Introduction to Algorithms October 5, 2004
Massachusetts Institute of Technology 6.046J/18.410J
Professors Piotr Indyk and Charles E. Leiserson Handout 8

Problem Set 3

Reading: Chapters 12.1-12.4, 13, 18.1-18.3

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered in the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date and the names of any students with whom you collaborated.

Three-hole punch your paper on submissions.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct algorithms which are *which are described clearly*. Convoluted and obtuse descriptions will receive low marks.

Exercise 3-1. Do Exercise 12.1-2 on page 256 in CLRS.

Exercise 3-2. Do Exercise 12.2-1 on page 259 in CLRS.

Exercise 3-3. Do Exercise 12.3-3 on page 264 in CLRS.

Exercise 3-4. Do Exercise 13.2-1 on page 278 in CLRS.

Problem 3-1. Packing Boxes

The computer science department makes a move to a new building offering the faculty and graduate students boxes, crates and other containers. Prof. Potemkin, afraid of his questionable tenure case, spends all of his time doing research and absentmindedly forgets about the move until the last minute. His secretary advises him to use the only remaining boxes, which have capacity exactly *1 kg*. His belongings consists of n books that weigh between 0 and 1 kilograms. He wants to minimize the total number of used boxes.

Prof. Potemkin realizes that this packing problem is NP-hard, which means that the research community has not yet found a polynomial time algorithm[1] that solves this problem *exactly*.

He thinks of the *heuristic approach* called BEST-PACK:

1. Take the books in the **order in which they appear** on his shelves.
2. For each book, scan the boxes in **increasing order of the remaining capacity** and place the book in the first box in which it fits.

(a) Describe a data structure that supports efficient implementation of BEST-PACK. Show how to use your data structure to get that implementation.

(b) Analyze the running time of your implementation.

Soon, Prof. Potemkin comes up with another heuristic WORST-PACK, which is as follows:

1. Take the books in the order in which they appear on his shelves.
2. For each book, find a partially used box which has the **maximum** remaining capacity. If possible, place the book in that box. Otherwise, put the book into a new box.

(c) Describe a data structure that supports an efficient implementation of WORST-PACK. Show how to use your data structure to get that implementation.

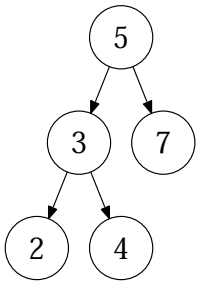
(d) Analyze the running time of your implementation.

[1] That is, an algorithm with running time $O(n^k)$ for some fixed k.

Problem 3-2. AVL Trees

An AVL tree is a binary search tree with one additional structural constraint: For any of its internal nodes, the height difference between its left and right subtree is at most one. We call this property *balance*. Remember that the height is the maximum length of a path to the root.

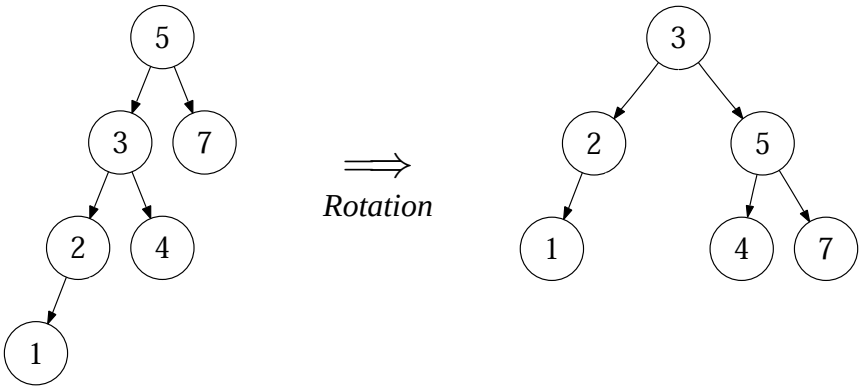
For example, the following binary search tree is an AVL tree:



Balanced AVL Tree

Nevertheless, if you insert 1, the tree becomes unbalanced.

In this case, we can rebalance the tree by doing a simple operation, called a rotation, as follows:



Unbalanced *Balanced*

See CLRS, p. 278 for the formal definition of rotations.

(a) If we insert a new element into an AVL tree of height 4, is one rotation sufficient to re-establish balance? Justify your answer.

(b) Denote the minimum number of nodes of an AVL tree of height h by $M(h)$. A tree of height 0 has one node, so $M(0) = 1$. What is $M(1)$? Give a recurrence for $M(h)$. Show that $M(h)$ is at least F_h, where F_h is the hth Fibonacci number.

(c) Denote by n the number of nodes in an AVL tree. Note that $n \geq M(h)$. Give an upper bound for the height of an AVL tree as a function of n.