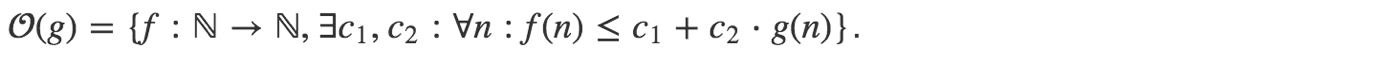
Algorithems:

A better algorithm is one that is faster for a big number of entries.

When we write an algorithm we can leave every constant out.



every function in O(g) grows asymptotically at most as fast as g.

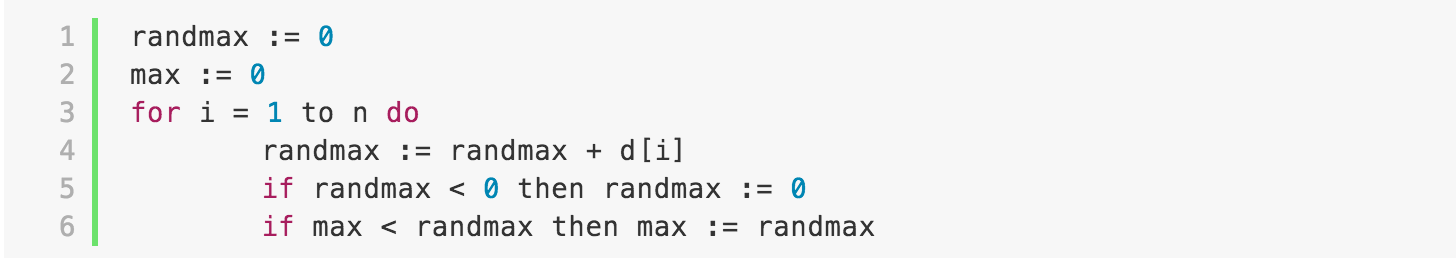
every function in Ω(g) grows at least as fast as g.

every function in θ(g) grows exactly like g. (O(g) combined withΩ(g) )

what makes an algorithmus good :

* It works
* Time and space consumption
* Quality of the result

Maximal subarray problem = kendanes algorithem:



searching something in an ordered array:

binary search (divide in to find the good side and recursively search there)

there is no faster algorithmus the binary search in an ordered array.

If the array is too big (infinit) we can use an exponential search to find the key:

We start with an array [1,1] and keep doubling it ([1,2] [1,4]) until we are shure that this array contains a number bigger then the number we are looking for then we do a binary search on this array.

Time complexity: If the loops are in each other multiply them, if they are one after the other you take the bigger one.

Telescoping:

Replace the T(n/m) by itself with m to the power of something, the do the sum of the rest, replace n with m to the power of the same thing and replace the value of T(1). Use the formule for the sum. Then replace the the power with n. Then show with induction. To show the induction step rereplace n with m to the power of k and show for m to the power of k+1.

Searching in an unordered array:

Quickselect:

Select a pivot put everything smaller on the left everything bigger on the right if the pivot has at least n/4 elements on both sides then ok or else start again with another pivot. Recursive on good side afterwards.

Median of medians:

Divide the array in to groups of 5 find the median of each group and find the median of these medians recursively. This will be a good pivot.

Hashing:

Important in hashing is search insert delete.



hashing function that transforms the value into a shorter key threw a function.

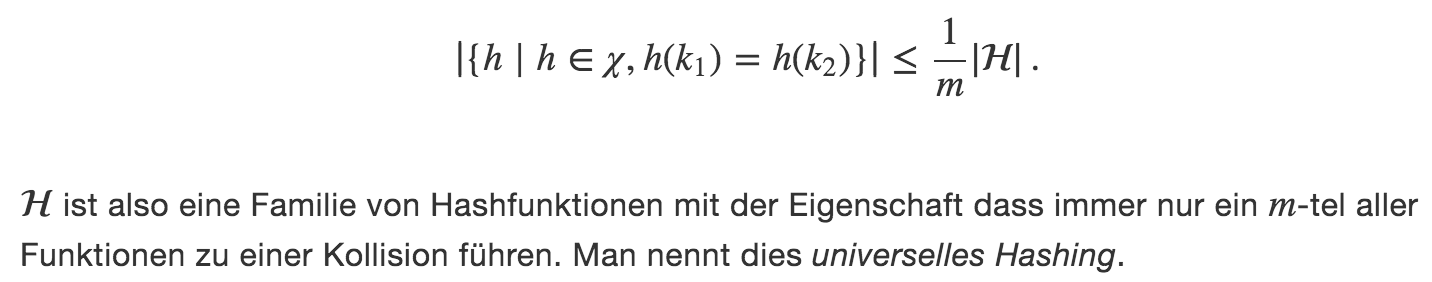
Problem with hashing is that there are often collisions (two different inputs hash to the same key).

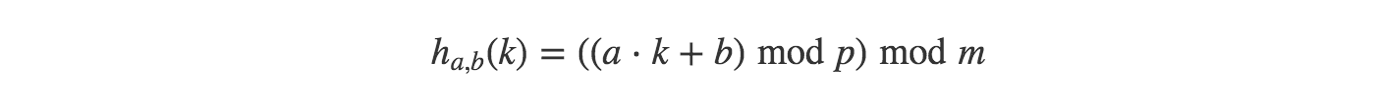
Two popular hash function:

* H(k) = k mod m
* H(k) =/Users/jakubkotal/Desktop/Screen Shot 2016-04-05 at 14.34.04.png

Perfect hashing is when there are now collisions.

Universal hashing is selecting a hash function from a family of hash functions that satisfy:

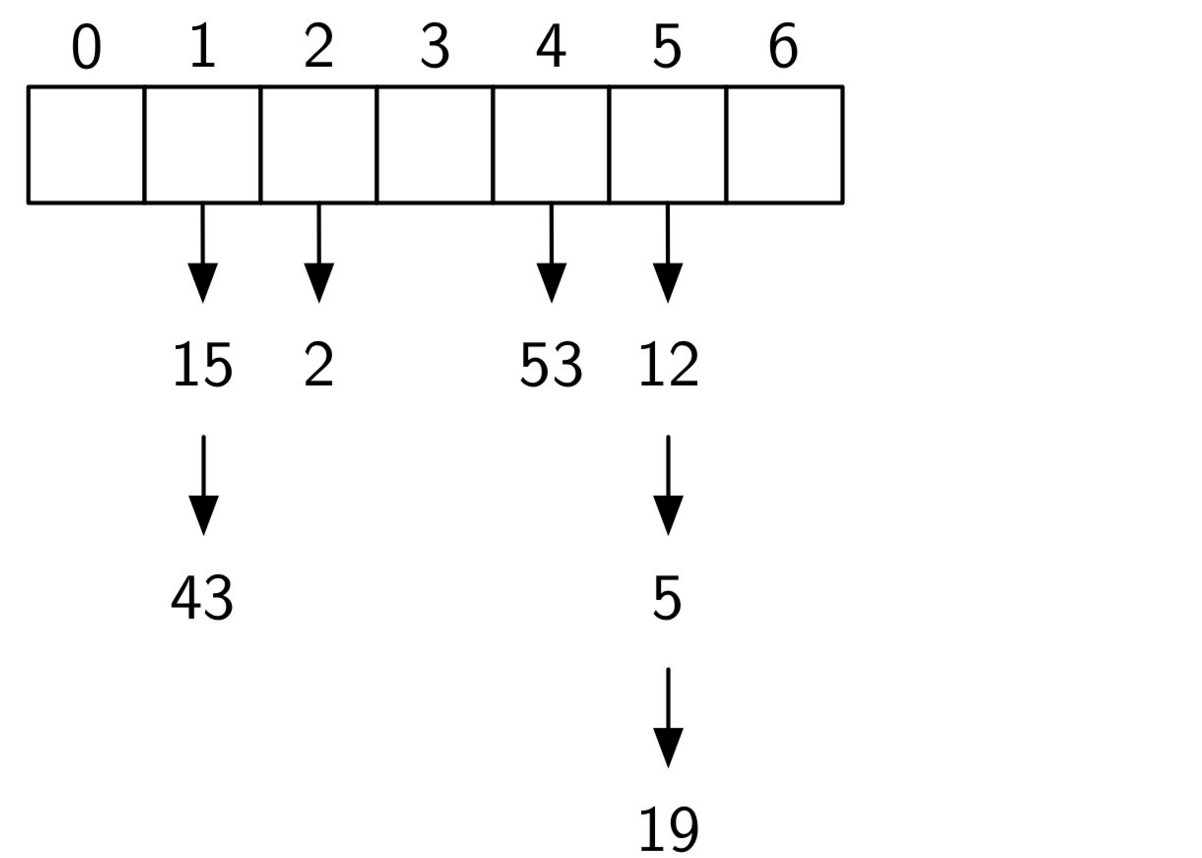
example of such a family:



collision handling:

chaining:

an array of array where if two input have the same key the second input will go into the 2 dimension of the array ex:



open addressing, sondieren:

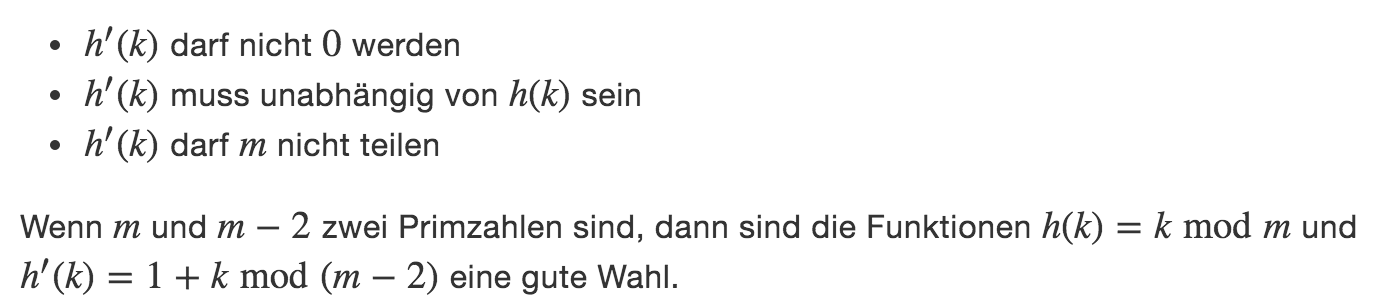
if h(k) is taken then just go to h(k)+1 or -1

quadratic sondering:

/Users/jakubkotal/Desktop/Screen Shot 2016-04-05 at 14.49.14.png

double hashing:

use a second hash function if a collision occurs



Sorting an unsorted array:

Bubble sort:

Take the key from the left if it is bigger then the one on it’s right then swap the two keys and so on until the key on it’s right is bigger and so on. It’s easily parelizeable.

O-(n^2)

Selection sort:

Go threw the list and find the smallest element and switch it with the element that is most on the left. The do the same for the list n-1 and so on.

O-(n^2)

Insertion sort:

Insertion sort [iterates](https://en.wikipedia.org/wiki/Iteration), consuming one input element each repetition, and growing a sorted output list. Each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

<https://en.wikipedia.org/wiki/Insertion_sort>

O(n log n) comparisons and O-(n^2) exchanges therefore time O-(n^2)

Heap sort:

Build a tree out of the array and if child is bigger than parent swap the two.

O(n log n)

Min/Max Heap:

Build a tree with the values of the array in order. Start with the index half of the length of the array, and in the tree if at this index the children of the node are bigger than the node key, then swap the biggest children node with the parent, so on until the root of the tree and then check if the children are always smaller than the parent nodes else swap.

Merge sort:

Divide into small ordered parts (divide and conquer) compare smallest element of both parts and the smaller one of both goes to the left and do the same until you only have one array left.

Exchanges O-(n log n)

Extra place O-(n)

Quicksort:

Like quickselect but repeat recursively on both sides to sort the whole array.

Best case O-(n log n)

Worst case O-(n^2)

Middle case O-(n log n)

Go from left and find the first element that is bigger than the pivot, from the right side find the first element that is smaller than the pivot and then switch the two.

It is impossible to sort an array faster than n log n (demonstration chap 7)

Bucketsort:

For a large number of numbers we can divide the numbers into multiple buckets with each bucket containing only the number that have a criteria. For Ex: 83405. Can go to the bucket 834 because it begins with these numbers. We can afterwards sort the buckets trivially (for ex quicksort).

Radix sort:

Sort the numbers into ten groups according to the last or first number then sort these groups.

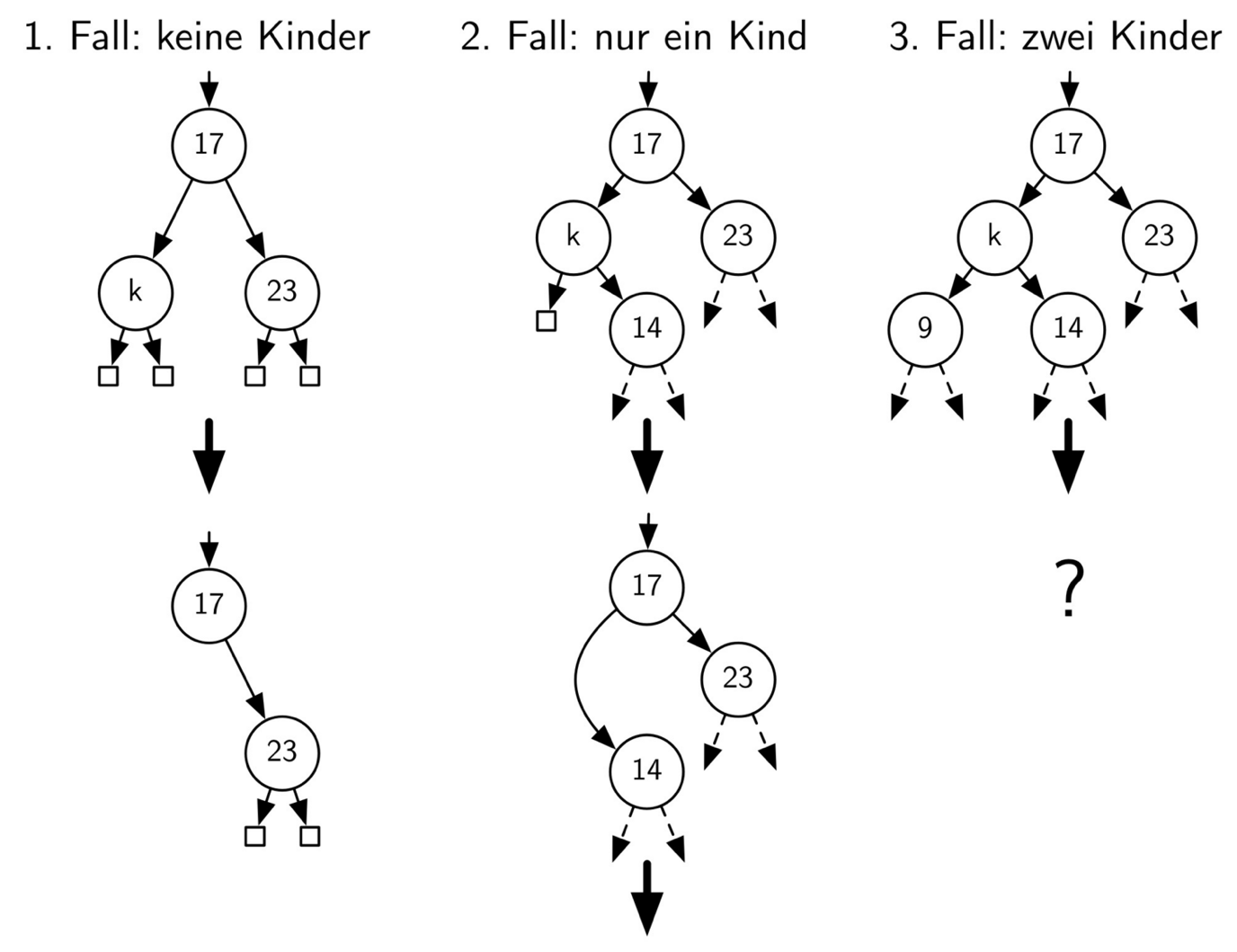
A stable sorting algorithm doesn’t change the order of two numbers if they have the same value.

Search Trees:

Insert :

If the input array is bad than the search tree can degenerate to a normal linear list and therefore have an insert time of O-(n).

Delete:



for the third case find the biggest child of the one on the left or the smallest child of the one on the right and replace with this one.

O-(size of the tree)

Worst case: O-(n)

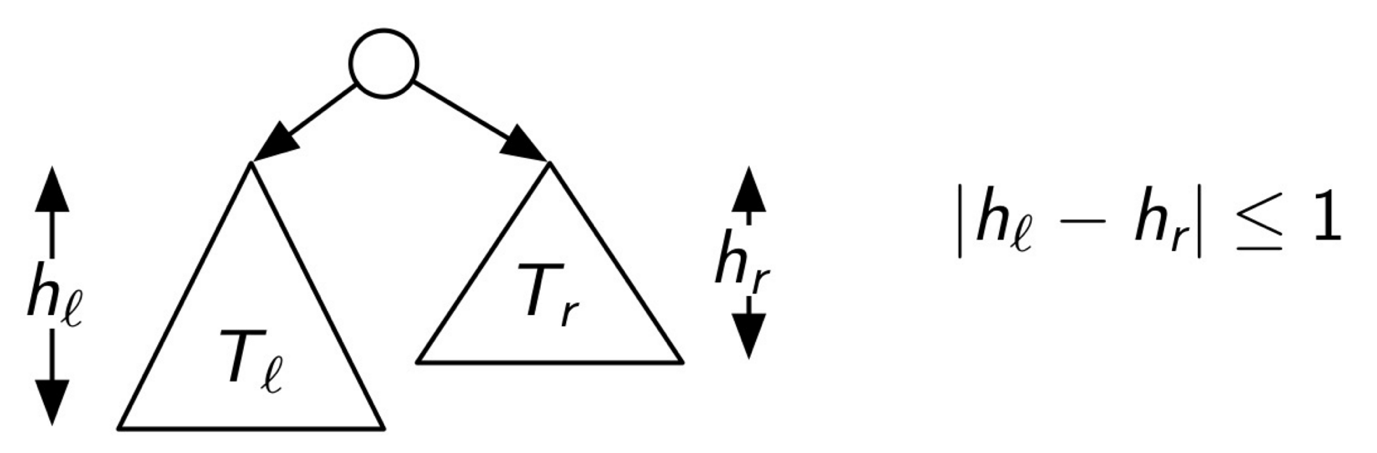
Average case: O-(log n)

Avl-search Trees:

Avl-search trees garantie are similar to search trees but they garantie a worst case search time of O-(log n).

For every not the size of the left child and the size of the right child can’t vary buy more than one:

Left left, left right, right right. Right



Insert:

Insert in an avl-tree is similar to the insert in a normal search tree but after every insert we have to rebalance the tree:

Every time that the difference between the hight of the children is bigger than one you have to take the children which are too big and invers set the middle child as the parent and the parent becomes the child of this one.(see example for explanation exercice 4).

Search: O(log n)

Insert : O(log n)

Splay trees:

Similar to avl trees but each time you perform an operation on the tree you move a node up the tree so it will be faster to get the next time.

Amortized analysis:

Elle consiste essentiellement à majorer le coût cumulé d'une suite d'opérations pour attribuer à chaque opération la moyenne de cette majoration, en prenant en compte le fait que les cas chers surviennent rarement et isolément et compensent les cas bon marché. Pour être utilisable, cette analyse suppose que l'on est capable de borner la fréquence des cas les plus coûteux. L'analyse amortie se place dans le cas le plus défavorable et garantit *la performance moyenne de chaque opération dans ce cas*[1](https://fr.wikipedia.org/wiki/Analyse_amortie#cite_note-1). À partir de l'analyse amortie on peut concevoir des structures de données efficaces.

Transpose:

If we have a list for wich we search elements in often. Every time an element is searched we move this one on spot to the left every time it is searched for so that the more popular elements will be found faster.

Tree traversal:

Preorder: vlr (value then left then right)

Postorder: lrv(first left then right then value)

Inorder: gives the nodes in increasing order.

Interval trees:

Max number of points = 2\*(closest power of two of the number of elements) – 1.

Closest power e.x if number of elements is 5 then 2^3 = 8 if number of elements 4 then 2^2 =4

B trees:

A B trees of order m, each node has at most m descendants and therefore each node has at most m-1 numbers. The bottom level of a b tree has to be full. Every node other than the root has t-1 children and therefore at least t keys and at most 2t children and 2t-1 keys.

Insertion: We insert the number in the correct node, if it is full we split the node in two and the median goes up a level.

Deletion:

If the node contains minimum number of keys and another node on the same level contains more than the minimum number of keys, then take a value from the root to the node where we are deleting and put a value in the root from the node with more keys. Then delete the value.

If all the nodes on this level contain the minimum number of keys. Merge two nodes together.

Graphs:

Spanning tree:

Sub graph that connects all vertices.

Prim’s algorithm to find minimal spanning tree(mst):

Pick a vertice from this vertice pick an Edge with the minimal cost and that leads to a vertic that isn’t in the visited set. And so on.

Kruskal’s algorithm:

Choose the smallest edge that does not connect two vertices that are already visited and add these two vertices to the visited set, and so on.

Topologic ordering:

Is an ordering where for every edge from u to v u comes before v(tasks where edges show which has to be done before which), a topologic ordering is possible if and only if there is no directed cycle.

Dijkstra:

Runtime : O(|E| + |V| log (|V|))

Bellman ford:

If slower than Dijkstra but works with negative edges runtime O(mn).

Flow:

Ford Fulkerson(runtime O(E max flow)), find augmenting path don’t forget backwards edges until the graph is full.

Dynamic programming:

For a problem to be resolvable using dynamic programming it has to have an optimal substructure (the answer must be derivable from the answers of its sub problems).

To solve it make an array that solves it’s sub problems by using the answer from the solution of a smaller sub problem and so on until you solve the whole thing.

Case 1: Placing objects somewhere to find the min amount of objects. initialize the table with value infinity ans use the minimum or max to find the answer.

Case 2: adding stuff the make a total amount. Matrix one side contains the elements and the other all the possible amounts you can make with these elements.