Name: .... *Neptun code:* ....

2. Algorithms and Data Structures I. example test

In exercises 1-2 do not write code. In exercises 3-4 try to write efficient code (structograms). Deletion includes explicit deallocation. Do not forget the types and modes of the formal parameters.

1.a Illustrate the operation removing the maximum of a priority queue on max-heap $\langle 15; 13; 9; 5; 12; 8; 7; 4; 0; 6; 2; 1\rangle$. Then redraw the resulting max-heap and illustrate the operation removing the maximum of it.

1.b Illustrate the operation of the insertion of 10 into a priority queue on max-heap $\langle 15; 13; 9; 5; 12; 8; 7; 4; 0; 6; 2; 1\rangle$. Then redraw the result and illustrate the operation of the insertion of 18 into it.
We suppose that the physical array containing the heap is long enough.

2. Illustrate heap sort on array
$\langle 5; 3; 17; 10; 84; 19; 6; 22; 9\rangle$.
Redraw the tree before each sinking operation modifying any node which is already modified on the actual drawing. When you redraw the tree, also draw the actual state of the array containing the tree.

3.a Write recursive function size(t) calculating the size of binary tree t:Node* ($MT(n) \in O(n)$). Do not use loops in the structogram.

3.b Write boolean function strictBinTree(t) deciding whether binary tree t:Node* is strictly binary tree or not ($MT(n) \in O(n)$). Do not use loops in the structogram.

3.c Write recursive procedure insert(t, k) inserting key k into binary sort tree t:Node* ($MT(h) \in O(h)$). Do not use loops in the structogram.

3.d Write procedure del1(t) deleting all the internal nodes with a single child from binary tree t:Node* ($MT(n) \in O(n)$). Do not use loops in the structograms.

Name: .... *Neptun code:* ....

4. The simple textual representation of a nonempty binary tree is the following: (leftSubtree Root rightSubtree)
The empty tree is represented by the empty string.
The lexemes of a textual representation are
- the opening bracket,
- the closing bracket,
- and the roots of the subtrees.
For example, a tree is represented by the string
"(((1)2)3((4(5))6(7)))"
if and only if it can be drawn like this:

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

The smart textual representation is similar, but we use curly brackets i.e. "{}" at *level* 0 of the tree, square brackets i.e. "[]" at *level* 1, normal brackets i.e. "()" at *level* 2, and so on; curly brackets if *level* mod 3 == 0, square brackets if *level* mod 3 == 1, normal brackets if *level* mod 3 == 2. For example the tree above is "{ [(1) 2] 3 [(4 {5}) 6 (7)] }" in smart textual representation.

4.a Give recursive structogram which prints the simple textual form of the binary tree identified by pointer t. $T(n) \in \Theta(n)$ where $n = n(t)$.

4.b Give recursive structogram which prints the smart textual form of the binary tree identified by pointer t. $T(n) \in \Theta(n)$ where $n = n(t)$.

4.c* Given the simple textual form of a nonempty binary tree in sequential file F. Write recursive structogram building the linked representation of the tree in $\Theta(n)$ time where n is the number of lexemes of the textual form. We suppose that we have a read statement that can read a lexeme x in time $\Theta(1)$. And we can also check whether $x ==' ('$ or $x ==')'$ in constant time.

Name: .... *Neptun code:* ....

5.a We can sort a given sequence of n numbers by first building a binary sort tree containing these numbers – using a modified tree insertion (which allows equal keys in the the tree) repeatedly to insert the numbers one by one – and then printing the numbers by an inorder tree walk. Write procedure binTreeSort(A) sorting array A with this sorting method where $mT(n) \in O(n \lg n)$ and $MT(n) \in O(n^2)$.

5.b Suppose that the type of the nodes of a binary search tree is `struct NodeS{ key: T; left, right, s: NodeS* }`
where the `s` pointers are undefined. Give recursive structogram to ensure that the attribute `s` in each node refers to the successor of the node according to the inorder traversal where $MT(n) \in O(n)$. Attribute `s` of the last node (containing the greatest key) must be $\oslash$.

5.c Write boolean function BST(t) deciding whether binary tree t:Node* is binary search tree (BST) or not ($MT(n) \in O(n)$). We can suppose that the keys in the nodes of the tree are numbers. Do not use loops in the structograms.

5.d Write boolean function BSortT(t) deciding whether binary tree t:Node* is binary sort tree (BSortT) or not ($MT(n) \in O(n)$). Do not use loops in the structograms.

5.e Write boolean function fullBinTree(t) deciding whether binary tree t:Node* is full binary tree (i.e. strictly binary tree with all the leaves at the lowest level of the tree) or not. ($MT(n) \in O(n)$.) Do not use loops in the structograms.

5.f A node of a binary tree is size-balanced, iff the difference between the sizes of its two subtrees is at most one.

A binary tree is size-balanced, iff all the nodes of the tree are size-balanced.

Write boolean function sizeBalancedBinTree(t) deciding whether binary tree t:Node* is size-balanced binary tree or not ($MT(n) \in O(n)$). Do not use loops in the structograms.

5.g Write procedure toBinTree(A, t) which creates the size-balanced binary tree t:Node* from array A ($MT(n) \in O(n)$) so that the inorder traversal of t is array A. Do not use loops in the structograms.