Chair of Applied Computer Science III

Prof. Dr. Guido Moerkotte
Email: moerkotte@uni-mannheim.de

Daniel Flachs
Email: daniel.flachs@uni-mannheim.de

Database Systems II
Spring Semester 2019

Solution to Exercise Sheet 1
Created February 27, 2019

Exercise 1

If you haven't done so already, install a C++ compiler on your computer. If in doubt how to do this, this webpage may be helpful to you:
`www.cs.odu.edu/~zeil/cs250PreTest/latest/Public/installingACompiler/`

Exercise 2

For each of the following functions, give the asymptotic runtime in "Big O" notation. All examples are taken from McDowell: *Cracking the Coding Interview: 189 Programming Questions and Solutions*. 6^{th} ed. Palo Alto, CA, 2016.

Exercise 2 a)

```
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;
    }
  }
}
```

Solution

The loop body is executed for each pair in $P := \{(i,j) \mid 0 \leq i < j < n\}$. It holds: $|P| = \underbrace{(n-1)+(n-2)+\ldots+1+0}_{n} = \frac{n \cdot (n-1)}{2}$. Therefore, the runtime is $O(n^2)$.

Exercise 2 b)

```
int factorial (int n) {
  if (n < 0) {              // case (i)
    return -1;
  } else if (n == 0) {      // case (ii)
    return 1;
  } else {                  // case (iii)
    return n * factorial(n-1);
```

```
  }
}
```

Solution

For an arbitrary $n \geq 0$, the recursion branch (iii) is entered n times (for n, $n-1$, $\ldots$, 1), and the base case (ii) once (for $n = 0$): $(n+1) \in O(n)$.

Exercise 2 c)

```
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);
}
```

Solution

- 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 $n \in \{0, \ldots, n\} \Rightarrow 2^0 + 2^1 + \ldots + 2^{n-1} + 2^n = 2^{n+1} - 1$.

  This yields a total asymptotic runtime of $O(2^n)$.

Exercise 3

Exercise 3 a)

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?
Optional: Measure the actual runtime of your function.

Solution

See code.

Exercise 3 b)

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.

Solution

See code.

Exercise 3 c)

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

Solution

See code.

Exercise 4

In the source code provided on the website you'll find a simple binary tree class, cf. `binaryTree.hh` and `binaryTree.cc`. In addition you'll find a file `main.cc` with a main function and several stubs of functions to be implemented.

On unix systems, you can use of the `makefile` to compile your code by simply typing `make` in the command line. Otherwise manually call the compiler with all source files (`.cc` ending) as input.

Solution

See code.

Exercise 4 a)

Implement the copy constructor of the class `Node` such that if performs a deep copy of its subtree. In addition, write `COPY` to the standard output.

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

Implement the destructor of the class `Node` such that, upon destruction of a Node, all child nodes are destructed and their memory is freed.

Note: Follow this link for information on copy-constructors, destructors and pointers: `http://www.cplusplus.com/doc/tutorial/classes2/`

Exercise 4 b)

Implement the `print` function of the class `Node` to output a tree in in-order ordering such that the value of a node and the values of its children are surrounded by brackets. See the following example for an illustration:
The tree:

```
     0
   /   \
  1     2
 / \   / \
3   4 5   6
```

The output:
`((3)1(4))0((5)2(6))`

Exercise 4 c)

Implement two functions `foo` and `bar`. Both take as parameter the root node of a tree. However, `foo` calls the root node by value whereas `bar` calls the root node by reference.
Note: Follow this link for information on call-by-value and call-by-reference:
`http://www.cplusplus.com/articles/z6vU7k9E/`

Exercise 4 d)

Implement a function `isSymmetric` that returns `true` if and only if the tree is axial symmetric around a vertical axis through the root node.
For instance, for this tree, `isSymmetric` must return `true`:

```
     0
   /   \
  1     1
 / \   / \
3   4 4   3
```

Exercise 4 e)

Implement a function `invert` that inverts a given tree. See the following example for an illustration:
The tree:

```
       0
     /   \
    1     2
   / \   / \
  3   4 5   6
```

Inverted tree:

```
       0
     /   \
    2     1
   / \   / \
  6   5 4   3
```

Exercise 4 f)

Implement a function `flatten` that transforms a given tree to a linked-list by reassigning the node's child pointers. Try to perform this transformation in-place, meaning that no new memory must be allocated except for auxiliary variables.
See the following example for an illustration:
The tree:

```
       0
     /   \
    1     2
   / \   / \
  3   4 5   6
```

Tree as linked-list:

```
0-1-3-4-2-5-6
```