Proeftentamen INFDEV02-6A

The grade of the written exam is the sum of the points obtained in each question.
The grade of the written exam must be **greater or equal to 5.5** in order to receive the grade of the practical examination. The grade of the practical examination is the final grade[1].

Question	**1**	**2**	**3**	**4**	**5**	**Total**
Points	2	2	2	2	2	10

Question 1

What is the (tightest) complexity class of the code below with the big-Oh notation?

```
public int factorial(int n)
{
  if (n == 0)
    return 1;
  else
    return n * factorial(n - 1);
}
```

Question 2

Complete the code below (in correspondence of …………………) so that it produces the desired result: insertion of a given new node (`newNode`) in a doubly linked list (`list`), after a specified node (`node`).

```
public void insertAfter(DLinkedList list, Node node, Node newNode)
{
  newNode.prev = ............................ ;
  newNode.next = ............................ ;
  if (node.next == null)
    list.lastNode = ............................ ;
  else
    node.next.prev = newNode;
  node.next = ............................ ;
}
```

[1] See modulewijzer.

Question 3

a) What is the output of the following algorithm if input is the array `{800, 11, 50, 771, 649, 770, 240, 9 };` ?

b) What is the worst-case (tightest) complexity class of the algorithm using the big-Oh notation?

```
    public void MysteryMethod(int[] numarray)
    {
      for (int i = 1; i < numarray.Length; i++)
      {
        int j = i;
        while (j > 0)
        {
          if (numarray[j - 1] < numarray[j])
          {
            int temp = numarray[j - 1];
            numarray[j - 1] = numarray[j];
            numarray[j] = temp;
            j--;
          }
          else
            break;
        }
      }
    }
```

Question 4

Complete the code below (in correspondence of …………………) so that it correctly performs the insertion of a new node in a binary search tree. The `Node` class contains three fields: `key` (integer value), `left` (left child node), `right` (right child node).

```
public void insert(int key)
{
  root = insertRec(root, key);
}
public Node insertRec(Node root, int key)
{
  if (root == null)
  {
    root = new Node(key);
    return root;
  }

  if (key < root.key)
    root.left = insertRec(.............................. , ..............................);
  else if (key .............................. )
```

```
    root.right = ……………………………   ;

  return root;
}
```

Question 5

Suppose that a graph is stored as a list of nodes (and each node contains information on its neighbours). What does the following algorithm (written in pseudocode) do and with which complexity?

```
MysteryMethod(Graph, root)
      for each node n in Graph:
               n.visited = FALSE

      create empty stack S
      root.visited = TRUE
      print(root)
      S.push(root)

      while S is not empty:
         currentTop = S.peek()
         while (exist v adjacent to currentTop that is not visited yet) {
            v.visited = TRUE
            print(v)
            S.push(v)
            currentTop = S.peek()
         }
         S.pop()
```