Data Structures Lab
Session 08

Course: Data Structures (CS2001) **Semester:** Fall 2022
Instructor: Mafaza Mohi **T.A:** N/A

Note:
● Lab manual covers the following elementary sorting algorithms
{Priority Queue, Binary Heap, Min Heap, Max Heap}
● Maintain discipline during the lab.
● Just raise your hand if you have any problem.
● Completing all tasks of each lab is compulsory.
● Get your lab checked at the end of the session.

Priority Queue

Recall, that an ordinary queue follows the **FIFO (** First In First Out **)** principle. Where the element which was enqueued first will be dequeued first, however, in many situations we are not concerned with the order of input but importantly concerned about the order of output. For example, an organization may be interested in finding the employee with the least holidays or most targets achieved. In such cases, the priority of the elements matter, and the most prioritized element should leave the queue first followed by the second most prioritized element until the queue is empty.

Functions of Queue

Recall all the methods of a Queue data structure:

● Enqueue (element): Inserts an item in the queue.
● Dequeue ( ): Removes the first element from the queue.
● IsFull ( ): Returns True if Queue is full and False otherwise.
● IsEmpty ( ): Returns True if Queue is empty and False otherwise.
● Peek ( ): Returns the element at the front of the queue without deleting it. ● Clear ( ): Dequeue all elements from the queue / Empties the Queue.

Updated Functions

To make sure that top priority elements leave the queue first, we have two options update the Enqueue or update the Dequeue function. Now we need to augment our Queue Class with the following functions

- Enqueue (element, priority)
  - Inserts the element having some value v, and priority p.
  - Put the element at its appropriate position as per priority.
  - Since we will be finding the position of the element among **n** elements, the enqueue takes O (**n**), however, the dequeue will only require one step hence it will be O (**1**)
- Update Priority (element, priority)
  - Check if an element exists in the Queue, and update its priority with the new value.

    - Update the position of the elements according to the priority.
- Dequeue ( )
    - Removes the most prioritized element from the queue.
    - If we don’t update the Enqueue then the element at the front might not be the top priority element in the Queue and before dequeue, we would have to find the top priority element.
    - Finding the top priority element would require **n** comparisons and hence the dequeue, in this case, would take O (**n**).

Task 1:

Create Priority Queue class with Updated Enqueue function. Here the element and the priority both will be integer values.

Task-2:

Create Priority Queue class with Updated Dequeue function.

Another Approach

To reduce the time complexity of the Priority Queue from O (**n**), we can have two lists (arrays)

- Short ordered list.
- An unordered list.

If the priority of the element is above a certain threshold it will be enqueued in the short ordered list otherwise it will be enqueued in the unordered list. Here, we will be sorting a small number of elements.

In some cases, the short ordered list might remain empty, as no elements have a priority greater than the threshold. To mitigate this, we allow the threshold to be dynamic and the user would have the freedom to update the threshold.

For dequeue the short ordered list will be looked at first, if it is empty then you will search for the top priority element in the unordered list.

Task-3:

Implement the Priority queue with two Arrays as discussed above.

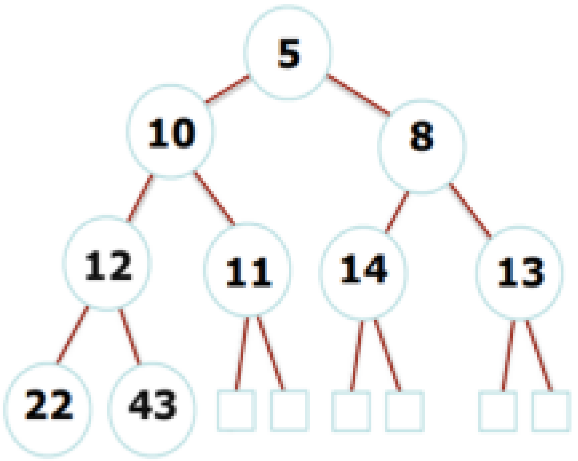
Note: Make sure the threshold is dynamic and upon changing the threshold the lists should be modified.

Binary Heap

Heap is a tree-based structure that satisfies the following heap properties.

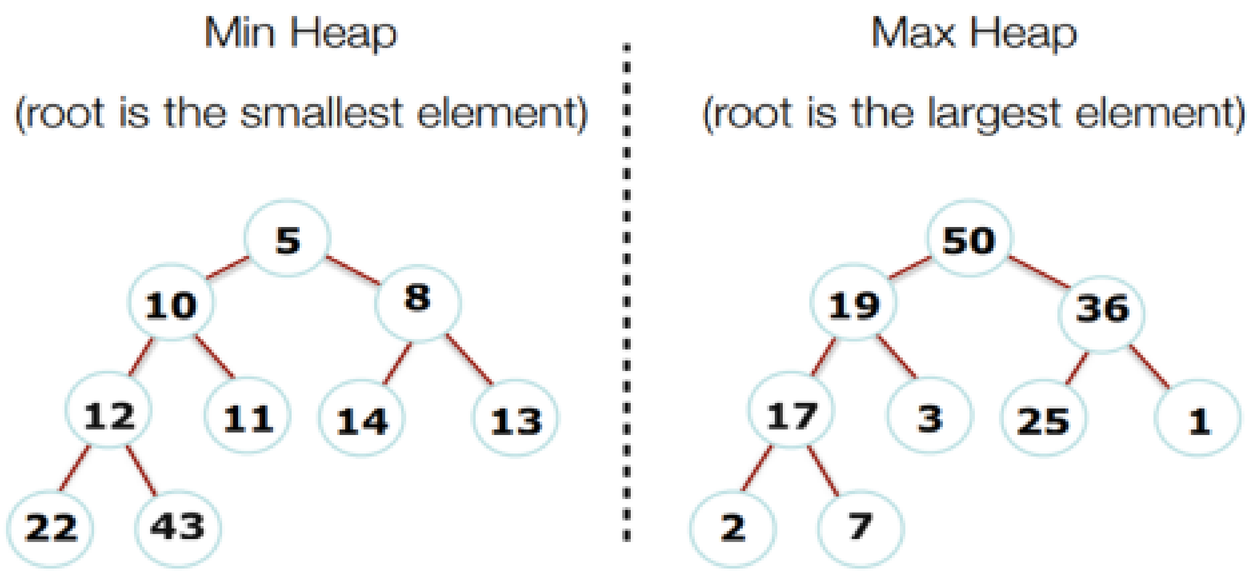
Parents have higher priority than any of their children

Heaps are completely filled, with the exception of the bottom level. They are, therefore, "complete binary trees".

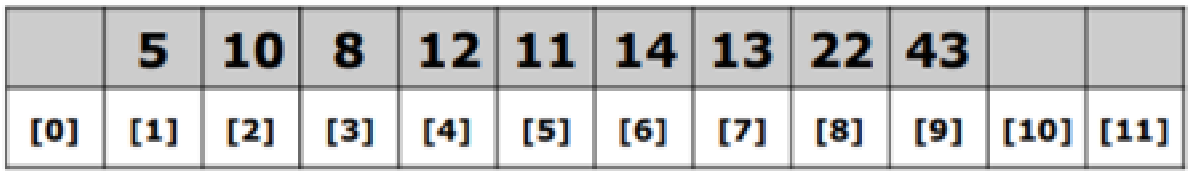


Complete: all levels filled except the bottom in the leftmost positions.
Binary: two children per node (parent)
Maximum number of nodes
Filled from left to right

Types



We can use a linked list, but an array works really well for sorting a binary heap. If we start from index 1 instead of 0 the arithmetic becomes a lot easier. Following diagram further illustrates the point.



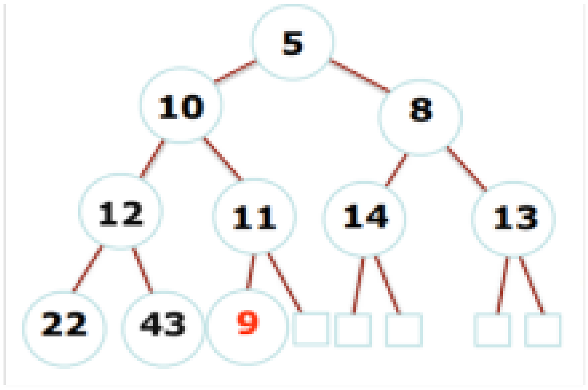
	5	10	8	12	11	14	13	22	43		
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]

Determining the parent and children of a node becomes simple arithmetic.

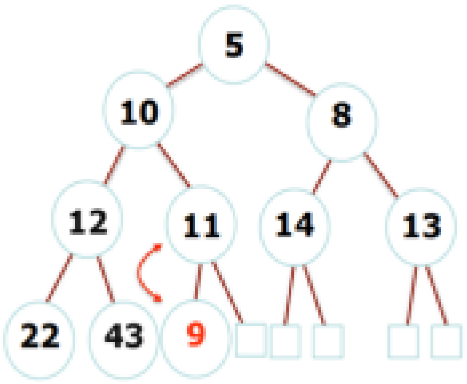
- The left child is at 2i
- The right Child is at 2i + 1
- The parent is at $\lfloor i / 2 \rfloor$
- Heap size is the number of elements in the heap.

Task 4:
Given an array of integers, find out if it is a Heap or not.

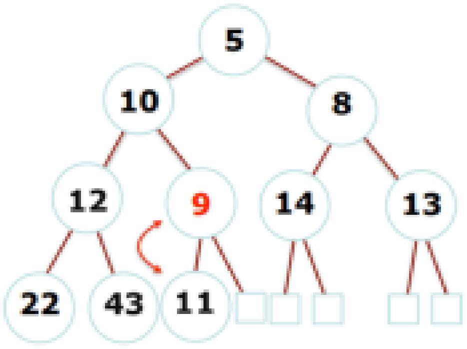
Enqueue In a Heap



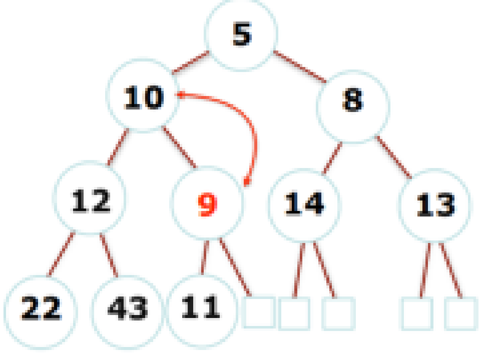
	5	10	8	12	11	14	13	22	43	9	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]



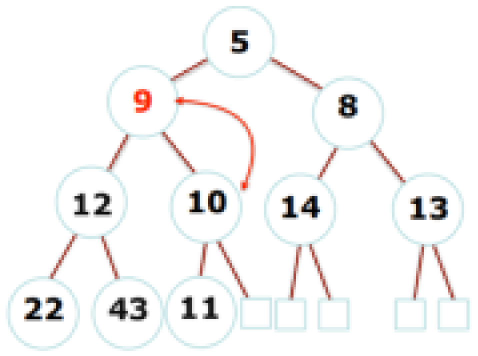
	5	10	8	12	11	14	13	22	43	9	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]



	5	10	8	12	9	14	13	22	43	11	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]



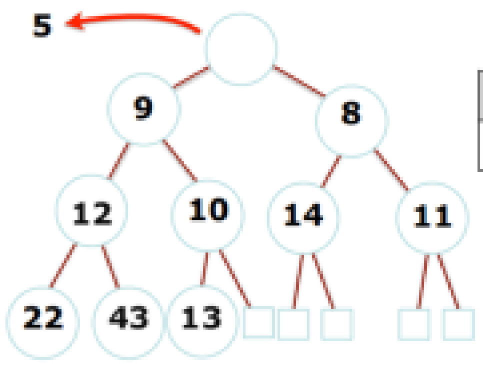
	5	10	8	12	9	14	13	22	43	11	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]



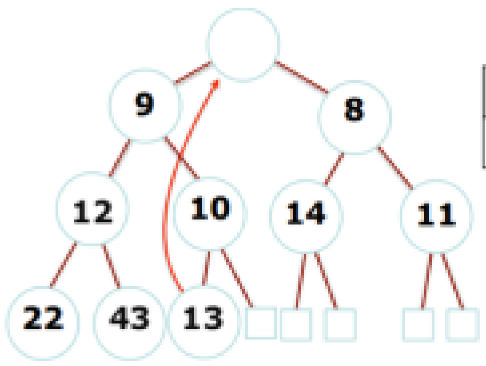
	5	9	8	12	10	14	13	22	43	11	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]

Deque In a Heap

Move last element (heap [heap.size ( )]) to the root heap [1].

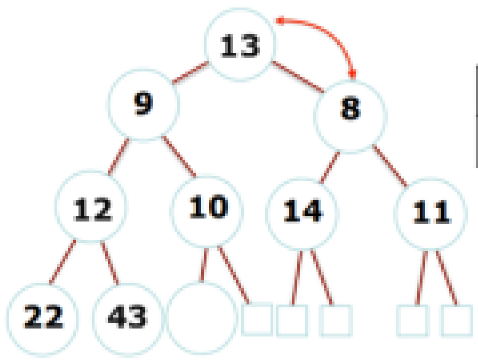


	5	9	8	12	10	14	11	22	43	13	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]

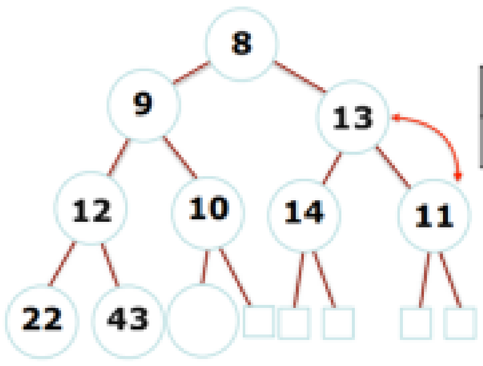


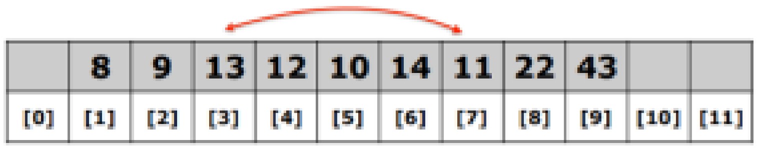
	5	9	8	12	10	14	11	22	43	13	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]

Don't forget to decrease heap size!

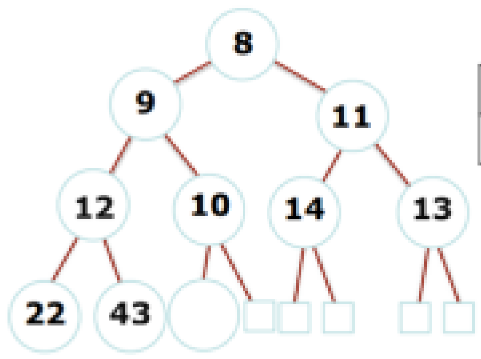


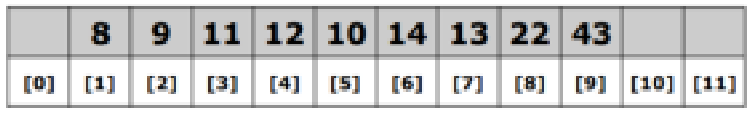
	13	9	8	12	10	14	11	22	43		
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]





	8	9	13	12	10	14	11	22	43		
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]





	8	9	11	12	10	14	13	22	43		
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]

Complexity? O(log n) - yay!

Task 5:

Create a class Heap that will store given elements in a heap. ● Implement Max Heap

● Implement Min Heap