# Database Systems II – Exercise #1

## Sheet #1: Big O Notation, Data Structures and Algorithms

Daniel Flachs

Chair of Practical Computer Science III:
Database Management Systems

27/02/2019

UNIVERSITY
OF MANNHEIM
School of Business Informatics
and Mathematics

# Contents

# Contents

# Task 2

## Recap: Big O Notation – Asymptotic Runtime

- Approximation for the runtime of an algorithm for sufficiently large inputs (→ asymptotic).
- Runtime (in number of instructions) is given as a function that depends on the size of the input, e. g., length of an array, number of bits needed to store a number.
- Big O only considers the highest-order terms of an expression and ignores constant factors, e. g., $0.5n^2 + 3n - 4$ is $O(n^2)$.
  **Reason**: For large input sizes $n$, the lower-order terms and constants become insignificant.
- Read more in Cormen, Leiserson, Rivest, Stein: *Introduction to Algorithms*. 3rd ed. Cambridge, Mass. MIT Press, 2009.
  Chapters 2 and 3. Online book available at the ▶ Uni MA Library.

# Task 2

## Recap: Big O Notation – Asymptotic Runtime

**Note**

For this exercise, we are mostly interested in upper bounds that are tight. Different from the formal definition, we use the letter $O$ to indicate such a bound, not considering the formal differences between $O$, $\Theta$, $\Omega$, $o$, and $\omega$. For details, see ▶ Cormen et al.

# Task 2a

```
void printUnorderedPairs (const std::vector<double>& V) {
  for (unsigned i = 0; i < V.size(); ++i) {
    for (unsigned j = i+1; j < V.size(); ++j) {
      std::cout << "{" << V[i] << "," << V[j] << "}" << std::endl;
    }
  }
}
```

Loop body is executed for all pairs $P := \{(i, j) \mid 0 \leq i < j < n\}$.

$$|P| = \underbrace{(n-1) + (n-2) + \ldots + 1 + 0}_{n} = \frac{n \cdot (n-1)}{2}$$

Asymptotic Runtime: $O(n^2)$

# Task 2b

```
int factorial (int n) {
  if (n < 0) {                // Error case
    return -1;
  } else if (n == 0) {        // Base case
    return 1;
  } else {                    // Recursion
    return n * factorial(n-1);
  }
}
```

For an arbitrary $n \geq 0$, the recursion branch is entered $n$ times, and the base case once: $n + 1$.

Asymptotic Runtime: $O(n)$

# Task 2c

```
void allFib (int n) {
  for (int i = 0; i <= n; ++i) {
    std::cout << "fib(" << i << "): " << fib(i) << std::endl;
  }
}

int fib (int n) {
  if (n == 0 || n == 1) {
    return n;
  }
  return fib(n-1) + fib(n-2);
}
```

- Each fib(n) call produces two recursive calls, fib(n-1) and fib(n-2).
- fib(n) has runtime $O(2^n)$.
- allFib calls fib(n) for all $n \in \{0, \ldots, n\}$
  $\Rightarrow 2^0 + 2^1 + \ldots + 2^{n-1} + 2^n = 2^{n+1} - 1.$

Asymptotic Runtime: $O(2^n)$

# Task 3a

Implement a function that finds all positive integer solutions to the equation

$$a^2 + b^2 = c^2 + d^2$$

where $a, b, c, d \in [0, 1000]$.

Try to find an efficient solution. What is the asymptotic runtime of your function? Measure the actual runtime of your function.

# Task 3b

Implement a function that compresses a string using counts of repeated characters[1]. For example, the string `aabcccccaaa` becomes `a2bc5a3`. Note that if a character occurs only once, then its count is not part of the compressed string.

[1] Run-length encoding, see https://en.wikipedia.org/wiki/Run-length_encoding

# Task 3c

Implement a stack class, i.e., a LIFO container.
Your class is expected to provide the following function members:

- `pop()`: Remove the top item from the stack.
- `push(item)`: Add an item to the top of the stack.
- `top()`: Return the top of the stack.
- `isEmpty()`: Return true if and only if the stack is empty.

# Task 4

Implement a binary tree class based on the skeleton code provided.

a) Copy constructor and destructor
b) In-order printing
c) Call by value and call by reference
d) Symmetry check
e) Symmetric inversion of a tree
f) Flatten a tree to a linked list

# Task 4d: Symmetry of Binary Trees

- A tree is symmetrical if the left subtree $T_l$ is the mirror image of the right subtree $T_r$.
- $T_l$ is the mirror image of $T_r$ if
  - the root nodes are identical, and
  - the right subtree of $T_r$ is the mirror image of the left subtree of $T_l$, and
  - the right subtree of $T_l$ is the mirror image of the left subtree of $T_r$.

# Task 4e: Symmetric Inversion of a Binary Tree

- The inversion of an empty tree is the empty tree.
- The inversion of a tree with a left subtree $T_l$ and a right subtree $T_r$ is a tree with the inversion of $T_r$ as its left child and the inversion of $T_l$ as its right child.

# Task 4f: Binary Tree to Linked List

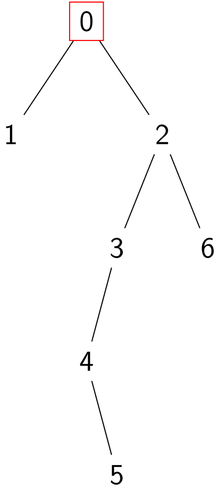
- To transform a tree with root $R$ and subtrees $T_l$ and $T_r$:
  - Transform $T_r$ into a list, then $T_r$.
  - The list $T_l$ becomes the right child of $R$.
  - The list $T_r$ becomes the left child of $R$.

# Task 4f: Binary Tree to Linked List

```
void flatten(Node* aNode) {
  if (!aNode) { return; }            // Recursion anchor: NULL nodes

  flatten(aNode->getRightChild()); // Recursively flatten both child nodes
  flatten(aNode->getLeftChild());  // => _left and _right are now chains!

  /* Go to the rightmost (deepest) node in the left subtree of aNode
   * and append the entire right subtree of aNode as rightmost's right child. */
  Node* lRightMost = aNode->getLeftChild();
  if (lRightMost) {
    while (lRightMost->getRightChild()) {
      lRightMost = lRightMost->getRightChild();
    }

    lRightMost->setRightChild(aNode->getRightChild());
    aNode->setRightChild(aNode->getLeftChild());
    aNode->setLeftChild(NULL);
  }
}
```

# Task 2f)

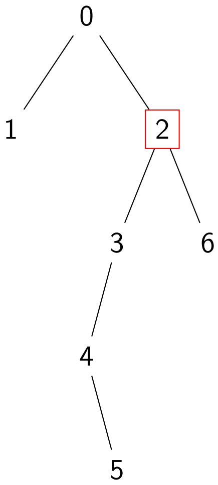
## Example (1/11)




flatten(0)

# Task 2f)

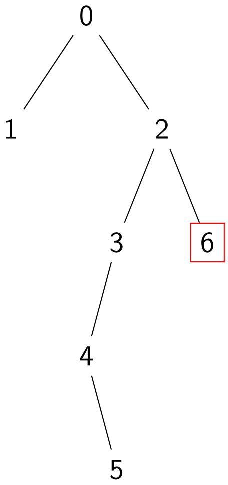
## Example (2/11)




`flatten(2)`

# Task 2f)

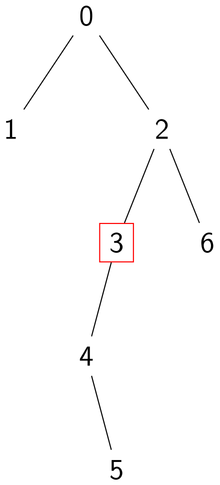
## Example (3/11)




flatten(6) ✓

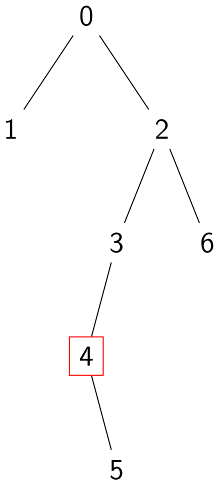
# Task 2f)

## Example (4/11)




flatten(3)

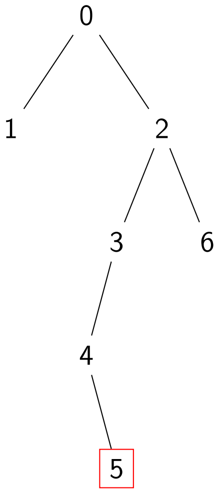
# Task 2f)
## Example (5/11)




flatten(4)

# Task 2f)

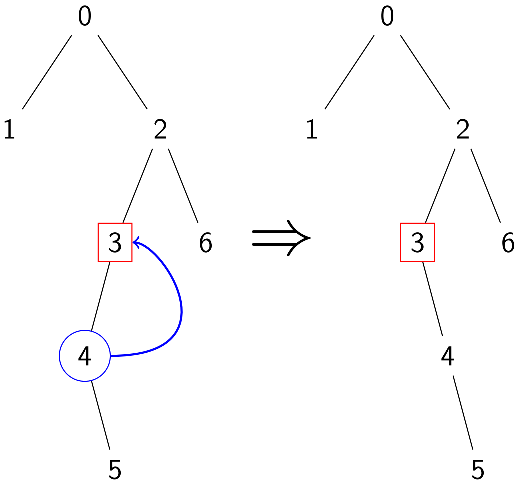
## Example (6/11)




flatten(5) ✓

# Task 2f)

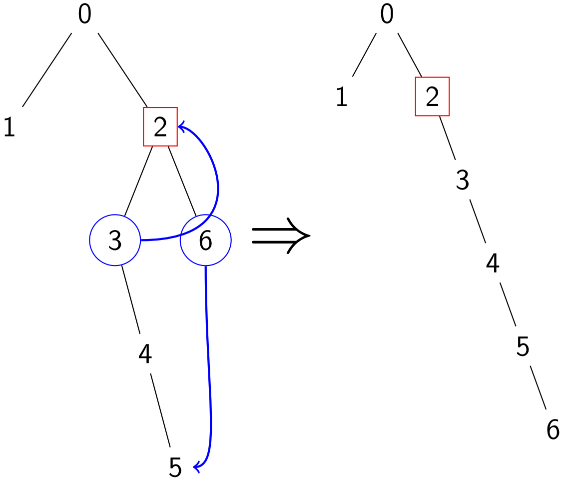
## Example (7/11)




flatten(3) ✓

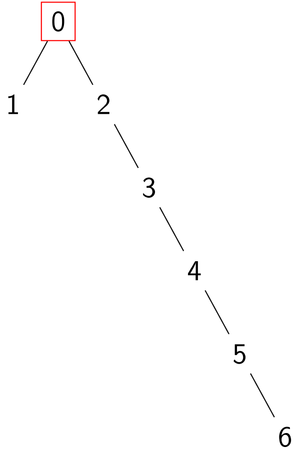
# Task 2f)

## Example (8/11)




flatten(2) ✓

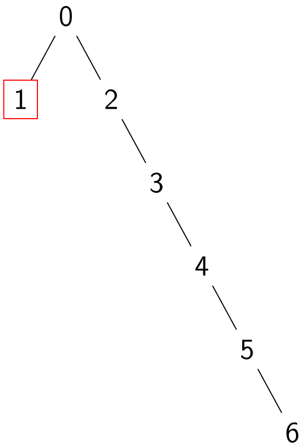
# Task 2f)
## Example (9/11)




flatten(0)

# Task 2f)

## Example (10/11)




flatten(1) ✓

# Task 2f)

## Example (11/11)