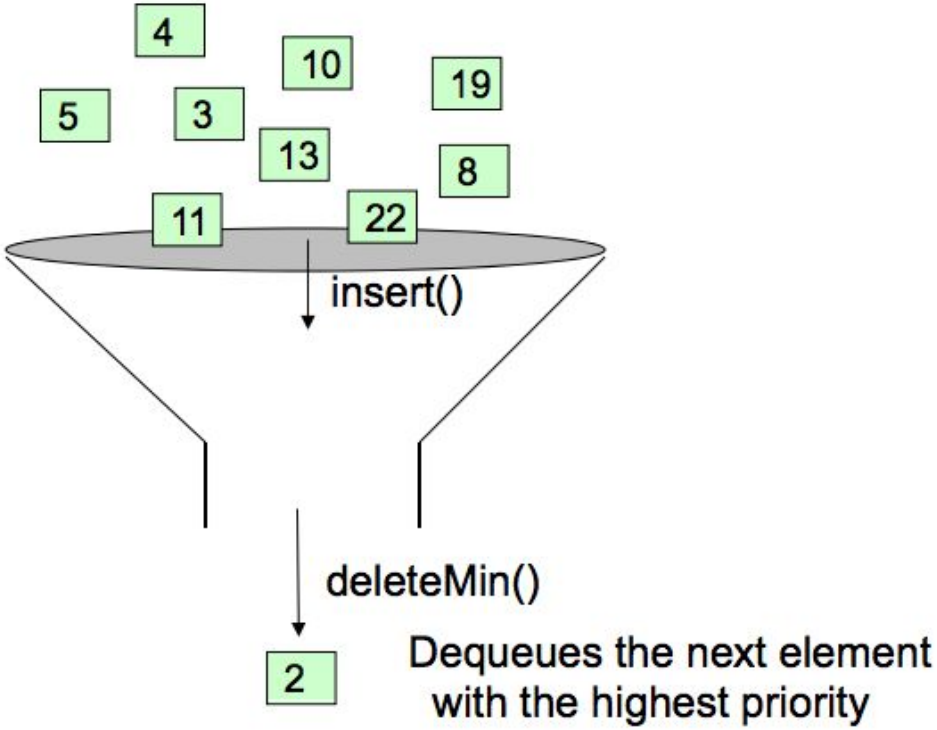
# PRIORITY QUEUE (HEAP)

Data Structures and Algorithms
Waheed Iqbal


Department of Data Science, FCIT
University of the Punjab, Lahore, Pakistan

# Introduction

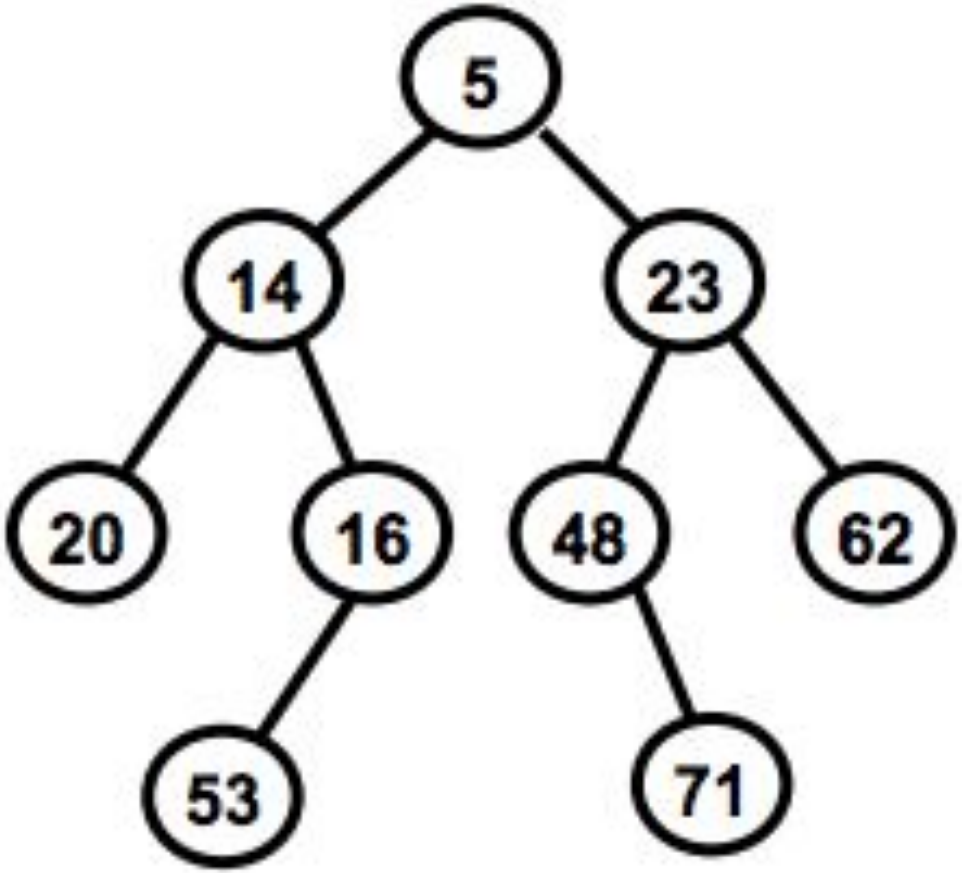
A **priority queue** is different from a "normal" queue, because instead of being a "first-in-first-out" data structure, values come out in order by priority.



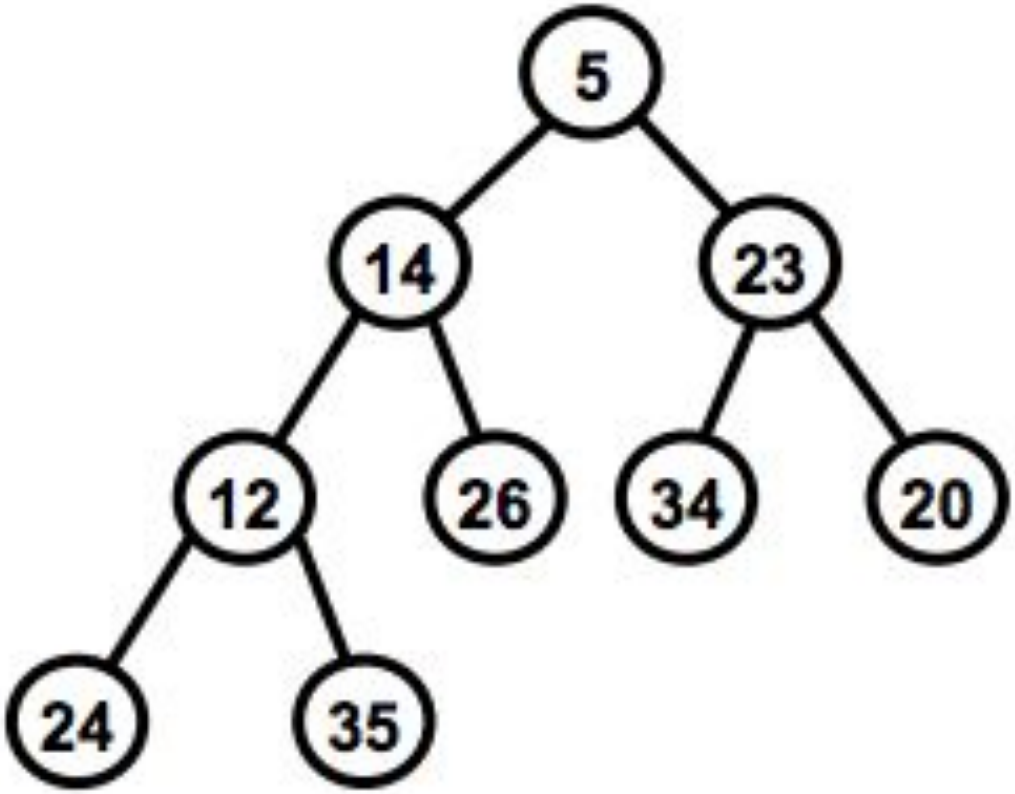
# Heap

- Heap is a binary tree with two properties:
  - **Structure/Shape property**
    - Each level (except possibly the bottom most level) is completely filled
    - The bottom most level may be partially filled (from left to right)
  - **Order property order property**
    - **max-heap**, the data contained in each node is greater than (or equal to) the data in that node's children.
    - **min-heap**, the data contained in each node is less than (or equal to) the data in that node's children.

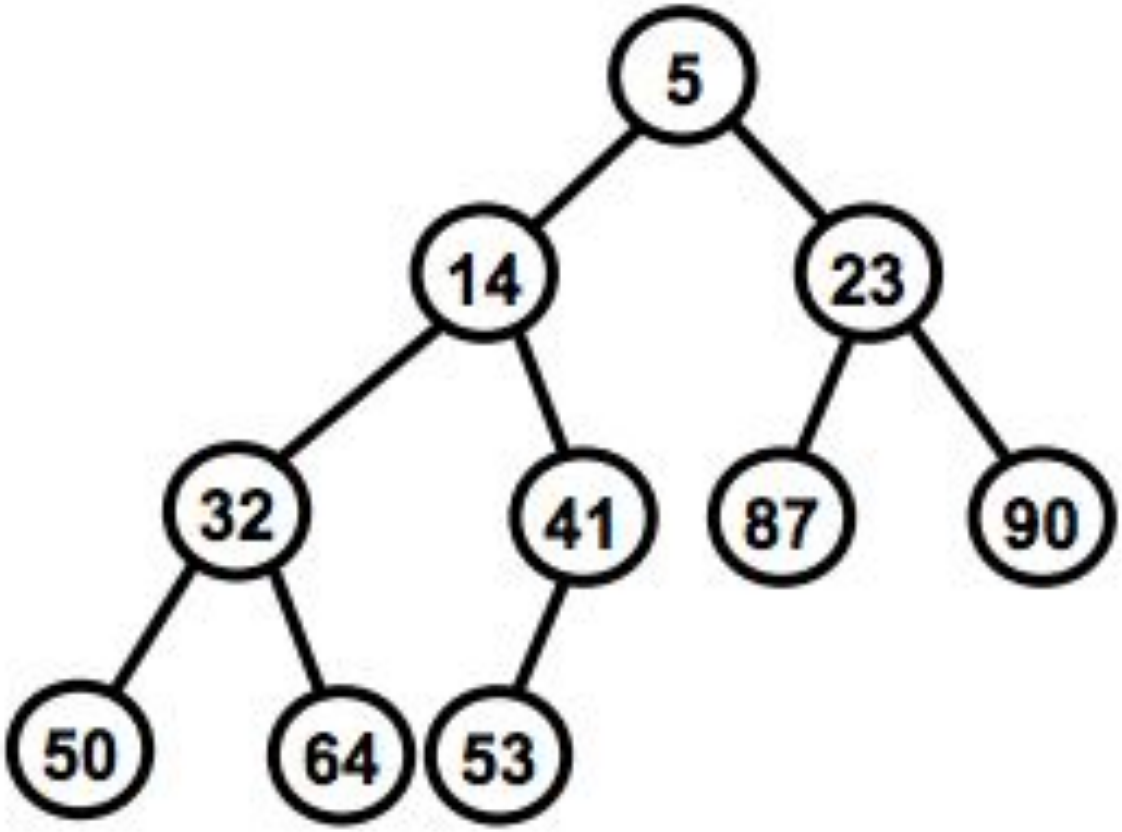
# Example 1: Is it a min-heap?



# Example 2: Is it a min-heap?



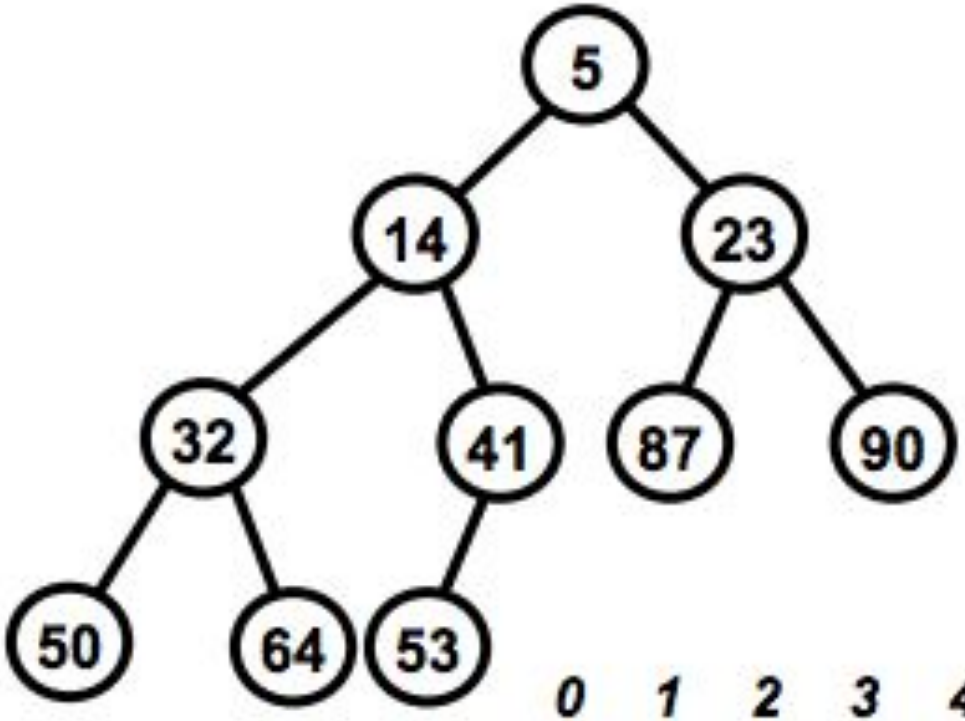
# Example 2: Is it a min-heap?



# Heap: A Simple Implementation

- Use an array to hold the data.
- Store the root in position 1.
  - We won't use index 0 for this implementation.
- For any node in position i,
  - its **left child** (if any) is in position **2i**
  - its **right child** (if any) is in position **2i + 1**
  - its parent (if any) is in position **i/2**

# Heap: A Simple Implementation (Cont.)




For node at i:
Left child is at 2i
Right child is at 2i12
Parent is at ⌊i/2⌋

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | 5 | 14 | 23 | 32 | 41 | 87 | 90 | 50 | 64 | 53 |

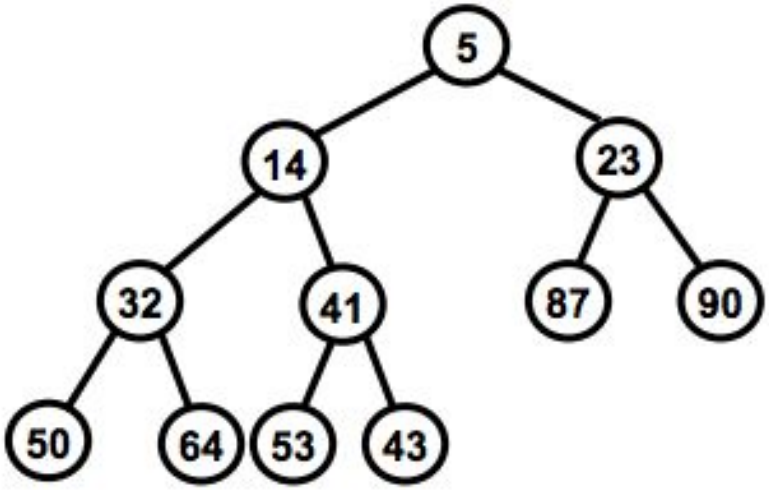
# Heap: Insertion

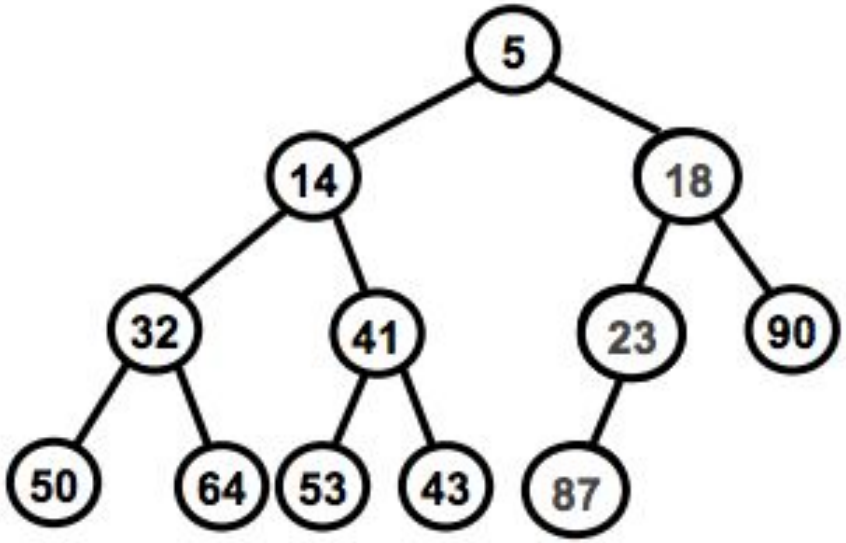
Given you have a min-heap.

- Place the new element in the next available position in the array.
- Compare the new element with its parent. If the new element is smaller, than swap it with its parent.
- Continue this process until either the new element's parent is smaller than or equal to the new element, or the new element reaches the root.

# Heap: Insertion

**Inset 18**

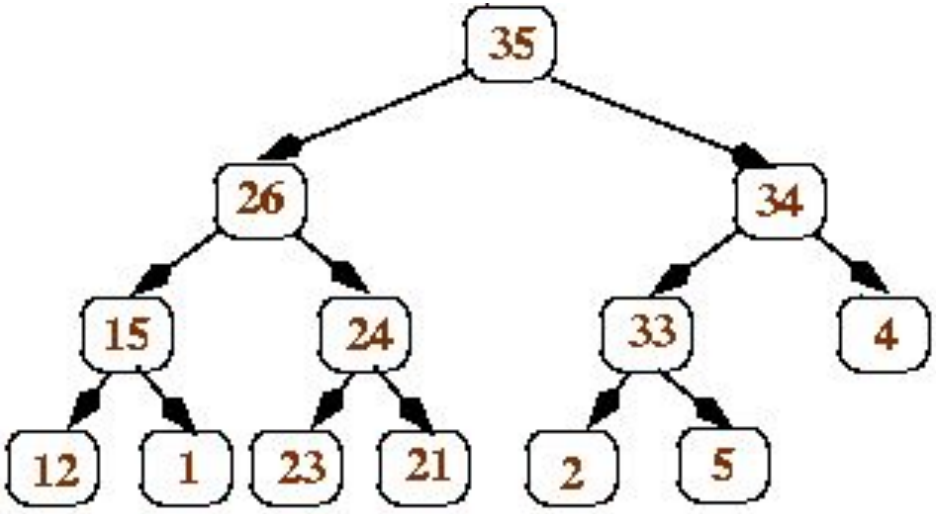


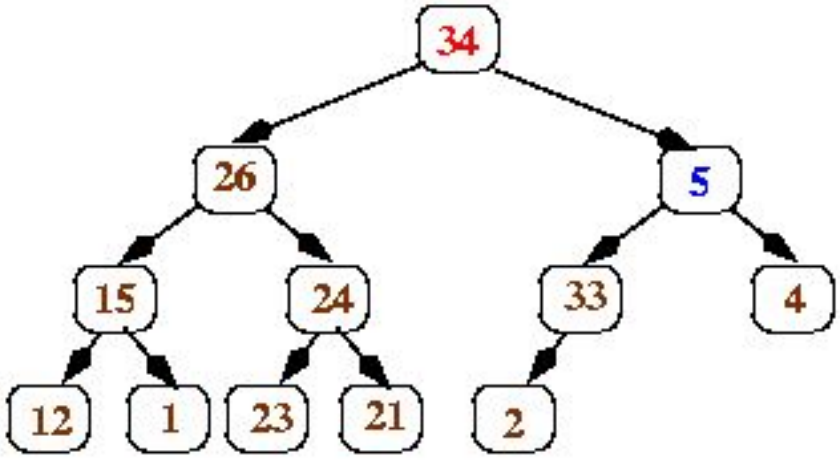
# Heap: Remove

- Replace the value in the root with the value at the end of the array (which corresponds to the heap's rightmost leaf at depth d). Remove that leaf from the tree.

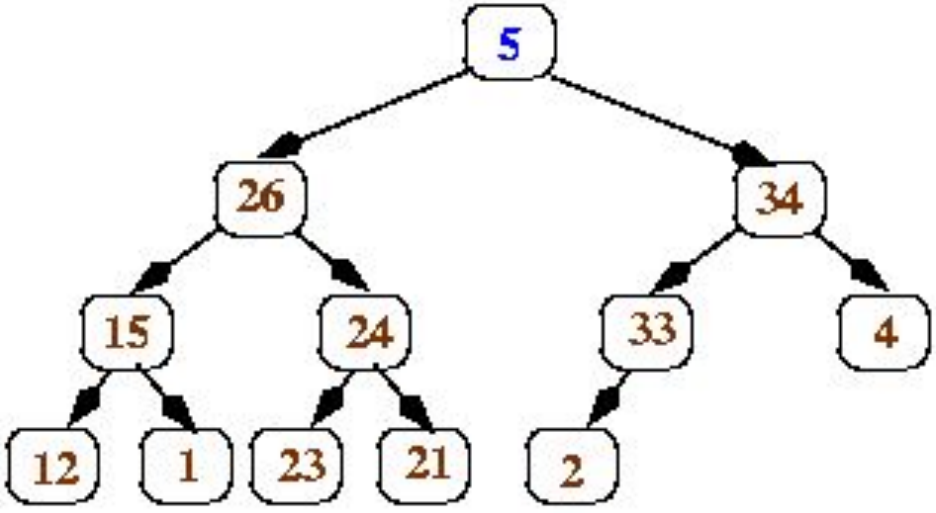
- Now work your way down the tree, swapping values to restore the order property:
  - each time, if the value in the current node is less than one of its children (if heap is max-heap), then swap its value with the larger child (that ensures that the new root value is larger than both of its children)



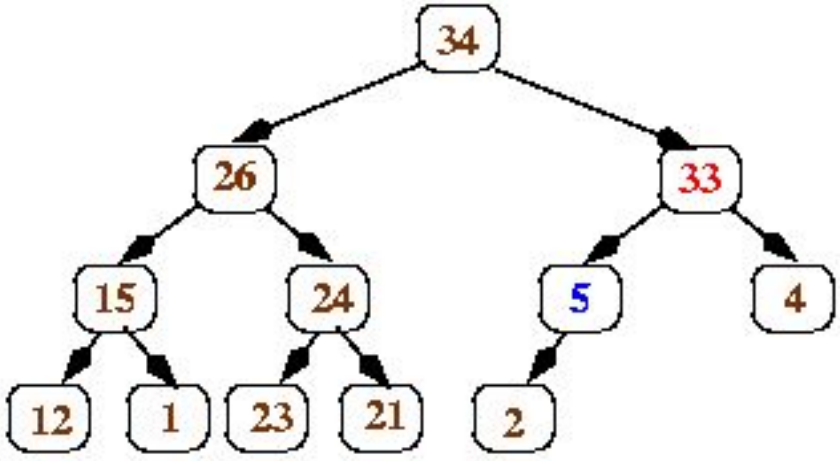

original heap




step 2: swap with larger child




step 1: extract root value and replace with last leaf




step 2: swap with larger child again

All done!

# Priority Queue Implementation using Linked List

Let's discuss how to implement these methods:

- isEmpty()
- peekMin()
- removeMin()
- add(object obj)

# Heap Sort

1. Build a min heap from the array elements.
2. Create an empty result array.
3. While the min heap is not empty:
   a. Remove the minimum element from the heap.
   b. Add the element to the result array.
4. Return the result array.

# Credit

This lecture notes includes material from:

- 15-121 Introduction to Data Structures, Carnegie Mellon University – CORTINA
- http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html
- http://www.eecs.wsu.edu/~ananth/CptS223/Lectures/heaps.pdf