Joanna Klukowska
joannakl@cs.nyu.edu

DNHI Homework 6 Solutions
Searching and Sorting

Problem 1

Write a method that provides recursive implementation for selection sort.

Answer: Note, it is not a good idea to use this in practice! This is done here just as an exercise.

```
1  void selectionSort ( int [] list ) {
2    //call recursive method starting at the zeroth index
3    selectionSortRecursive( list, 0 );
4  }
5
6  void selectionSortRecursive( int [] list, int indexToStart ) {
7    //if end of array is reached, stop
8    if (indexToStart == list.length() - 1 )
9      return;
10   //get the index of the smallest element in the rest of the array
11   int indexOfMin = findIndexOfSmallest ( list, indexToStart );
12   //swap the smallest element with the current front of the array
13   if (indexOfMin != indexToStart ) {
14     int tmp = list[indexToStart];
15     list[indexToStart] = list[indexOfMin];
16     list[indexOfMin] = tmp;
17   }
18   //continue starting at indexToStart+1
19   selectionSortRecursive( list, indexToStart + 1);
20 }
21
22 //find the index of the smallest element in the list starting
23 //at indexToStart
24 int findIndexOfSmallest ( int [] list, int indexToStart ) {
25   int smallest = indexToStart;
26   for (int i = indexToStart+1; i< list.length; i++ )
27     if (list[i] < list[smallest] )
28       smallest = i;
29   return smallest;
30 }
31
```

Problem 2

Given the following array of integers, show what the array will look like after each iteration of the outer loop of insertion sort. Indicate where the "sorted" part ends after each iteration. The elements should be sorted from smallest to largest.

index	0	1	2	3	4	5	6	7	8	9
item	7	32	12	3	0	64	23	12	3	45

Joanna Klukowska
joannakl@cs.nyu.edu

index	0	1	2	3	4	5	6	7	8	9
item	7	32	12	3	0	64	23	12	3	45
before iteration 1	7	32	12	3	0	64	23	12	3	45
after iteration 2	7	32	12	3	0	64	23	12	3	45
after iteration 3	7	12	32	3	0	64	23	12	3	45
after iteration 4	3	7	12	32	0	64	23	12	3	45
after iteration 5	0	3	7	12	32	64	23	12	3	45
after iteration 6	0	3	7	12	32	64	23	12	3	45
after iteration 7	0	3	7	12	23	32	64	12	3	45
after iteration 8	0	3	7	12	12	23	32	64	3	45
after iteration 9	0	3	3	7	12	12	23	32	64	45
after iteration 10	0	3	3	7	12	12	23	32	64	45

Problem 3

Apply quick sort to the following integer array. The array should be sorted from smallest to largest. The index of the pivot is selected to be `(first+last)/2` where first and last are the indexes of the `first` and `last` elements in the considered partition (do not use the median of three pivot selection that we discussed in class). The pivot should be moved to the last position before the partitioning step. After the partition it should be moved back to its proper position. You should show the following steps:

- content of the array after pivot is moved to the last position,
- content of the array after completing partitioning and moving the pivot to its proper position (indicate the position of the pivot in bold and/or in color).

index	0	1	2	3	4	5	6	7	8	9	10
value	12	13	27	65	43	59	45	67	7	5	55

Joanna Klukowska
joannakl@cs.nyu.edu

index	0	1	2	3	4	5	6	7	8	9	10
value	12	13	27	65	43	59	45	67	7	5	55
move pivot	12	13	27	65	43	55	45	67	7	5	59
partition	12	13	27	5	43	55	45	7	67	65	59
move pivot back	12	13	27	5	43	55	45	7	59	65	67
move pivot	12	13	27	7	43	55	45	5		67	65
partition	12	13	27	7	43	55	45	5		67	65
move pivot back	5	13	27	7	43	55	45	12		65	67
move pivot, base case reached		13	27	7	12	55	45	43			67
partition		13	27	7	12	55	45	43			
move pivot back		13	27	7	12	43	45	55			
move pivot		13	12	7	27		55	45			
partition		13	12	7	27		55	45			
move pivot back		13	12	7	27		45	55			
move pivot base case reached		13	7	12				55			
partition		7	13	12							
move pivot back		7	12	13							
move pivot (base cases reached)		7		13							
FINAL ARRAY	5	7	12	13	27	43	45	55	59	65	67

Problem 4

Apply merge sort to the following integer array. The array should be sorted from smallest to largest. Show all the steps of splitting the array and then merging the pieces.

index	0	1	2	3	4	5	6	7	8	9	10
value	2	76	27	62	43	59	45	87	13	5	99

Assuming the partitions are made aftert indexes: (first+last)/2, where first and last are beginning and ending index of each subarray on which merge sort is called.

0	1	2	3	4	5
2	76	27	62	43	59

6	7	8	9	10
45	87	13	5	99

0	1	2
2	76	27

3	4	5
62	43	59

6	7	8
45	87	13

9	10
5	99

0	1
2	76

2
27

3	4
62	43

5
59

6	7
45	87

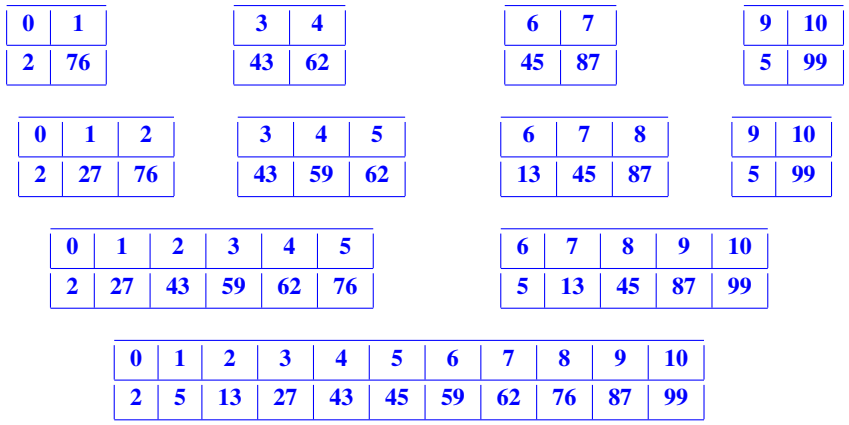
8
13

9
5

10
99

Joanna Klukowska
joannakl@cs.nyu.edu

Now, we can start merging (note, in practice, this would not be happening in "parallel", but written on paper it looks as it does):



Problem 5

Write an implementation of the generic method that given two sorted arrays, merges them into a single array and returns the array containing all elements in sorted order. The original arrays passed as a parameter should not be modified.
What do you need to assume about the generic type?

Answer: the generic type `E` has to implement the `Comparable<E>` interface

```
/**
  * Merge method merges to arrays of sorted data of generic type E
  * into a single sorted array that is returned.
  * Warning: this method assumes that list1 and list2 are sorted.
  * It does not validate this.
  *
  * @param list1  list of sorted elements of type E
  * @param list2  list of sorted elements of type E
  * @return
  *     an array of type E containing all elements from list1 and list2
  *     in sorted order
  */
private static <E extends Comparable<E>> E[] merge(E[] list1, E[] list2) {
  //Java does not let us create an array of generic type in
  //a "traditional" way.
  //The following code shows how this can be done.
  E[] temp = (E [] ) Array.newInstance(list1.getClass().getComponentType(),
          list1.length + list2.length);

  int current1 = 0; // Current index in list1
  int current2 = 0; // Current index in list2
  int current3 = 0; // Current index in temp

  //as long as there are elements in both lists,
  //find the smaller one and copy it to the merged list
  while (current1 < list1.length && current2 < list2.length) {
    if (list1[current1].compareTo(list2[current2]) < 0)
      temp[current3++] = list1[current1++];
    else
      temp[current3++] = list2[current2++];
  }
  //copy remaining elements from either list1 or list2
```

Joanna Klukowska
joannakl@cs.nyu.edu

```
33   while (current1 < list1.length)
34     temp[current3++] = list1[current1++];
35
36   while (current2 < list2.length)
37     temp[current3++] = list2[current2++];
38
39   return temp;
40 }
```

Problem 6

Write an implementation of the generic method that given a sorted `ArrayList<E>` object removes all repeated elements (leaving only one copy of each element). Make sure your method is efficient - it should run in linear time (not quadratic)!
What do you need to assume about the generic type?

This is my implementation. Therea re other, equally correct, implementations. For this version, the only assumption that we make for type E is that the `equals` method is implemented for it.

```
1 <E> boolean findDuplicates( ArrayList<E> names ) {
2   boolean modified = false;
3   int i = 1;
4   while (i < names.size()) {
5     if (names.get(i).equals(names.get(i - 1)))
6       names.remove(i);
7       modified  = true;
8     else
9       i++;
10  return modified;
11  }
12 }
```

If one used the `compareTo` method instead of `equals` method, the specification for the generic would need to be restricted to classes that implement the Comparable interface: `<E extends Comparable<E> >`.

Problem 7

Given an array of n elements, specify an algorithm for finding mode (the element which apprears most often). What is the complexity of your solution? Can you think of a different way of achieving the same task? What is teh complexity of the other solution?

Solution 1 (brute force)
The first solution is not the most efficient, but should be straight forward to think of and to implement. For each element in the array, check how many times it repeats. If the count is larger than the previously known maximum count, revise the maximum count.

```
1 int findMode ( int [] list ) {
2   int curCounter = 0;
3   int maxCounter = 0;
4   int mode = list[0];
5   for (int i = 0; i < list.length; i++ ) {
6     curCounter = 0;
7
8     for (int j = 0; j < list.legnth; j++ ) {
9       if (list[i] == list[j])
10        curCounter++;
11    }
12    if ( curCounter > maxCounter )
13      maxCounter = curCounter;
14      mode = list[i];
15    }
16  }
17  return mode;
```

Joanna Klukowska
joannakl@cs.nyu.edu

```
18 }
```

Because we have nested loops, the complexity of this solution is $O(n^2)$.

Solution 2:
The alternative and more efficient solution is to sort the elements in the array first. This can be done in $O(n \log_2 n)$ time. Once the array is sorted, we can traverse it once (complexity of $O(n)$) and figure out the mode since the same values are adjacent to each other. The overall complexity of this solution is $O(n \log_2 n)$.

Extra Challenge [1]

Write a sort method that sorts an array of strings so that all of the anagrams are next to one another.

[1] Not for the quiz, but maybe useful for a job interview.