Programming Abstractions

CS106B

Cynthia Bailey Lee
Julie Zelenski

Topics:

- Priority Queue ADT
  - Heap data structure implementation
    - What are binary trees?
    - What are heaps?
    - How do we do insert/remove operations on heaps?

Priority Queue

Emergency Department waiting room operates as a priority queue: patients are sorted according to priority (urgency), not "first come, first serve" (in computer science, "first in, first out" or FIFO).

Contents of one element of a Priority Queue

- Individual elements of our priority queue will have two pieces to them:
  - An integer indicating the **priority** of this element
    - We will use smaller number means higher priority, but could be done either way
  - A “**payload**” of whatever the actual element data is
    - Examples:
      - **a class MedicalRecord** that has many fields and is the patient’s entire medical history
      - **a string** that is the name of a student waiting in the Lair queue (in a world where Lair is based on urgency of request, rather than FIFO)
      - etc.

6	"SooMin"	13	"Diego"	15	"Muhammad"	22	"Sasha"

Two priority queue implementation options

0		1		2		3		4	
22	"Sasha"	6	"SooMin"	15	"Muhammad"	13	"Diego"		

Unsorted array

- Always insert new element *at the end of the array*
- Remove by searching entire array for highest-priority item, then removing it, and (if needed) scooting elements over to fill in the gap

0		1		2		3		4	
22	"Sasha"	15	"Muhammad"	13	"Diego"	6	"SooMin"		

Sorted array

- Always insert new elements *where they go* in priority-sorted order, with the highest-priority item at the end of the array
- Remove by taking the last element of the array

Priority queue implementations

Unsorted array

Add is FAST

- Just throw it in the array at the back
- **O(1)**

Remove/peek is SLOW

- Hard to find item the highest priority item—could be anywhere
- Might need to scoot over elements to fill gap
- **O(N)**

0		1		2		3		4	
22	"Sasha"	6	"SooMin"	15	"Muhammad"	13	"Diego"		

Priority queue implementations

Sorted array

Add is **SLOW**

- Need to step through the array to find where item goes in priority-sorted order
- If proper place is in the front/middle, need to scoot over other elements to make room
- **O(N)**

Remove/peek is **FAST**

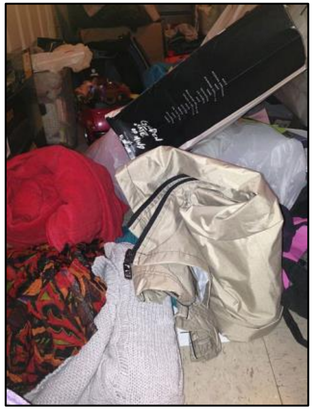
- Easy to find item you are looking for (last in array)
- No need to scoot over elements when removing last
- **O(1)**

Image is in the public domain.
http://commons.wikimedia.org/wiki/File:Wall_Closet.jpg

0		1		2		3		4	
6	"SooMin"	13	"Diego"	15	"Muhammad"	22	"Sasha"		

Would be great if we could get the best of both…

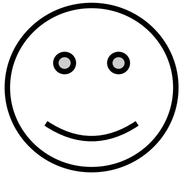
Fast add *and* fast remove/peek


Fast add

+

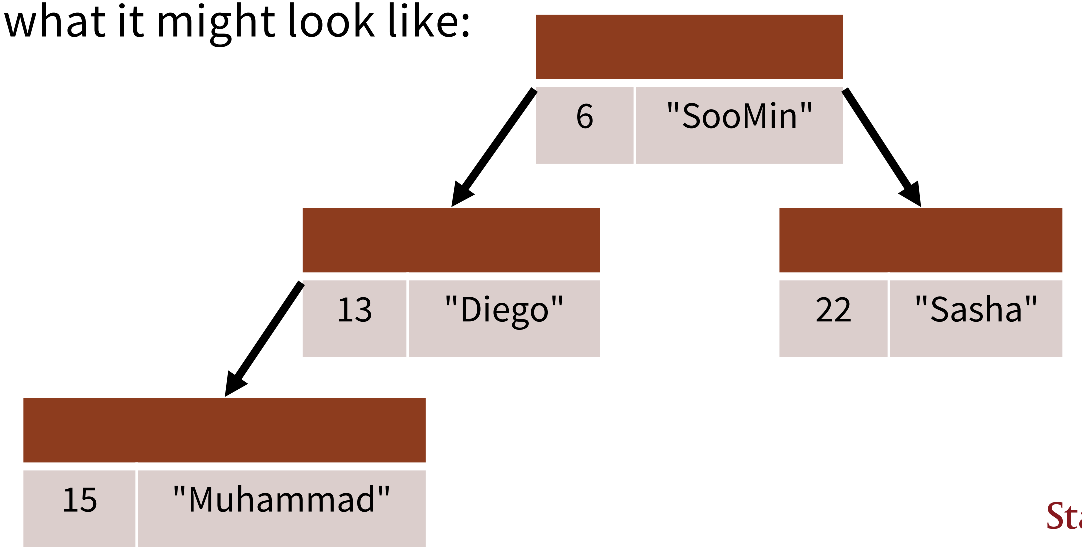
Fast remove/peek

=



Binary heap for our priority queue

- Instead of storing our priority queue nodes entirely sorted or entirely unsorted, we will store them *partially-sorted*.
- The partial sorting will still be stored in an array, but it's best to imagine it as what we call a "tree" in computer science (computer science trees are upside-down for some reason ¯_(ツ)_/¯)
- Here's what it might look like:



Stanford University

Binary trees

Before we delve into how to construct a binary heap, let’s take a step back and introduce computer science binary trees generally

A binary tree

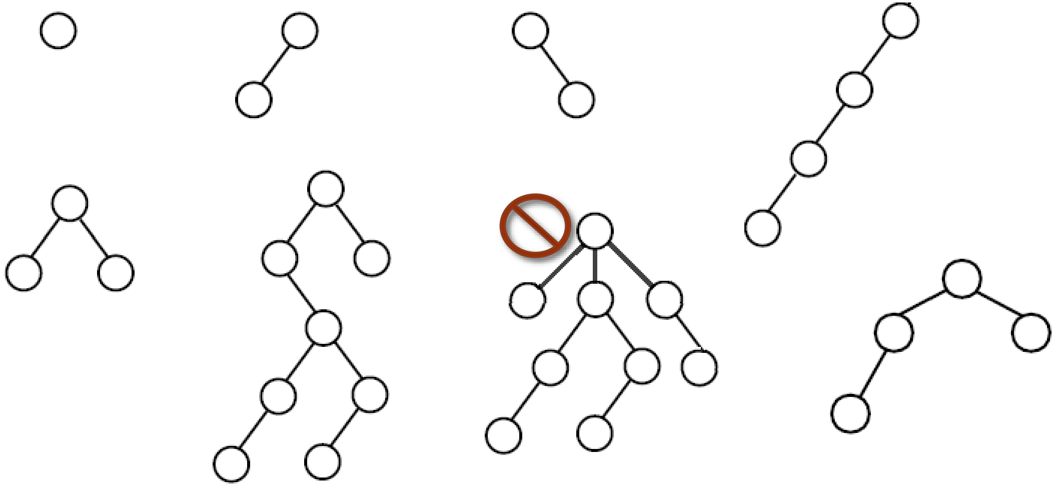
“In computer science, a **binary tree** is a tree data structure in which each node has at most two child nodes, usually distinguished as "left" and "right."”

(Thanks, Wikipedia!)

How many of these are valid binary trees?

“In computer science, a **binary tree** is a tree data structure in which each node has at most two child nodes, usually distinguished as "left" and "right."”

(Thanks, Wikipedia!)



Stanford University

Heaps!

Binary Heaps*

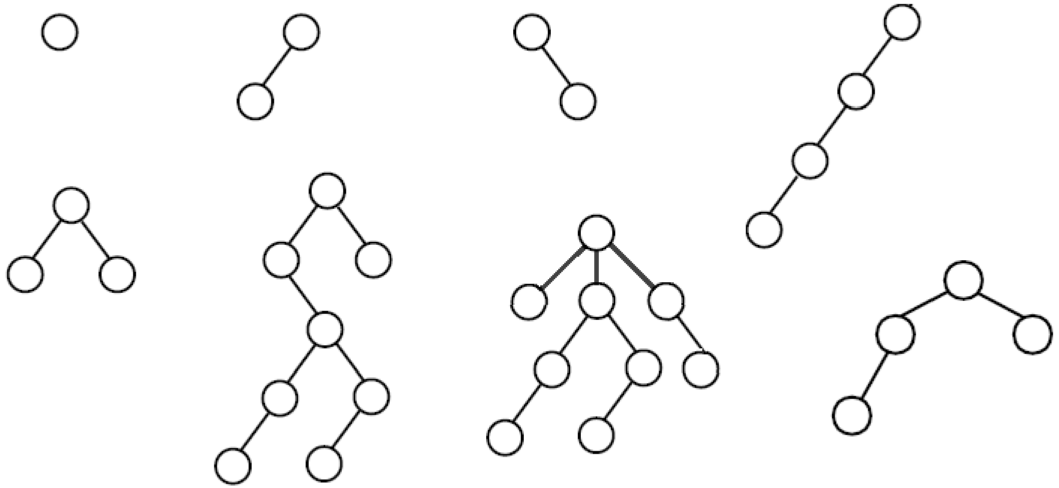
Binary heaps are **one kind** of binary tree

They have a few special restrictions, in addition to the usual binary tree:

- Must be **complete**
  - No “gaps”—nodes are filled in left-to-right on each level (row) of the tree
- Ordering of data must obey **heap property**
  - Min-heap version: a parent’s priority is always ≤ both its children’s priority
  - Max-heap version: a parent’s priority is always ≥ both its children’s priority

* There are other kinds of heaps as well. For example, binomial heap is an extra-fun one!

How many of these could be valid binary heaps?



A. 0-1

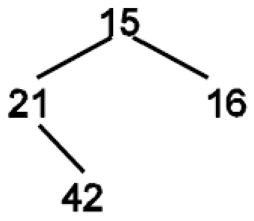
B. 2

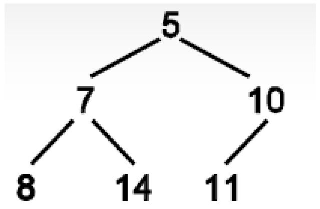
C. 3

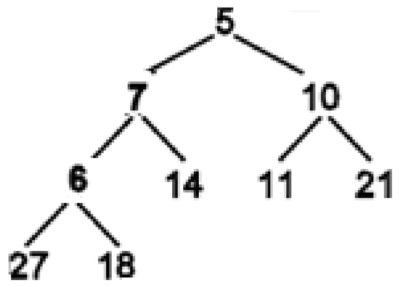
D. 4

E. 5-8

How many of these are valid min-binary-heaps?



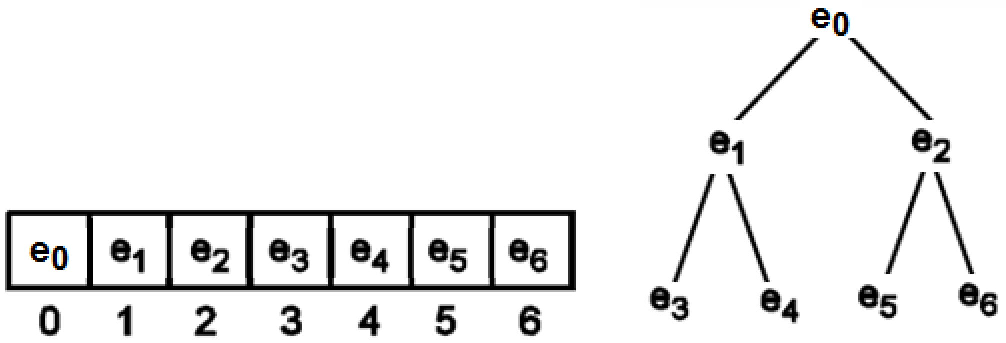




Binary heap in an array

Binary heap in an array

- Because of the special constraint that they must be **complete**, binary heaps fit nicely into an **array**
  - *As we'll see later, this is not true of some other kinds of tree data structures, and we'll use a different approach for those*

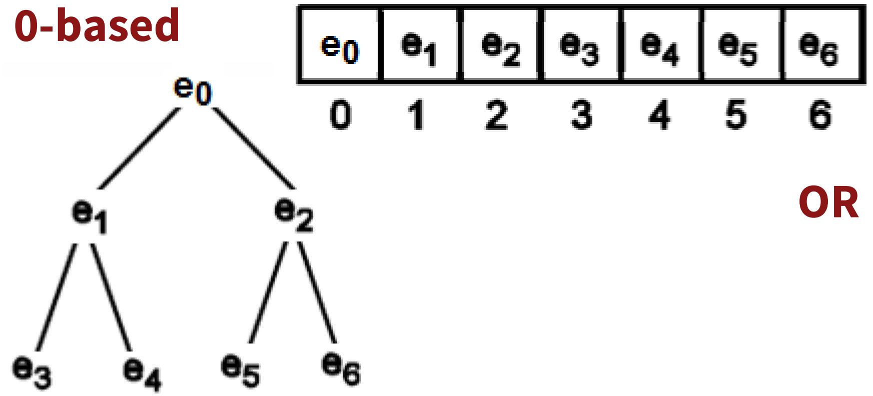


Two approaches: Binary heap in an array

Wait, but the homework handout starts storing the elements at array index 1!

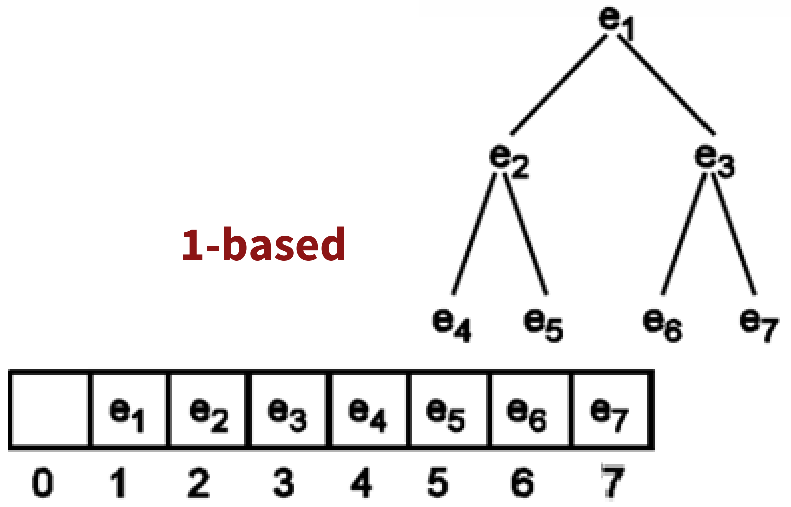
- Either way is ok for the assignment.
- You should understand both ways, so we're teaching both ways

0-based

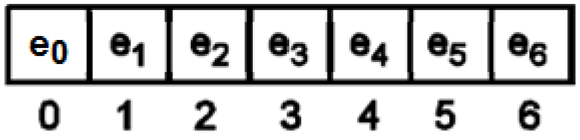


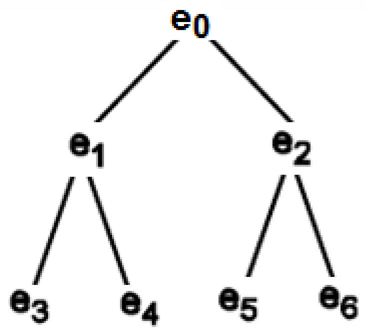
OR

1-based



Heap in an array

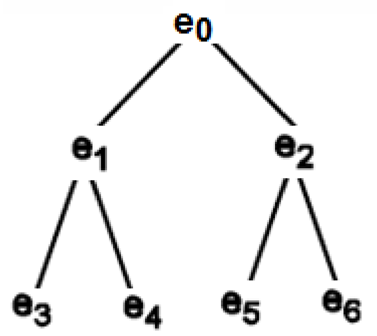


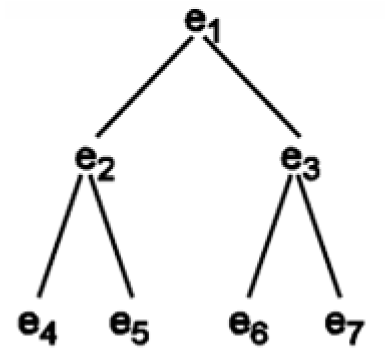


For a node in array index i:

- Q: The parent of that node is found where?
- A: at index:
  - A. i – 2
  - B. i / 2
  - C. (i – 1)/2
  - D. 2i

Fact summary: Binary heap in an array





0-based:

For tree of height h, array length is 2^h-1

For a node in array index i:

- Parent is at array index: (i – 1)/2
- Left child is at array index: 2i + 1
- Right child is at array index: 2i + 2

1-based:

For tree of height h, array length is 2^h

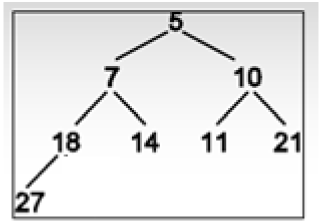
For a node in array index i:

- Parent is at array index: i /2
- Left child is at array index: 2i
- Right child is at array index: 2i + 1

Stanford University

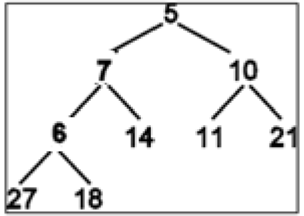
Binary heap enqueue and dequeue

Binary heap enqueue example (insert 6 + "bubble up")



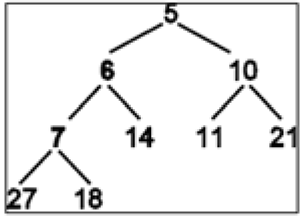
Size=8, Capacity=15

0	1	2	3	4	5	6	7	8	9	…	14
5	7	10	18	14	11	21	27	6	?	…	?



Size=9, Capacity=15

0	1	2	3	4	5	6	7	8	9	…	14
5	7	10	6	14	11	21	27	18	?	…	?

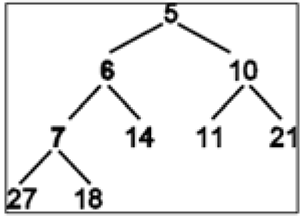


Size=9, Capacity=15

0	1	2	3	4	5	6	7	8	9	…	14
5	6	10	7	14	11	21	27	18	?	…	?

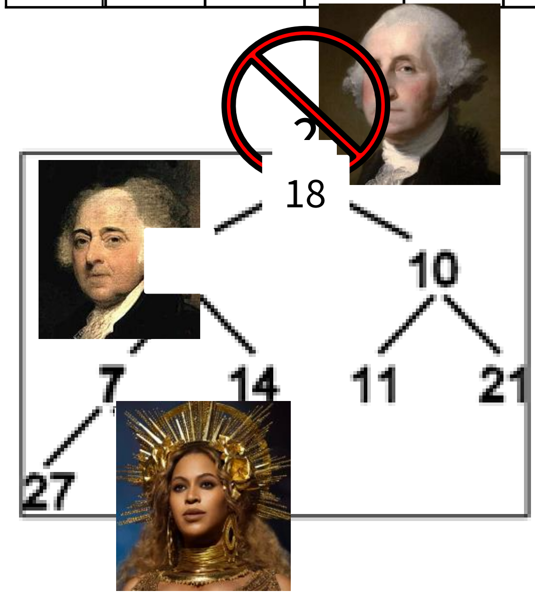
Stanford University

Binary heap dequeue (delete min)

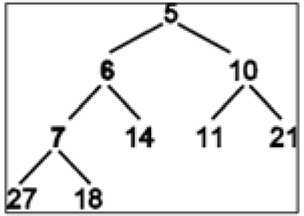


```
Size=9, Capacity=15
```

0	1	2	3	4	5	6	7	8	9	…	14
18	6	10	7	14	11	21	27		…	…	…

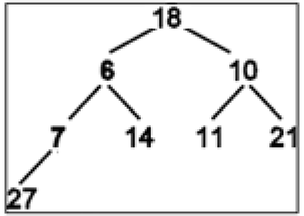


Binary heap dequeue (delete min + "trickle down")



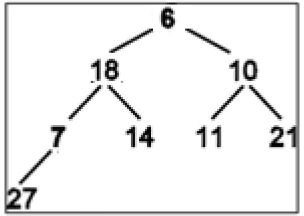
Size=9, Capacity=15

0	1	2	3	4	5	6	7	8	9	…	14
5	6	10	7	14	11	21	27	18	?	…	?



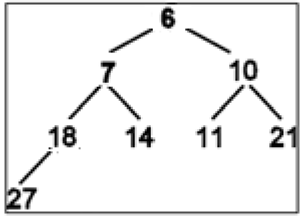
Size=**8**, Capacity=15

0	1	2	3	4	5	6	7	8	9	…	14
18	6	10	7	14	11	21	27	18	?	…	?



Size=8, Capacity=15

0	1	2	3	4	5	6	7	8	9	…	14
6	**18**	10	7	14	11	21	27	18	?	…	?



Size=8, Capacity=15

0	1	2	3	4	5	6	7	8	9	…	14
6	**7**	10	**18**	14	11	21	27	18	?	…	?

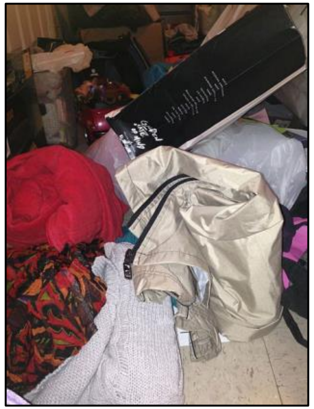
Summary analysis

Comparing our priority queue options

Stanford University

Would be great if we could get the best of both…

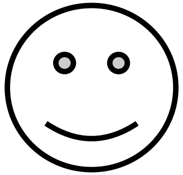
Fast add *and* fast remove/peek


Fast add

+

Fast remove/peek

=



Review: priority queue implementation options performance

Unsorted array

- Insert new element in back: O(1)
- Remove by searching list and scooting over: O(N)

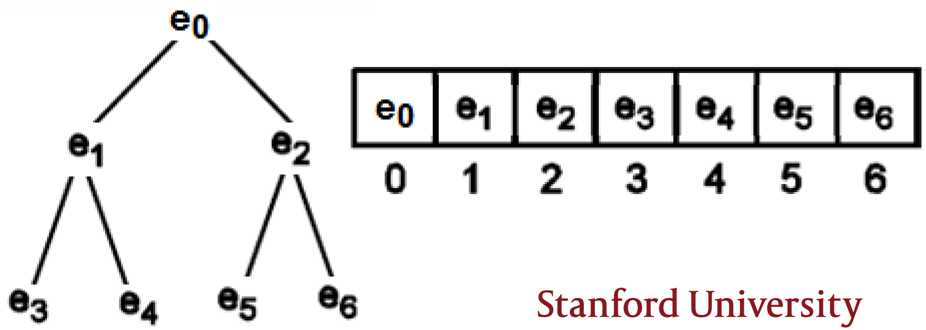
Sorted array

- Always insert in sorted order: O(N)
- Remove from back: O(1)

Binary heap

- Insert + "bubble up": O(logN)
- Delete + "trickle down": O(logN)

☺

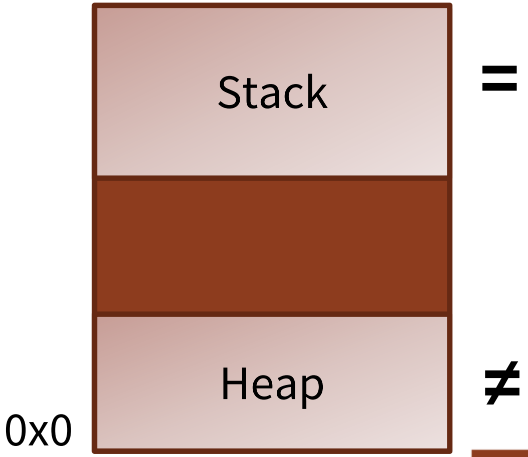


Final aside on terminology

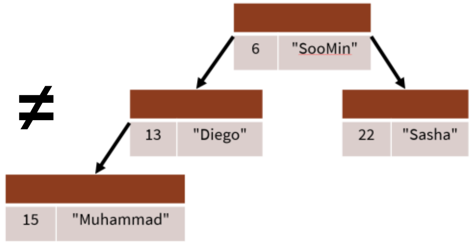
Stanford University

Aside: Binary Heap, not to be confused with Heap memory!

- The Stack section of memory **is** a Stack like the ADT
- The Heap section of memory **has nothing to do** with the Heap structure.



Stack ADT



Heap data structure

- Probably just happened to reuse the same word ☹

Stanford University

Source: http://www.flickr.com/photos/35237093334@N01/409465578/
Author: http://www.flickr.com/people/35237093334@N01 Peter Kazanjy]