Data Structures & Algorithms Problem Sheet 3

This is the printable version of the online problem sheet.

Worth 7% of your final grade
Due Friday 10/12, 7pm in Moodle

Motivation: Sorting is a very popular topic in coding interviews. When applying for a job at a tech company, it definitely helps if you can say "yes, I know that sorting algorithm – and I have implemented it, too." Sorting is also used everywhere: whenever you do an Internet search or look at online bank transactions there is likely some sorting happening in the background – because sorting data makes many algorithms faster (e.g. search). Similarly, trees are among the most popular and most widely used data structures and many problems can be solved with them. Implementing the AVL tree balancing is a challenge, but it makes your trees much faster.

Can you help Gabriel sort his toys?

Please read this first:

1. Marks are given for passing tests. Every passing test gives you a few marks. In order to pass the tests, your code needs to work as the automarker expects it to. This is described in the questions, so please read them carefully and follow their instructions to make sure you get the marks.
2. Make sure you always use the Precheck button to test your answer to a coding question. Precheck runs a subset of the tests that will be used to mark your asnwer, and you can use Precheck as many often as you like without using marks. This will help you to submit your code in a format that the automarker understands, i.e. it will show you if there are problems, so please **use Precheck well before the deadline!** Close to the deadline many people will be using the system and it will often be very slow.
3. Once you have created a complete answer for a question that you feel is correct, **use the Check button once**. Check will run all the tests used for marking and tell you which tests passed or failed. You can then use this to improve your answer. **Check will give you a penalty if you use it multiple times**: first 10% of the maximum marks of the question, then 20%, then 30% etc. Close to the deadline Check will often be very slow due to many people using it, therefore it is recommended to use it ealier, which will give you time to improve your answer.
4. It is recommended that you create yoru own tests to test your code before you use Check, and especially afterwards to make sure your answer is correct.
5. You don't need to use the Submit button to submit your answers. Your answers will be submitted automatically once the deadline has passed. Just make sure your answers are all stored in the answer boxes. If you don't press Submit, you can still change your answers up to the deadline without penalty.
6. Change only the parts in the given Java code files that are marked for your code. Do not change the method signature (i.e. parameters, name, types) when implementing a given method. Any changes other than the allowed parts will likely cause tests to fail.
7. You should comment your code, i.e. leave Java comments that explain briefly what you are doing and why (this may give you partial marks in case of problems).

8. You may use arrays, but you may not use predefined Java data structures such as ArrayList or API methods such as in java.util.Arrays.
9. You can assume that your methods will only be used with correct inputs, i.e. your code does not need to handle errors caused by incorrect inputs.
10. Please respect the Java conventions for naming of methods and classes, e.g. the use of upper/lower case: http://www.oracle.com/technetwork/java/codeconventions-135099.html
11. Do not copy code or answers from others. More about plagiarism and how to avoid it here: http://www.bath.ac.uk/library/help/infoguides/plagiarism.html
12. If you are struggling with Java or programming tools such as IntelliJ, have a look at the Topic 0 resources such as the video tutorials on YouTube. Many students found them useful.
13. If you have questions about this coursework please ask them in our Moodle forum or (even better) our Discord sever. You will get answers more quickly as the whole teaching team can answer, and you are helping your classmates.

Q1: Comparing Contacts (10% of marks)

Download the file Contact.java here:

```
public class Contact implements Comparable<Contact> {
    public String firstName;
    public String lastName;

    public Contact(String f, String l) {
        firstName = f; lastName = l;
    }

    public String toString() {
        return firstName + " " + lastName;
    }

    public int compareTo(Contact c) {
        // TODO implement this
        return 0;
    }
}
```

It implements a Java class which stores a contact with a first and last name. Implement the method compareTo(Contact c) which compares a Contact object with another given Contact object c. It returns an integer number which indicates whether the contact is before, equal to, or after the contact c, similar to the compareTo method of class String (have a look at the description below to understand what it should do):
http://docs.oracle.com/javase/7/docs/api/java/lang/String.html#compareTo(java.lang.String)
However, the compareTo method of class Contact should be implemented so that it compares the two objects as follows: for contacts with different last names, it should compare their last names using lexicographic order (for that you may use the compareTo method of class String); for contacts with the same last name it should compare their first names using lexicographic order (again you may use compareTo of class String).

Submit the source code of your Contact class using the answer box below. Use the Precheck button without penalty as often as you like to test your answer with a subset of the tests used for marking. You can use Check only once without penalty to get the results of all the tests used for marking. Using Check several times will incur a penalty (first 10% of the maximum question marks, then 20%, then 30% etc.). You can still change your answer without penalty anytime until the deadline or until you submit manually, just make sure that Moodle has recorded your answer. Your answer will be submitted automatically after the deadline, so there is no need to press Submit.

For example:

Test	Result

`Contact a = new Contact("Anne", "Abel");`
`Contact b = new Contact("Bertie", "Brenner");`
`System.out.println(a.compareTo(b) < 0);`
`System.out.println(b.compareTo(a) > 0);`
	true
	true

`Contact a = new Contact("Anne", "Abel");`
`Contact b = new Contact("Anne", "Abel");`
`System.out.println(a.compareTo(b) == 0);`
`System.out.println(b.compareTo(a) == 0);`
	true
	true

Q2: Sorting (40% of marks)

In this question you are asked to implement several sorting algorithms to sort arrays of Contact objects, by completing class Sorter.java, which can be downloaded here:

```
public class Sorter {
    public static void selectionSort(Contact[] contacts) {
        // TODO implement this
    }
    public static void insertionSort(Contact[] contacts) {
        // TODO implement this
    }
    public static void quickSort(Contact[] contacts) {
        // TODO implement this
    }
    public static void mergeSort(Contact[] contacts) {
        // TODO implement this
    }
    // TODO you may add additional methods here. Make them static just like
    // the other methods defined here.
}
```

You are allowed to use the code from the lecture slides. In order to compare two Contact objects in your code, you must use the compareTo() method of class Contact. A correct implementation of class Contact is automatically included in this question. The automarker will test that you have implemented the sorting methods correctly by looking printouts generated by compareTo() which state the objects that are being compared during sorting. For this to work correctly, you need to avoid calling compareTo() in the code on the same objects twice in a row. That is, instead of calling a.compareTo(b) and then again

a.compareTo(b), use a local variable to store the result and then use it, e.g. int result = a.compareTo(b);

a) Implement method selectionSort() of class Sorter, so that it sorts the given array of Contacts in ascending order using selection sort, as presented in the lecture.

b) Implement method insertionSort() of class Sorter, so that it sorts the given array of Contacts in ascending order using insertion sort, as presented in the lecture.

c) Implement method quickSort() of class Sorter, so that it sorts the given array of Contacts in ascending order using quicksort, as presented in the lecture. Use the first element of a subarray as pivot element. Note that you may need to add another internal quickSort() method with more parameters to Sort, which gets called by the given quickSort() method.

d) Implement method mergeSort() of class Sorter, so that it sorts the given array of Contacts in ascending order using mergesort, as presented in the lecture. Note that you may need to add another internal mergeSort() method with more parameters to Sort, which gets called by the given mergeSort() method.

Submit the source code of your Sorter class using the answer box below. Use the Precheck button without penalty as often as you like to test your answer with a subset of the tests used for marking. You can use Check only once without penalty to get the results of all the tests used for marking. Using Check several times will incur a penalty (first 10% of the maximum question marks, then 20%, then 30% etc.). You can still change your answer without penalty anytime until the deadline or until you submit manually, just make sure that Moodle has recorded your answer. Your answer will be submitted automatically after the deadline, so there is no need to press Submit.

For example:

Test	Result

`Contact[] a = new Contact[]{ new Contact("Bert", "Brenner"), new Contact("Anne", "Aalbert") };`
`Sorter.selectionSort(a);`
`System.out.println(java.util.Arrays.toString(a));`
	`Comparing Anne Aalbert with Bert Brenner`
	`[Anne Aalbert, Bert Brenner]`

`Contact[] a = new Contact[]{ new Contact("Bert", "Brenner"), new Contact("Anne", "Aalbert") };`
`Sorter.insertionSort(a);`
`System.out.println(java.util.Arrays.toString(a));`
	`Comparing Anne Aalbert with Bert Brenner`
	`[Anne Aalbert, Bert Brenner]`

`Contact[] a = new Contact[]{ new Contact("Bert", "Brenner"), new Contact("Anne", "Aalbert") };`
`Sorter.quickSort(a);`
`System.out.println(java.util.Arrays.toString(a));`
	`Comparing Bert Brenner with Bert Brenner`
	`Comparing Anne Aalbert with Bert Brenner`
	`Comparing Anne Aalbert with Bert Brenner`

Test	Result

`Contact[] a = new Contact[]{ new Contact("Bert", "Brenner"), new Contact("Anne", "Aalbert") };`
`Sorter.mergeSort(a);`
`System.out.println(java.util.Arrays.toString(a));`
	`[Anne Aalbert, Bert Brenner]`
	`Comparing Bert Brenner with Anne Aalbert`
	`[Anne Aalbert, Bert Brenner]`

Q3: Binary Trees (50% of marks)

Given the following incomplete Java class AVLTree and the class AVLTreeNode, which can be downloaded here:

```
class AVLTree {
  public AVLTreeNode root;
  // TODO you may define additional variables here

  public AVLTree() {
    // TODO implement this
  }

  public void createTestTree() {
    // TODO implement this
  }

  public void print() {
    // TODO implement this
  }

  public boolean inTree(String e) {
    // TODO implement this
  }

  public void insert(String e) {
    // TODO implement this
  }
}
class AVLTreeNode {
    public String element;
    public int height;
    public AVLTreeNode left = null;
    public AVLTreeNode right = null;

    AVLTreeNode(String e) {
        element = e;
        height = 1;
    }

    public int getRightHeight() {
        if (right == null) {
            return 0;
        } else {
            return right.height;
        }
    }
```

```
    public int getLeftHeight() {
        if (left == null) {
            return 0;
        } else {
            return left.height;
        }
    }

    public int getBalance() {
        return getRightHeight() - getLeftHeight();
    }

    public int getChildHeight() {
        return Math.max(getRightHeight(), getLeftHeight());
    }

    public int compareStringToThis(String s) {
        System.out.println("Comparing " + s + " with " + element);
        return s.compareTo(element);
    }

    public String toString() {
      return element;
    }
}
```

a) Implement the AVLTree constructor, which creates a new binary tree (very easy -- not much to do for this one).

b) Implement the createTestTree() method, which should construct a perfectly balanced binary tree for testing, using the strings “1”, “2”, “3”, “4”, “5”, “6”, “7” as elements.

c) Implement the print() method, which prints out the current AVL tree based on one of the tree traversals described in the lecture, using one line per element and space characters for indentation to signify the depth of an element in the tree. The printed output for the tree created by createTestTree should look like this:

```
4
 2
  1
  3
 6
  5
  7
```

d) Implement the inTree() method, which checks whether the given string element e is in the current tree. If it is in the tree, the method should return true, and false otherwise. Use the method compareStringToThis() of class AVLTreeNode to compare a given String with an AVLTreeNode and avoid calling compareStringToThis() twice with the same two elements (call it once and store the result in a local variable instead), as this is important for the automarker.

e) Implement the insert() method, which can use the usual insertion algorithm for binary search trees as described in the lecture, without re-balancing. You may assume the tree can

never contain any duplicates. Use the method compareStringToThis() of class AVLTreeNode to compare a given String with an AVLTreeNode and avoid calling compareStringToThis() twice with the same two elements (call it once and store the result in a local variable instead), as this is important for the automarker.

f) Modify and extend your code so that the tree is balanced after each insertion, based on the AVL tree algorithm described in the lecture. This is hard, so you may need to study the AVL algorithm in more detail to do this. You may want to use some of the variables and methods defined in AVLTreeNode.

Submit the source code of your AVLTree class using the answer box below. Use the Precheck button without penalty as often as you like to test your answer with a subset of the tests used for marking. You can use Check only once without penalty to get the results of all the tests used for marking. Using Check several times will incur a penalty (first 10% of the maximum question marks, then 20%, then 30% etc.). You can still change your answer without penalty anytime until the deadline or until you submit manually, just make sure that Moodle has recorded your answer. Your answer will be submitted automatically after the deadline, so there is no need to press Submit.

For example:

Test	Result
`AVLTree t = new AVLTree();` `System.out.println(t.root);`	`null`

`AVLTree t = new AVLTree();`
`t.createTestTree();`
`System.out.println(t.root);`
	`4`

`AVLTree t = new AVLTree();`
`t.createTestTree();`
`t.print();`
	`4`
	` 2`
	`  1`
	`  3`
	` 6`
	`  5`
	`  7`

`AVLTree t = new AVLTree();`
`t.createTestTree();`
`System.out.println(t.inTree("4"));`
	`Comparing 4 with 4`
	`true`

`AVLTree t = new AVLTree();`
`t.insert("100");`
`System.out.println(t.root);`
	`100`