Algorithms and Data Structures

All necessary data and source code files can be downloaded from the web site.

Exercise 1 *(O-notation)*

Determine good upper bounds for the following functions in O- notation:
$f(n) = 2^n + n^2$, $g(n) = n^2 + n \log_2 n$, $h(n) = 3n^3 + 7n^2 - 17n$.

Give short explanations for your findings.

Exercise 2 *(insertion sort)*

Apply insertion sort to the input $23, 15, 7, 42, 9, 13, 3, 11$. After each sorting step, write down the current sequence.

Exercise 3 *(shell-command sort)*

Use the command `sort` to sort the instance given in file `shellsort.txt`. Sorting has to be done according to the following criteria:

- sort according to the data in the third column
- numbers are output in alphanumeric order
- only one data record is output in case it appears multiple times

Write down the command with which you did the sort.

Hint: For linux, the shell command `man sort` accesses the manual for `sort`.

Exercise 4 *(programming: merge sort)*

1. Get the implementation of `merge sort` from the web site and test it for the instance `merge-sort-small.txt`.
2. Change the implementation such that it sorts in decreasing instead of increasing order.
3. Let $a_1 \ldots, a_n \in \mathbb{Z}$ be pairwise different numbers. Then we call a pair (i, j) with $i < j$ an *inversion* of $a_1, \ldots, a_n$, if it is $a_i < a_j$. (Remember we sort in decreasing order.) Determine all inversions of the sequence 3, 2, 6, 7, 5.
4. Modify `merge sort` such that additionally all inversions are counted, without changing the asymptotic running time of $O(n \log n)$. Test your implementation on instances `merge-sort-small.txt` and `merge-sort-large.txt`. How many inversions do the sequences have?

Exercise 5 *(stable sort)*

A sorting algorithm is called *stable* if the order of records with the same keys is not changed during the sorting. Argue whether `merge sort` and `insertion sort` are stable or not.

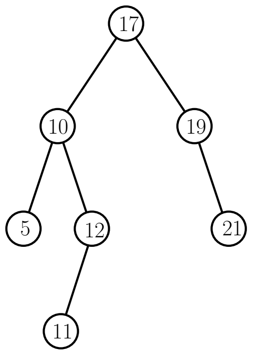


Figure 1: A binary search tree.

Exercise 6 *(binary search)*

Perform a binary search for the entry 130 in

$$1, 2, 3, 4, 12, 51, 52, 60, 82, 84, 90, 116, 121, 127, 130, 132.$$

For each step in the algorithm, write down the element it currently checks and whether it continues in the left or in the right subsequence.

Exercise 7 *(programming: binary trees)*

Get the framework for implementing a `binary tree` from the shared directory. Several functionalities are missing and need to be implemented. In particular, write the following functions efficiently.

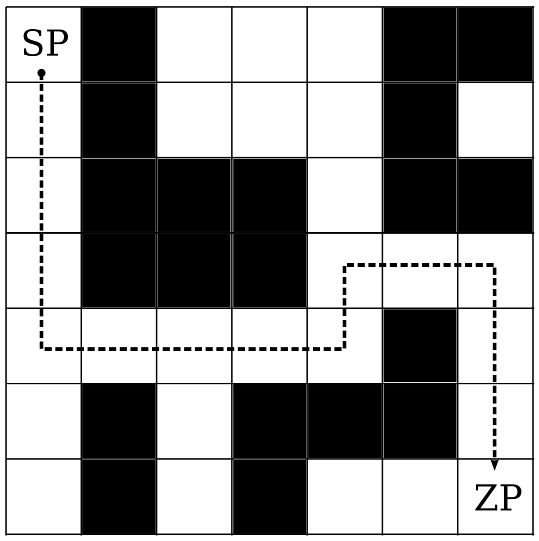
1. Determine the element with maximum key and print it to the screen.
2. The `successor(x)` of a node `x` returns a pointer to the node in the tree whose key value is next higher than that of `x`. What are the successors of nodes 17 and 10 in the tree from Figure 1? Implement a function that outputs the successor of a node `x`.
3. Different ways of traversing a tree are possible. Using recursion, all nodes of the tree can be visited. In each recursion step, *preorder* visits (outputs) the key of the current node, traverses to the left child, and traverses to the right child. Write down the preorder traversal of the tree from Figure 1.
4. Implement a tree traversal using preorder.
5. Extend the above implementation such that the level of each node is printed as well. Additionally, the maximum level that can be found in the tree should also be printed. The level of the root node is 0, the level of its childen is 1, and so on.

What are the worst-case running times of your functions?

Test your implementations with the files `treedata-small.txt` and `treedata-large.txt`.

Additional exercise 8 if you still have time... *(programming: labyrinth)*

A small robot has to find a way through a labyrinth from SP to ZP:



The robot can move horizontally and vertically along white fields. Black fields are walls and cannot be entered. A labyrinth always has quadratic shape. As the robot does not know how the labyrinth looks like, we need an algorithm that moves the robot from the start SP to an arbitrary final field ZP. The length of the path does not matter, as long as the robot finds ZP at all.

In file `labyrinth.c` some functions are missing and need to be implemented.

1. Design an algorithm for solving the robot problem and implement it in the framework from `labyrinth.c`. Use one or several stacks for your implementation.

2. Test your implementation on instances `labyrinth.txt` and `labyrinth2.txt`. Print the sequence of visited fields.