CMSC 641 Design & Analysis of Algorithms Spring 2023
Section 01 Homework 5

Faisal Rasheed Khan

VB02734

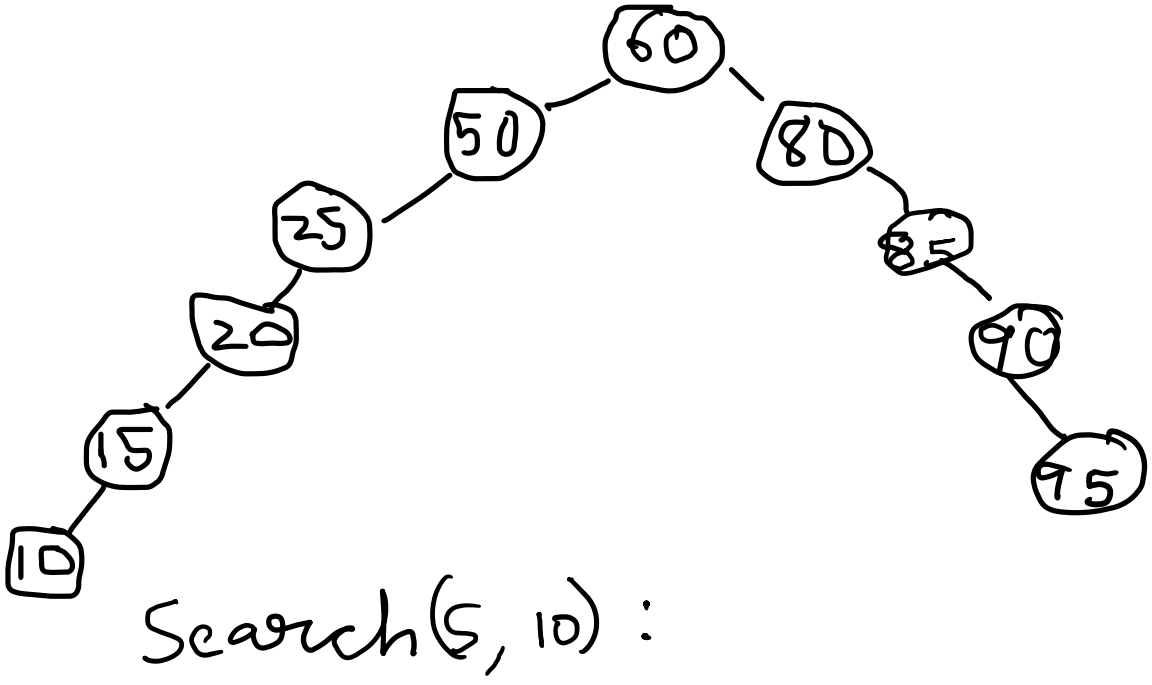
vb02734@umbc.edu

1 Chang's Folly

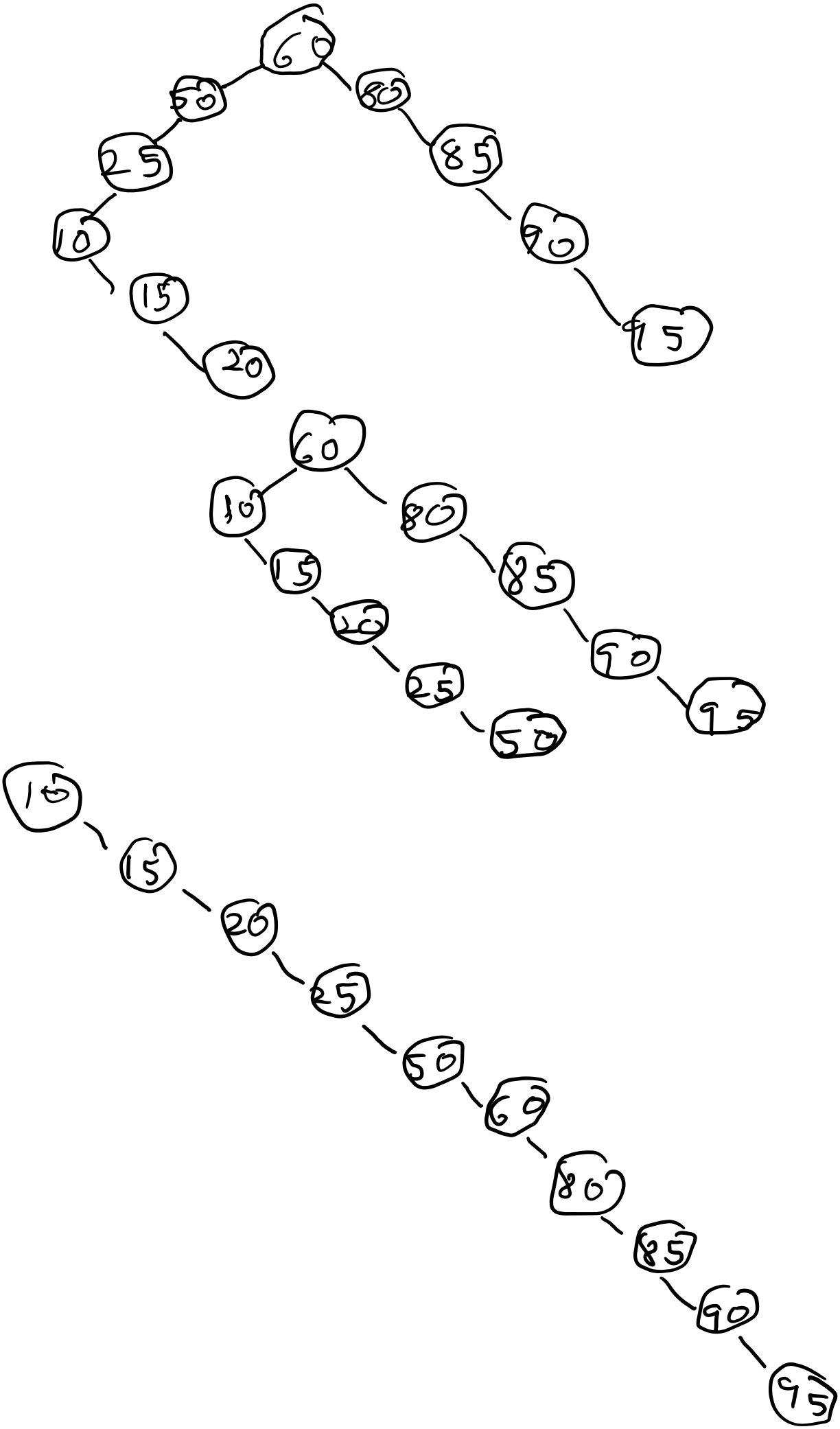
In a previous semester, Prof. Chang suggested in lecture that as operations are applied to a splay tree, it becomes more and more balanced. Show that this is not necessarily true by providing a sequence of search() operations that will turn any n node splay tree into a right-going chain. That is, in the resulting tree, all of the nodes are along the path starting at the root that keeps visiting the right child.

Briefly justify your answer. Why does your example not contradict the amortized analysis?

A. Operations applied to splay tree becomes more and more balanced is not necessarily true, as sequence of search operations turns n node splay tree into right going chain, in the next step after becoming the right going chain, search operation takes worst n time to find the element assuming if the element is the last node in the right going chain.
Consider the following sequence and search(s,10):



Search for the element 10 and after that apply splay operations such that 10 becomes root


60
50
80
25
85
10
90
15
95
20
60
10
80
5
85
15
90
25
95
50
10
15
20
25
50
60
80
85
90
95

The Ammortized Analysis of Splay operation is O(log n) where we define:
Size(x) : # of nodes in subtree rooted at x
Rank(x) : $\lfloor log(Size(x)) \rfloor$
Rank(S) : $\lfloor log(|S|) \rfloor$
Invariant : Each node x has rank(x) credits
Need to maintain the invariant.
The reason why my example doesnot contradicts amortized analysis because Amortized analysis considers the average # of the operations, above example applies splay operation 3 times i.e. log n time amortized. After each rotation we will add or subtract credits at nodes where rank has changes such that invariant is maintained.

2 Alternate Splay Tree Insertion

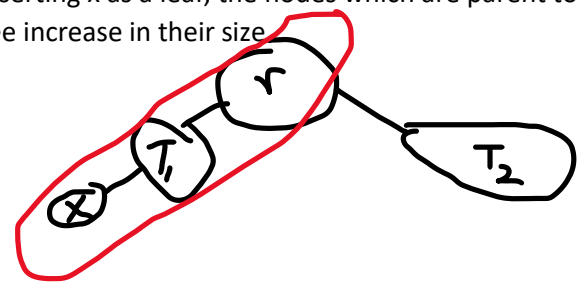
Here is an alternate implementation of insert for Splay trees. First, insert x as you would normally do in a binary search tree. After this insertion x becomes a leaf. Then, rotate x as you would in the splay operation until x is the root of the tree.

This implementation is actually different from the one discussed in lecture. In this implementation, x becomes the root of the tree, which is not the case in the original implementation.

Fortunately, we do not have to redo the entire amortized analysis of Splay trees for this change, because the amortized analysis is valid as long as each node in the tree has enough credits (i.e., as long as the invariant holds). It does not matter that the tree was not constructed using splay tree operations. We just have to make sure that every node y still has rank(y) credits after we insert x at the leaf (and before we run the splay operation).

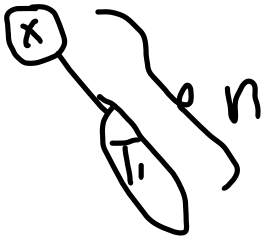
1. After inserting x as a leaf in the alternate implementation of insert, which nodes might see an increase in their size?

A. After inserting x as a leaf, the nodes which are parent to x and its parents up to root (i.e. x to root) see increase in their size



2. How many nodes are there that see an increase in their size? Briefly justify your answer. Note: recall that the height of a splay tree could be linear.

A. The number of nodes that see increase in their size is n (linear) worst case considering the splay is a linear chain. Then the size is worst case n.



3. How many nodes are there that see an increase in their rank? Briefly justify your answer. Note: it is possible that a node's size increases, but its rank does not.

A. The rank of x rooted is defined as, Rank(x) : $\lfloor log(Size(x)) \rfloor$
In the worst case n its still will not exceed log n.
Therefore, log n nodes see an increase in their rank.

4. If a node's rank does increase, how much can the rank change? Briefly justify your answer.

A. We are inserting x at the leaf node, so if the rank increase, it increases by at most 1. The length of the path increases by 1 and Rank(x) : $\lfloor log(Size(x)) \rfloor$
Therefore, rank will increase at most one time.

5. How many credits would you need to add to make sure that every node y still has rank(y) credits? Briefly justify your answer.

A. For rank(y) credits to be left, we need to add extra is log(n) credits such that the variant is maintained such that rotation uses not more than 3(rank(s) – rank(x))+1 credits
The additional credits added (log n) maintains the variant such that the operations take amortized time.

6. What is the amortized running time of the entire alternate insert operation (including inserting x at the leaf and splaying at x)? Briefly justify your answer.

A. <u>Proof of Induction on alternate insert operation (including inserting x at leaf and splaying at x)</u>:

<u>Induction Hypothesis P(n):</u>
The insertion of binary search tree T with n items is O(log n) amortized for alternate splay insertion.

The Amortized Analysis of Splay operation is O(log n) where we define:
Size(x) : # of nodes in subtree rooted at x
Rank(x) : $\lfloor log(Size(x)) \rfloor$
Rank(S) : $\lfloor log(|S|) \rfloor$
Invariant : Each node x has rank(x) credits
Need to maintain the invariant.

<u>Base Case:</u>
P(1) holds since insertion of a binary search tree and splay operation with one element is log 1 constant time

<u>Induction Case:</u>
Assume P(k) holds for all k<n, k less than n items. We want to show that P(n) holds.

Consider k items =n-1

By Inductive Hypothesis, insertion of binary search=log(k)

=log(n-1)

Splay Operation=log(k)

=log(n-1)

By induction hypothesis, the subtree with k items has height O(log k)=O(log(n/2))=O(log n)

Thus,

insertion of binary search tree T for n items=O(log(n-1 + 1))

=O(log n)

Splay Operation for n items=O(log(n-1 + 1))

=O(log n)

Therefore, P(n) holds.

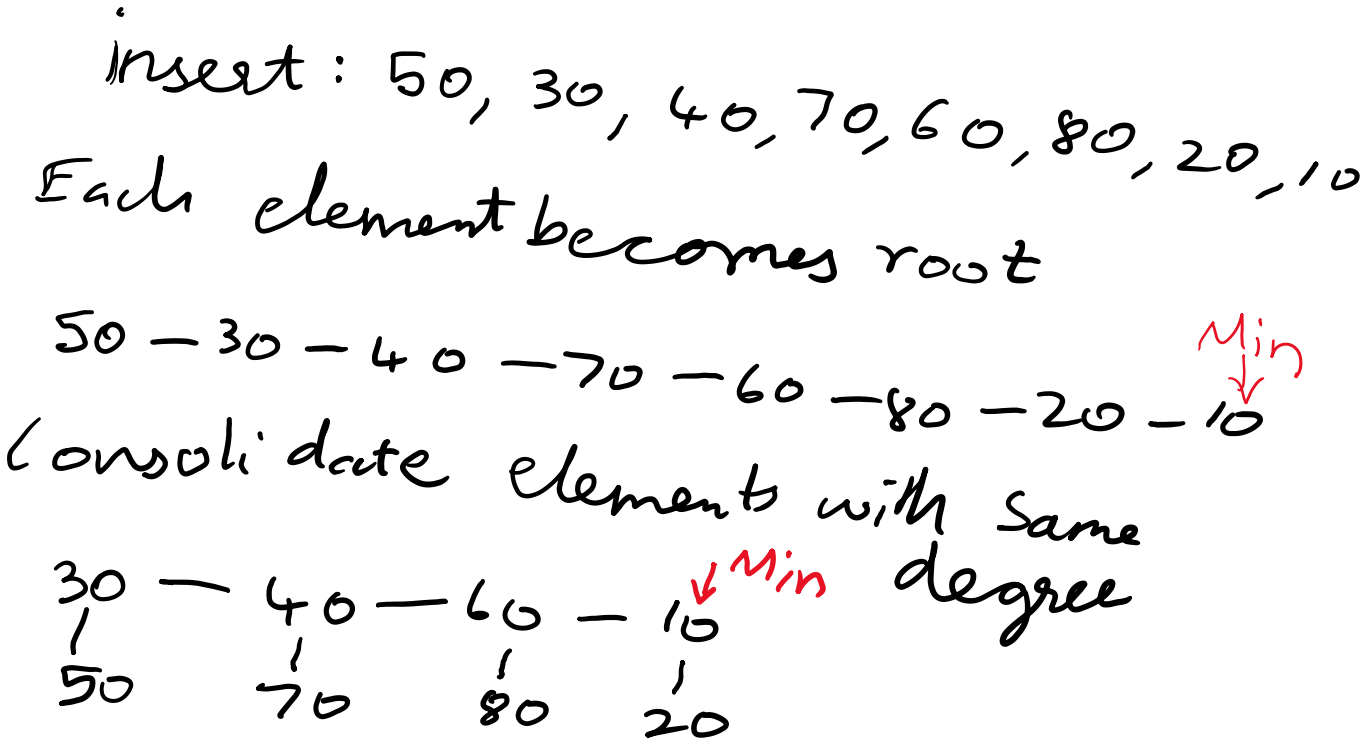
alternate insert operation (including inserting x at the leaf and splaying at x)=O(log n)

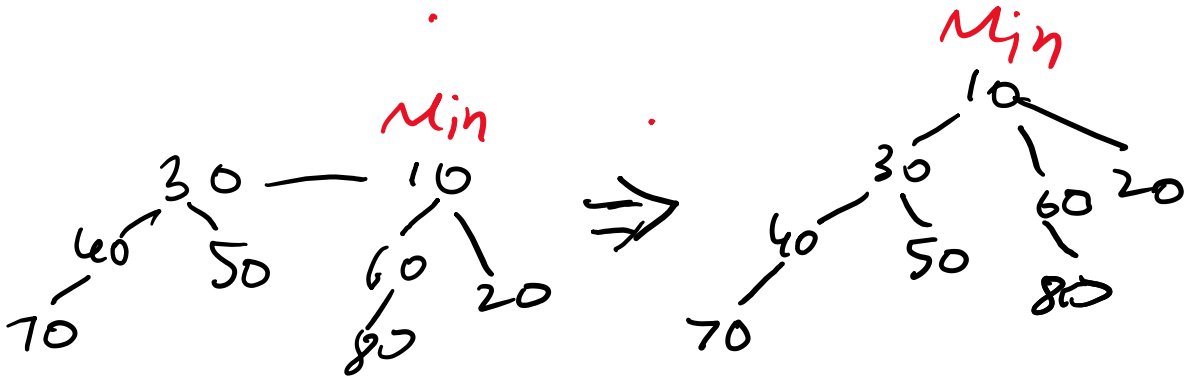
3 Tall Skinny Fibonacci Heap

Specify a sequence of Insert, Decrease-Key and Extract-Min operations that would create a Fibonacci Heap with 4 nodes in a single chain:

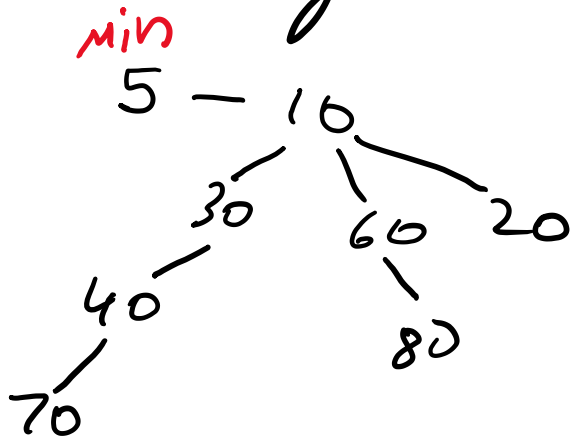
Show what your Fibonacci Heap looks like at important steps (i.e., you can bunch together the boring parts). Use actual numbers for the items

A. :

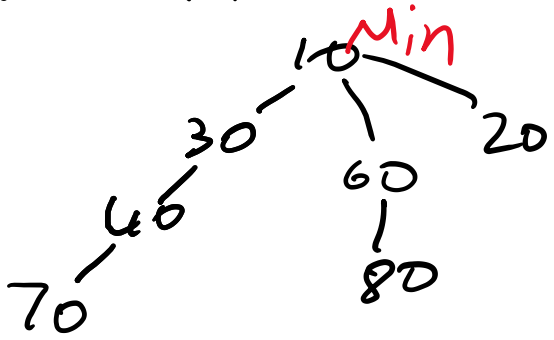




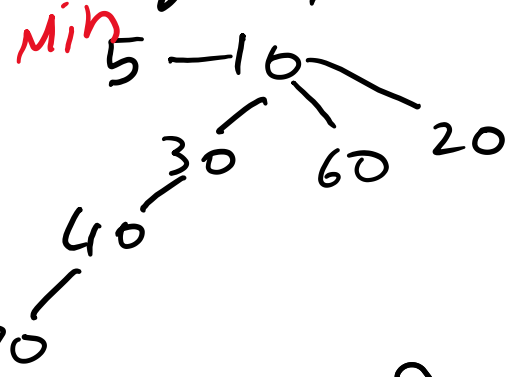
decrease key (50, 5):



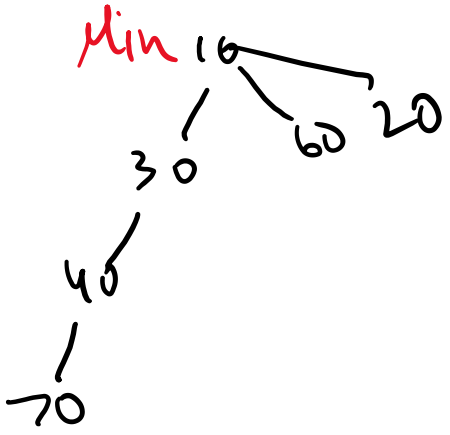
Extract Min:



Decrease key (80, 5):

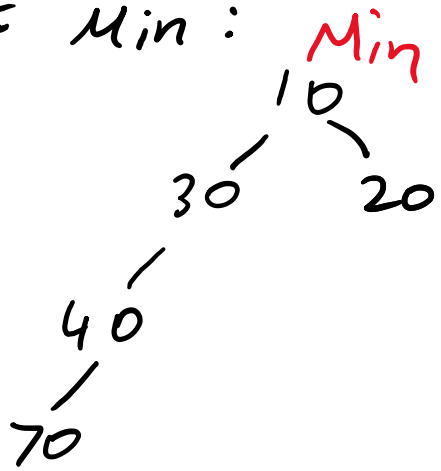


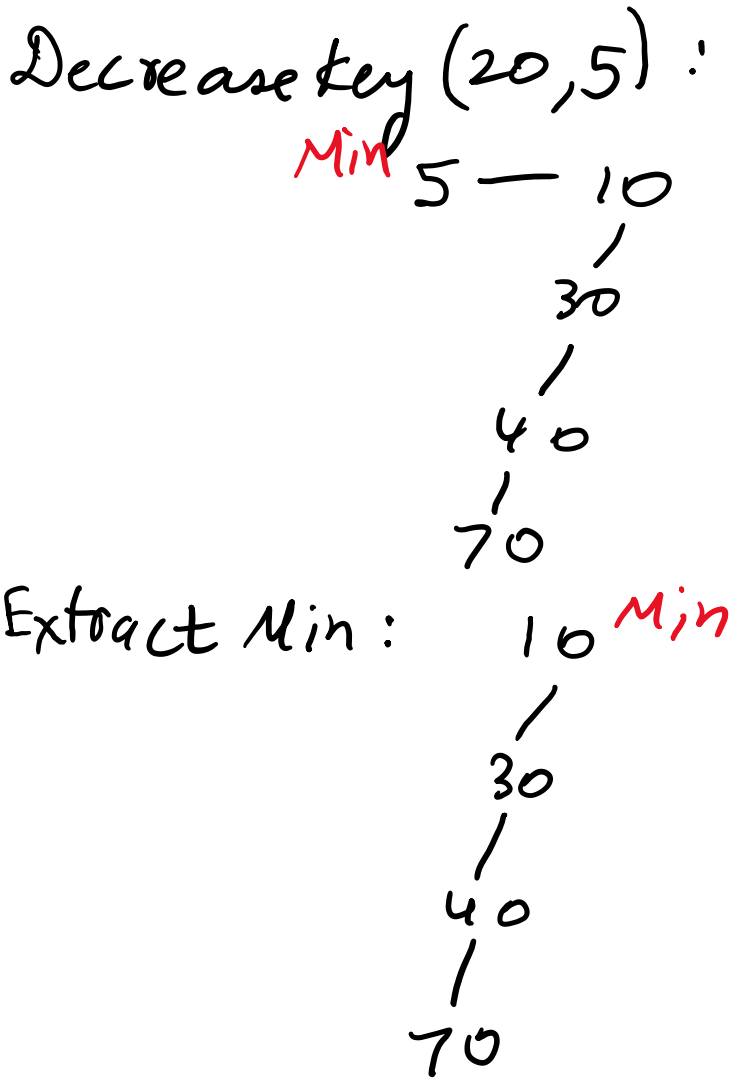
Extract Min:



⇒ Decrease key (60, 5)

Extract Min:





References:

https://archive.org/details/introduction-to-algorithms-by-thomas-h.-cormen-charles-e.-leiserson-ronald.pdf/page/427/mode/2up

https://archive.org/details/AlgorithmDesign1stEditionByJonKleinbergAndEvaTardos2005PDF/page/n201/mode/2up

09-Fibonacci-Heaps-post.pdf - Google Drive

06.1-Amortized-Analysis-post.pdf - Google Drive

06.2-Skew-Heaps-post.pdf - Google Drive

CMSC 641-01 Spring 2023 Shared Folder - Google Drive