CO322: DS & A
The heap data structure

Dhammika Elkaduwe

Department of Computer Engineering
Faculty of Engineering

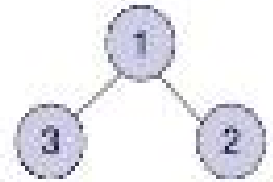
University of Peradeniya

Heap data structure

- Something between a priority queue and binary search tree.
- Recall the simple implementation of priority queue:
  - inserts are $O(N)$ and removal was $O(1)$
  - (or inserts are $O(1)$ and removal was $O(N)$)
  - Can we do better?

Heap basics: a min-heap

- A heap looks like a binary tree (each node has two children)
- Each node has a value smaller than both its children.
- Every level of the heap is full, except for possibly the last level
  - The shape matters: more examples later on
- Elements can be removed or accessed from the root node
  - So the element with the minimum value will be removed first

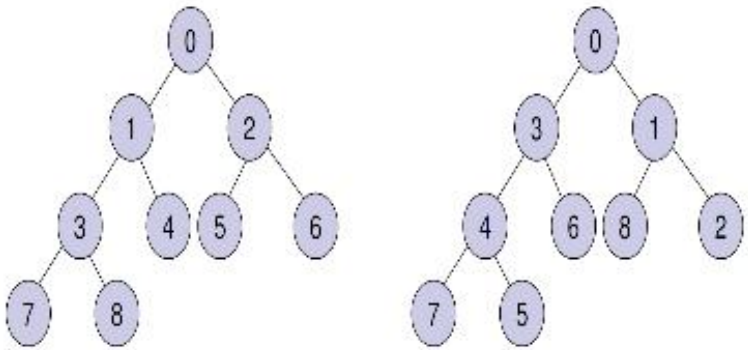


Questions:

- Can you implement a priority queue using a heap?
- Recall the student example where we need the student with the highest GAP. Can you use a min-heap for this?

Min-heap example

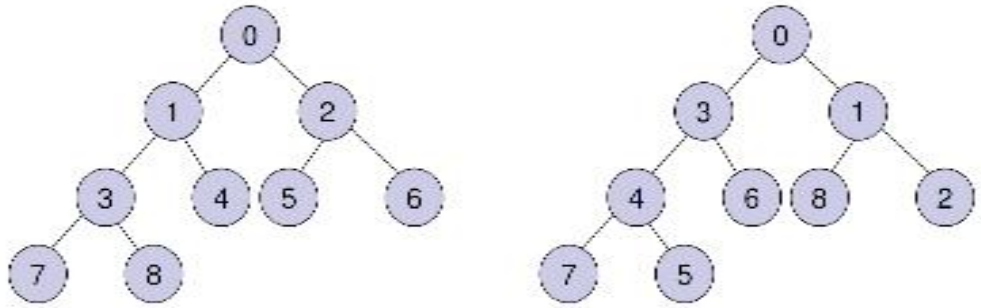
Example of 2 possible heaps on numbers from 0 to 8.



- Min-heap means the minimum value is always at root
- So, minimum value will leave the heap
- shape make sure that tree is *balanced*

Min-heap operations: Adding

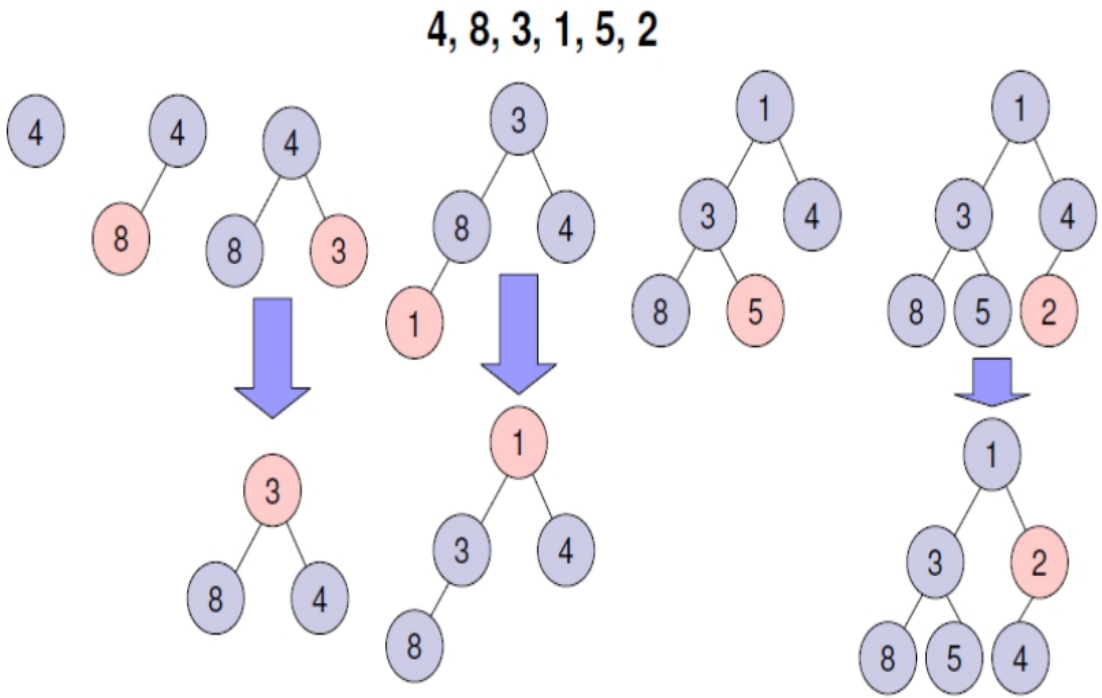
Enter value 5 to the heap.



- Put the new value in the next position at the bottom of the heap (do not worry about the ordering now)
- To maintain min-heap *"bubble"* or *"shift"* the value up the heap by swapping the value with its parent if required.
- Note that adding (or inserting) is $O(log(N))$. (height of the tree)

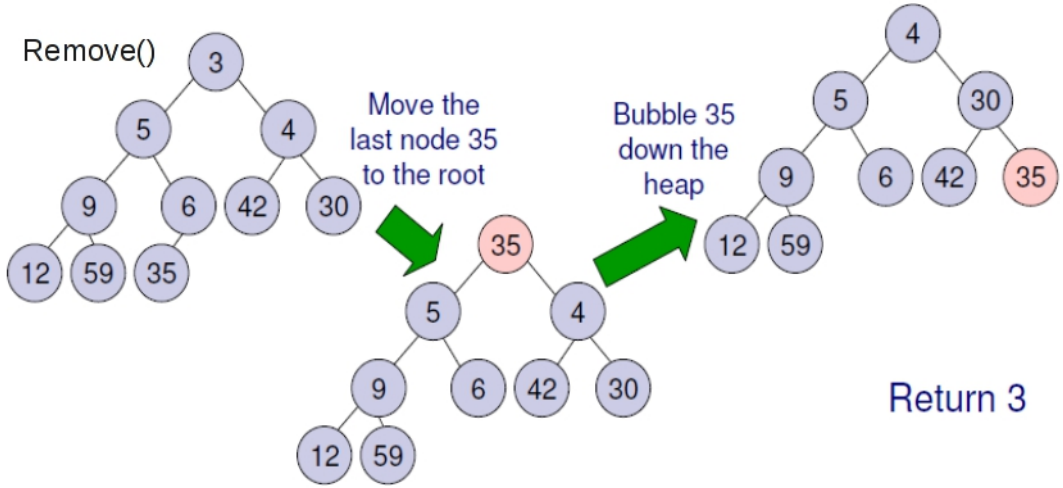
Building a min-heap

Building a heap by inserting following numbers: 4, 8, 3, 1, 5, 2



Min-heap operations: Removing

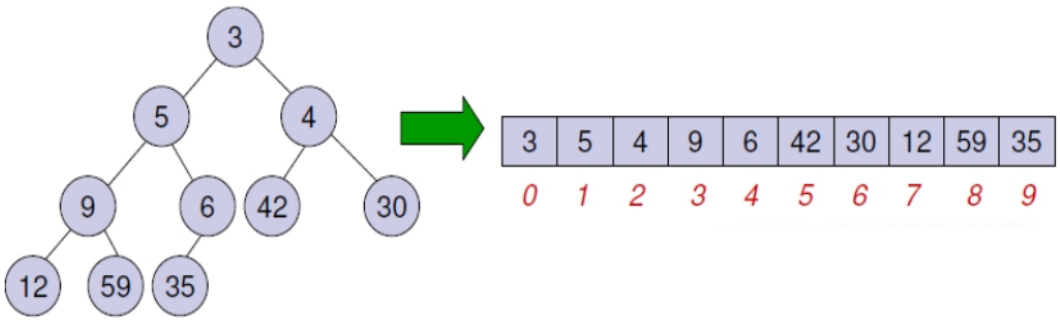
Calling remove function on a heap:



- Only the root node can be removed from the heap.
- But, it has to be replaced with the correct value. Algorithm for this is follows:
  - Replace the root with the last element
  - *Bubble* that value down by swapping till you re-establish the heap properties.
- Removing from a heap is $O(log(N))$

Heap implementation: using an array

- Since the heap is full, we can implement using an array
- Of cause we need to know how many elements are expected
- (some language provides methods to re-size arrays (how?))



For the element at $heap[i]$:

- Left child at: $heap[2i + 1]$
- Right child at: $heap[2i + 2]$
- Parent at: $heap[(i - 1)/2]$ (integer division)

Heap implementation: in Java (preliminaries)

```
public class Heap<T extends Comparable<T>> {
  private int default_size = 10;
  private T[] array;
  private int size;
  public Heap() {
    array = (T[]) new Comparable[default_size];
    size  = 0;
  }
  boolean isRoot(int i) { return (i == 0); }
  int leftChild(int i) { return 2 * i + 1; }
  int parent(int i)   { return (int)((i - 1) / 2); }
  int rightChild(int i) { return 2 * i + 2; }
  T myParent(int i) { return array[parent(i)]; }
```

Heap implementation: in Java (preliminaries...)

```
boolean hasLeftChild(int i) {
  return leftChild(i) < size;
}
boolean hasRightChild(int i){
  return rightChild(i) < size;
}

private void swap(int a, int b) {
  T tmp = array[a];
  array[a] = array[b];
  array[b] = tmp;
}

public boolean isEmpty() { return (size == 0); }
```

Heap implementation: in Java – add a value

```
public void add(T value) {
  if(size == default_size) throw new
      IllegalStateException("Full array");
  array[size++] = value;
  bubbleUp();
}

public void bubbleUp() {
  if(size == 0)
      throw new IllegalStateException("Shape error");
  int index = size - 1;
  while(!isRoot(index)) {
    if(myParent(index).compareTo(array[index]) <= 0)
        break;
    /* else part */
    swap(parent(index), index);
    index = parent(index);
  } /* while */ } /* function */
```

Heap implementation: in Java – Remove a value

You task is to implement the remove function. Algorithm:

- replace the root with the last element
- bubble (down) till heap property is satisfied.

use the skeleton code from Moodle

The heap sort

- We can sort a given data set using a heap: add all elements to the heap, and then remove. Elements will be removed in sorted order.
- Each addition/removal takes $O(log(N))$ and we need N additions and removals. So total time complexity of heap sort is $O(Nlong(N))$
- Additional memory $O(N)$ for constructing the heap

Heaps in Java

Home work:

- find an API for a heap implmentation in Java
- Use that to implement heap sort