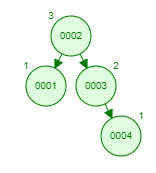
***R-4.4*** A certain Professor Amongus claims that the order in which a fixed set of elements

is inserted into an AVL tree does not matter—the same tree results every

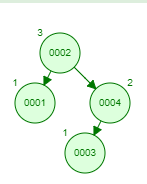
time. Give a small example that proves Professor Amongus wrong.

**Answer:** Amongus is incorrect and a small example to respectfully prove professor Amongus incorrect would be to point on that the insertion of 4 values {1,2,3,4} would not lead to the same binary tree as insertion of {2,1,4,3}

For AVL Tree Insertion of {1,2,3,4}



For AVL Tree Insertion of {2,1,4,3}



For diagrams AVL Simulator is used- <https://www.cs.usfca.edu/~galles/visualization/AVLtree.html>

***R-4.7*** What is the minimum number of nodes in a red-black tree of height 8?

**Answer:** The red black tree is a binary search tree that is self-balancing where the node is either red or black. A red black tree with nodes n and height 8 is

h≤ 2 Log2 (n+1),

8≤ 2 Log2 (n+1),

8/2 ≤ Log2 (n+1),

For the minimum number of nodes so n is minimum therefore,

4 = Log2 (n+1),

(n+1) = 24,

n+1 = 16,

n =15.

Hence, minimum number of nodes in a red-black tree of height 8 is 15.

***C-4.15*** A *mergeable heap* supports operations insert (*k, x*), remove(*k*), unionWith(*h*),

and min (), where the unionWith(*h*) operation performs a union of the mergeable

heap *h* with the present one, destroying the old versions of both, and min () returns

the element with minimum key. Describe an implementation of a mergeable

heap that achieves *O* (log *n*) performance for all its operations. For simplicity,

you may assume that all keys in existing mergeable heaps are distinct, although

this is not strictly necessary.

**Answer:** In mergeable heap Binary heap achieves O (log n) for most operations but, Binomial heap achieves O (log n) performance for all operations except Make-Heap.

Insert (k,x) inserts node x, whose key field has already been filled in, into heap k. a new element is inserted to a heap by creating a new heap by adding the inserted element to original heap.

remove(k)/delete deletes the node k from the heap, the key in the heap is decreased to negative infinity- lower than smallest value in the heap and then delete the minimum value in the him.

unionWith(h)/merge it creates and returns the value where two values of the trees that has same order is merged. The root values are compared smaller root value from two becomes the minimum key and becomes new root node. In short two heaps are merged in this h1 and h2 would be destroyed and there would be just h containing all the values of both heaps.

min() returns the pointer position to node in heap whose key is minimum.

***A-4.4*** Suppose you are working for a victim-support group to build a website for maintaining

a set, *S*, containing the names of all the registered sex offenders in a given

area. The system should be able to list out the names of the people in *S* ordered

by their Zip codes, and, within each Zip code, ordered alphabetically. It should

also, be able to list out the names of the people in *S* just for a given Zip code. The

running time for a full listing should be *O*(*n*), where *n* is the number of people in

*S*, and the running time for a listing for a given Zip code should be *O* (log *n*+*s*),

where *s* is the number of names returned. Insertions and removals from *S* should

run in *O* (log *n*) time. Describe a scheme for achieving these bounds.

Answer:

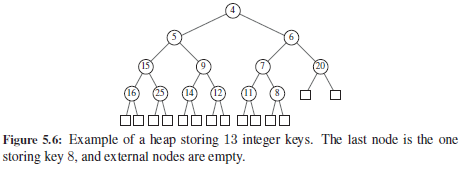
***R-5.6*** Give an example of a worst-case list with *n* elements for insertion-sort, and show

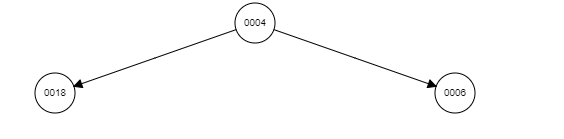
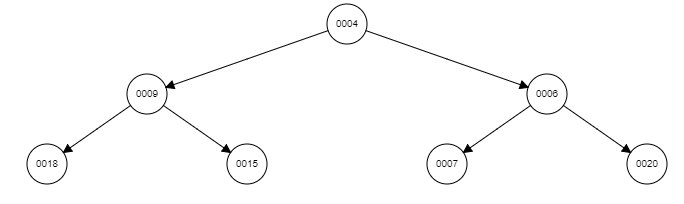
that insertion-sort runs in Ω(*n*2) time on such a list.

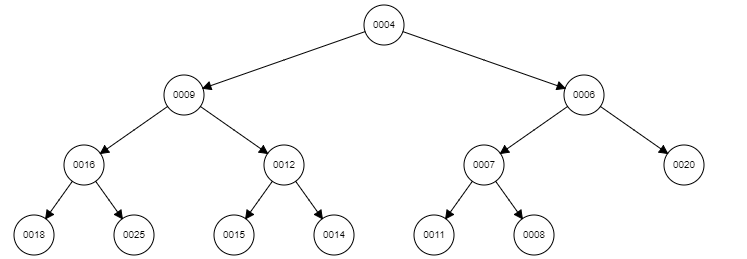
**Answer:** An example of a worst-case list with n elements for insertion-sort can be n, n-1, n-2, … 1 or (9,7,5,3,1). In the n sequence n-1 = 1, n-2 =2, n-3 =3 and it goes on. Hence, comparisons are obtained 1+2+3+4+5+…. +n whose sum would be n(n+1)/2. So, the complexity becomes Ω(*n*2)

R-5.14 Show the steps for replacing 5 with 18 in the heap of Figure 5.6.

Answer:





***C-5.9*** Let *T* be a heap storing *n* keys. Give an efficient algorithm for reporting all the keys in *T* that are smaller than or equal to a given query key *x* (which is not necessarily in *T*). For example, given the heap of Figure 5.6 and query key *x* = 7, the algorithm should report 4, 5, 6, 7. Note that the keys do not need to be reported in sorted order. Ideally, your algorithm should run in *O*(*k*) time, where

*k* is the number of keys reported.

**Answer:** SmallerThan(Node node, Object x )

{

If ( node.isInternal() && node.element().isLessThanOrEqualTo( x ))

{

System.out.println(node.element );

SmallerThan(node.leftChild(), x );

SmallerThan(node.rightChild(), x );

}

}

The algorithm starts at the root of the tree. When left or right subtree/left or right child has a value less than x of k than it would search both subtrees. This algorithm runs on O(k).

***A-5.3*** Suppose you work for a major airline and are given the job of writing the algorithm

for processing upgrades into first class on various flights. Any frequent

flyer can request an upgrade for his or her up-coming flight using this online

system. Frequent flyers have different priorities, which are determined first by

frequent flyer status (which can be, in order, silver, gold, platinum, and super)

and then, if there are ties, by length of time in the waiting list. In addition, at any time prior to the flight, a frequent flyer can cancel his or her upgrade request (for instance, if he or she wants to take a different flight), using a confirmation code they got when he or she made his or her upgrade request. When it is time to determine upgrades for a flight that is about to depart, the gate agents inform the system of the number, *k*, of seats available in first class, and it needs to match those seats with the *k* highest-priority passengers on the waiting list. Describe a system that can process upgrade requests and cancellations in *O*(log *n*) time and can determine the *k* highest-priority flyers on the waiting list in *O*(*k* log *n*) time, where *n* is the number of frequent flyers on the waiting list.

**Answer:**

The algorithm for this task can be created using the maximum heap.

If the user requests to upgrade or cancel it would take O (log n).

Hashmap can be used to map the priority queue data structure which would have the data values such as flyer number, status as the key and the priority queue would be the value.