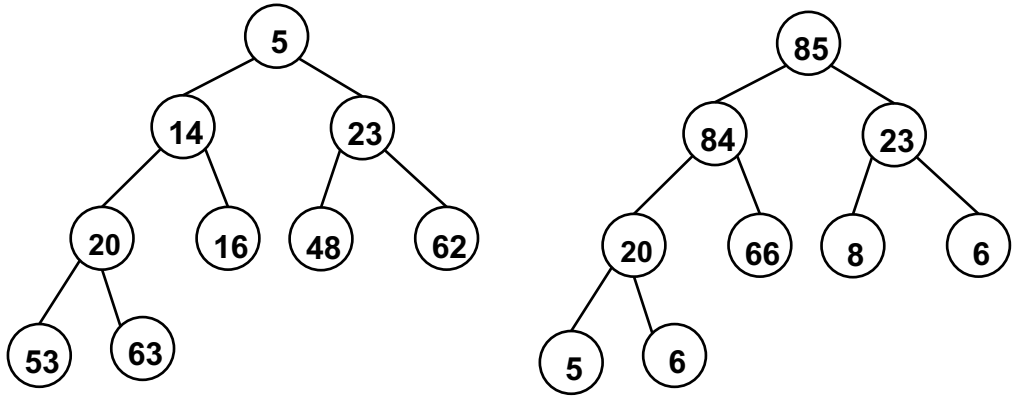
DS:
Binary Heap

Liwei

What is Binary Heap

- Binary Heap is a speical Binary Tree
  - Heap property
    - Value of any given node must be <= value of its children (Min-Heap)
    - Value of any given node must be >= value of its children (Max-Heap)
  - Complete Tree
    - All levels are completely filled except possibly the last level and the last level has all keys as left as possible
    - This makes Binary Heap ideal candidate for Array Implementation

Max-Heap



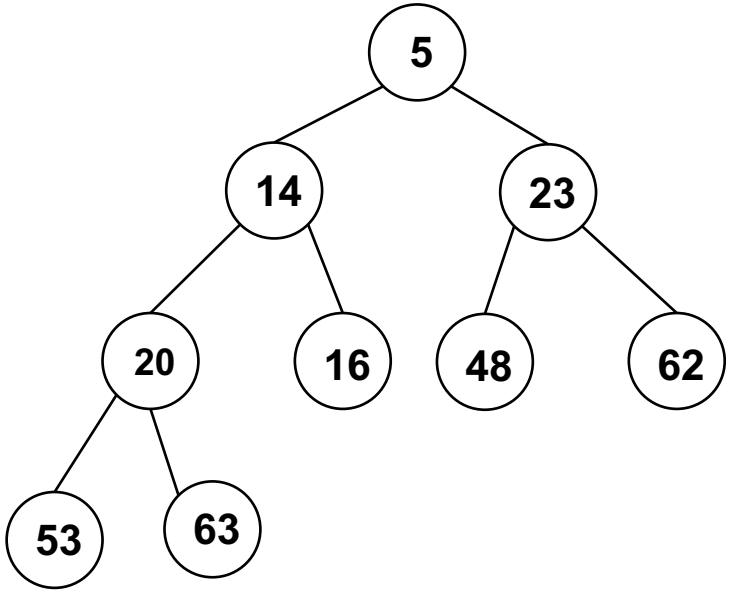
Min-Heap Max-Heap

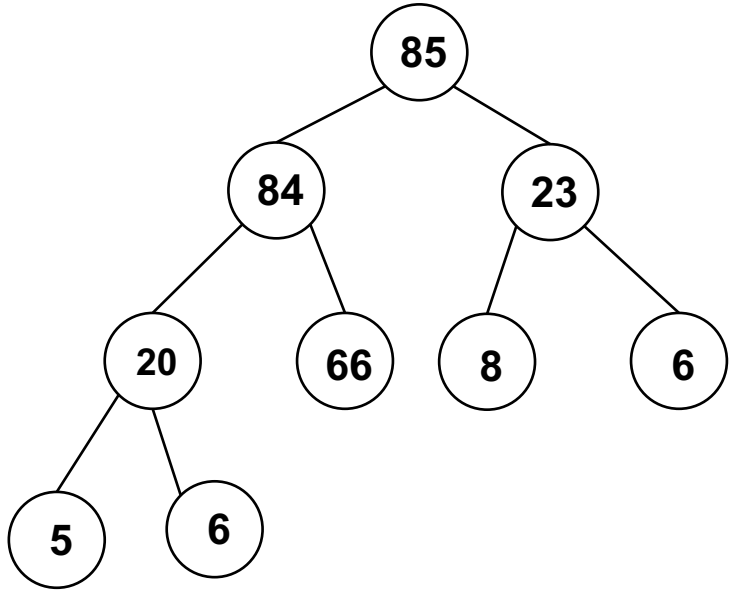
Why should we learn Binary Heap

- There are cases when we want to find min/max number among set of numbers in less than O(log n) time. Also, we want to make sure that inserting additional numbers dose not take more than O(log n) time.
- Possible solutions:
  - Store the numbers in sorted array
    - Issue here is that once we insert/delete a new number, our array needs to be adjusted again to keep it sorted which will take O(n) time
  - Store the numbers in linked list in sorted manner
    - Search takes O(n)
- Practical use
  - Prime's Algorithm, Heap Sort, Priority Queue

Types of Binary Heap

- Min-Heap: if the value of each node is less than or equal to value of both of its children
- Max-Heap: if the value of each node is more than or equal to value of both of its children





Common operations

- createHeap – creates a blank array to be used for storing heap
- peekTopOfHeap – returns min/max from Heap
- extractMin / extractMax – extracts Min/Max from Heap.
- sizeOfHeap – returns the size of the Heap
- insertValueInHeap – inserts a value in Heap
- deleteHeap – Deletes the entire Heap

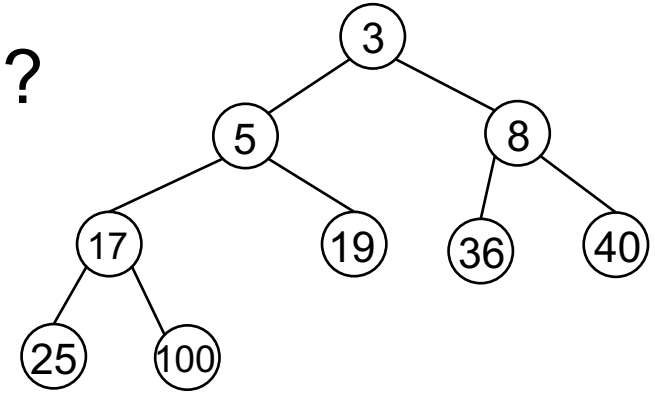
We can extract only this node.

Implementation Options

- Array based implementation
- Reference / Pointer based implementation

Binary Heap – Array Representation

- How does Binary Heap looks like at logical level?



- How does Binary Heap looks when implemented via Array

cell #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
value	X	3	5	8	17	19	36	40	25	100								

Left Child – cell[2x]
Right Child – cell[2x + 1]

Creation of Heap

createHeap(size)

- Create a blank array of “size+1”
- Initialize sizeOfHeap with 0

cell #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
value	X																	

Time complexity: O(1)

Peak of Heap

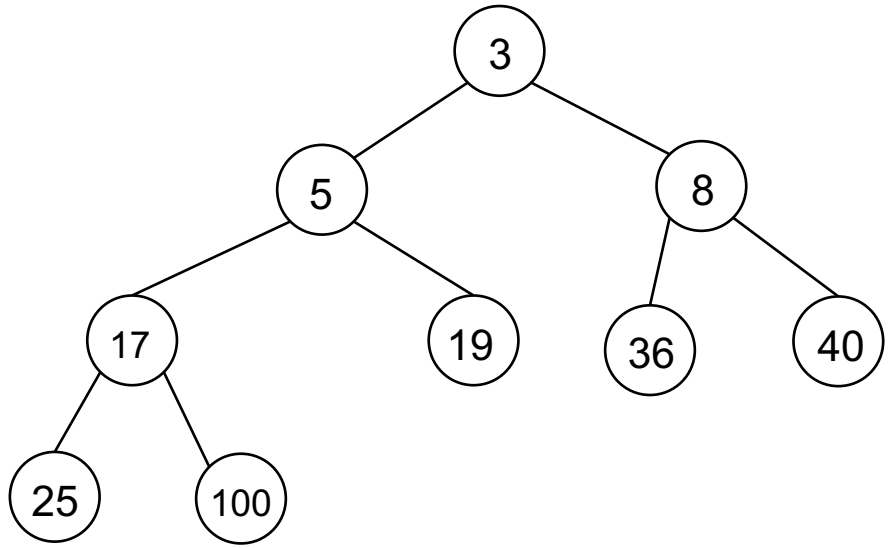
peakTopOfHeap()

    If tree does not exist

        return error message

    else

        1st cell of the array

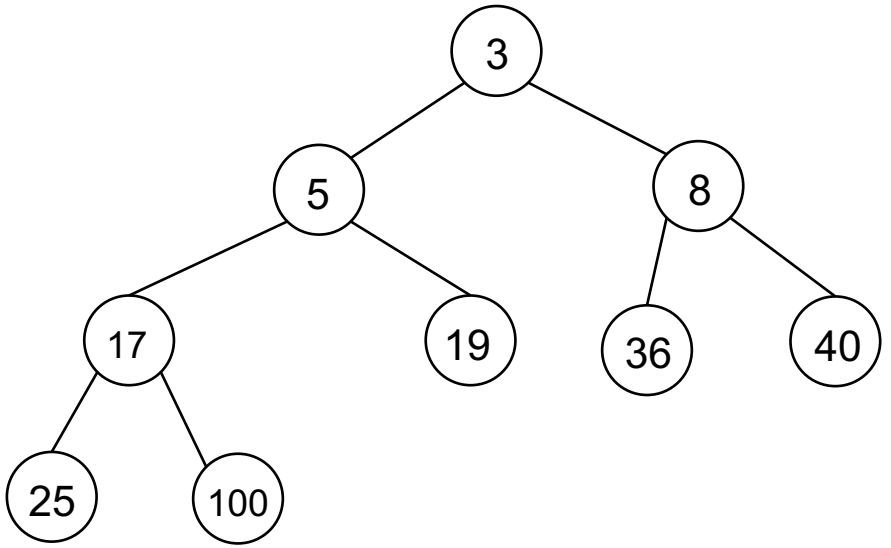


cell #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
value	X	3	5	8	17	19	36	40	25	100								

Time complexity: O(1)

Size of Heap

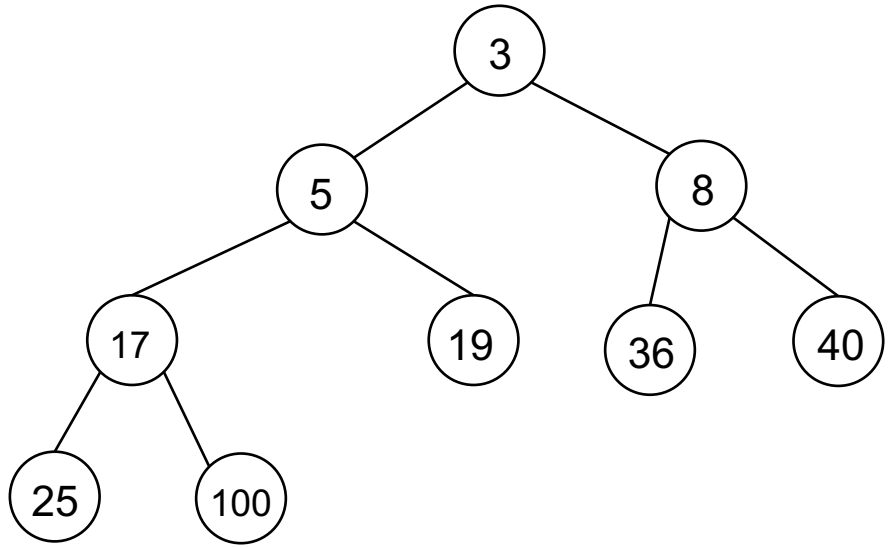
```
sizeOfHeap()
    return sizeOfHeap
```



cell #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
value	X	3	5	8	17	19	36	40	25	100								

Time complexity: O(1)

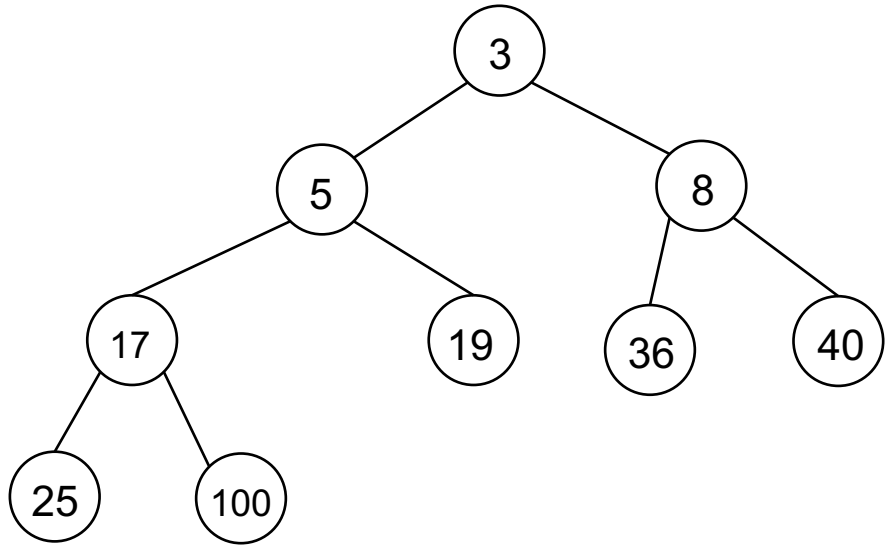
Insertion in Heap



cell #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
value	X	3	5	8	17	19	36	40	25	100								

Insertion in Heap

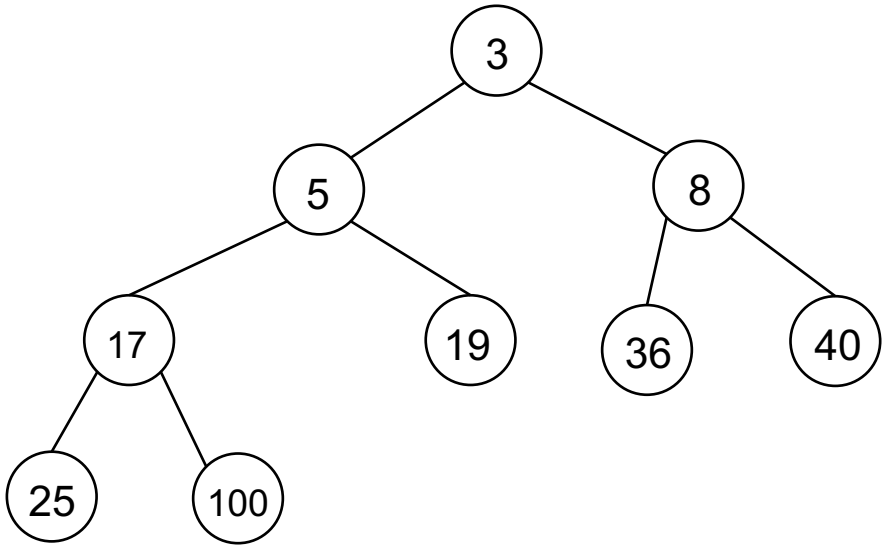
```
insertValueInHeap (value)
    If tree does not exist
        return error message
    else
        insert value in first unused cell of array
        sizeOfHeap ++
        heapifyBottomToTop(sizeOfHeap)
                                    O(logn)
```



cell #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
value	X	3	5	8	17	19	36	40	25	100								

Time complexity: O(logn)

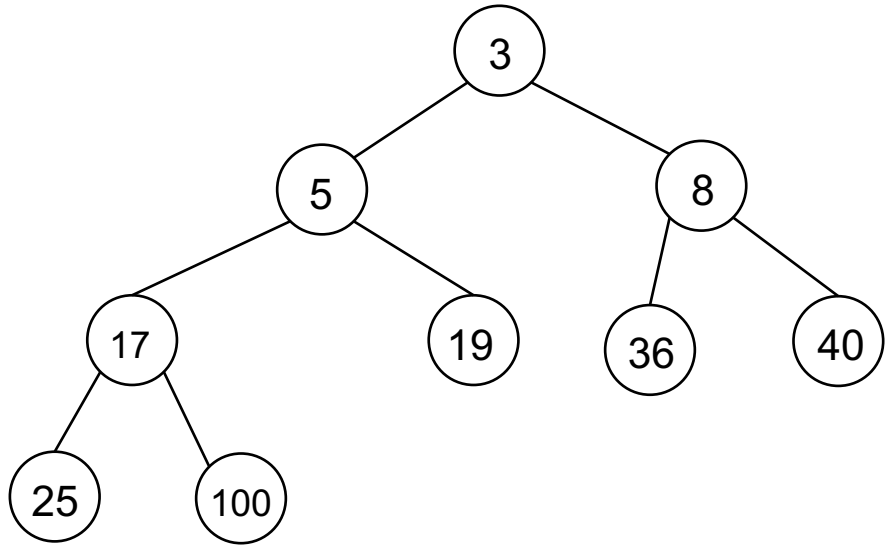
ExtractMin from Heap



cell #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
value	X	3	5	8	17	19	36	40	25	100								

ExtractMin from Heap

```
extractMin ()
    If tree does not exist
        return error message
    else
        extract 1st cell of array
        promote last element to first
        sizeOfHeap --
        heapifyTopToBottom (1)    O(logn)
```



cell #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
value	X	3	5	8	17	19	36	40	25	100								

Time complexity: O(logn)

Delete Heap

deleteHeap ()

Free the array from memory

O(1)

Time complexity: O(nlogn)

Time and Space Complexity (Binary Heap)

	Time Complexity
CreateHeap	O(1)
PeekTopOfHeap	O(1)
sizeOfHeap	O(1)
insertValueInHeap	O(logn)
extractMin / extractMax	O(logn)
deleteHeap	O(1)

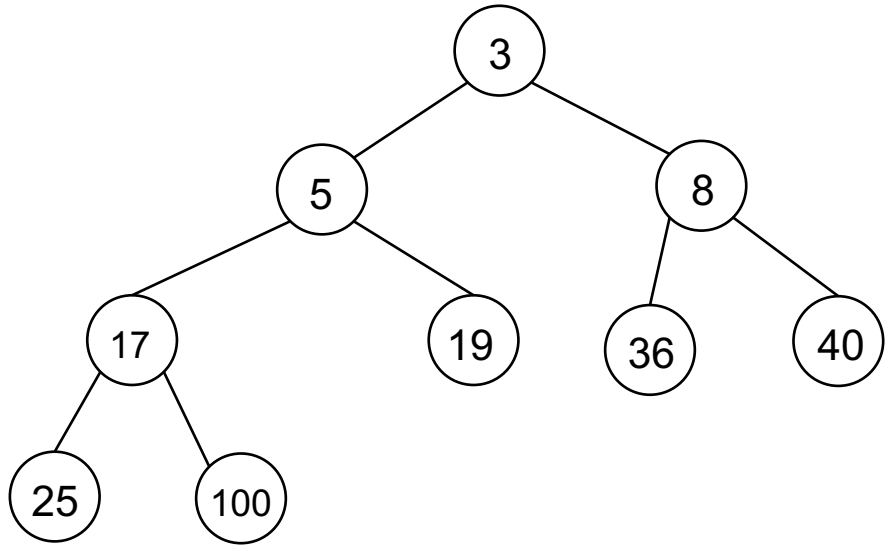
Binary Heap
Reference-based Implementation

Common operations

- createHeap – creates a blank array to be used for storing heap
- peekTopOfHeap – returns min/max from Heap
- extractMin / extractMax – extracts Min/Max from Heap.
- sizeOfHeap – returns the size of the Heap
- insertValueInHeap – inserts a value in Heap
- deleteHeap – Deletes the entire Heap

ExtractMin from Heap

```
extractMin ()
    If tree does not exist
        return error message
    else
        extract 1st cell of array
        promote last element to first
        sizeOfHeap --
        heapifyTopToBottom (1)      → O(n)
```



cell #	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
value	X	3	5	8	17	19	36	40	25	100								