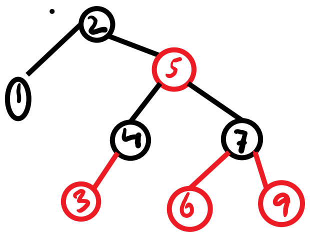


Habib University
shaping futures

CS 201 Data Structure II (L2 / L5)

Binary Heaps

Chapter 10, Open Data Structure

Muhammad Qasim Pasta

qasim.pasta@sse.habib.edu.pk

Slides are designed to be filled during the lectures. Some details are intentionally mentioned to be discussed in the class. These slides should not be used as reading.

Priority Queue

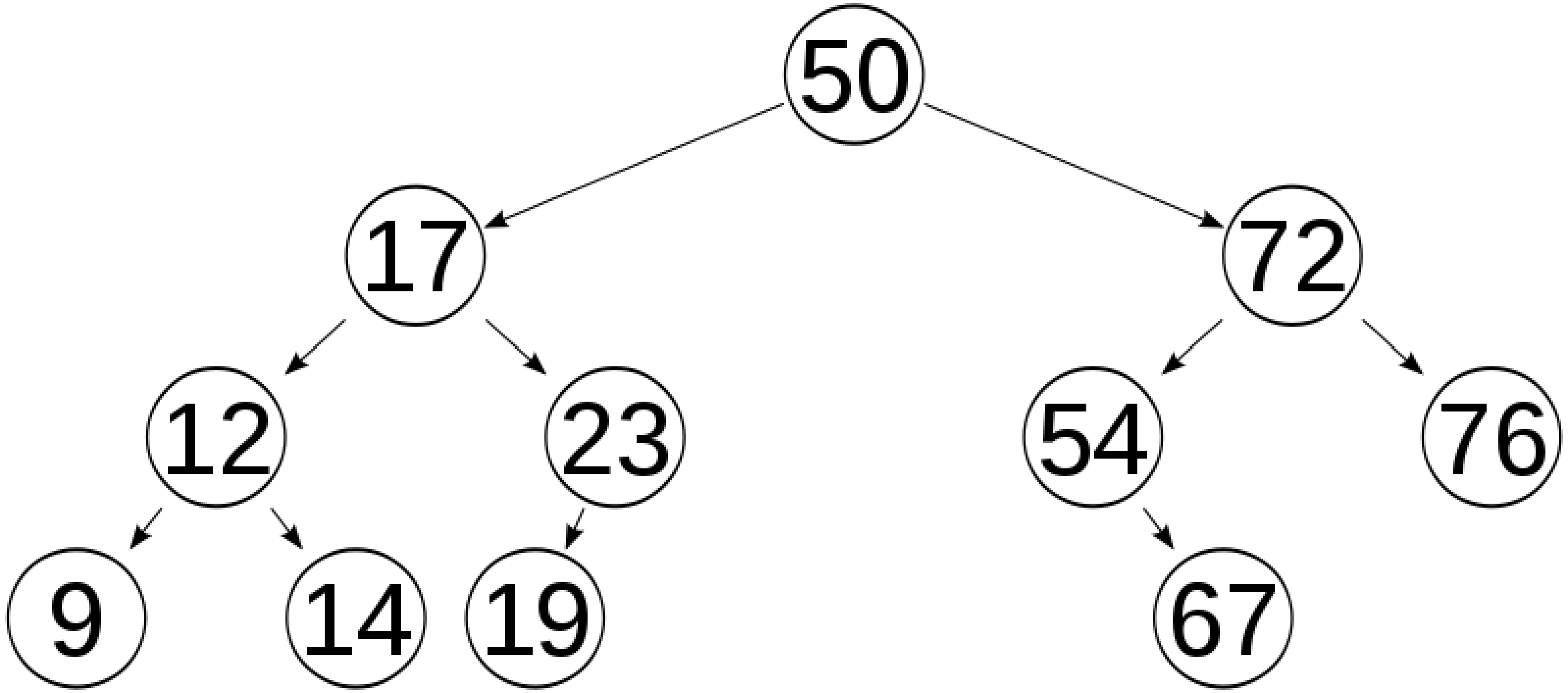
- We want to extract the values of the basis of priority
- Consider, the values are priorities

- Max Priority Queue: Dequeue() will return the maximum value
- Min Priority Queue: Dequeue() will return the minimum value

- Let’s we implement this using List:
  - Insertion will take:
  - Removal will take:

Binary Trees

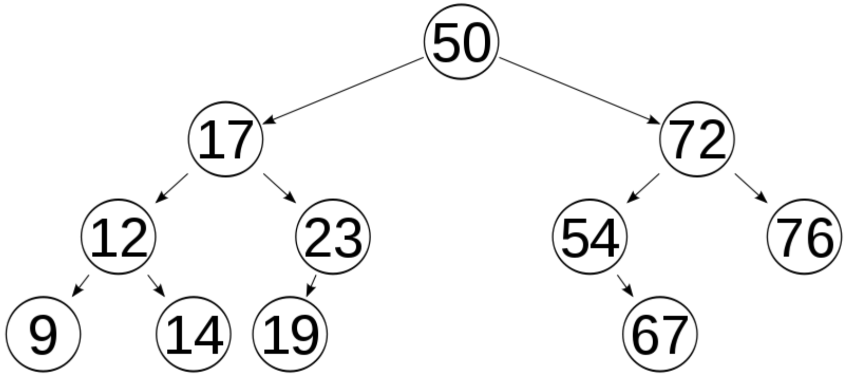
- Complete Binary Tree
- Full Binary Tree
- Left aligned/balanced Binary Tree

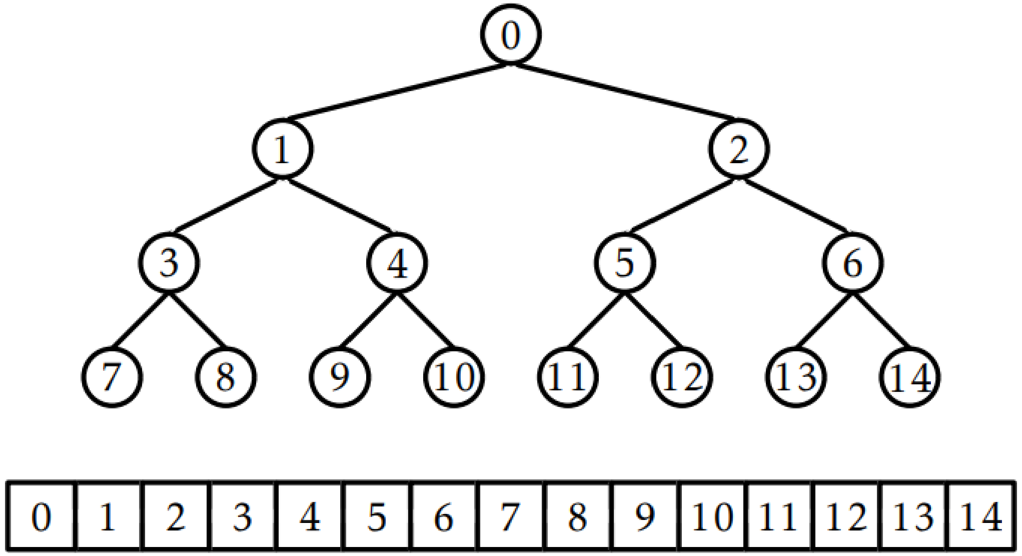


https://stackoverflow.com/questions/7275586/left-balanced-binary-trees/7275697#7275697

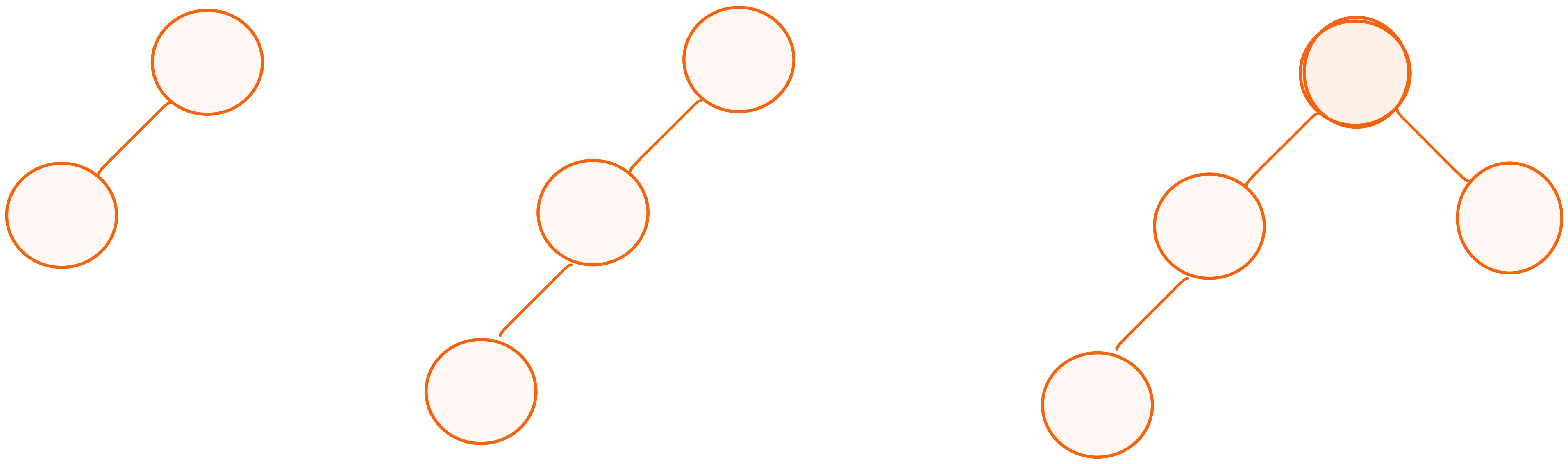
Representing Binary Trees as Array

- left(i) = 2 * i + 1
- right(i) = 2 * (i+1)   [OR 2i+2]
- parent(i) = (i – 1) / 2





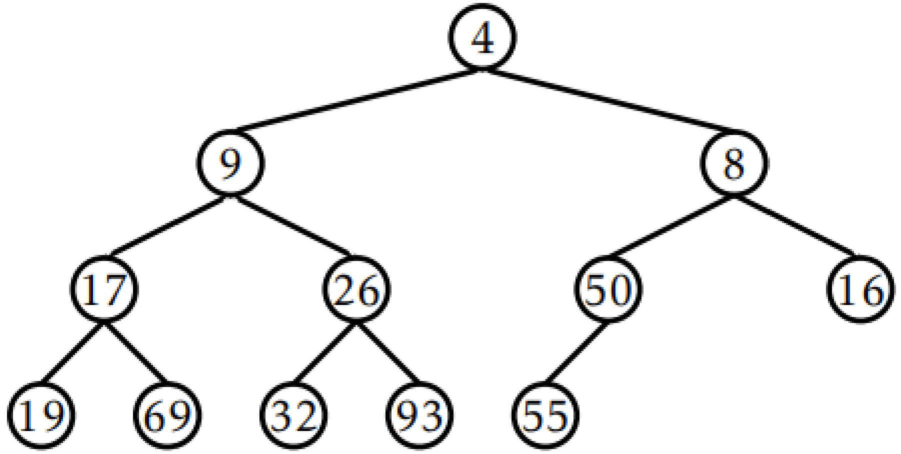
Valid left-balanced trees?



Binary Heaps

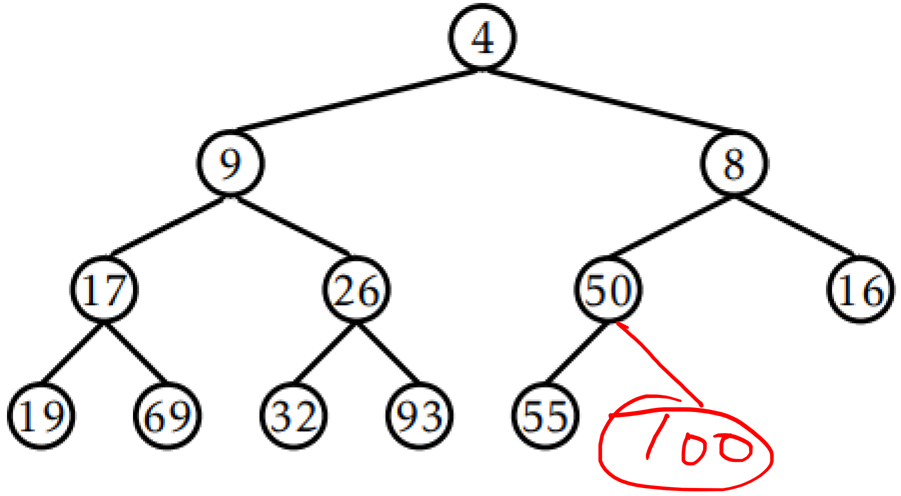
- Left balanced binary tree (no gaps)
- Ordering of data must obey heap property
- Min Heap: children values must be **greater** than parent's value
  - Root must be: ?
- Max Heap: children values must be **smaller** than parent's value
  - Root must be: ?
- No BST property! It's not a search tree

Binary heap



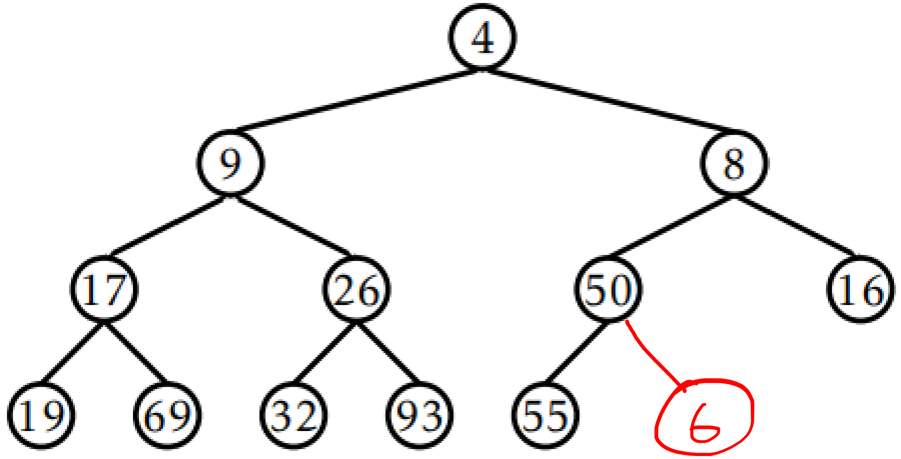
4	9	8	17	26	50	16	19	69	32	93	55			

Binary heap



4	9	8	17	26	50	16	19	69	32	93	55			

Binary heap



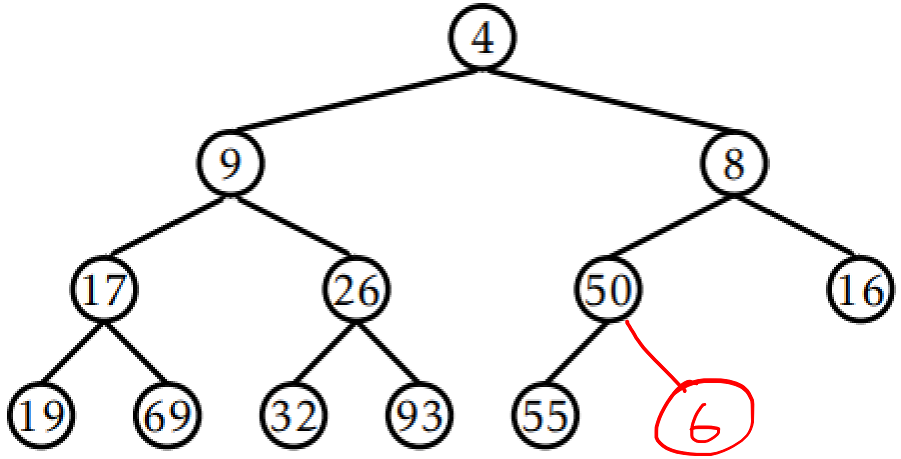
4	9	8	17	26	50	16	19	69	32	93	55	6		

Insertion in binary heap

- Add element at the last location of the array
  - To make sure the tree is left balanced
- Bubble up
  - Fix the new element by moving up, if violate
  - Repeat this until we react at root

```
BubbleUp(i)
        p = parent(i)
        if i>0 and A[p] < A[i]
                A[i],A[p] = A[p], A[i]
                BubbleUp(p)
```

Binary heap



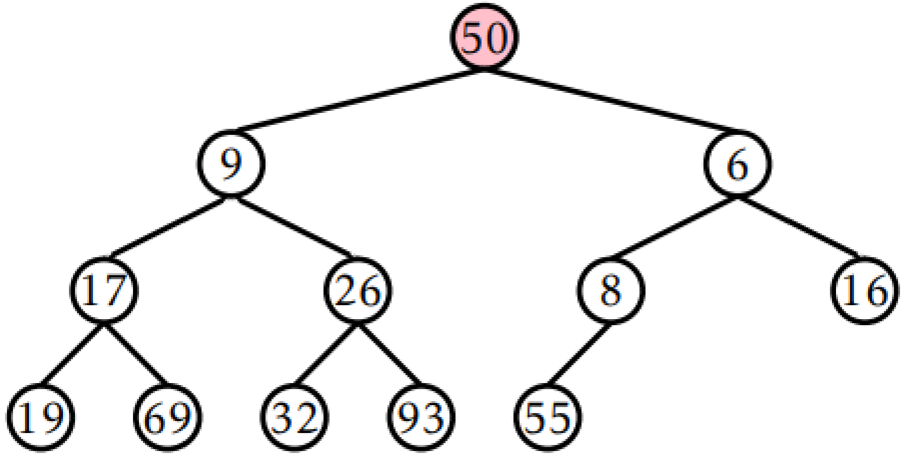
4	9	8	17	26	50	16	19	69	32	93	55	6		

Remove: Extract min/max

- Minimum /Maximum at Root
- Replace Root with the last value (n-1)
- Trickle Down: Place root’s new value at the right place

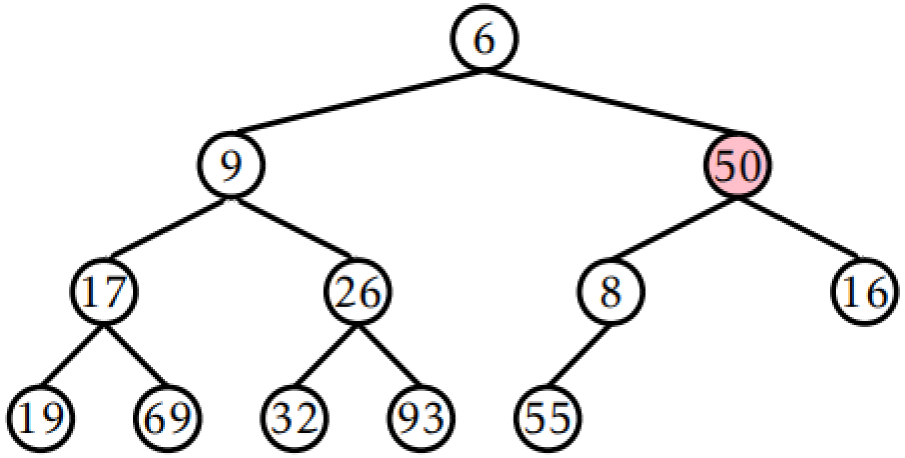
- TrickleDown(i)
  - rt = right(i)
  - lt = left(i)
  - If A[rt] < A[i] and A[rt] < A[lt]: swap A[i] with A[rt] and TrickleDown(rt)
  - If A[lt] < A[i] and A[lt] < A[rt]: swap A[i] with A[lt] and TrickleDown(lt)

Example: Trickle Down



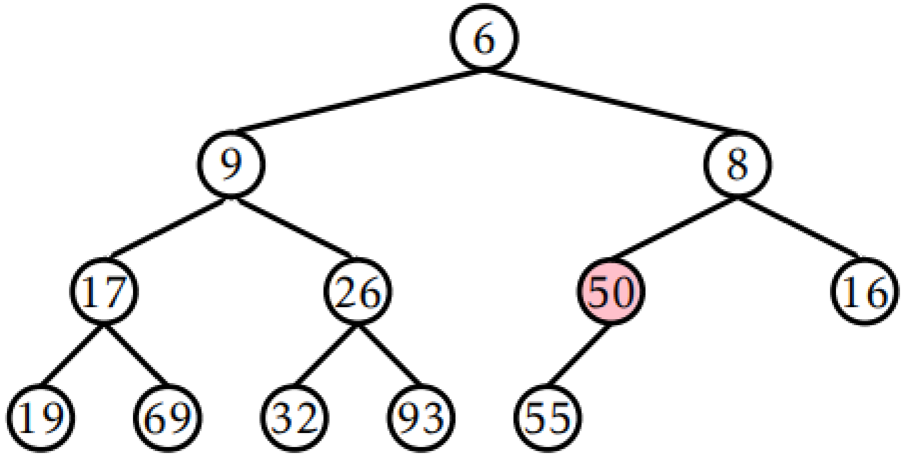
50	9	6	17	26	8	16	19	69	32	93	55			

Example: Trickle Down



6	9	50	17	26	8	16	19	69	32	93	55			

Example: Trickle Down



6	9	8	17	26	50	16	19	69	32	93	55			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Analysis

Theorem 10.1. *A BinaryHeap implements the (priority) Queue interface. Ignoring the cost of calls to* resize()*, a BinaryHeap supports the operations* add(x) *and* remove() *in* $O(\log n)$ *time per operation.*

Furthermore, beginning with an empty BinaryHeap, any sequence of m add(x) *and* remove() *operations results in a total of* $O(m)$ *time spent during all calls to* resize().

- You must be able to prove it yourself
  - What would be height of a heap of n elements?
  - Maximum number of nodes we need to travel to fix an element?