);  
 leftPtr++;  
 rightPtr--;  
 }  
 }  
 }  
 return mid;  
 }  
  
 public static int findSmallestKth(int[] a, int p, int r, int k) throws KthElementNotFound {  
 if (k > a.length || k < 1) {  
 //Taking care of out of bounds  
 throw new KthElementNotFound(k);  
 }  
 else {  
 int pivot = **partition**(a, p, r);  
 if (pivot == k-1) {  
 return a[pivot];  
 }  
 else if (pivot > k-1) {  
 //on the left  
 return **findSmallestKth**(a, p, pivot-1, k);  
 }  
 else if (pivot < k-1) {  
 //on the right  
 return **findSmallestKth**(a, pivot+1, r, k);  
 }  
 else {  
 throw new KthElementNotFound(k);  
 }  
 }  
  
 }  
  
 public static void main(String[] args) {  
 /\*  
 \* Given an input array (assume no duplicates),  
 \* use recursion to find the kth smallest entry  
 \*/  
 int[] a = { 10, 9, 6, 5, 4 };  
 int mid = (int) Math.**ceil**(a.length/2);  
 int k = 4;  
 int kthSmallest = 0;  
 System.**out**.println("Given: ");  
 System.**out**.println(Arrays.**toString**(a));  
 System.**out**.println("where k = " + k);  
 System.**out**.println("Finding the smallest kth element w/out sorting...");  
 try {  
 kthSmallest = **findSmallestKth**(a, 0, a.length, k);  
 }  
 catch (KthElementNotFound e) {  
 System.**out**.println(e.toString());  
 }  
 System.**out**.println("answer: " + kthSmallest + "\n");  
  
 /\*  
 \* Given an input array (contains duplicate), find the  
 \* median of the array.  
 \* Solution: 3-Way/ Dutch Flag partition algorithm  
 \*/  
 int[] c = { 1,5,3,4,4,3,2,5 };  
 System.**out**.println("Given: ");  
 System.**out**.println(Arrays.**toString**(c));  
 int median2 = **threeWayPartition**(c, 0, c.length);  
 System.**out**.println("\nWith 3-Way partition...");  
 System.**out**.println(Arrays.**toString**(c));  
 System.**out**.print("\nmedian index: " + median2);  
 }  
}



1. **A binary radix sort will sort an array a of n integer values based on their binary bits instead of their decimal digits. This sort will need only two buckets. Represent the buckets as a 2×n array. You can avoid some work by not copying the contents of the buckets back into the array a at the end of each pass. Instead just place the values from the second bucket to the end of the first bucket. Implement this algorithm.**

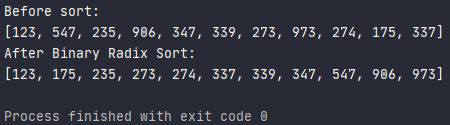
(The answer to this question is the same with question 3b below)

1. **Given the following numbers:**

**123 547 235 906 347 339 273 973 274 175 337**

1. **Sort using Quick sort (same answer with Question 5)**
2. **Sort using Radix sort**

/\*  
\* Author: Ken Ho  
\* Instructor: Prof. Faramarz Mortezaie  
\* Class: CS146, SUM20  
\* Topic: Binary Radix Sort using two buckets  
\* Reference: https://sjsu.instructure.com/courses/1368116/pages/recording-july-2?module\_item\_id=10781749  
\*/  
import java.util.Arrays;  
  
public class Question3 {  
 public static int getBit(int v, int bit) {  
 for (int i = 0; i < bit; i++) {  
 v /= 2;  
 }  
 return v % 2;  
 }  
  
 public static void binaryRadixSort(int[] a, int[][] temp, int f, int l) {  
 int maxPass = (int) Math.**ceil**(Math.**log**(Integer.**MAX\_VALUE**) / Math.**log**(2.0));  
 int bucketZero = 0;  
 int bucketOne = 0;  
  
 for (int scan = f; scan <= l; scan++) {  
 temp[0][bucketZero++] = a[scan];  
 }  
 int size = bucketZero;  
 for (int bit = 0; bit < maxPass; bit++) {  
 bucketZero = 0;  
 bucketOne = 0;  
 for (int scan = 0; scan < size; scan++) {  
 if (**getBit**(temp[0][scan], bit) == 0) {  
 temp[0][bucketZero++] = temp[0][scan];  
 }  
 else {  
 temp[1][bucketOne++] = temp[0][scan];  
 }  
 }  
 for (int scan = 0; scan < bucketOne; scan++) {  
 temp[0][bucketZero++] = temp[1][scan];  
 }  
 }  
 bucketZero = 0;  
 for (int scan = f; scan <= l; scan++) {  
 a[scan] = temp[0][bucketZero++];  
 }  
 }  
 public static void binaryRadixSort(int[] a, int f, int l) {  
 final int BUCKET\_SIZE = 2;  
 int[][] temp = new int[BUCKET\_SIZE][a.length];  
 **binaryRadixSort**(a, temp, f, l);  
 }  
 public static void main(String[] args) {  
 int[] a = { 123, 547, 235, 906, 347, 339, 273, 973, 274, 175, 337 };  
 System.**out**.println("Before sort: \n" + Arrays.**toString**(a));  
 **binaryRadixSort**(a, 0, a.length-1);  
 System.**out**.println("After Binary Radix Sort: \n" + Arrays.**toString**(a));  
 }  
  
}



1. **Implement quick sort and test it with**

**123 547 235 906 347 339 273 973 274 175 337**

/\*  
 \* Author: Ken Ho  
 \* Instructor: Prof. Faramarz Mortezaie  
 \* Class: CS146, SUM20  
 \* Topic: Quick sort implementation  
 \* Reference: https://sjsu.instructure.com/courses/1368116/pages/recording-july-2?module\_item\_id=10781749  
 \*/  
  
import java.util.Arrays;  
  
public class Question5 {  
 public static void swap(int[] a, int i, int j) {  
 int temp = a[i];  
 a[i] = a[j];  
 a[j] = temp;  
 }  
  
 public static int partition(int[] a, int p, int r) {  
 //select pivot  
 int x = a[r];  
 int i = p - 1;  
 for (int j = p; j < r; j++) {  
 if (a[j] < x) {  
 i++;  
 **swap**(a, i, j);  
 }  
 }  
 **swap**(a, i+1, r);  
 return i+1;  
 }  
  
 public static void quickSort(int[] a, int p, int r) {  
 if (p < r) {  
 int q = **partition**(a, p, r);  
 **quickSort**(a, p, q-1);  
 **quickSort**(a, q+1, r);  
 }  
 }  
  
 public static void main(String[] args) {  
 int[] a = {123, 547, 235, 906, 347, 339, 273, 973, 274, 175, 337};  
 System.**out**.println(Arrays.**toString**(a));  
 **quickSort**(a, 0, a.length-1);  
 System.**out**.println("\nAfter Quicksort...");  
 System.**out**.println(Arrays.**toString**(a));  
 }  
}

}

