ITS 231: Data Structures and Algorithms Laboratory — Semester 1

School of Information, Computer, and Communication Technology

Sirindhorn International Institute of Technology

Lab 8: Max Heap and its Applications

1. Objectives

- Understand the basic concept of heap data structure and its implementation.
- Practice simple applications that use heap data structure such as heap sorting.

Review

- A heap is a binary tree with the two properties
  1. The tree is either *complete* (full in all levels) or

nearly complete (full in all levels except the last level but data are fill contiguously from left to right)

  2. The key value of each node is greater than or equal to the key value in each of its descendents.

  This type of Heaps are called max-heap

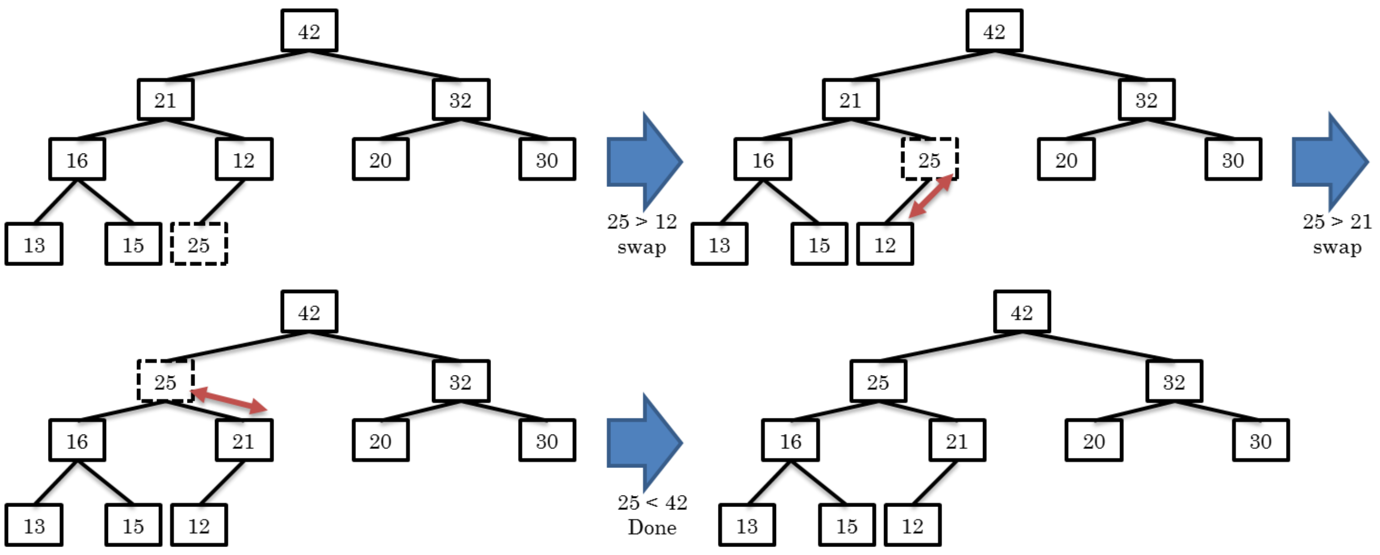
- The largest value will always be at the top root.
- A BT tree is complete if and only if all levels in it are full.
- A BT tree is nearly complete if and only if all levels excepts the last are full and the element in the last level are filled consecutively.

Maintenance Operations

- **reheap up** operation repairs the structure by floating up the last element that we have the problem with until that element is in its correct location in the tree
- **reheap down** operation repairs the structure by sinking down the focused element to it correct location at the lower part of the tree

Inserting data to a heap

- **Step 1:** Put the new data next to the last one in the tree.
- **Step 2:** Perform reheap up to maintain the heap structure

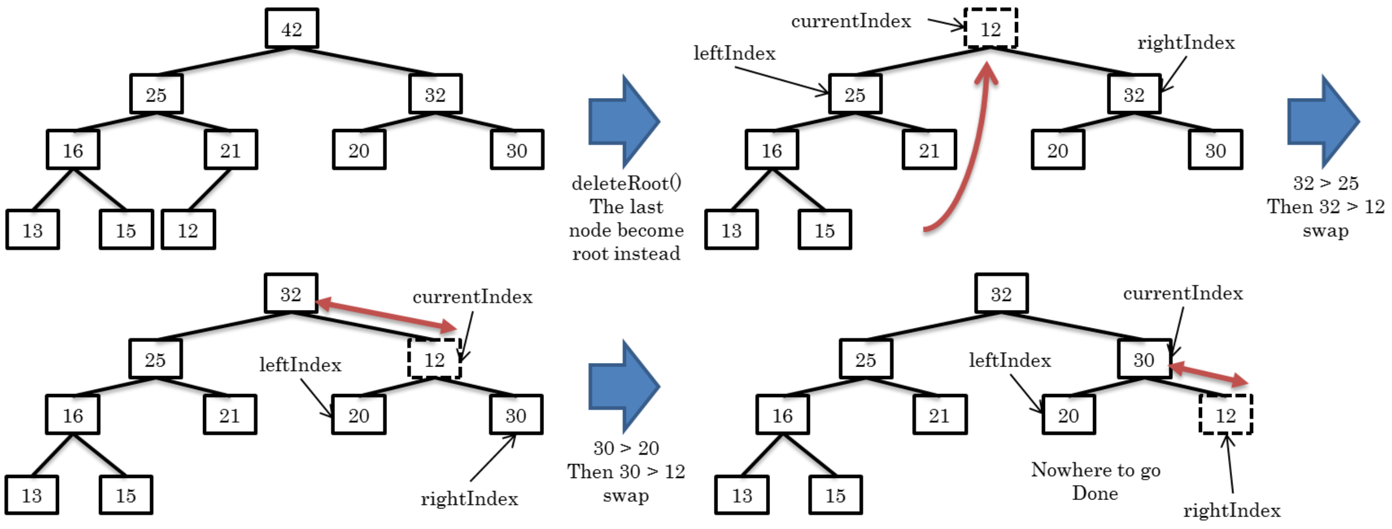


- Reheap up when 25 in inserted into heap.

Deleting the element from a heap

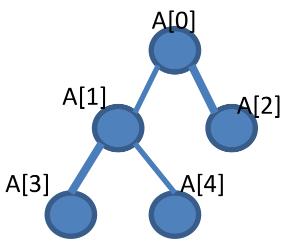
Step 1: Delete the last element and replace the root data with the deleted data.

Step 2: To fix the heap structure, perform reheap down

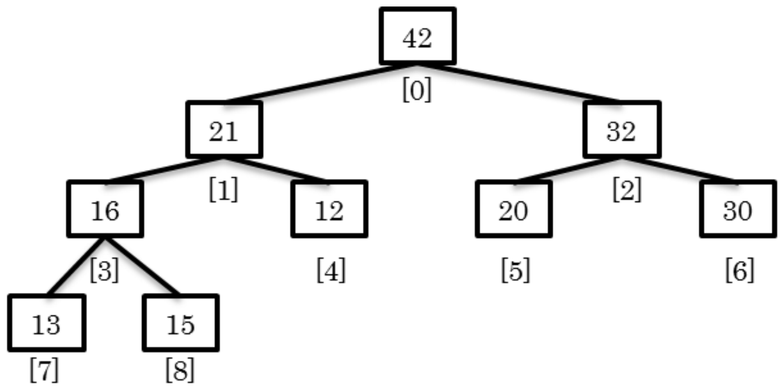


Heap Implementation

- Heaps are often implemented in an array rather than a linked list because of 2 main reasons. First the structure is dense and has no gaps inside of it so the locations of elements in the heap tree are fixed. Second doing it with a heap tree, in a reheapUp process using BT.java and BTNode.java is not possible with our existing BT codes because the BTNode has only the information of data at the node and its left and right children but no information of the the parent.
- The index of the array is given according to the order of nodes visited in breadth first traversal (BFT) fashion



- When we implement a heap using an array, we can calculate the locations of the parent, the left and right children in the array using the following formula.
  - Parent of A[i] is A[floor ((i-1)/2)]
  - Left Child of A[i] is A[2*i+1]
  - Right Child of A[i] is A[2*i+2]



Heap in its logical form

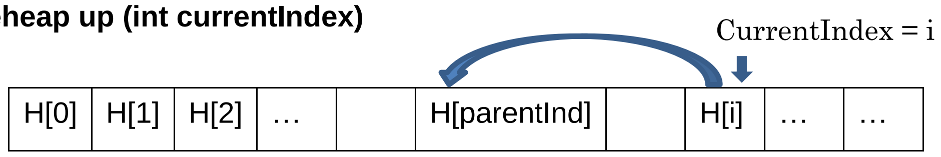
42	21	32	16	12	20	30	13	15
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

Heap in its array form

Heap
hArray: int []   load:int
Heap()   reheapUp(currentIndex: int): void   reheapDown(currentIndex: int): void   insert(data: int): void   deleteRoot(): int   swap(hArray: int [], ind1: int, ind2: int): void

Maintenance Operations using an Array

- **Reheap up (int currentIndex)**

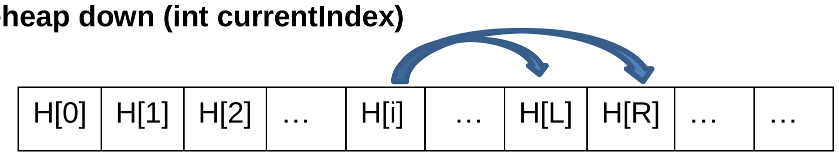


Step 1: Find the parentIndex

Step 2: If the value at the currentIndex is larger than the value at the parentIndex, swap the values

Repeat these steps until currentIndex is not larger than the value at the parentIndex or currentIndex is less than 0.

- **reheap down (int currentIndex)**



Step 1: Find the indices of the two children of the currentIndex

Step 2: Find the largest value and its index of the two children

Step 3: If the value at the currentIndex is smaller than the largest value, swap the values at the current index and at the index holding largest value

Step4: update the currentIndex to the index holding largest value

Repeat these steps until currentIndex is not small than the value at the index holding largest value **or** data at currentIndex has no more children

Insertion Operation

Step 1: Add the new data next to the current last data.

Step 2: update the load value

Step 2: Call reheapUp at the current location

DeleteRoot

Step 1: Replace the data at root with the current last data.

Step 2: Clear the last data.

Step 3: update the load value

Step 4: Call reheapDown starting at index 0

Heap Application

Sorting data using a heap

As the root of the max heap guarantees to be the largest, heapsort utilizes this fact to sort this data from the largest to smallest by repeatedly removing and printing data at the root from the heap array.

Exercises

Exercise 1 Implement **reheapUp**(int currentIndex) that move the data at currentIndex to a correct position.

Exercise 2 Implement **insert**(T data) that add the new data to the last position of arraylist and then call reheap up to the appropriate location. Test the methods in your main() method. Use the given template source code to complete this exercise. The method swap is provided in the template.

Exercise 3 Implement **reheapDown**(int currentIndex) that sinks the data at the currentIndex to the appropriate location in the list.

Exercise 4 Implement **the method deleteRoot**(): int that delete and return the value at the root of the heap. Test the method in the main method.

Exercise 5 Implement the method **makeHeapSort**()to print out the data in the heap Array in the descending order.