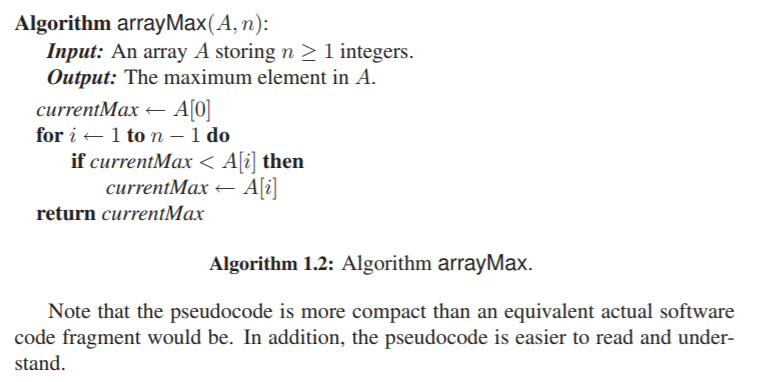
Algorithm Analysis

Pseudo code

**Expressions:** We use standard mathematical symbols to express numeric and Boolean expressions. We use the left arrow sign (←) as the assignment operator in assignment statements (equivalent to the = operator in C, C++, and Java) and we use the equal sign (=) as the equality relation in Boolean expressions (equivalent to the “==” relation in C, C++, and Java).

**Method declarations:** Algorithm name(param1, param2,...) declares a new method “name” and its parameters.

**Decision structures**: if condition then true-actions [else false-actions]. We use indentation to indicate what actions should be included in the true-actions and false-actions, and we assume Boolean operators allow for short-circuit evaluation.

**While-loops**: while condition do actions. We use indentation to indicate what actions should be included in the loop actions.

**Repeat-loops**: repeat actions until condition. We use indentation to indicate what actions should be included in the loop actions.

**For-loops**: for variable-increment-definition do actions. We use indentation to indicate what actions should be included among the loop actions.

**Array indexing**: A[i] represents the ith cell in the array A. We usually index the cells of an array A of size n from 1 to n, as in mathematics, but sometimes we instead such an array from 0 to n − 1, consistent with C, C++, and Java.

**Method calls**: object.method(args) (object is optional if it is understood).

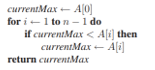
**Method returns:** return value. This operation returns the value specified to the method that called this one.

RAM Model

Define high-level primitive operations that are independent from programming language used and can be identified in pseudocode

* Assigning a value to a variable
* Calling a method
* Performing an arithmetic operation (for example, adding two numbers)
* Comparing two numbers
* Indexing into an array
* Following an object reference
* Returning from a method.

Instead of determining execution time of each operation, simply count how many are executed, and use number as high-level estimate of running time. Views computer as CPU connected to bank of memory cells containing words.

Showing how to count using example algorithm given above:

* Initializing the variable currentMax to A[0] corresponds to two primitive operations (indexing into an array and assigning a value to a variable) and is executed only once at the beginning of the algorithm. Thus, it contributes two units to the count.
* At the beginning of the for loop, counter i is initialized to 1. This action corresponds to executing one primitive operation (assigning a value to a variable).
* Before entering the body of the for loop, condition i<n is verified. This action corresponds to executing one primitive instruction (comparing two numbers). Since counter i starts at 1 and is incremented by 1 at the end of each iteration of the loop, the comparison I < n is performed n times. Thus, it contributes n units to the count.
* The body of the for loop is executed n − 1 times (for values 1, 2,...,n – 1 of the counter). At each iteration, A[i] is compared with currentMax (two primitive operations, indexing and comparing), A[i] is possibly assigned to currentMax (two primitive operations, indexing and assigning), and the counter i is incremented (two primitive operations, summing and assigning). Hence, at each iteration of the loop, either four or six primitive operations are performed, depending on whether A[i] ≤ currentMax or A[i] > currentMax. Therefore, the body of the loop contributes between 4(n − 1) and 6(n − 1) units to the count
* Returning the value of variable currentMax corresponds to one primitive operation, and is executed only once.

To summarize, the number of primitive operations t(n) executed by algorithm arrayMax is at least:

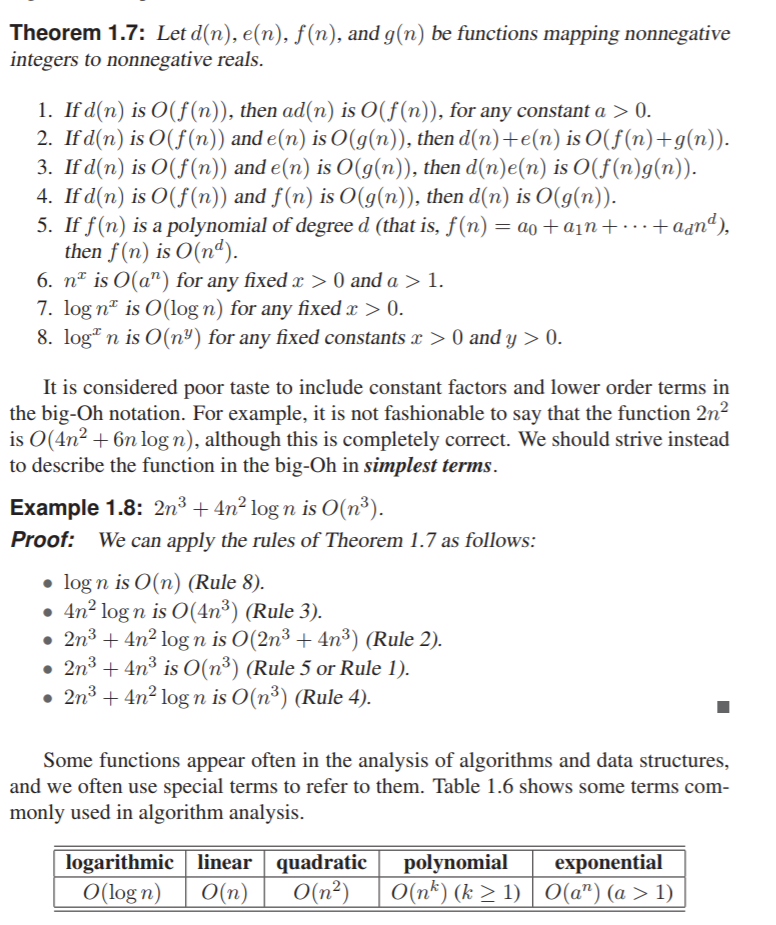
2+1+ n + 4(n − 1) + 1 = 5n

and at most

2 + 1 + n + 6(n − 1) + 1 = 7n – 2

Asymptotic Notation

Allows us to characterize the main factors affecting an algorithm’s running time without going into all the details of exactly how many primitive operations are performed for each constant-time set of instructions.

Big-Oh Notation

* Log(n), log2(n), √n, n, nlog(n), n2, n3, 2n

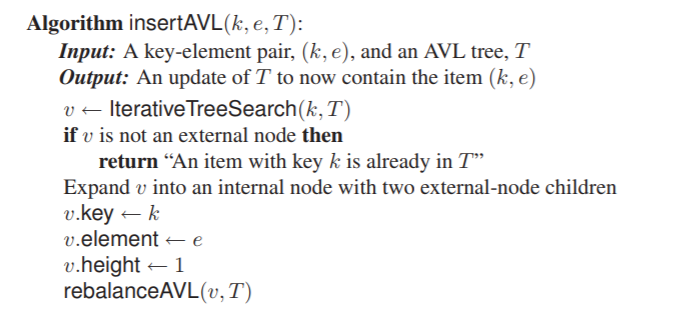
Amortization Notation

Allows us to characterize the main factors affecting an algorithm’s running time without going into all the details of exactly how many primitive operations are performed for each constant-time set of instructions.

* the worst-case running time of the series of operations divided by the number of operations

Accounting method:

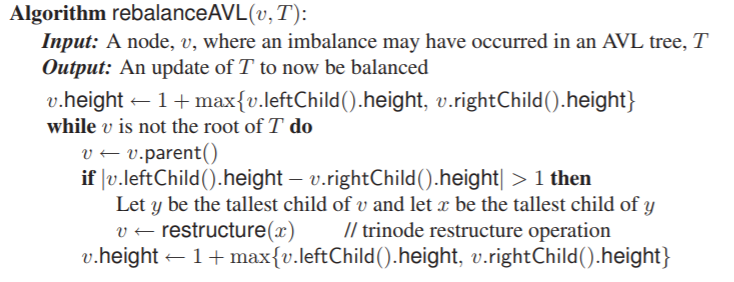
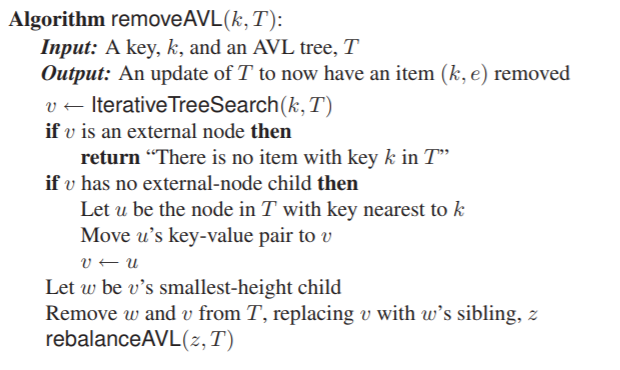
* We view the computer as a coin-operated appliance that requires the payment of 1 cyber-dollar for a constant amount of computing time. We also view an operation as a sequence of constant time primitive operations, which each cost 1 cyber-dollar to be executed. When an operation is executed, we should have enough cyber-dollars available to pay for its running time
* Of course, the most obvious approach is to charge an operation a number of cyber-dollars equal to the number of primitive operations performed.
* However, the interesting aspect of using the accounting method is that we do not have to be fair in the way we charge the operations. Namely, we can overcharge some operations that execute few primitive operations and use the profit made on them to help out other operations that execute many primitive operations. This mechanism may allow us to charge the same amount a of cyber-dollars to each operation in the series, without ever running out of cyber-dollars to pay for the computer time. Hence, if we can set up such a scheme, called an amortization scheme, we can say that each operation in the series has an amortized running time that is O(a).
* When designing an amortization scheme, it is often convenient to think of the unspent cyber-dollars as being “stored” in certain places of the data structure, for example, at the elements of a table.

Balanced Binary Search

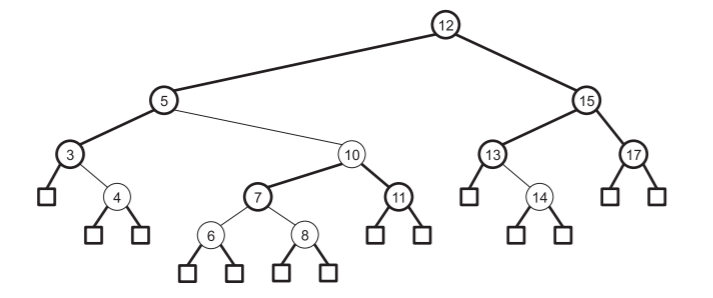
AVL Trees

Height-balance property:

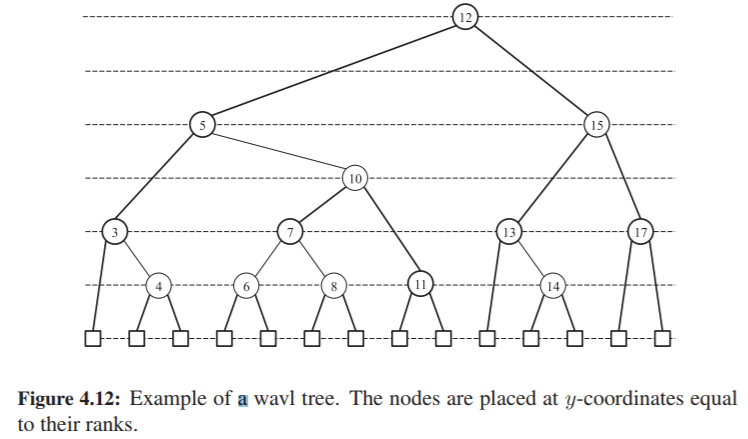
* For every internal node, V, the heights of v may differ by at most 1
* Any subtree of AVT tree is AVT tree



Red-Black Trees

Properties

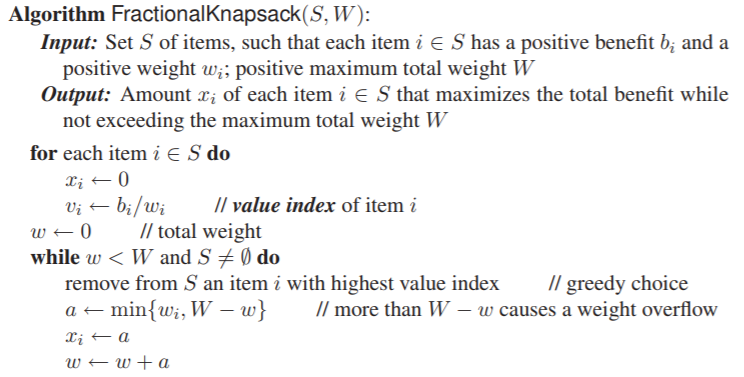
* Every external node is black
* The children of a red node are black.
* All the external nodes have the same black depth, that is, the same number of black nodes as proper ancestors.

Weak Avl Trees

Properties:

* Rank-difference property: Renk difference of any non-root node is 1 or 2
* External-node property: Every external node has rank 0
* Internal-node property: An internal node with two external-node children cannot be a 2, 2-node

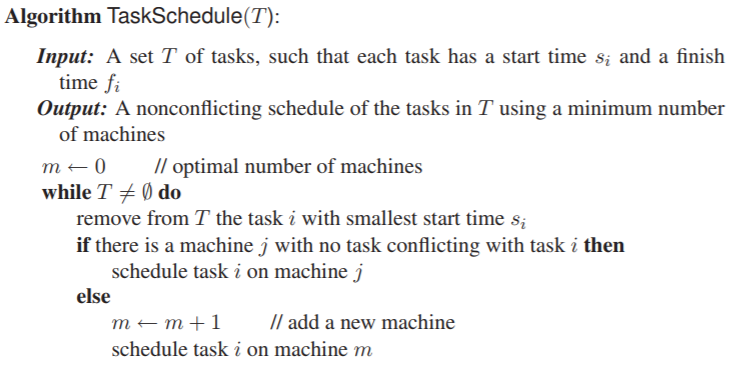
Greedy Methods

In order to solve a given optimization problem, proceed by a sequence of choices that start from well-understood starting configuration, then iteratively make the decision that is best from all of those, in terms of improving objective function.

Knapsack problem:

* Given a set S of n items, such that each item I has positive benefit bi and a positive weight wi, we wish to find maximum-benefit subset that does not exceed weight W.

Task scheduling:

*  Given set of tasks such that each has a start time and a finish time. Each task has to be performed on a machine and each machine can executive one task at a time. Two tasks are said to be nonconflicting if they do not overlap. Two tasks can be scheduled to be executed if they are nonconflicting.
* We consider scheduling tasks on fewest machines possible in nonconflicting way.