CSC 172– Data Structures and Algorithms

Lecture #19

Spring 2018

Please put away all electronic devices

HEAP DATA STRUCTURE

- Illustration of a binary tree in use
- Heap sort algorithm
- Magic behind Priority Queues

PRIORITY QUEUE

Recap: Priority Queues

- Stack: FILO
  - `push(), pop(), peek()`
- Queue: FIFO
  - `offer(), poll(), peek()`
- Priority Queue:
  - Is a Queue
  - Each element has a “priority”
  - `offer()` stores new element (with “priority”)
  - `poll()` removes element with highest/lowest priority
  - `peek()` shows that element but doesn’t remove

Code

```
public static void main(String args[])
   {
       // Creating empty priority queue
       PriorityQueue<Integer> pq =
                          new PriorityQueue<Integer>();

       // Adding items to the pq
       pq.offer(7);
       pq.offer(3);
       pq.offer(5);

       // Printing the most priority element
       System.out.println("Head value :"+ pq.peek());

       // Printing all elements
       System.out.println("The pq elements:");
       Iterator itr = pq.iterator();
       while (itr.hasNext())
           System.out.println(itr.next());

       // Removing the top priority element (or head) and
       // printing the modified pQueue
       pq.poll();
       System.out.println("After removing an element");
       itr = pq.iterator();
       while (itr.hasNext())
           System.out.println(itr.next());

       pq.offer(2);
       System.out.println("After adding 2");
       itr = pq.iterator();
       while (itr.hasNext())
           System.out.println(itr.next());
   }
```

```
Head value using peek function:3
The pq elements:
3
7
5
After removing an element
5
7
After adding 2
2
7
5
```

How to implement a PQ with what's known

- Using an unsorted ArrayList
  - offer takes O(1)
  - peek, poll take O(n)
- Using a sorted ArrayList
  - offer takes O(n)
  - peek, poll take O(1)
- Similarly for a LinkedList
- O(*n*) is way too slow for high-speed routers!

Need New Designs!

Heap

Illustration of a binary tree in use

- Heap sort algorithm
- Priority Queues

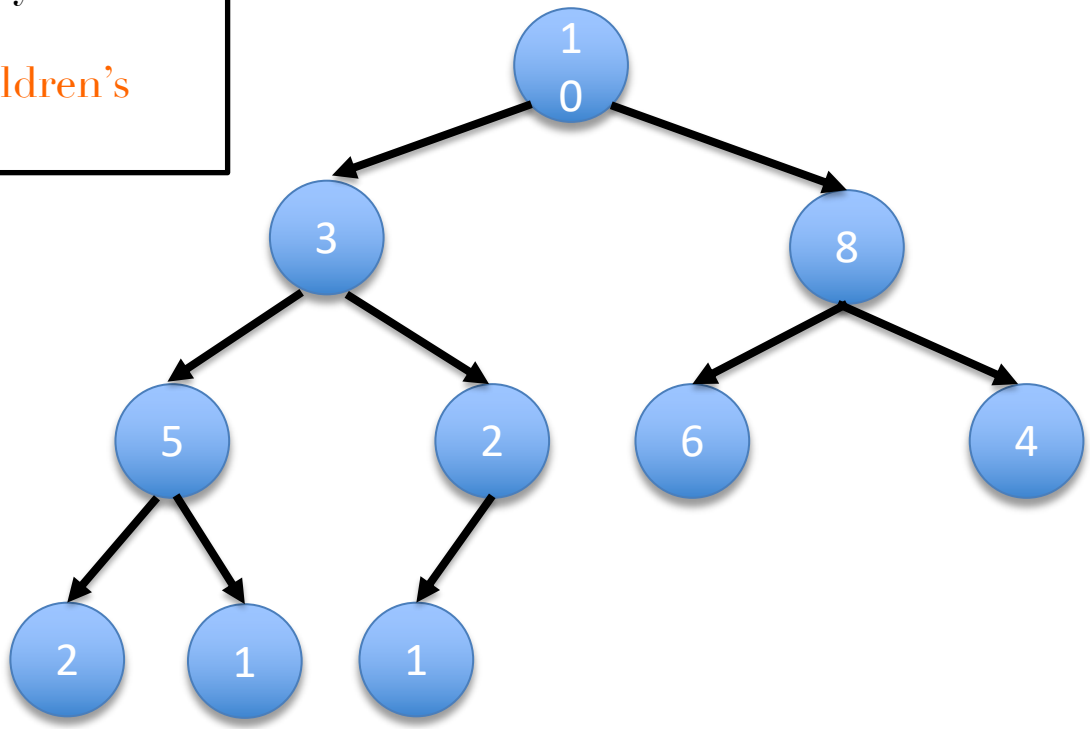
HEAP DATA STRUCTURE AND HEAP SORT ALGORITHM

How can we implement a PQ?

Binary Max Heap

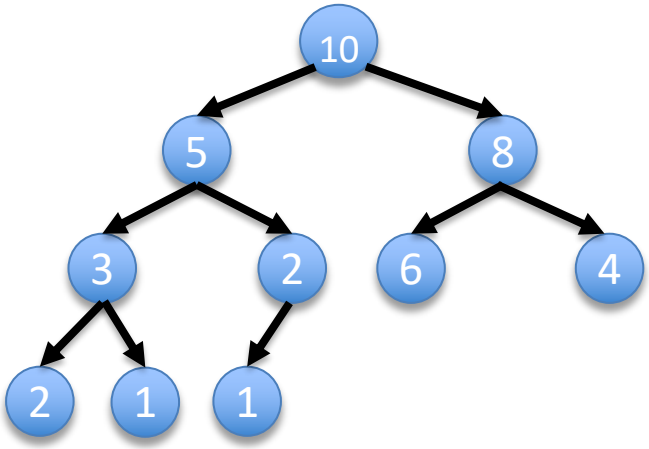
Complete binary tree
- Every level is filled
- Except possibly the last

Node's key ≥ children's keys

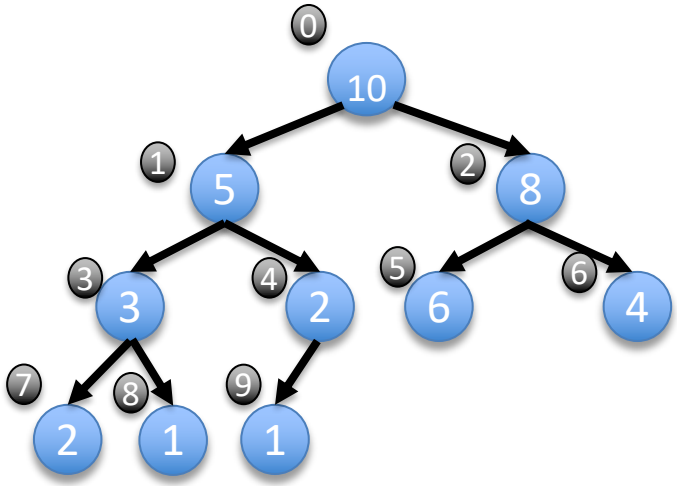


Need to Switch 3 and 5 for making it a heap

Array representation of Heap



Array representation of Heap

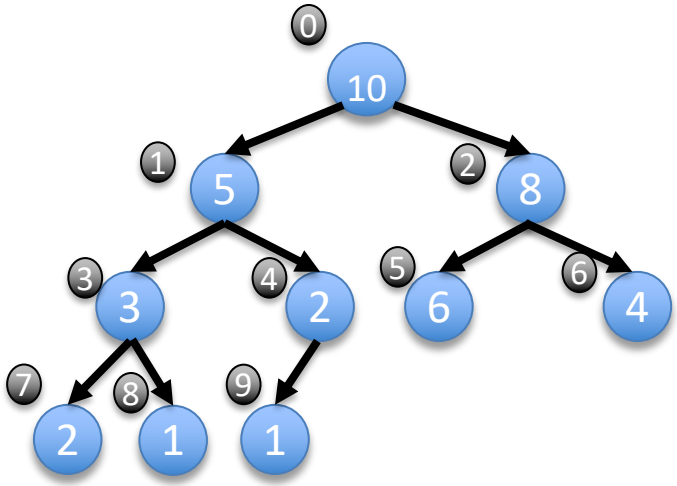


$$left(i) = 2 * i + 1$$
$$right(i) = 2 * i + 2$$
$$parent(i) = (i - 1)/2$$

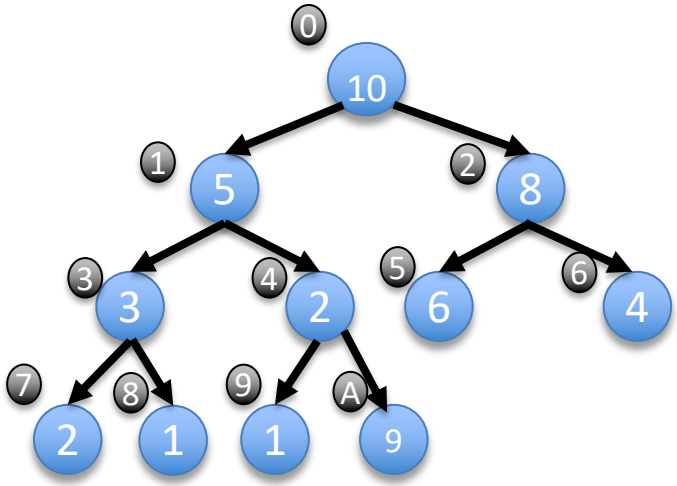
peek

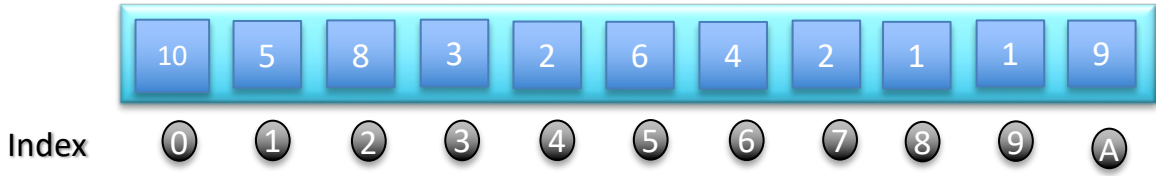
```
int peek() {
  return a[0];
}
```

Adding (`offer`) 9

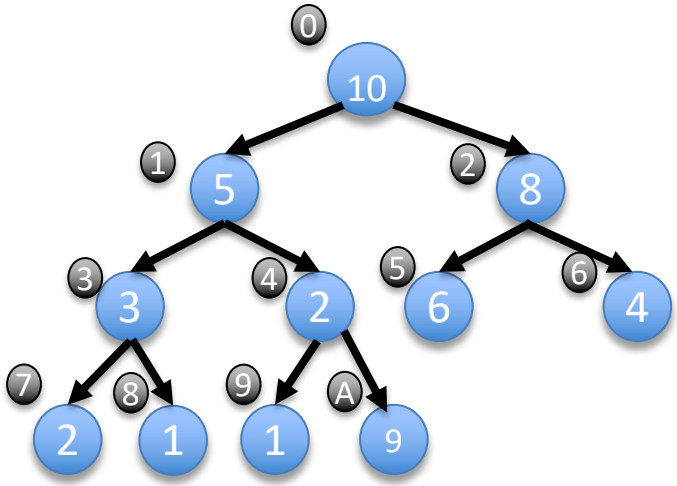


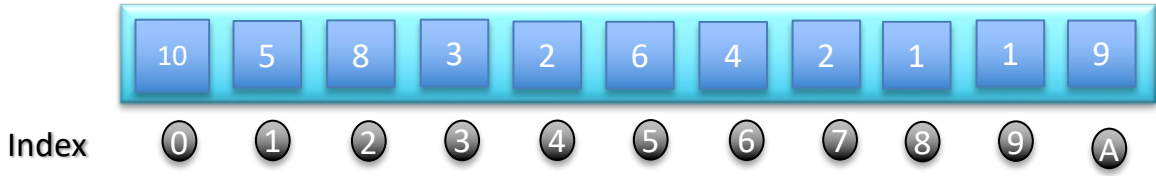
Adding elements (`offer`)



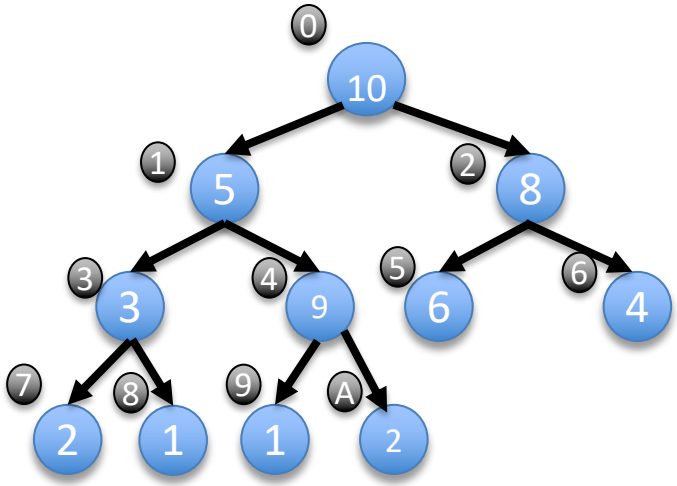


Adding elements (`offer`)

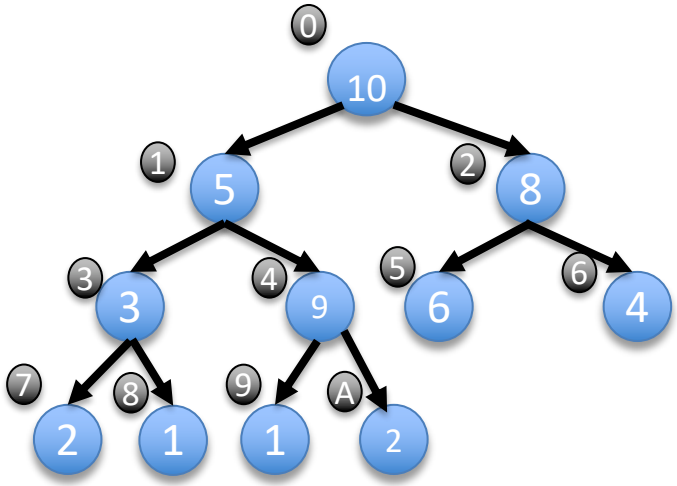




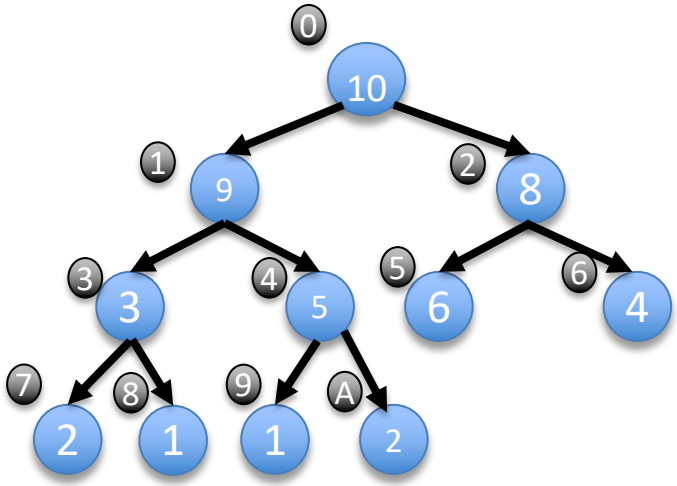
Adding elements (`offer`)



Adding elements (`offer`)



Adding elements (`offer`)



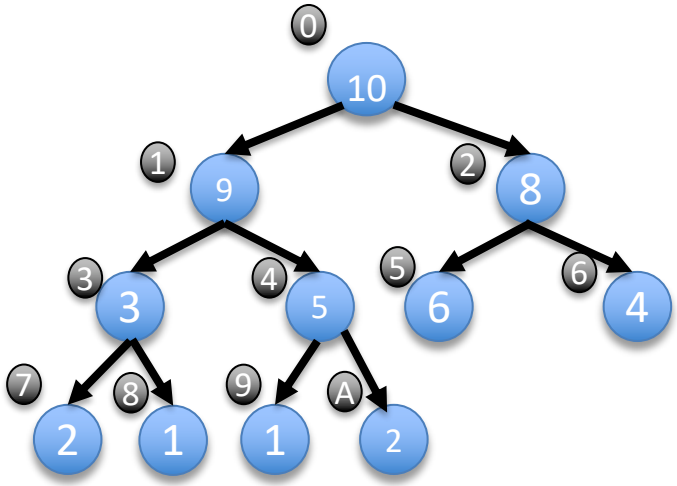
Done!

offer

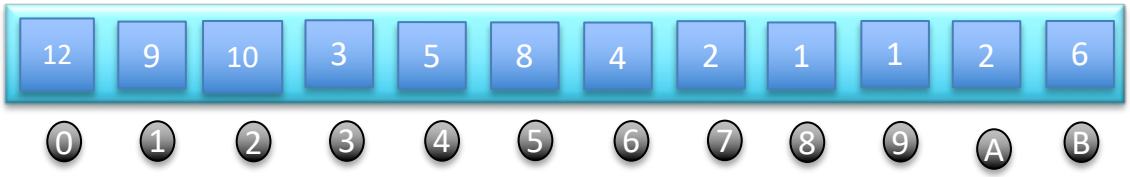
```
void offer(int data) {
   a[size] = data;
   bubbleUp(size);
   size++;
}

void bubbleUp(int pos) {
   if (pos > 0 && a[pos] > a[(pos-1)/2]) {
      swap(a, pos, (pos-1)/2);
      bubbleUp((pos-1)/2);
   }
}
```

Before we proceed…

- Add 12



Answer



This is the format you need to use for the final result. Though, you have to draw the tree and show each modification for full credit.

Another practice problem

- Create a max heap by inserting given integers from left to write.

- 2, 4, 3, 7, 9, 8

Answer: 9, 7, 8, 2, 4, 3

Another practice problem

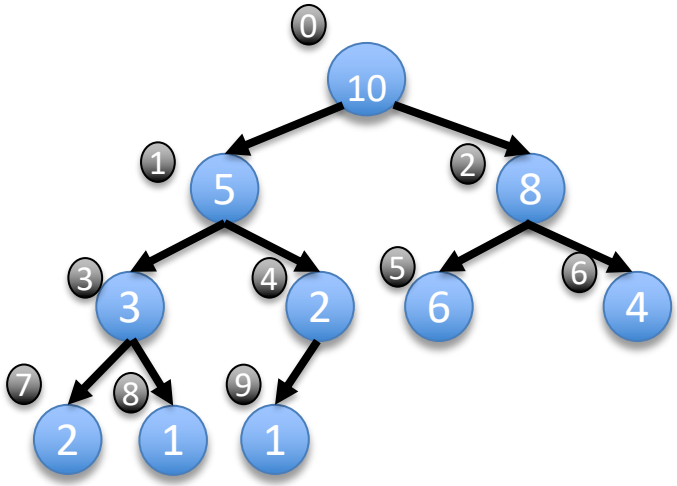
- Create a max heap by inserting given integers from left to write.

- 2, 4, 3, 7, 9, 8

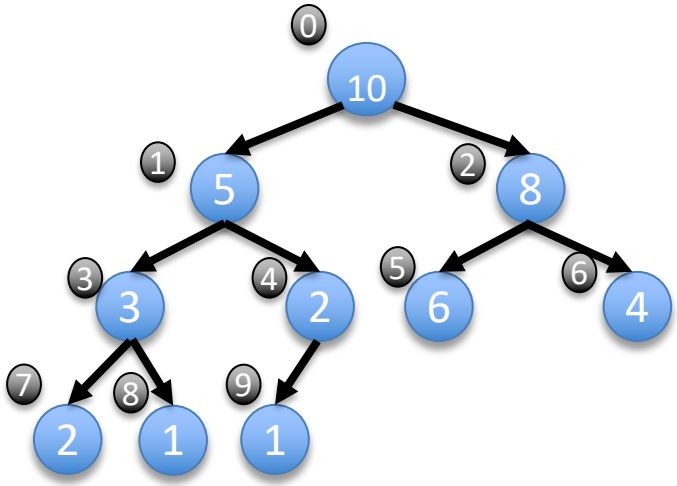
Last one!

- CHEMISTRY
  - Is it a heap?

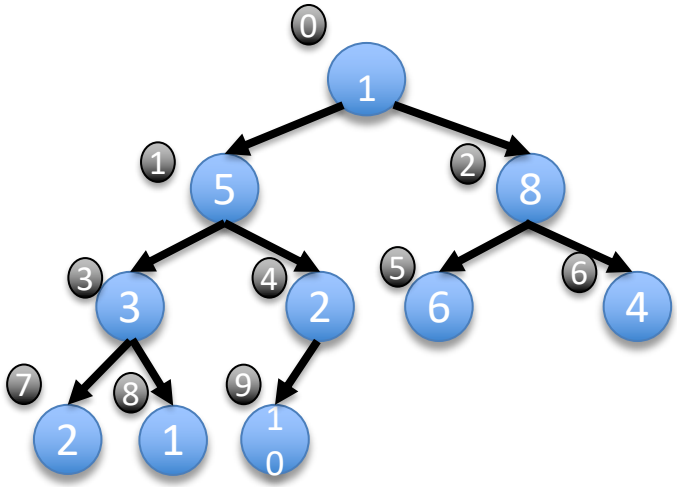
Remove (poll)



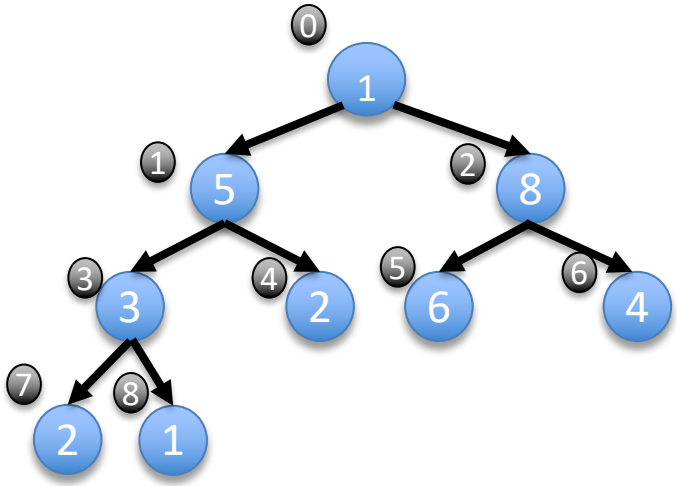
Remove (poll)



Remove (poll)



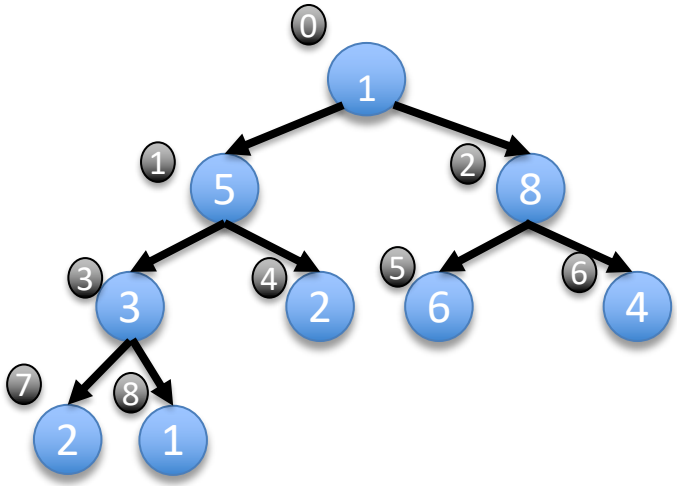
Remove (poll)



Not done yet!
It's a complete tree but not a heap.

What should we do?

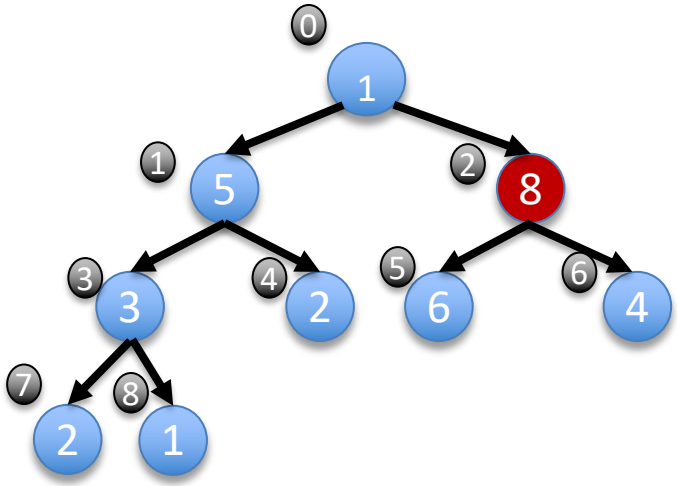
Remove (poll)



Trickle down 1

Find the max of it's two neighbors and swap with the maximum one.

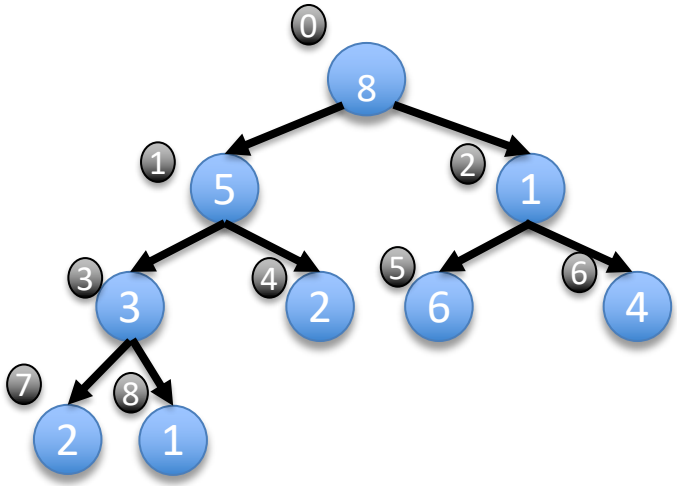
Remove (poll)



Trickle down 1

Find the max of it's two neighbors and swap with the maximum one.

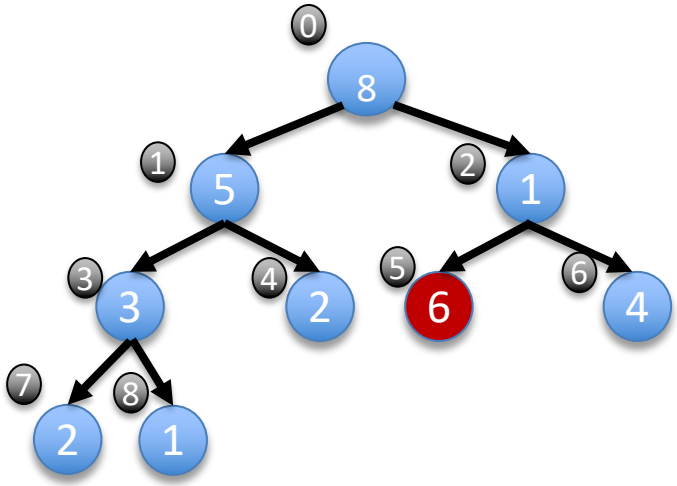
Remove (poll)



Trickle down 1

Find the max of it's two neighbors and swap with the maximum one.

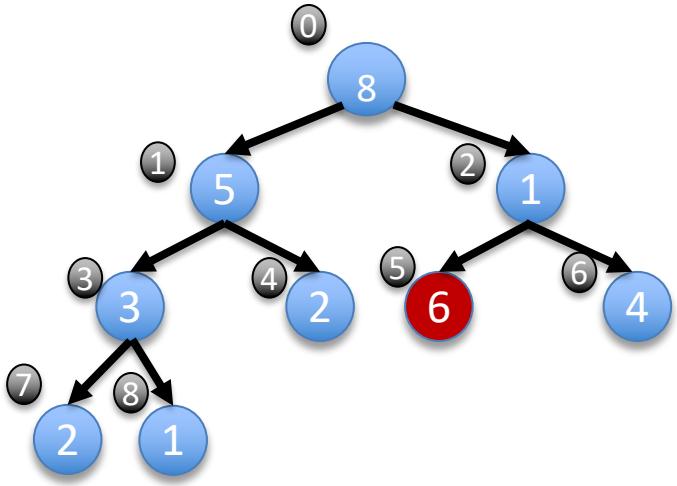
Remove (poll)



Trickle down 1

Find the max of it's two neighbors and swap with the maximum one.

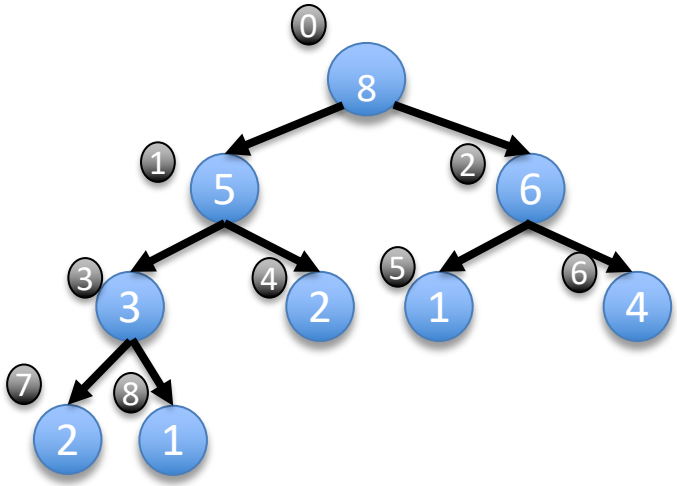
Remove (poll)



Trickle down 1

Find the max of it's two neighbors and swap with the maximum one.

Remove (poll)



Done!

poll

```
int poll() {
    int tmp = a[0];
    swap(a, 0,size-1);
    size--;
    trickleDown(0);
    return tmp;
}
```

```
void trickleDown(int pos) {
   int left = 2 * pos + 1;
   int right = left + 1; //
   possibly >= size
   if (left < size) {
    int greaterChild = left;
    if (right < size && a[left] <
   a[right]) {
        greaterChild = right;
    }
    if (a[greaterChild] > a[pos])
   {
        swap(a, greaterChild,
   pos);
        trickleDown(greaterChild);
    }
   }
}
```

Binary Heap as Priority Queue

- `offer` takes O(log *n*)
- `peek` takes O(1)
- `poll` takes O(log *n*)

- Drawback: search takes a long time

- Extremely simple to implement

Where is a Priority Queue used?

- Scheduling jobs on a server
- In all kinds of networking protocols
  - Buffer/bandwidth management at routers
  - Dijkstra's shortest path in link-state routing
- Huffman coding algorithm
- Prim's algorithm for minimum spanning tree.

Heap sort

- `heapify`: turn a vector/array into a heap

Note: Heapify is an in-place algorithm. So, it's different from inserting elements into the heap one after the other

- `sink(i, n, array)`
  - Makes sub-tree rooted at `i` a max heap
  - Assumes left & right sub-trees are already max heap
  - Sinks node `i` down to the correct level

Note: sink is same as trickle down

- `heap_sort(array)`:
  - `heapify(array)`
  - swap `root` to `array[n]`
  - `sink(0, n-1, array)`

Heapify and heap_sort

```
void heapify(int[] arr) {
    for (int i=arr.length/2; i>=0; i--)
        r_sink(arr, i, arr.length);
}
```

Note: Heapify is an in-place algorithm. So, it's different from inserting elements into the heap one after the other

```
void heap_sort(int[] arr) {
    heapify(arr);
    for (int j=arr.length-1; j>=1; j--) {
        swap(arr, 0, j);
        r_sink(arr, 0, j);
    }
}
```

Sinking, Recursively

```
void r_sink(int[] arr , int i, int n) {
    int left = 2*i + 1;
    if (n > arr.length || left >= n) return;
    int right = left + 1; // possibly >= n
    int my_pick = (right >= n) ?  left :
                      (arr[right] > arr[left]) ? right : left;

    if (arr[i] < arr[my_pick]) {
        swap(arr, i, my_pick);
        r_sink(arr, my_pick, n);
    }
}
```

Heap Sort Runtime

- Runtime to heapify:
  - Not $O(nlog\ n)$
  - Rather: $O(n)$

  - Why: http://www.cs.umd.edu/~meesh/351/mount/lectures/lect14-heapsort-analysis-part.pdf

- Total Runtime = Heapify + n * Sinking
- $= O(n) + O(nlogn)$
- $= O(nlogn)$

Summary

- Heap
  - Min and Max Heap
  - Used for implementing Priority Queue
  - Used for Sorting
  - We never build the tree --- it's always an array
  - Heapify is an in-place algorithm and it's runtime is O(n)

Acknowledgement

- Douglas Wilhelm Harder.
  - Thanks for making an excellent set of slides for ECE 250 *Algorithms and Data Structures* course

- Prof. Hung Q. Ngo:
  - Thanks for those beautiful slides created for CSC 250 (Data Structures) course at UB.

- Many of these slides are taken from these two sources.