Discussion 1F Week 10

Spring 2021

TA: Manoj Reddy

LA: Katherine Zhang

Credit: Prof. Carey Nachenberg

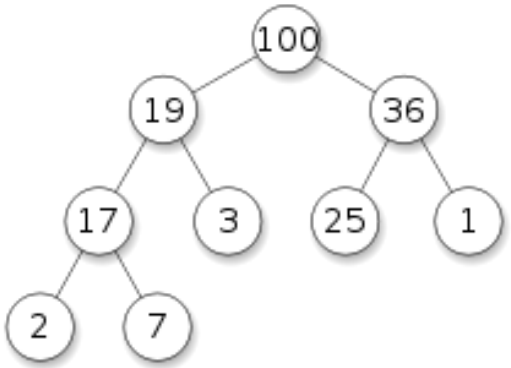
Outline

- Priority Queues
- Heaps
- Review
- Course Evaluation

Priority Queues

- Priority Queue is a special type of queue that allows to keep a prioritized list of items
- Insert:
  - Each item you insert into the queue has a “priority rating”
- Dequeue:
  - Removes the item with the highest priority
- Example:
  - ER: Priority can be based on the severity of the injury
- Used in Operating Systems, Network Packet Routing
- Implementation (Simple Approach):
  - Store n queues, one for each priority level
  - Doesn’t scale???

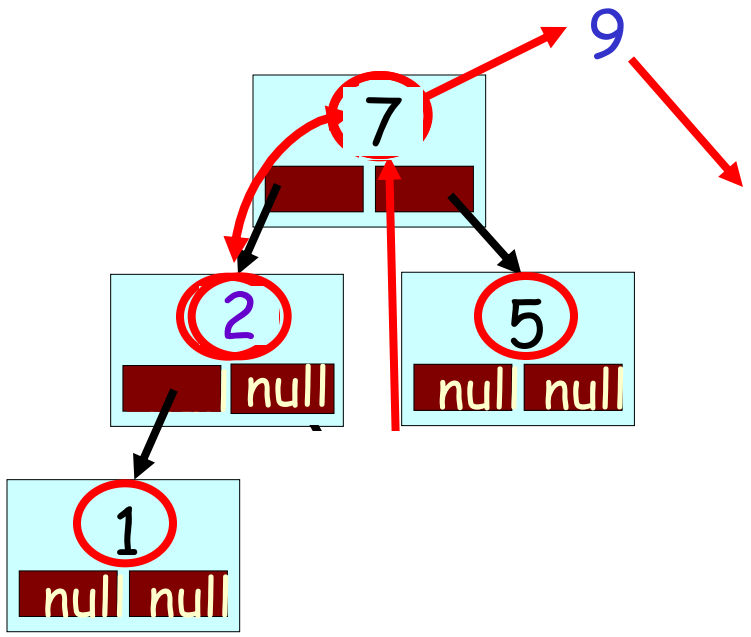
Heap



- Data structure to efficiently implement Priority Queues
- Relies on Complete Binary Tree
  - Top N-1 levels of the tree are completely filled with nodes
  - All nodes on the bottom-most level must be as far left as possible (with no empty slots between nodes!)
- Heaps are not Binary Search Trees (BST)
- Heaps are designed to retrieve the largest/smallest element in the tree efficiently
- Two types of heaps:
  - Max Heaps: Value contained by a node is always greater than or equal to the values of the node's children
  - Min heaps: Value contained by a node is always smaller than or equal to the values of the node's children

Extracting the Biggest Item

1. If the tree is empty, return error.
2. Otherwise, the top item in the tree is the biggest value. Remember it for later.
3. If the heap has only one node, then delete it and return the saved value.
4. Copy the value from the right-most node in the bottom-most row to the root node.
5. Delete the right-most node in the bottom-most row.
6. Repeatedly swap the just-moved value with the larger of its two children until the value is greater than or equal to both of its children. ("sifting DOWN")
7. Return the saved value to the user.

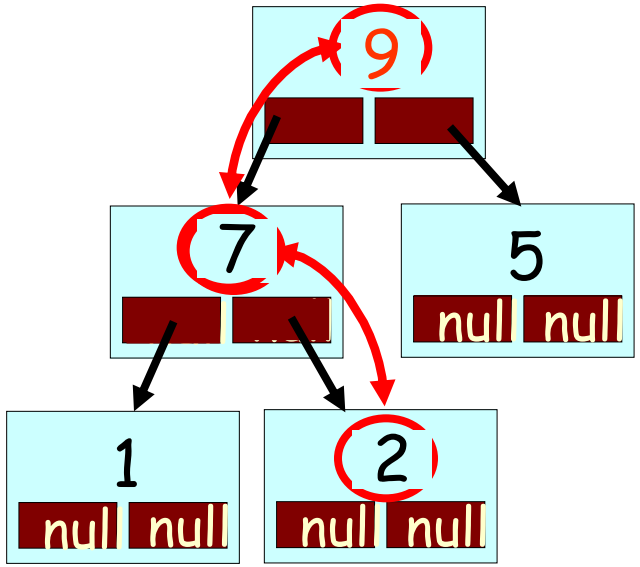


When we're done, the largest value is on the top again, and the heap is consistent.

Adding a Node to a Maxheap

(Let's see how to add a value of 9)

1. If the tree is empty, create a new root node & return.
2. Otherwise, insert the new node in the bottom-most, left-most position of the tree (so it's still a complete tree).
3. Compare the new value with its parent's value.
4. If the new value is greater than its parent's value, then swap them.
5. Repeat steps 3-4 until the new value rises to its proper place.



This process is called "reheapification."

Implementing A Heap

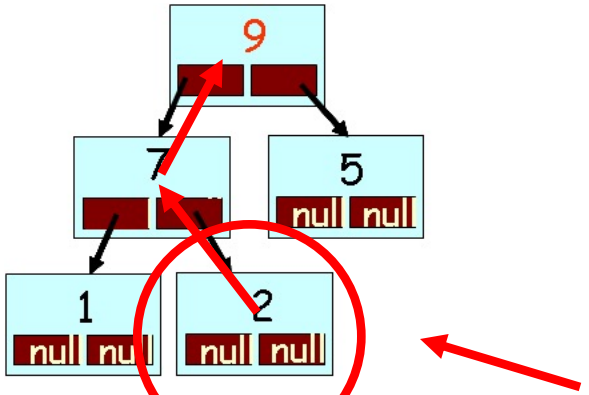
Question:

What data structure can we use to implement a heap?

How about a classical binary tree node with links?

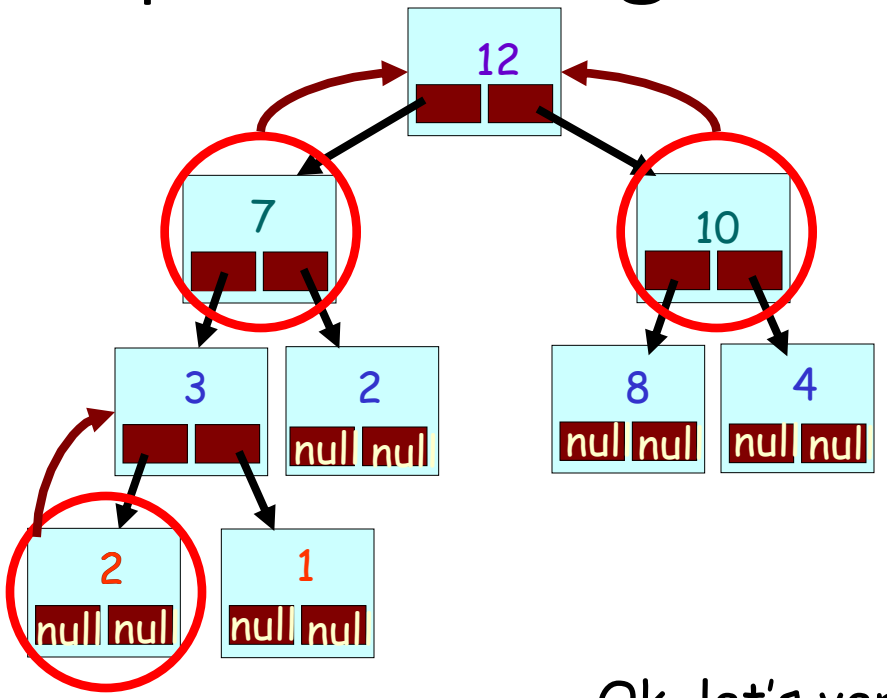
Hmmm… But this has some challenges. What are they?

```
struct node
{
   int value;
   node *left, *right;
};
```



1. It's not easy to locate the bottom-most, right-most node during extraction.
2. It's not easy to locate the bottom-most, left-most open spot to insert a new node during insertion!
3. It's not easy to locate a node's parent to do reheapification swaps.

Implementing A Heap



int count;

9

int heap[1000];

0	12
1	7
2	10
3	3
4	2
5	8
6	4
7	2
8	1
9	
10	
11	
12	
13	
...	

Ok, let's verify that it works...

The parent of slot #1 is... (1-1)/2 = 0

The parent of slot #2 is... (2-1)/2 = 0

The parent of slot #7 is... (7-1)/2 = 3

$$\text{parent} = \frac{\text{child}-1}{2}$$

Cool stool! So now we know how to locate the children of a node, find the parent of a node, and add and remove nodes!

Heap in an Array Summary

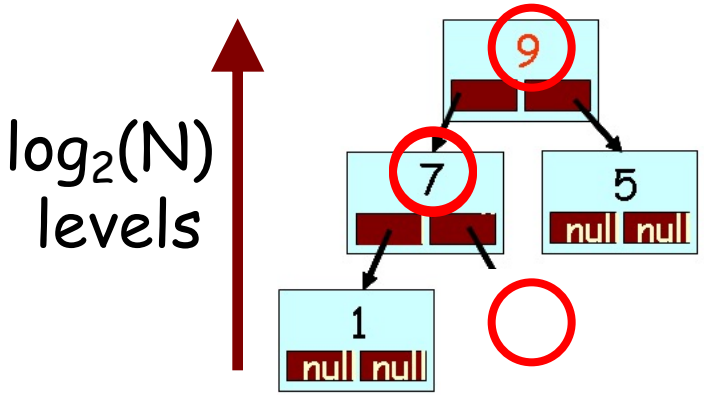
So, now we know how to store a heap in an array!

Here's a recap of what we just learned :

1. The root of the heap goes in array[0]
2. If the data for a node appears in array[i], its children, if they exist, are in these locations:
   Left child: array[2i+1]
   Right child: array[2i+2]
3. If the data for a non-root node is in array[i], then its parent is always at array[(i-1)/2] (Use integer division)

Complexity of the Heap

Question: What is the big-oh cost of inserting a new item into a heap?



Every time we insert a new item, we need to keep comparing it with its parent until it reaches the right spot or not our heap is stored in an array!)

So in the worst case, we'll have to do $\log_2(N)$ comparisons and $\log_2(N)$ swaps of our new value, (This is the worst case.) Since our tree is a COMPLETE binary tree it has N entries, it's guaranteed to be exactly $\log_2(N)$ levels deep.

Question: What is the big-oh cost of extracting the maximum/minimum item from a heap?

Just as with heap insertion, when we extract a value we need to bubble an item from the root down the tree. Since the maximum number of levels in our tree is $\log_2(N)$, the worst case that this requires $\log_2(N)$ swaps.

So inserting and extracting from a heap is $O(\log_2(n))$

The Heapsort

Question:
How can we use a heap to sort a bunch of items?

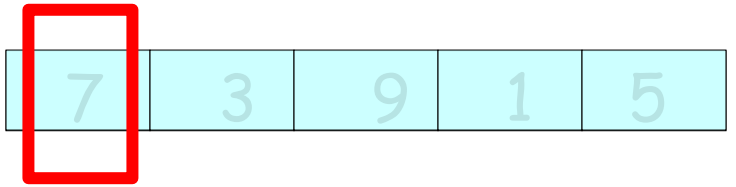
Answer:
Here's a "naïve" way to do it…

Given an array of N numbers that we want to sort:

1. Insert all N numbers into a new maxheap
2. While there are numbers left in the heap:
   A. Remove the biggest value from the heap
   B. Place it in the last open slot of the array

1

And voila!
Our array is sorted!



The Naïve Heapsort

Question:

What's the complexity of our simple version of heapsort?

Hints:

What is the cost of inserting an item into a maxheap?
What is the cost of extracting an item from a maxheap?
How many items must be inserted and then extracted?

Insertions: N items * $\log_2(N)$ steps per item → N $\log_2(N)$ steps

Extractions: N items * $\log_2(N)$ steps per item → N $\log_2(N)$ steps

That comes to 2 * N $\log_2(N)$ total steps, or $O(N \log_2(N))$.

Not bad! But in fact there's an even faster way to use a heap to sort an array… Let's see it!

The Efficient Heapsort

In our naïve algorithm, we took every item from the array and inserted it into a separate, brand-new maxheap.

So first we built a separate maxheap from scratch, copying every one of our items over!

And then we had to remove each one from our new heap and stick them back into our original array. So SLOW!

Question:
Could we have avoided creating a whole new maxheap and moving our numbers back and forth?

Answer:
Yes! That's the way the "official" Heapsort works!

Please complete Course Evaluations on MyUCLA