INFDEV026A - Algoritmiek Week 7

G. Costantini, F. Di Giacomo, G. Maggiore

costg@hr.nl, giacf@hr.nl, maggg@hr.nl – Office H4.204

Today

- ~~Why is my code slow?~~
  - ~~Empirical and complexity analysis~~
- ~~How do I order my data?~~
  - ~~Sorting algorithms~~
- ~~How do I structure my data?~~
  - ~~Linear, tabular, recursive data structures~~
- ~~How do I represent relationship networks?~~
  - ~~Graphs~~

→ Proeftentamen

Proeftentamen – 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);
  }
```

Proeftentamen – Question 1

- Answer:
  - $\boldsymbol{O(n)}$
- Why?
  - The factorial method is called n times
  - The base case ($n == 0$) has constant complexity

Proeftentamen – 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 = …… ;
 }
```

Proeftentamen – 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 = node;
   newNode.next = node.next;
   if (node.next == null)
     list.lastNode = newNode;
   else
     node.next.prev = newNode;
   node.next = newNode;
 }
```

Proeftentamen – Question 3

- What is the output of the following algorithm if input is the array $\{800, 11, 50, 771, 649, 770, 240, 9\}$; ?
- What is the worst-case 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;
        }
      }
    }
```

Proeftentamen – Question 3

- Answers
  1. The output is $\{\mathbf{800}, \mathbf{771}, \mathbf{770}, \mathbf{649}, \mathbf{240}, \mathbf{50}, \mathbf{11}, \mathbf{9}\}$;
  2. The worst-case complexity is $\boldsymbol{O(n^2)}$, where n = length of the array
     - The outer **for** loop is executed approximately n times
     - For each iteration of the outer loop there is a **while** loop which could (in the worst case) be executed approximately n times
     - The body of the **while** loop takes constant time

Proeftentamen – 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;
}
```

Proeftentamen – 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( root.left , key );
  else if (key > root.key )
    root.right = insertRec( root.right, key );

  return root;
}
```

Proeftentamen – 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()
```

Proeftentamen – Question 5

- Answers

1. The code performs a **depth first traversal** of the graph

2. The complexity is $\boldsymbol{O(|V| + |E|)}$
   - Where V is the vertices set (and thus $|V|$ is the number of vertices)…
   - …and E is the edges set (and thus $|E|$ is the number of edges)

Proeftentamen feedback

- How did you find it?
- Where do you have more difficulties?
- Do you have specific doubts/questions?