Data Structures 2018
Exercise 8 (Week 44)

- **Notice that based on university's new regulations on degrees, you can have a degree fail from the course which is put to the register.**

  If a student does not participate in the course and does not cancel his/her enrollment, or if he/she discontinues the course, he/she will be assigned a fail grade for the course in question.

- **Students who participate in exercise group must be in place before the exercise group begins (12.15/14.15/16.15). Students who come late do not get the exercise points.**
- **Check the numbers of exercises made before you come to the exercise group. By this means we can save a lot of time when filling the exercise point list.**
- **Notice that pseudocode does not mean the same this as Java code. Pseudocode is not a programming language dependent presentation for an algorithm.**

1.-3. Implement the operations `insert` and `extractMin` for a heap containing integers in the file `MinHeap.java`. The heap is to be implemented in an array. Test the heap with the program `MinHeapTest` in the file `MinHeapTest.java`; you will have to change the test program a little as it visualizes the operations on heap by default and the testing is commented out. You get marks for this assignment only if all the tests pass without errors.
First implement the operations `percolateUp` and `percolateDown`, which takes a key to its position in the heap either upward or downward. `PercolateUp` works as follows: the key is exchanged with its parent until the key in the parent is smaller or it becomes the root. `PercolateDown` does the same, but downward and stops when both children have larger key or the key hits the bottom of the heap.
Use the functions `parent`, `leftChild` and `rightChild` that compute the indices of the parent and children of a heap node given the index of the node and the function `swap`, which swaps two keys in the heap given their indices. The heap as a binary tree starts from the index 1 of *table* and *last* contains the index of the last element in the heap (e.g. when the size of the heap is 1 it only contains the root and then $last = 1$)

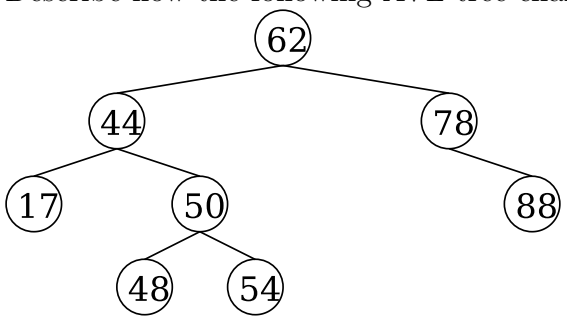
4. Describe how an AVL-tree is formed, when then keys

   a) 1, 2, 3, 4, 5, 6, 7, 8

   b) 1, 8, 2, 7, 3, 6, 4, 5

   are inserted to an empty AVL-tree in this order.

5. Describe how the following AVL-tree changes when the key 62 is removed.



6. How would you implement delete and find in closed hashing (e.g. linear probing and quadratic probing) ? Look at situations where keys collide and keys that have collided are deleted. Does the insert operation also change ? Why can't you just simply delete the element once it's found in delete ?

7. We are using an 11-element hashtable with the hash function

$$h(i) = (3i + 2) \mod 11.$$

   and the keys 3, 20, 100, 15, 9, 16, 45, 11, 5, 33 are inserted. What result do we get when collisions are solved with

   a) chaining ?

   b) linear probing ?

   c) quadratic probing ?