Lecture Notes – Binary Heap

Consider how we can find the largest value in a list in Python if we do not have the max( ) function.

```
# Python3 program to find maximum
# in arr[] of size n

# python function to find maximum
# in arr[] of size n
def largest(arr,n):

    # Initialize maximum element
    max = arr[0]

    # Traverse array elements from second
    # and compare every element with
    # current max
    for i in range(1, n):
        if arr[i] > max:
            max = arr[i]
    return max

# Driver Code
arr = [10, 324, 45, 90, 9808]
n = len(arr)
Ans = largest(arr,n)
print ("Largest in given array is", Ans)

# This code is contributed by Smitha Dinesh Semwal
```

Big O notation of this *largest* function is *O(n)*.

Consider the Linked List project (project 6). You implemented an ordered list (priority list according to the student's name – alphabetically).
Is there a way to organize your data structure to make the list operation of *O(n)* to move closer to *O(log n)*?

Complete Binary Tree – is a binary tree that has the maximum number of nodes / entries for its height, i.e. the tree is full and all leaves are at the bottom level.

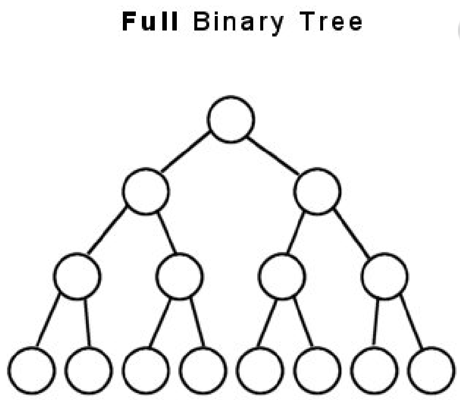


image from web.cecs.pdx.edu

Almost Complete Binary Tree / Nearly Complete Binary Tree – is a binary tree that has all its leaves in either bottom level or (bottom-1) level. At the bottom level, all leaf nodes are found on the left.

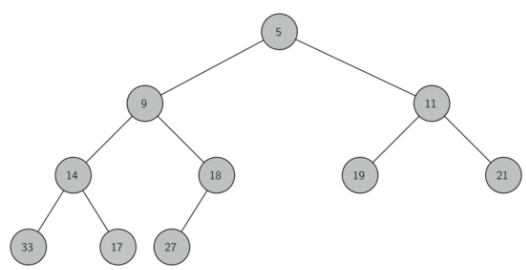


image from e-text section 7-10

Then we ask 2 questions:

1. Consider the priority queue / priority list scenario. Enqueue and dequeue / adding and deleting an item in the list has time complexity of $O(n)$. Can we improve that?
2. How can we take advantage of the near complete binary tree structure?

Binary Heap

- Unique relationship between parent and its 2 children. A min heap has the smallest key in the parent node. A max heap has the largest key in the parent node.
- if we arrange items in this tree form, searching, enqueue and dequeue of an item will take at most *O(log n)* in time complexity.
- A heap looks a lot like a tree, but it can be implemented as a list. Consider any parent node, leftchild and rightchild nodes, note the 2p and 2p+1 relationship between parent and children. Simple multiplication can be used to locate a node's left or right child. Simple division can be used to locate a node's parent. Note that the first cell with subscript zero is not used for this reason.

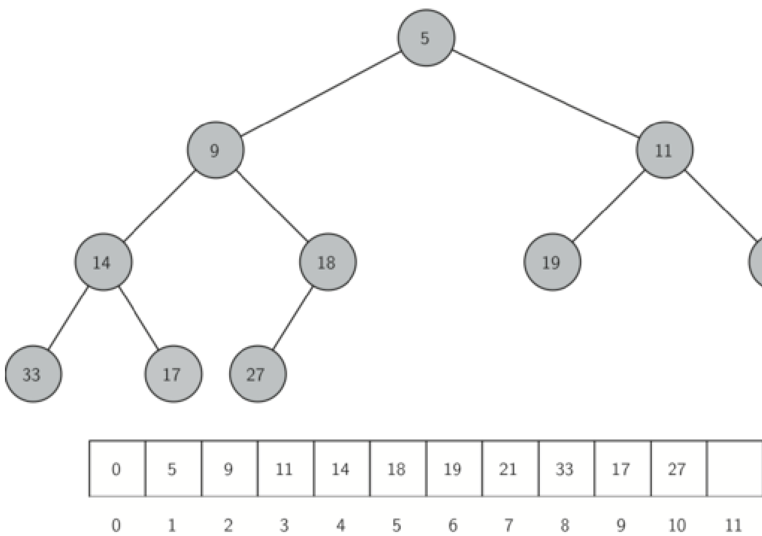


Figure 2: A Complete Binary Tree, along with its List Representation

image from e-text section 7.10.2

Example of a Min Heap

In this example, consider we are building a min heap.
For every node x with parent p, the key in p is smaller or equal to the key in x.

Heap Operations

To **insert** a new node:

1. Add new node to the end of the list
2. Recheck heap property by comparing new node to parent node. If new node is smaller, percolate new node up the tree by swapping with the parent node.

Let’s consider adding a newNode with key 7 to this example.

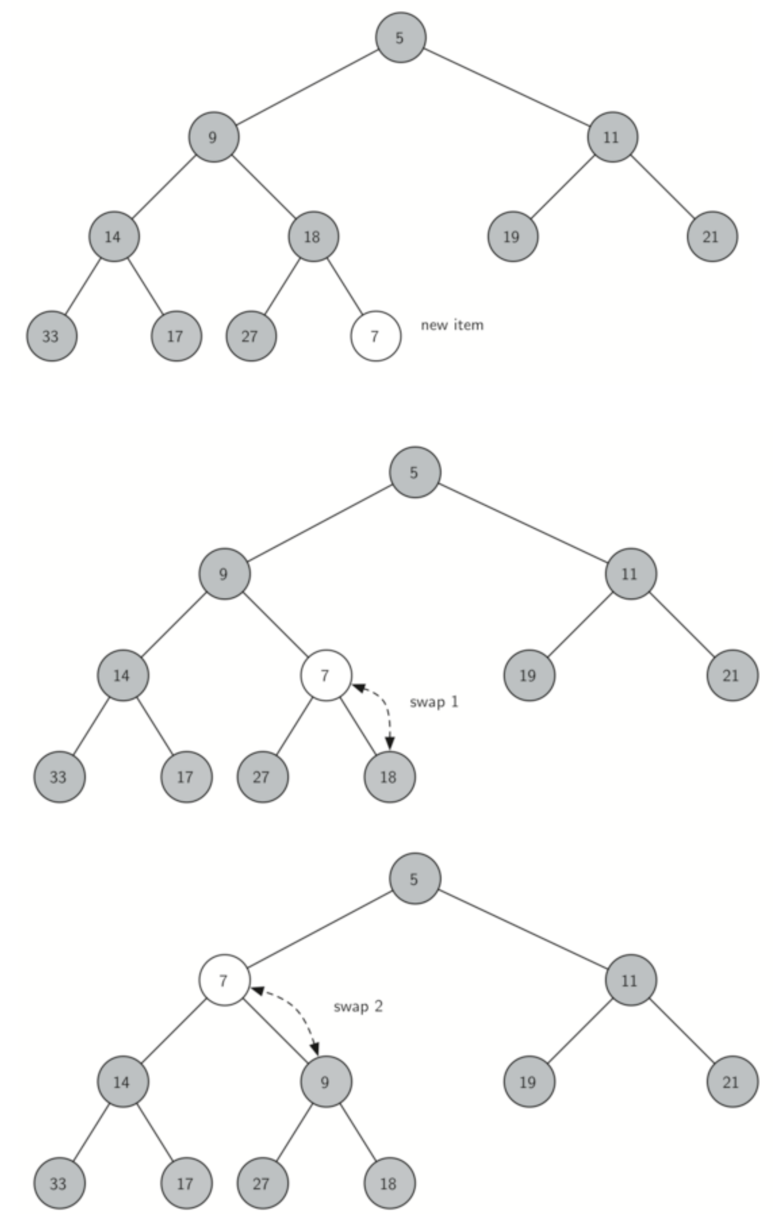


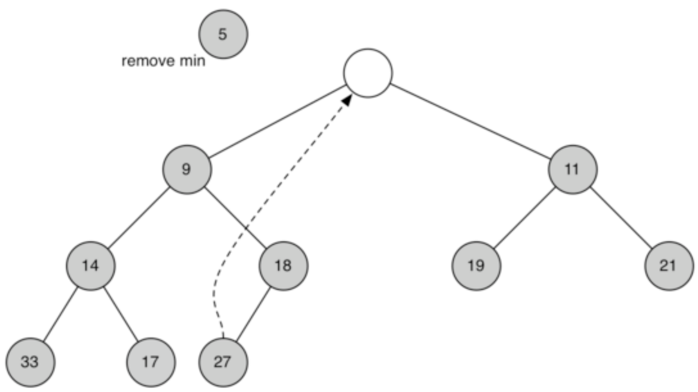
Figure 2: Percolate the New Node up to Its Proper Position

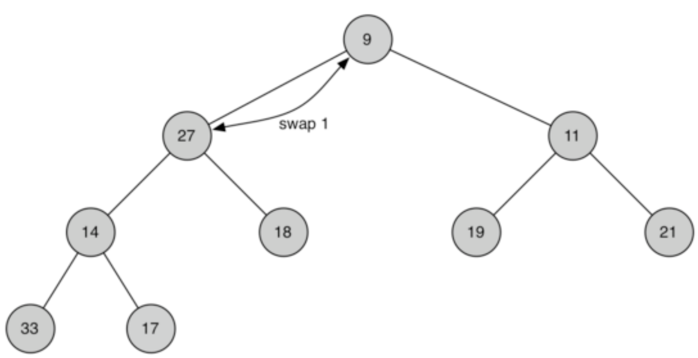
image from etext section 7.10.3

As a min heap, the smallest value in the tree is at the root.
If this min heap is implemented as a priority queue, to delete the smallest is to delete the root and then to rebalance the heap structure.

To **delete** the minimum and rebalance heap:

1. remove root
2. take the last value in the tree (rightmost child in the bottom level) and move it to the root position
3. if this node is larger than either one of its children, swap node with the smaller child node
4. keep percolating down until node is in right place – restoring the heap structure





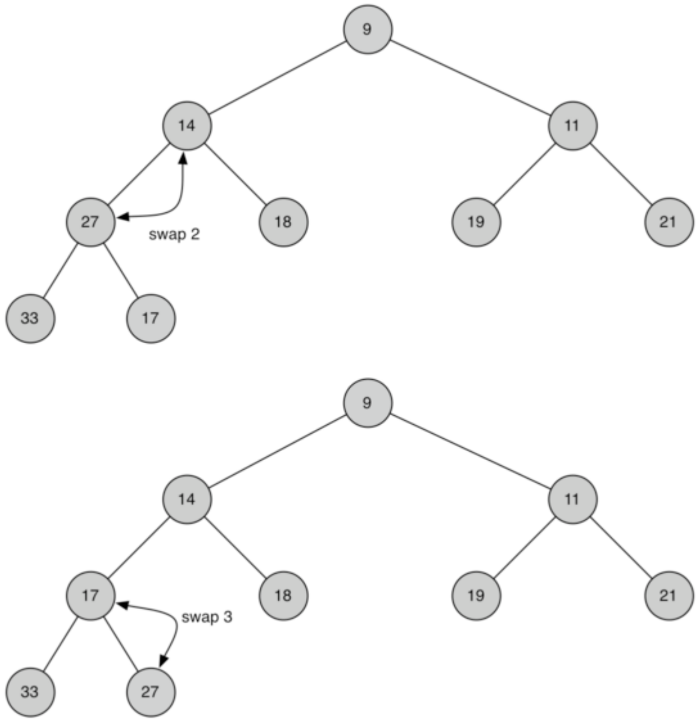


Figure 3: Percolating the Root Node down the Tree

image from etext section 7.10.3

Now consider a max heap – when all nodes are organized in a max heap structure.
The root of the max heap is the largest value.
What happens if

- we remove the root and swap it to the last node in the heap?
- we then take the 2nd last node and move it to the root position?
- Compare this new root with its left or right child and swap with the larger of the two. Percolate (heapify) through the heap.
- The new root is then the largest of these remaining nodes.
- Repeat this process to cover all nodes in the heap

Here is another sorting algorithm you can look up --- Heapsort – *O(n log n)*