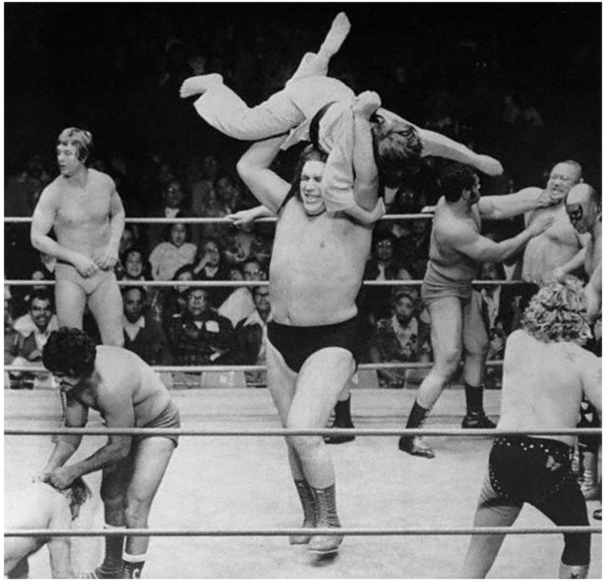


Lecture 21 (Data Structures 5)

Priority Queues and Heaps

CS61B, Spring 2025 @ UC Berkeley

Slides credit: Josh Hug

Introducing the Priority Queue

Lecture 21, CS61B, Spring 2025

The Priority Queue Interface

```java
/** (Min) Priority Queue: Allowing tracking and removal of the
* smallest item in a priority queue. */
public interface MinPQ<Item> {
   /** Adds the item to the priority queue. */
   public void add(Item x);
   /** Returns the smallest item in the priority queue. */
   public Item getSmallest();
   /** Removes the smallest item from the priority queue. */
   public Item removeSmallest();
   /** Returns the size of the priority queue. */
   public int size();
}
```

Useful if you want to keep track of the “smallest”, “largest”, “best” etc. seen so far.

Using a PQ

Lecture 21, CS61B, Spring 2025

Usage Example: Recording the Highest Energy Particles

Suppose we have a particle detector that records the energy of incoming particles.

Suppose we want to record the M highest energy particles in a given day.

Naive approach: Create a list of all particles detected during the day. Sort it using a particle energy comparator. Return the M particles that have highest energy.

Naive Implementation: Store and Sort

```java
public List<Particle> highestEnergyParticles(Detector det, int M) {
    ArrayList<Particle> allParticles = new ArrayList<>();

    for (Timer timer = new Timer(); timer.hours() < 24; ) {
        allParticles.add(det.getNextParticle());
    }

    Comparator<String> cmptr = new EnergyComparator();
    Collections.sort(allParticles, cmptr, Collections.reverseOrder());

    return allParticles.sublist(0, M);
}
```

Potentially uses a huge amount of memory $\Theta(N)$, where N is number of particles.

Naive Implementation: Store and Sort

```
public List<Particle> highestEnergyParticles(Detector det, int M) {
    ArrayList<Particle> allParticles = new ArrayList<>();

    for (Timer timer = new Timer(); timer.hours() < 24; ) {
        allParticles.add(det.getNextParticle());
    }

    Comparator<String> cmptr = new EnergyComparator();
    Collections.sort(allParticles, cmptr, Collections.reverseOrder());

    return allParticles.sublist(0, M);
}
```

Potentially uses a huge amount of memory Θ(N), where N is number of particles.

- Goal: Do this in Θ(M) memory using a MinPQ.

```
MinPQ<Particle> highEnergyParticles = new HeapMinPQ<>(cmptr);
```

Try to Solve Using a MinPQ

```
public List<Particle> highestEnergyParticles(Detector det, int M) {
    Comparator<Particle> cmptr = new EnergyComparator();
    MinPQ<Particle> highEnergyParticles = new HeapMinPQ<>(cmptr);
    for (Timer timer = new Timer(); timer.hours() < 24; ) {
        // Do something with det.getNextParticle(); ??
        ...
    }
```

Potentially uses a huge amount of memory Θ(N), where N is number of particles.

- Goal: Do this in Θ(M) memory using a MinPQ.

```
MinPQ<Particle> highEnergyParticles = new HeapMinPQ<>(cmptr);
```

Better Implementation: Track the M Best

```java
public List<Particle> highestEnergyParticles(Detector det, int M) {
    Comparator<Particle> cmptr = new EnergyComparator();
    MinPQ<Particle> highEnergyParticles = new HeapMinPQ<>(cmptr);
    for (Timer timer = new Timer(); timer.hours() < 24; ) {
        highEnergyParticles.add(det.getNextParticle());
        if (highEnergyParticles.size() > M)
        { highEnergyParticles.removeSmallest(); }
    }
    ArrayList<String> returnList = new ArrayList<String>();
    while (highEnergyParticles.size() > 0) {
        returnList.add(highEnergyParticles.removeSmallest());
    }
    return returnList;
}
```

Can track top M transactions using only M memory. API for MinPQ also makes code very simple (don't need to make explicit comparisons).

Some Bad Implementations

Lecture 21, CS61B, Spring 2025

How Would We Implement a MinPQ?

Some possibilities:

- Ordered Array
- Bushy BST: Maintaining bushiness is annoying. **Handling duplicate priorities is awkward.**
- HashTable: No good! Items go into random places.

	Ordered Array	Bushy BST	Hash Table	Heap
add	Θ(N)	Θ(log N)	Θ(1)	
getSmallest	Θ(1)	Θ(log N)	Θ(N)	
removeSmallest	Θ(N)	Θ(log N)	Θ(N)	
Caveats		Dups tough		

Worst Case Θ(·) Runtimes

Heap Definitions

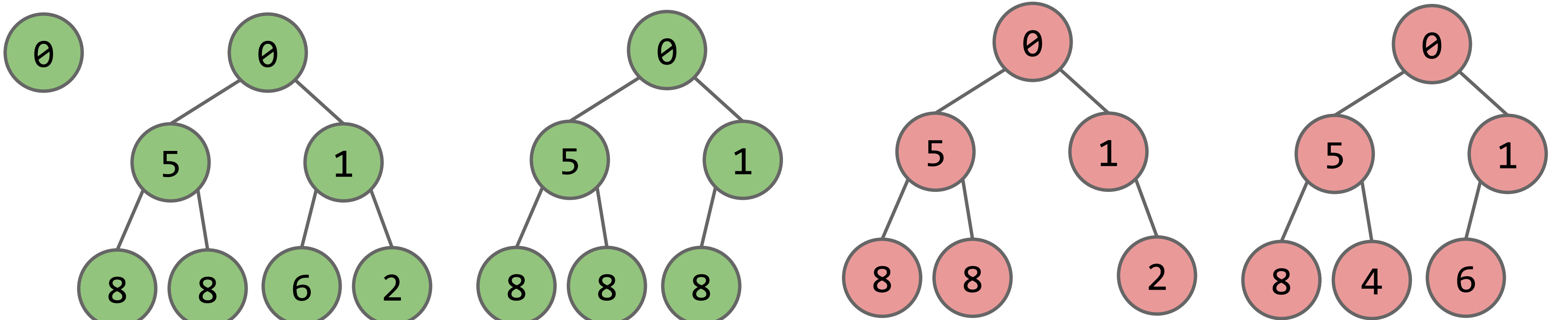
Lecture 21, CS61B, Spring 2025

Introducing the Heap

BSTs would work, but need to be kept bushy and duplicates are awkward.

Binary min-heap: Binary tree that is ***complete*** and obeys ***min-heap property***.

- Min-heap: Every node is less than or equal to both of its children.
- Complete: Every level is full, except the bottom level may be partially empty. All nodes in the bottom level are as far left as possible.



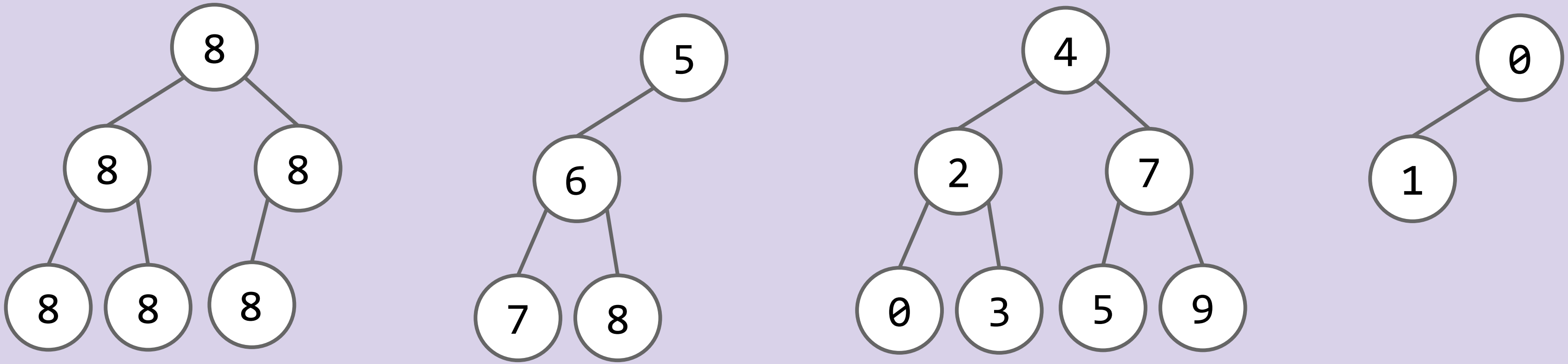
Incomplete

Lacks min-heap property

Heap Comprehension Test: www.yellkey.com/kid

How many of these are min heaps?

A. 0
B. 1
C. 2
D. 3
E. 4

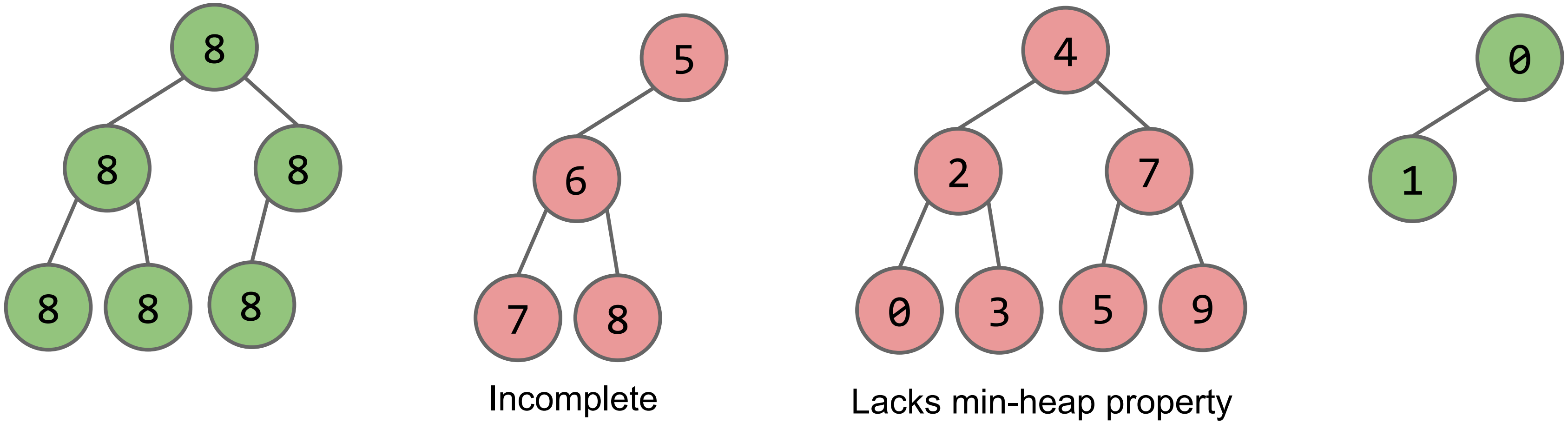


Heap Comprehension Test

How many of these are min heaps?

A. 0
B. 1
C. 2
D. 3
E. 4

Completeness: **Every level is full, except the bottom level may be partially empty**. All nodes in the bottom level are as far left as possible.

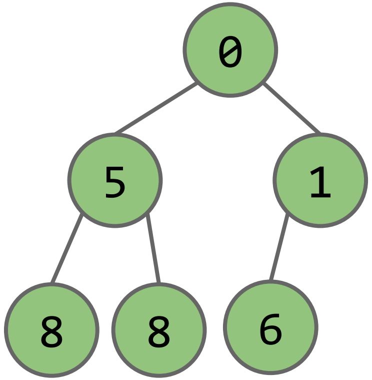


What Good Are Heaps?

Heaps lend themselves very naturally to implementation of a priority queue.

Hopefully easy question:

- How would you support `getSmallest()`?



Heap Add

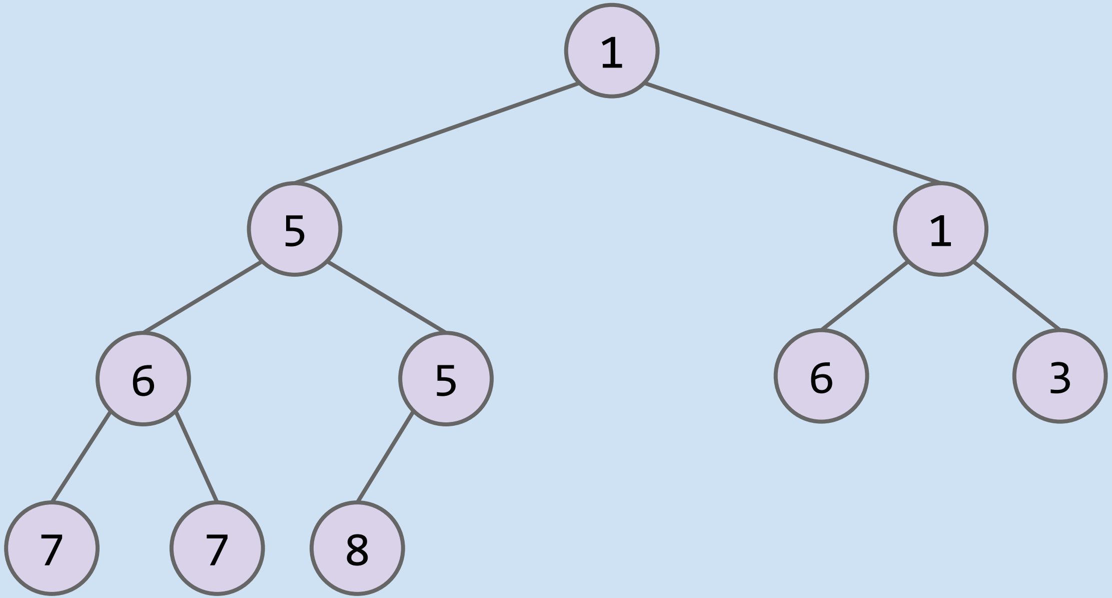
Lecture 21, CS61B, Spring 2025

How Do We Add to a Heap?

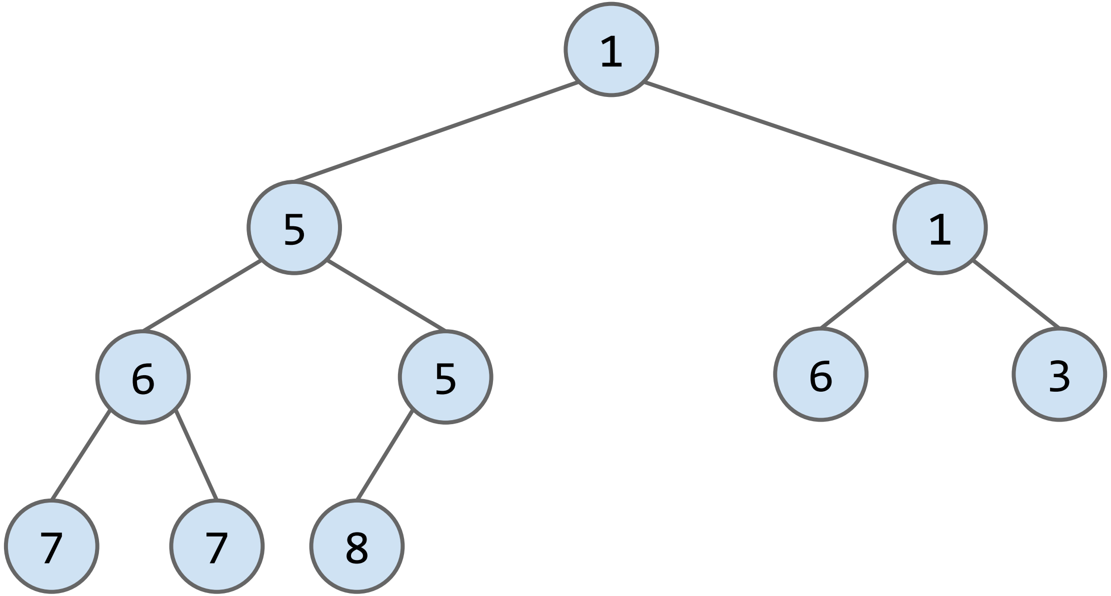
Challenge: Come up with an algorithm for `add(x)`.

- How would we insert 3?

Runtime must be logarithmic.

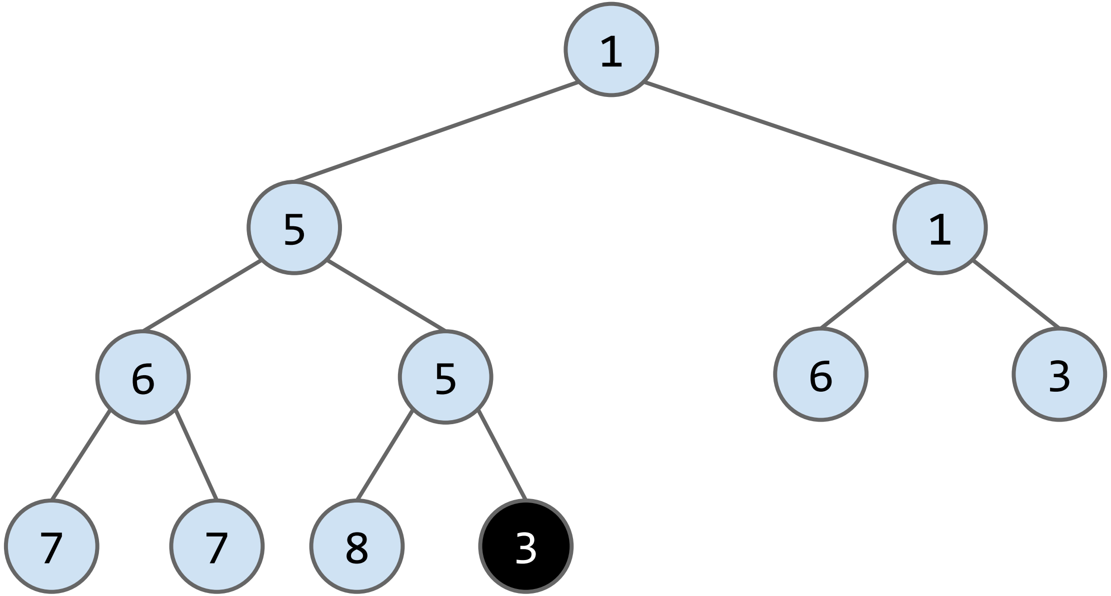


Heap Add Demo



Insert 3?

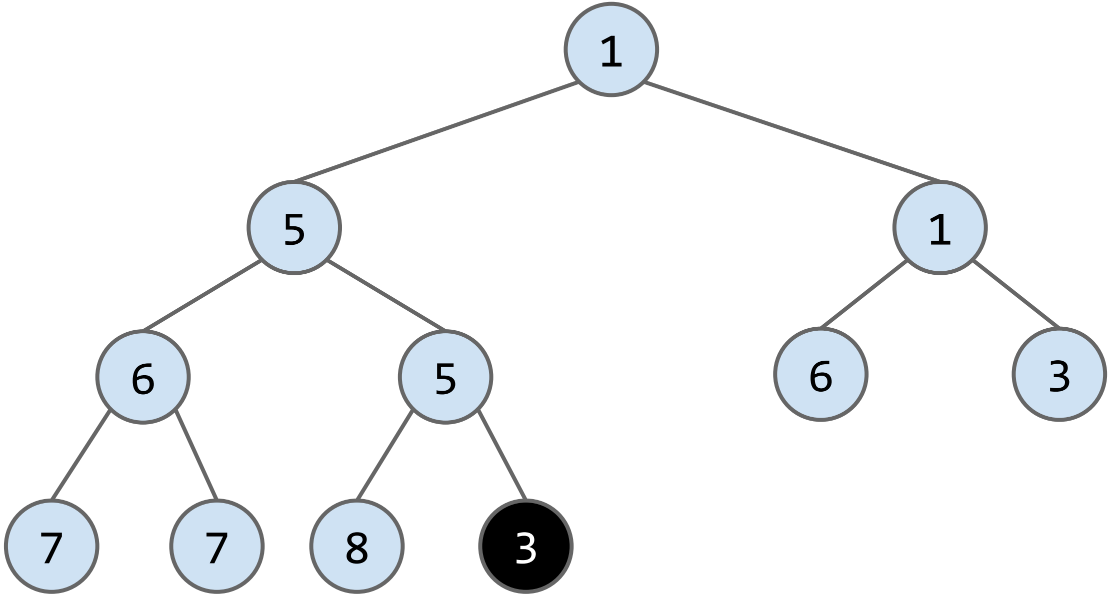
Heap Add Demo



Insert 3.

- Add to end of heap temporarily.

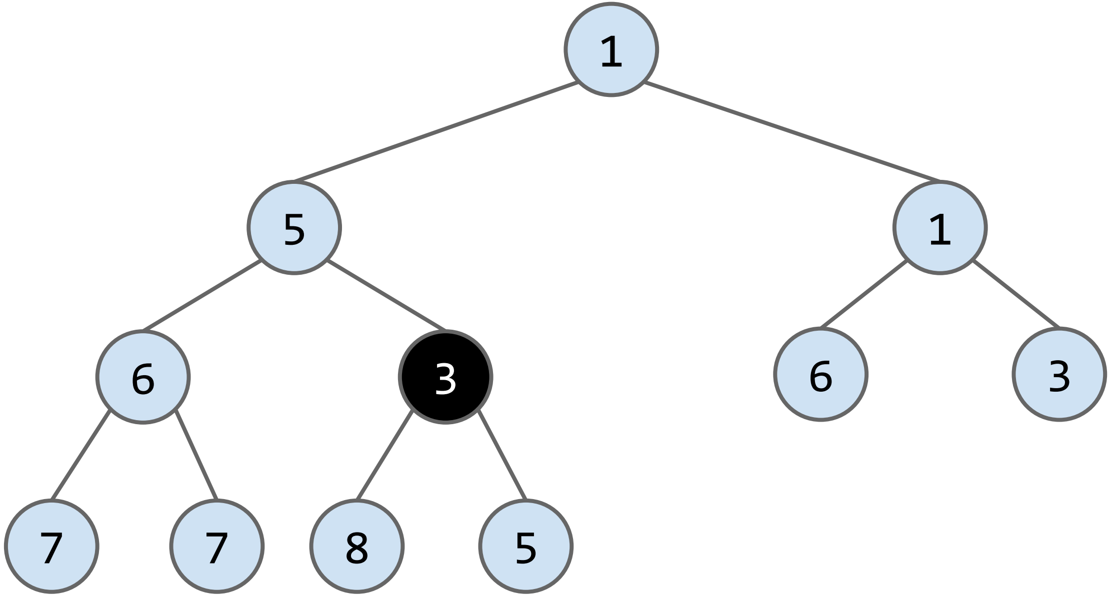
Heap Add Demo



Insert 3.

- Add to end of heap temporarily.
- Swim up the hierarchy to your rightful place...

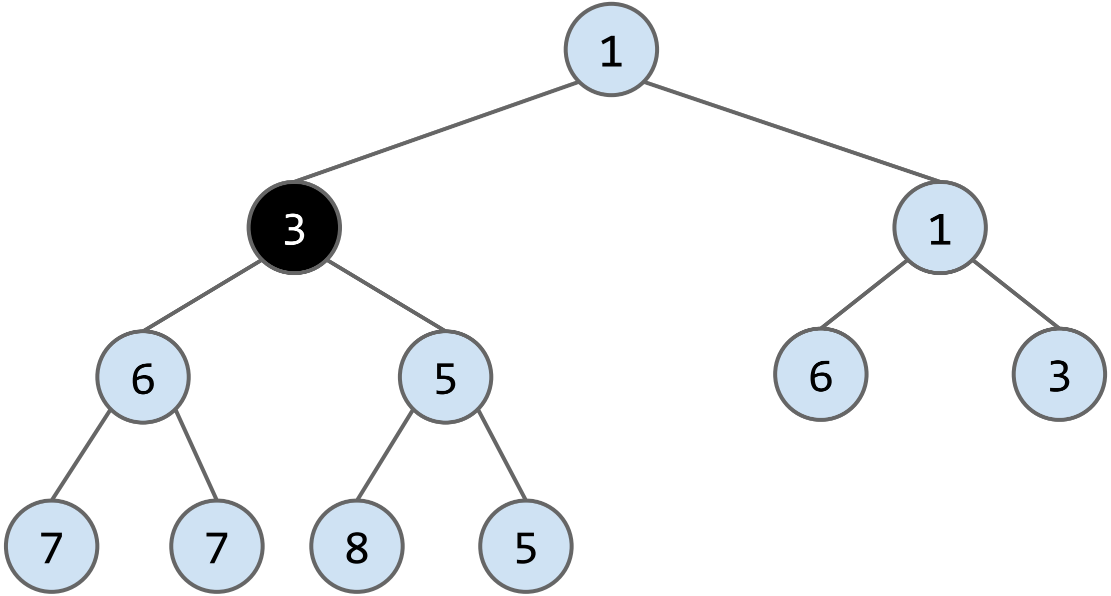
Heap Add Demo



Insert 3.

- Add to end of heap temporarily.
- Swim up the hierarchy to your rightful place...

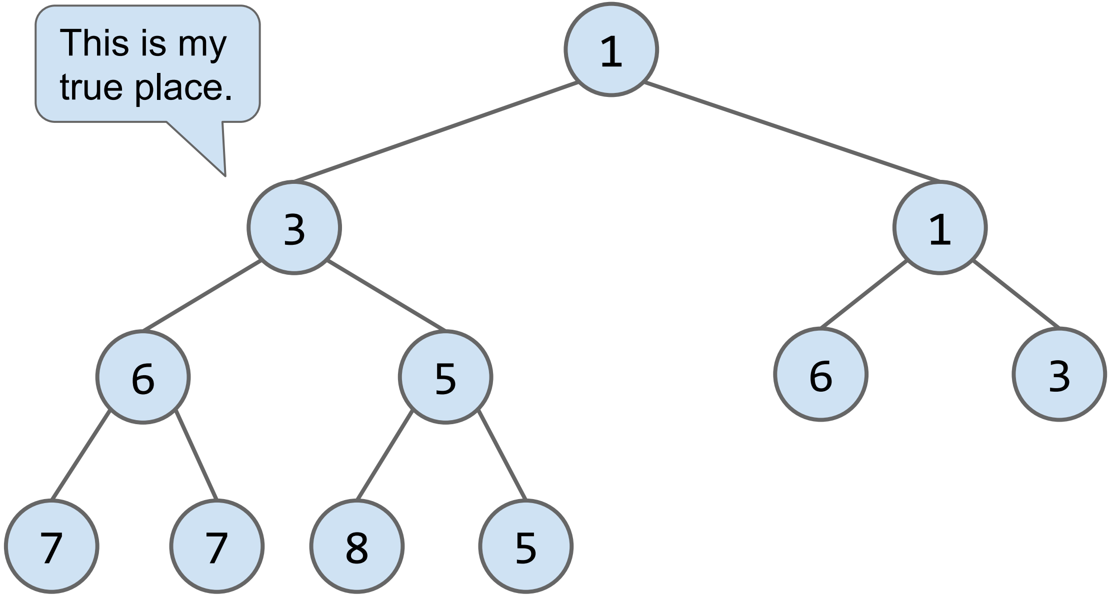
Heap Add Demo



Insert 3.

- Add to end of heap temporarily.
- Swim up the hierarchy to your rightful place...

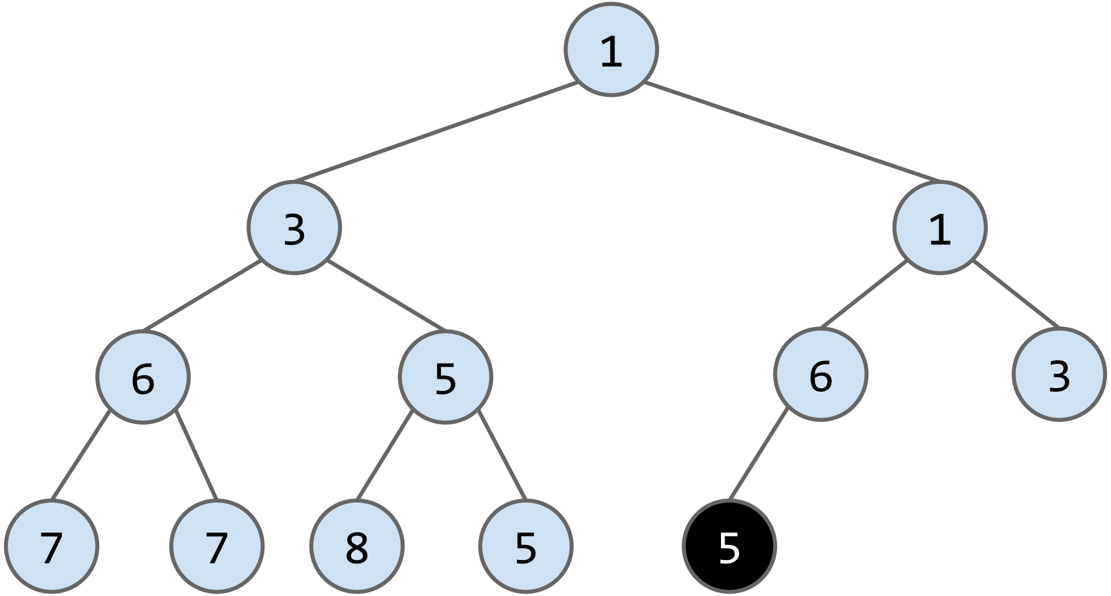
Heap Add Demo



Insert 3.

- Add to end of heap temporarily.
- Swim up the hierarchy to your rightful place.

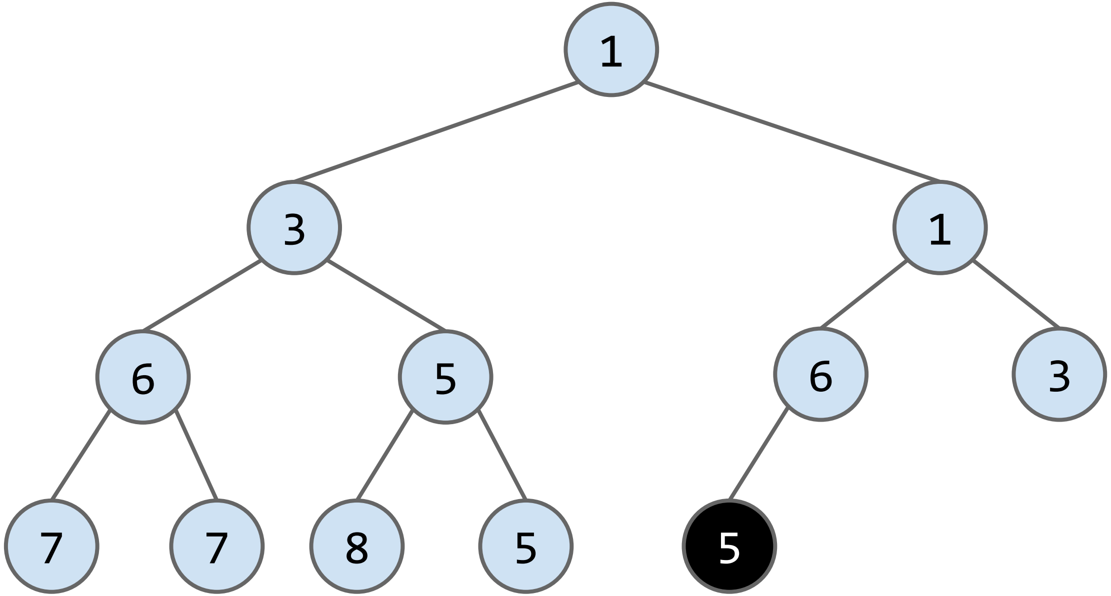
Heap Add Demo



Insert 5.

- Add to end of heap temporarily.

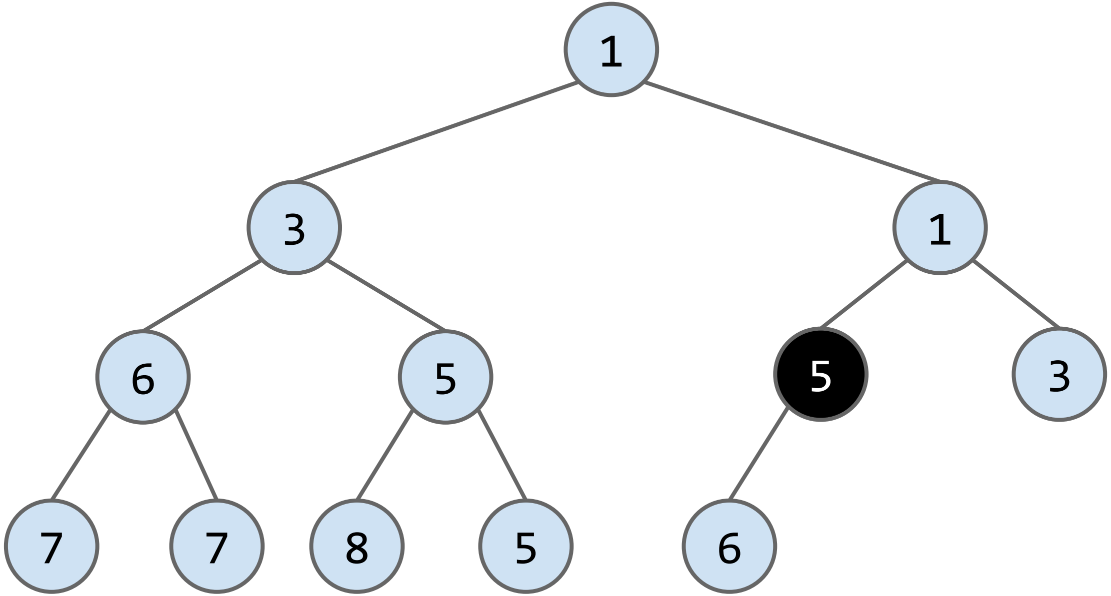
Heap Add Demo



Insert 5.

- Add to end of heap temporarily.
- Swim up the hierarchy to your rightful place...

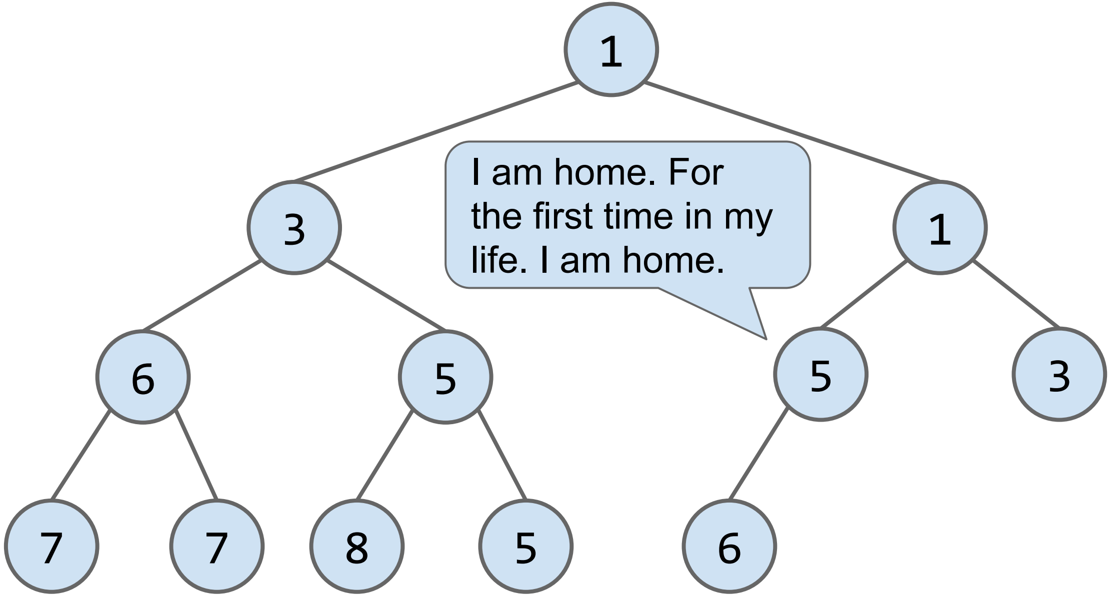
Heap Add Demo



Insert 5.

- Add to end of heap temporarily.
- Swim up the hierarchy to your rightful place...

Heap Add Demo



Insert 5.

- Add to end of heap temporarily.
- Swim up the hierarchy to your rightful place.

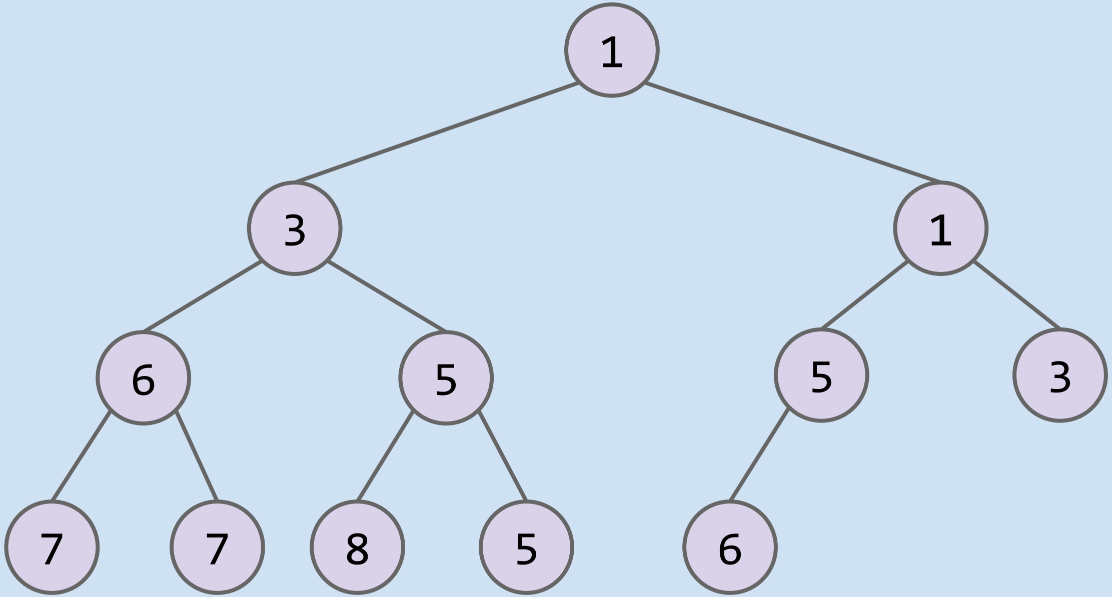
Heap Delete

Lecture 21, CS61B, Spring 2025

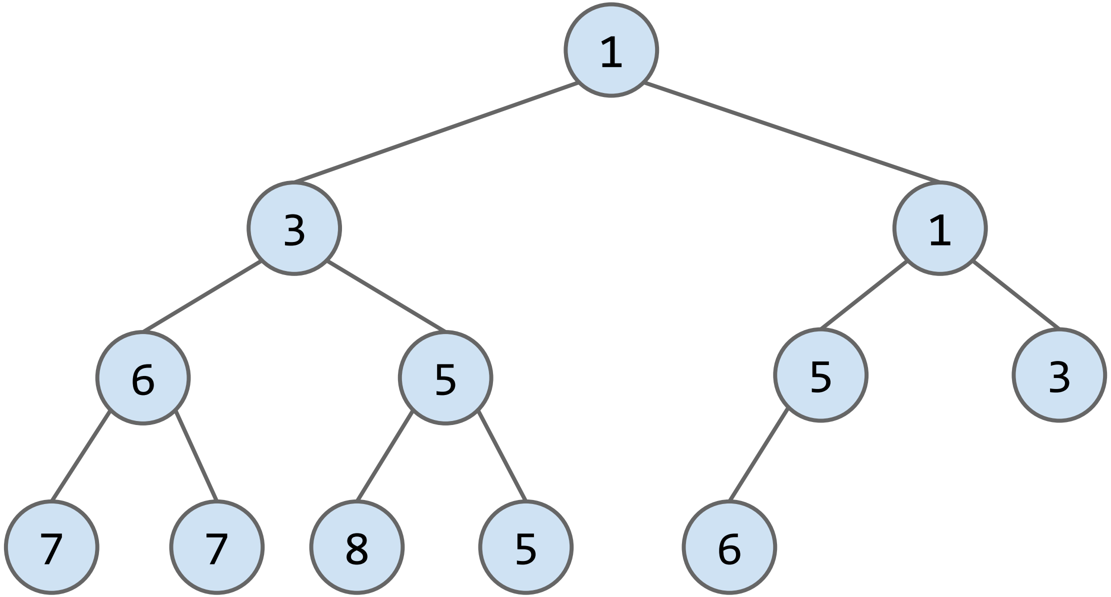
How Do We Remove from a Heap?

Challenge: Come up with an algorithm for `removeSmallest()`.

Runtime must be logarithmic.

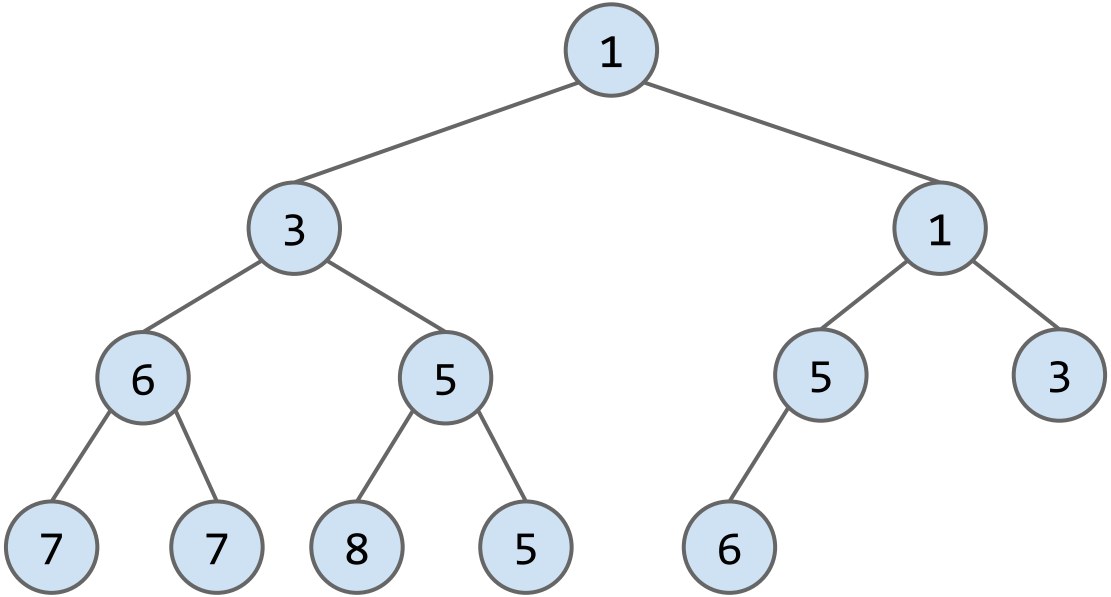


Heap Delete Demo



Delete min.

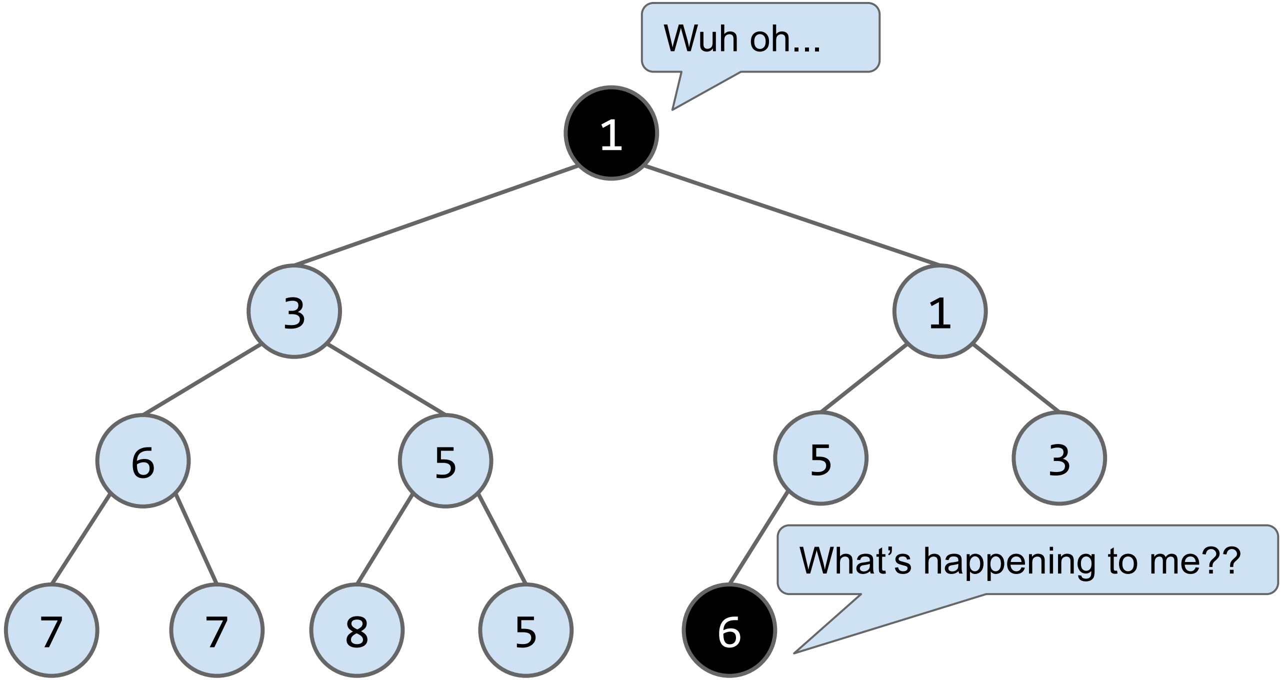
Heap Delete Demo



Delete min.

- Swap the last item in the heap into the root.

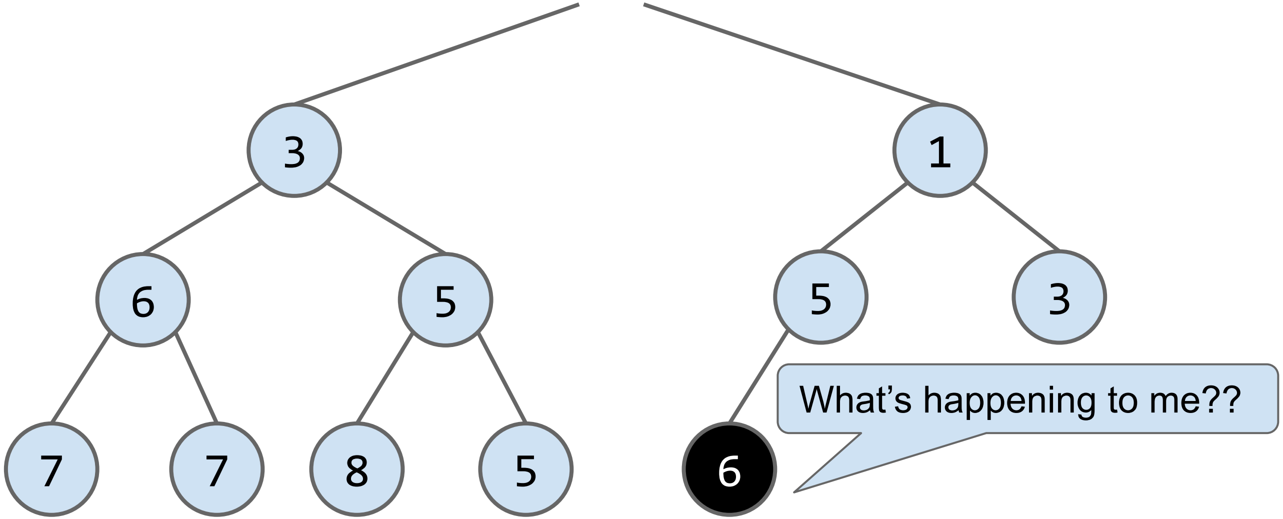
Heap Delete Demo



Delete min.

- Swap the last item in the heap into the root.

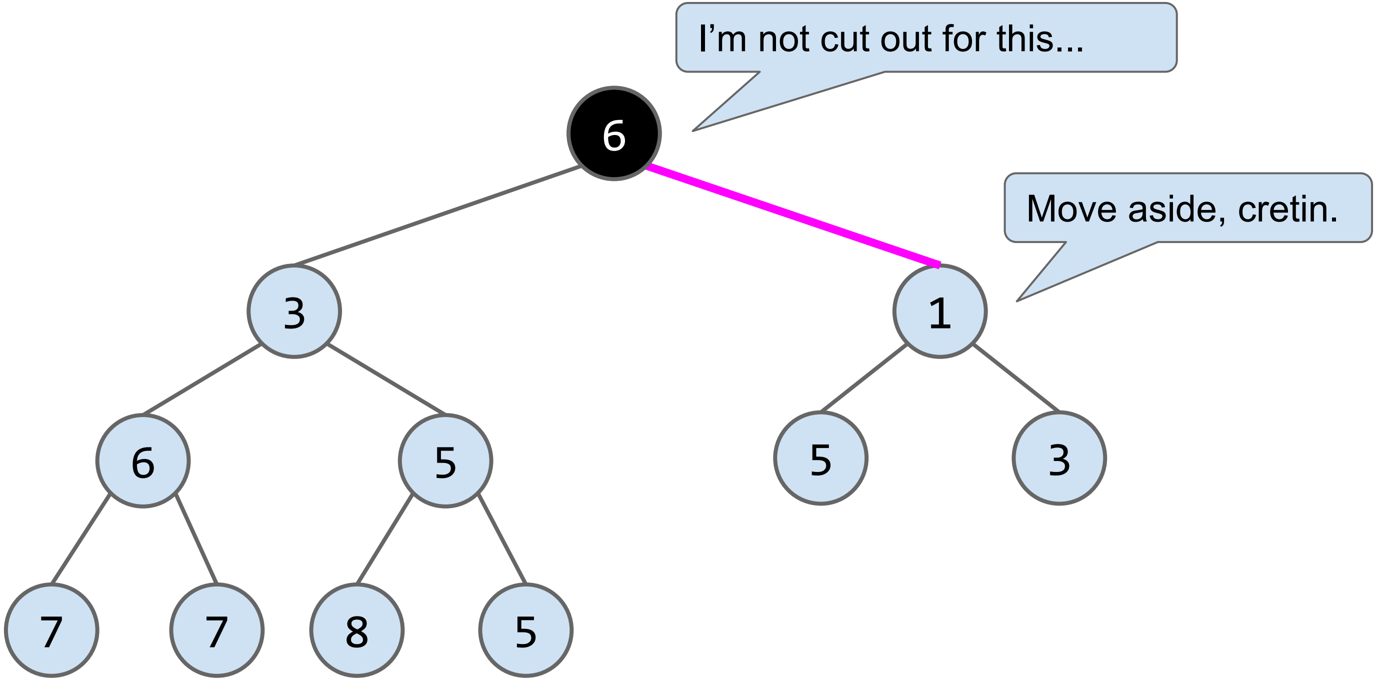
Heap Delete Demo



Delete min.

- Swap the last item in the heap into the root.

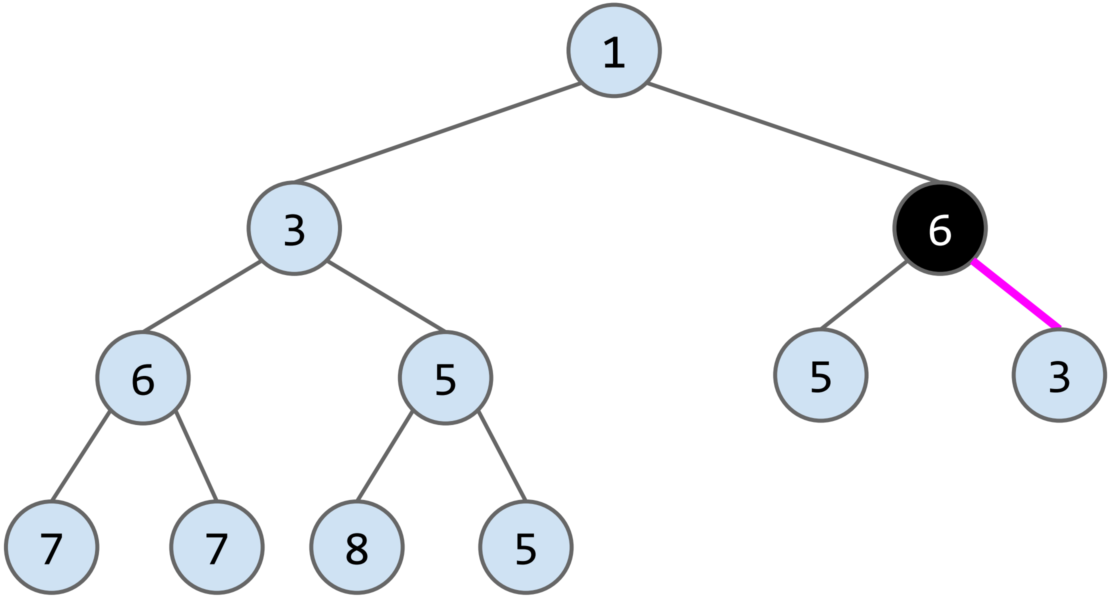
Heap Delete Demo



Delete min.

- Swap the last item in the heap into the root.
- Then sink your way down the hierarchy, yielding to most qualified folks...

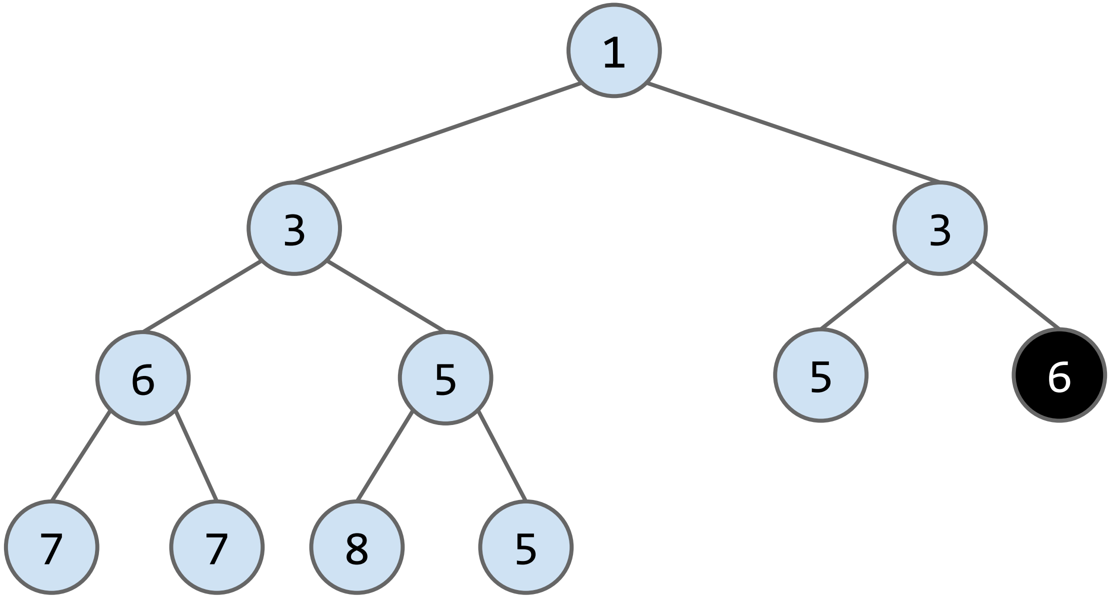
Heap Delete Demo



Delete min.

- Swap the last item in the heap into the root.
- Then sink your way down the hierarchy, yielding to most qualified folks...

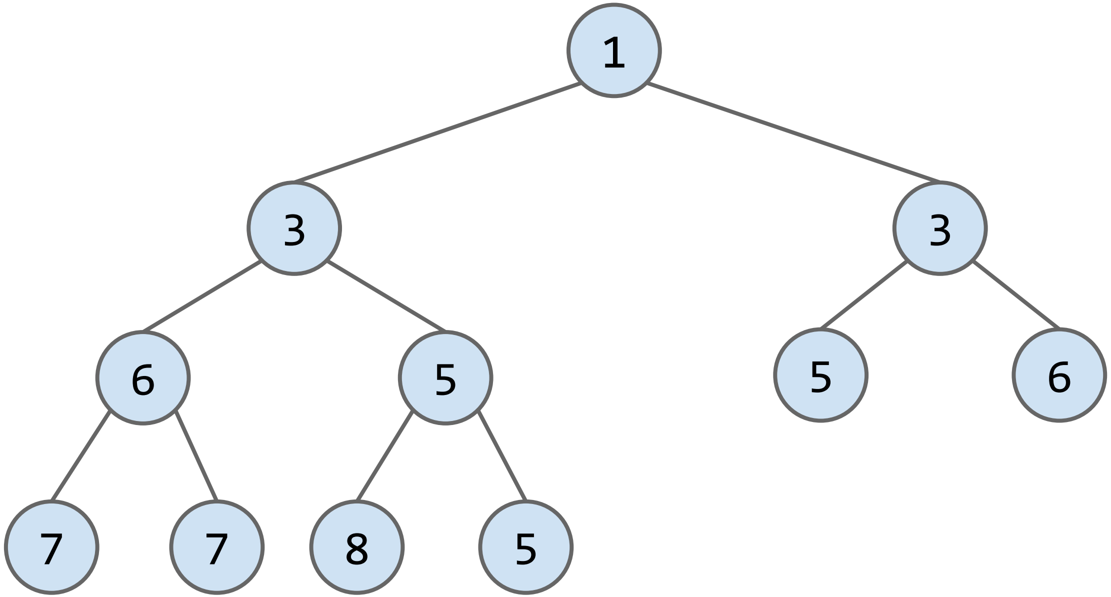
Heap Delete Demo



Delete min.

- Swap the last item in the heap into the root.
- Then sink your way down the hierarchy, yielding to most qualified folks...

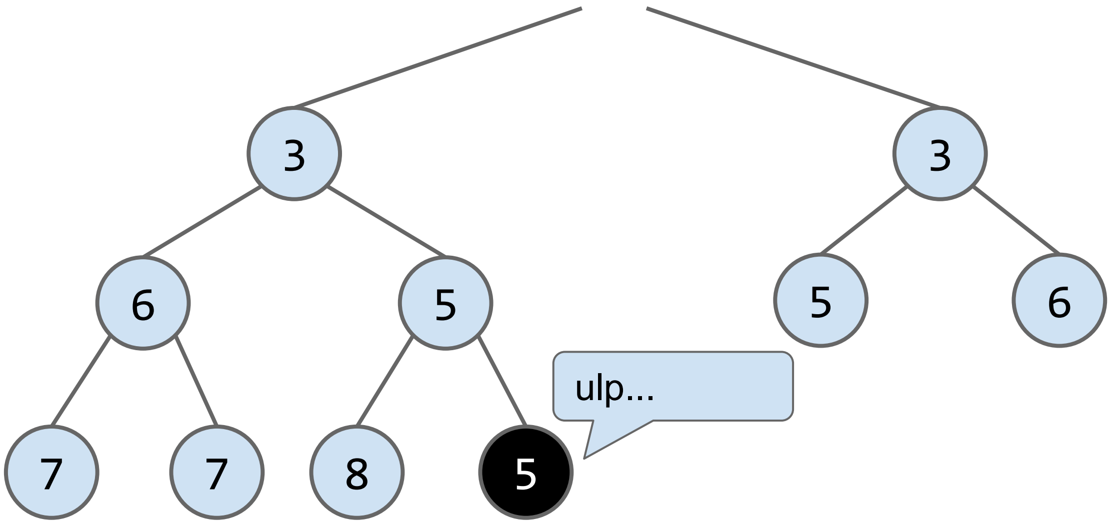
Heap Delete Demo



Delete min.

- Swap the last item in the heap into the root.
- Then sink your way down the hierarchy, yielding to most qualified folks.

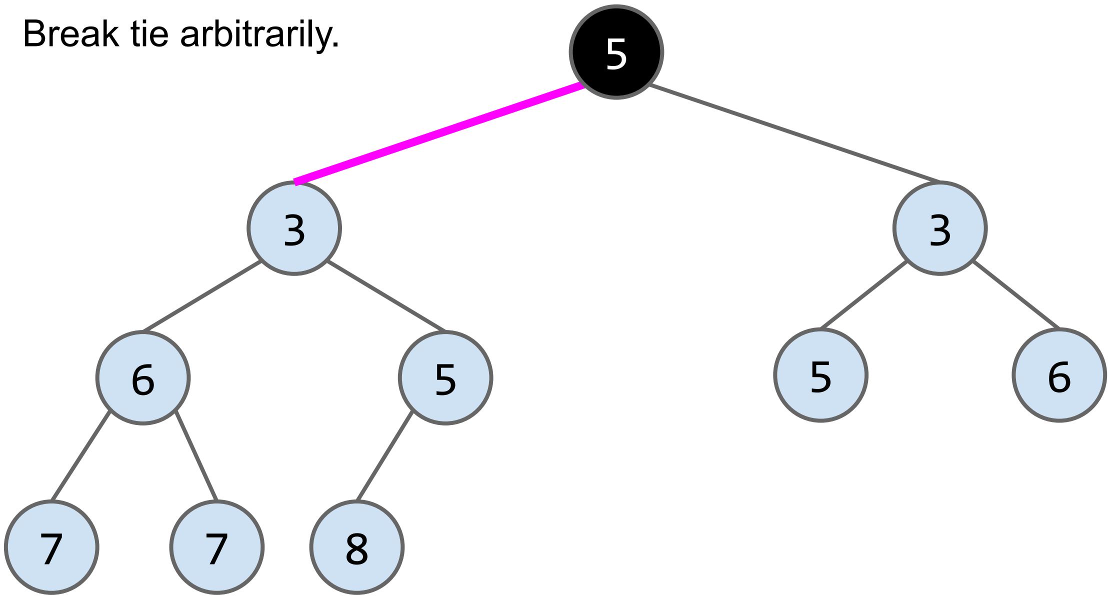
Heap Delete Demo



Delete min.

- Swap the last item in the heap into the root.

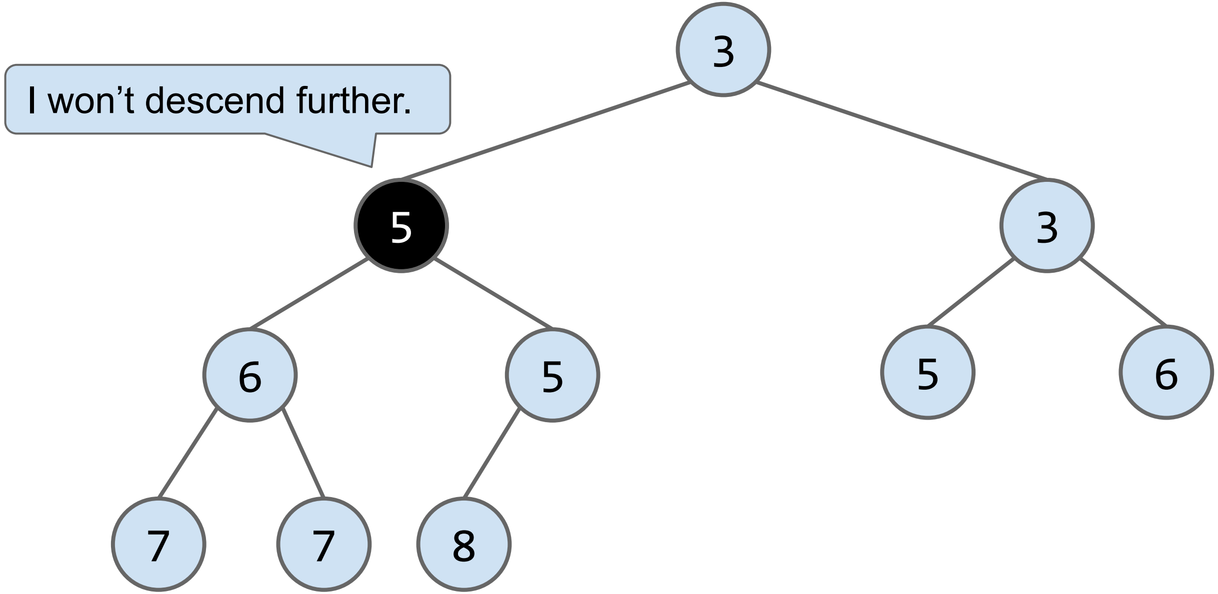
Heap Delete Demo



Delete min.

- Swap the last item in the heap into the root.
- Then sink your way down the hierarchy, yielding to most qualified folks...

Heap Delete Demo



Delete min.

- Swap the last item in the heap into the root.
- Then sink your way down the hierarchy, yielding to most qualified folks...

Recursive Representation (1)

Lecture 21, CS61B, Spring 2025

Heap Operations Summary

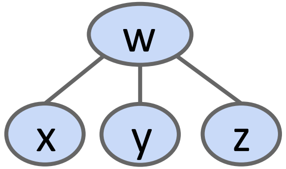
Given a heap, how do we implement PQ operations?

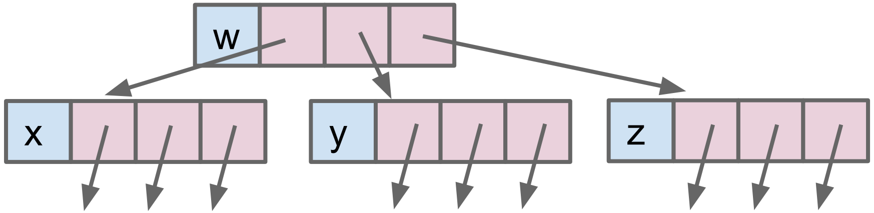
- `getSmallest()` - return the item in the root node.
- `add(x)` - place the new employee in the last position, and promote as high as possible.
- `removeSmallest()` - eliminate the president (of the company), promote the rightmost person x in the company to president. Then demote x repeatedly, always taking the 'better' successor.

Remaining question: How would we do all this in Java?

How do we Represent a Tree in Java?

Approach 1a, 1b and 1c: Create mapping from node to children.





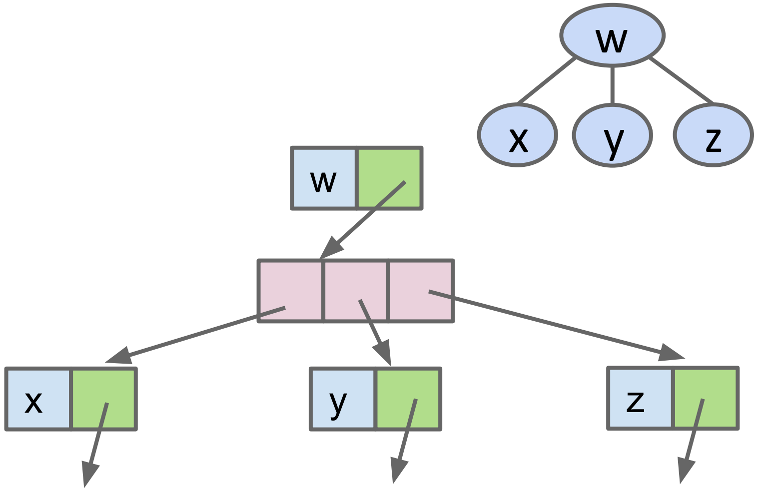
1a: Fixed-Width Nodes (BSTMap used this approach)

```
public class Tree1A<Key> {
   Key k; // e.g. 0
   Tree1A left;
   Tree1A middle;
   Tree1A right;
  ...
```

How do we Represent a Tree in Java?

Approach 1a, 1b and 1c: Create mapping from node to children.

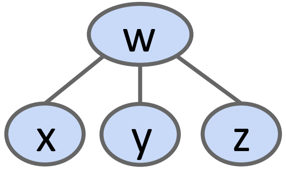
```java
public class Tree1B<Key> {
    Key k; // e.g. 0
    Tree1B[] children;
    ...
```



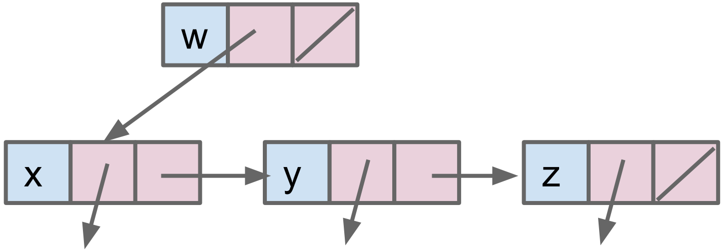
1b: Variable-Width Nodes

How do we Represent a Tree in Java?

Approach 1a, 1b and 1c: Create mapping from node to children.



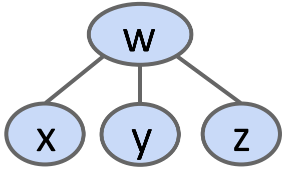

```java
public class Tree1C<Key> {
   Key k; // e.g. 0
   Tree1C favoredChild;
   Tree1C sibling;
   ...
```

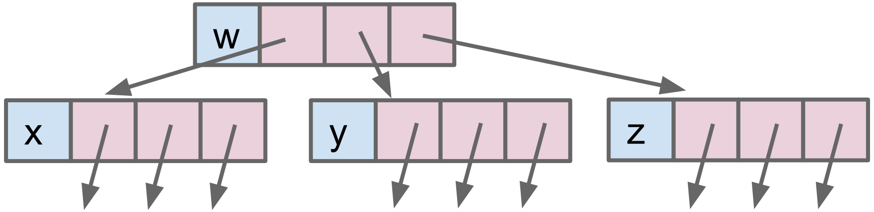


1c: Sibling Tree

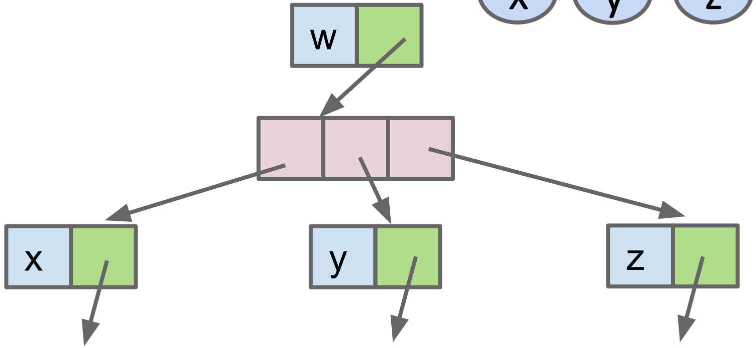
How do we Represent a Tree in Java?

Approach 1a, 1b and 1c: Create mapping from node to children.

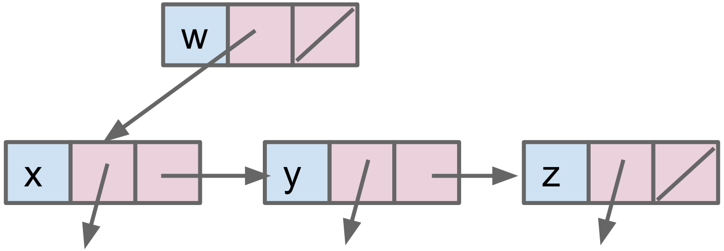




1a: Fixed-Width Nodes



1b: Variable-Width Nodes



1c: Sibling Tree

Array Representations (2, 3, 3b)

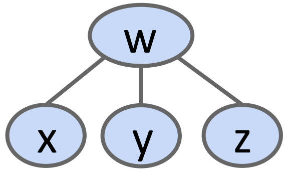
Lecture 21, CS61B, Spring 2025

How do we Represent a Tree in Java?

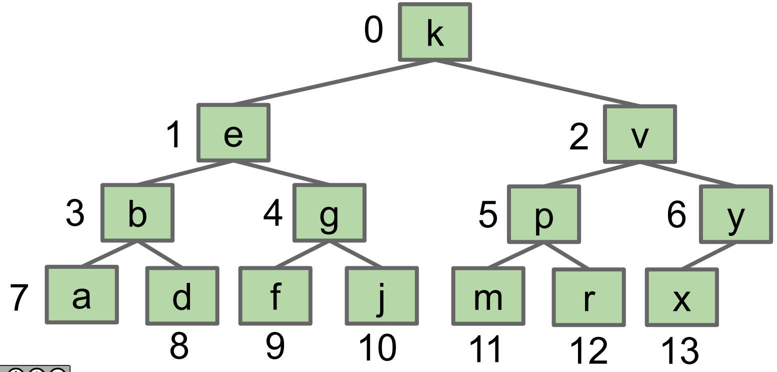
Approach 2: Store keys in an array. Store parentIDs in an array.

- Similar to what we did with disjointSets.

```
public class Tree2<Key> {
   Key[] keys;
   int[] parents;
   ...
```



	0	1	2	3
Key[] keys	w	x	y	z
int[] parents	0	0	0	0



Key[] keys

k	e	v	b	g	p	y	a	d	f	j	m	r	x
0	1	2	3	4	5	6	7	8	9	10	11	12	13

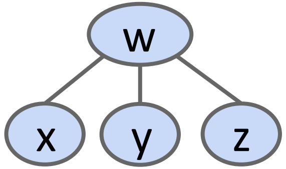
int[] parents

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

How do we Represent a Tree in Java?

Approach 3: Store keys in an array. Don't store structure anywhere.

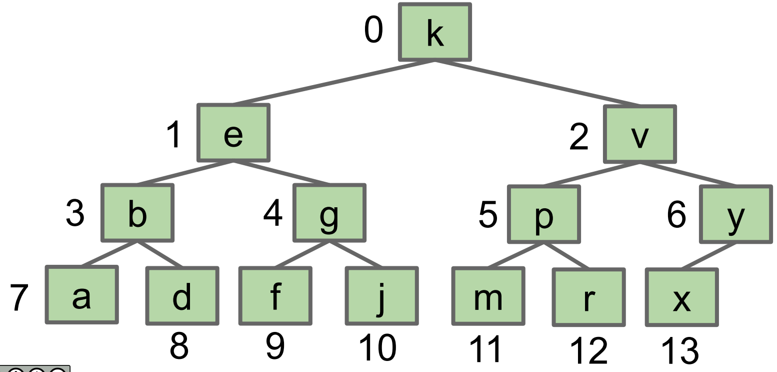
- To interpret array: Simply assume tree is complete.
- Obviously only works for "complete" trees.



Key[] keys

w	x	y	z
0	1	2	3

```java
public class Tree3<Key> {
    Key[] keys;
    ...
```



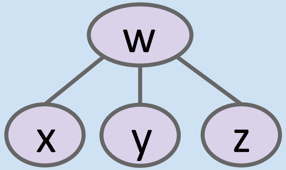
Key[] keys

k	e	v	b	g	p	y	a	d	f	j	m	r	x
0	1	2	3	4	5	6	7	8	9	10	11	12	13

A Deep Look at Approach 3

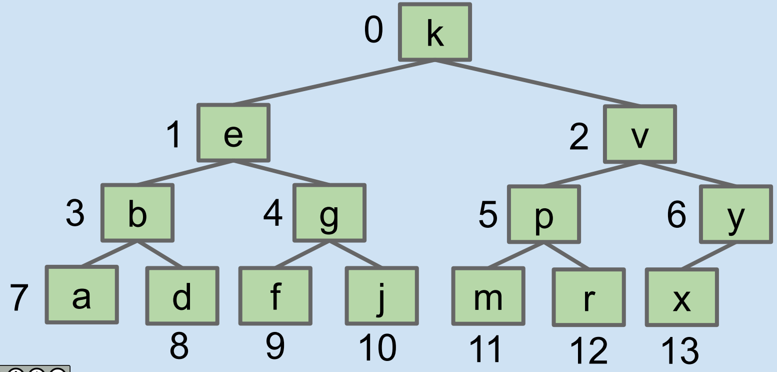
Challenge: Write the `parent(k)` method for approach 3.

```java
public void swim(int k) {
   if (keys[parent(k)] > keys[k]) {
       swap(k, parent(k));
       swim(parent(k));
   }
}
```



Key[] keys

w	x	y	z
0	1	2	3



Key[] keys

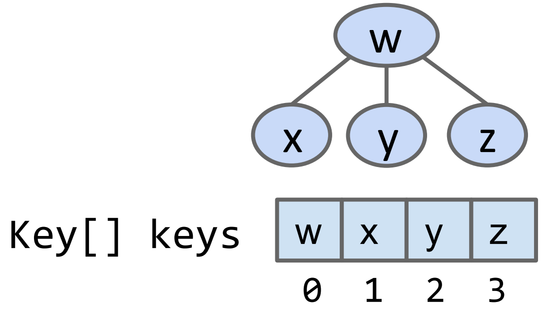
k	e	v	b	g	p	y	a	d	f	j	m	r	x
0	1	2	3	4	5	6	7	8	9	10	11	12	13

```java
public class Tree3<Key> {
   Key[] keys;
   ...
```

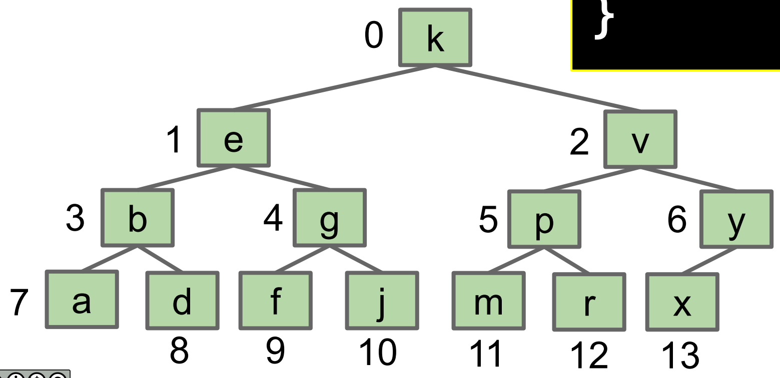
A Deep Look at Approach 3

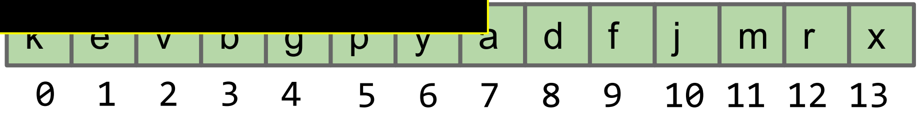
Challenge: Write the `parent(k)` method for approach 3.

```java
public void swim(int k) {
   if (keys[parent(k)] > keys[k]) {
      swap(k, parent(k));
      swim(parent(k));
   }
}
```



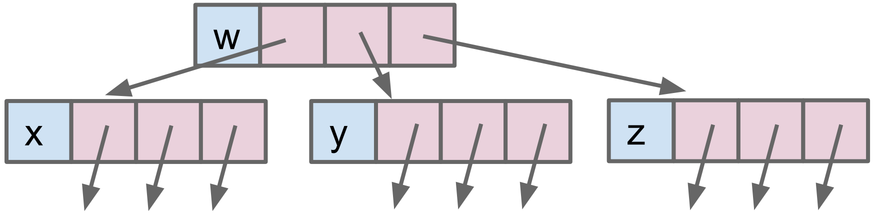

```java
public int parent(int k) {
   return (k - 1) / 2;
}
```



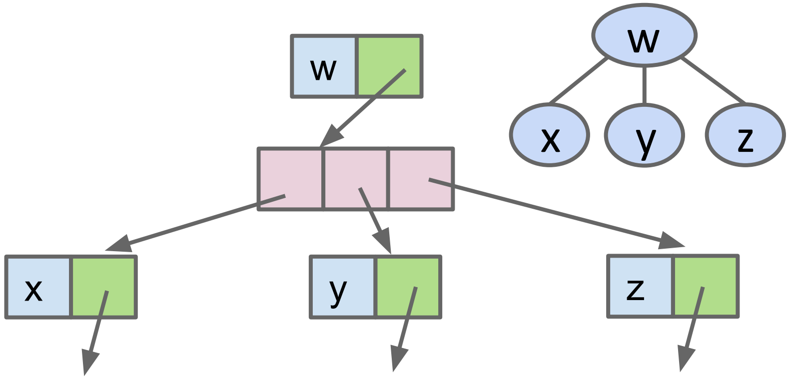



```java
public class Tree3<Key> {
   Key[] keys;
   ...
```

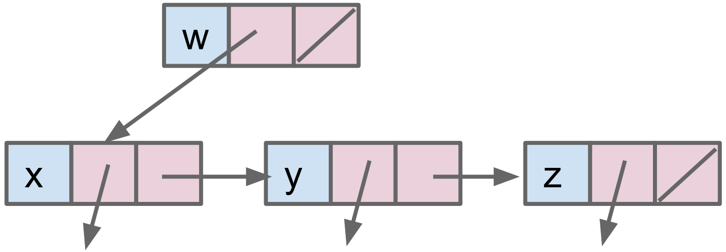
Tree Representations (Summary)



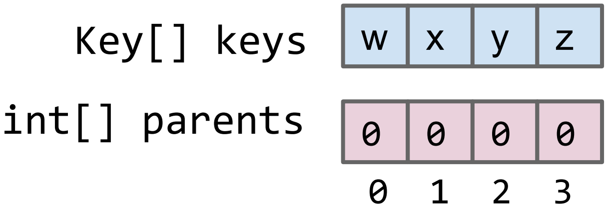
1a: Fixed Number of Links (One Per Child)



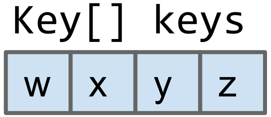
1b: Array of Child Links



1c: FirstBorn/Sibling Links



2: Array of Keys, Array of Structure

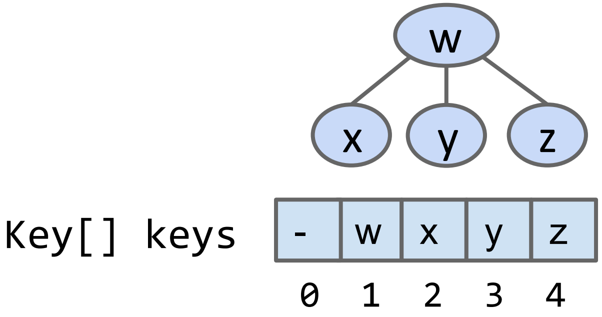


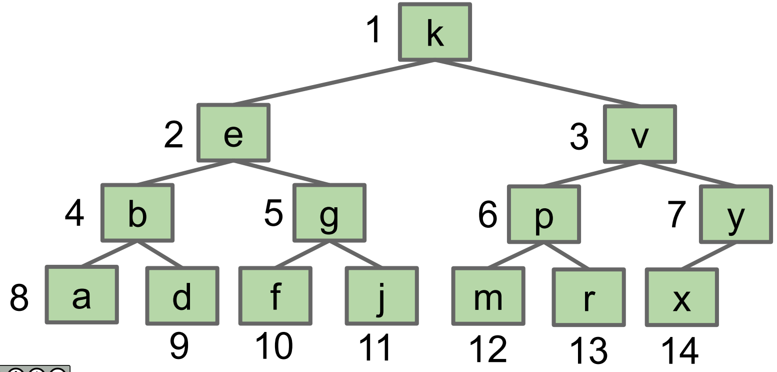
3: Array of Keys

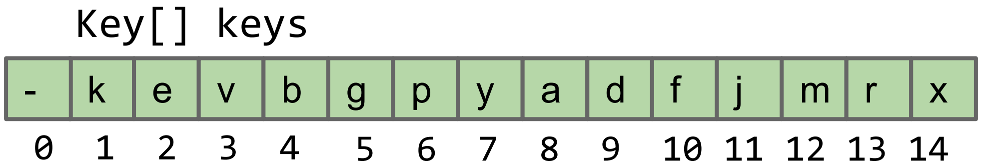
Approach 3B (book implementation): Leaving One Empty Spot

Approach 3b: Store keys in an array. Offset everything by 1 spot.

- Same as 3, but leave spot 0 empty.
- Makes computation of children/parents “nicer”.
  - `leftChild(k) = k*2`
  - `rightChild(k) = k*2 + 1`
  - `parent(k) = k/2`







Priority Queue Summary

Lecture 21, CS61B, Spring 2025

Heap Implementation of a Priority Queue

	Ordered Array	Bushy BST	Hash Table	Heap
add	Θ(N)	Θ(log N)	Θ(1)	Θ(log N)
getSmallest	Θ(1)	Θ(log N)	Θ(N)	Θ(1)
removeSmallest	Θ(N)	Θ(log N)	Θ(N)	Θ(log N)

Items with same priority hard to handle.

Notes:

- Why "priority queue"? Can think of position in tree as its "priority."
- Heap is log N time AMORTIZED (some resizes, but no big deal).
- BST can have constant getSmallest if you keep a pointer to smallest.
- Heaps handle duplicate priorities much more naturally than BSTs.
- Array based heaps take less memory (very roughly about 1/3rd the memory of representing a tree with approach 1a).

Some Implementation Questions

1. How does a PQ know how to determine which item in a PQ is larger?
    a. What could we change so that there is a default comparison?
2. What constructors are needed to allow for different orderings?

```java
/** (Min) Priority Queue: Allowing tracking and removal of the
* smallest item in a priority queue. */
public interface MinPQ<Item> {
   /** Adds the item to the priority queue. */
   public void add(Item x);
   /** Returns the smallest item in the priority queue. */
   public Item getSmallest();
   /** Removes the smallest item from the priority queue. */
   public Item removeSmallest();
   /** Returns the size of the priority queue. */
   public int size();
}
```

Data Structures Summary

Lecture 21, CS61B, Spring 2025

The Search Problem

Given a stream of data, retrieve information of interest.

- Examples:
  - Website users post to personal page. Serve content only to friends.
  - Given logs for thousands of weather stations, display weather map for specified date and time.

Search Data Structures (The particularly abstract ones)

Name	Storage Operation(s)	Primary Retrieval Operation	Retrieve By:
List	`add(key)` `insert(key, index)`	`get(index)`	index
Map	`put(key, value)`	`get(key)`	key identity
Set	`add(key)`	`containsKey(key)`	key identity
PQ	`add(key)`	`getSmallest()`	key order (a.k.a. key size)
Disjoint Sets	`connect(int1, int2)`	`isConnected(int1, int2)`	two int values

Diagram of Data Structures and ADTs (past semester version)

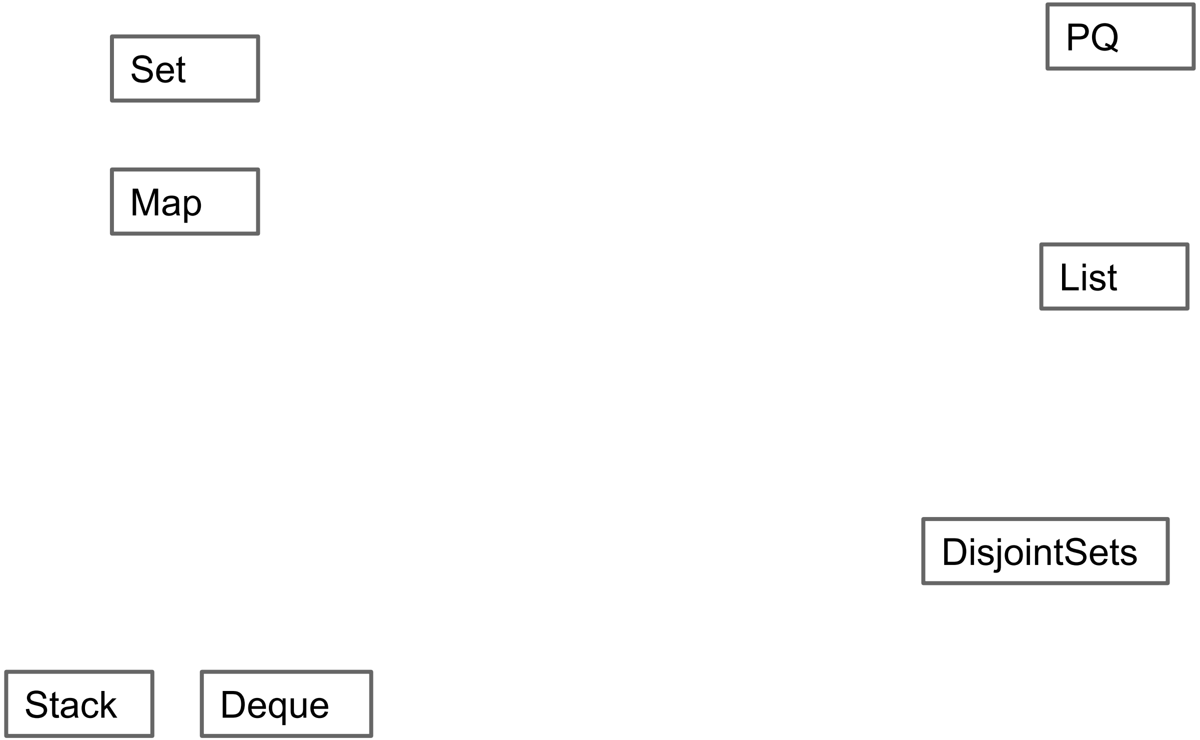


Diagram of Data Structures and ADTs (past semester version)

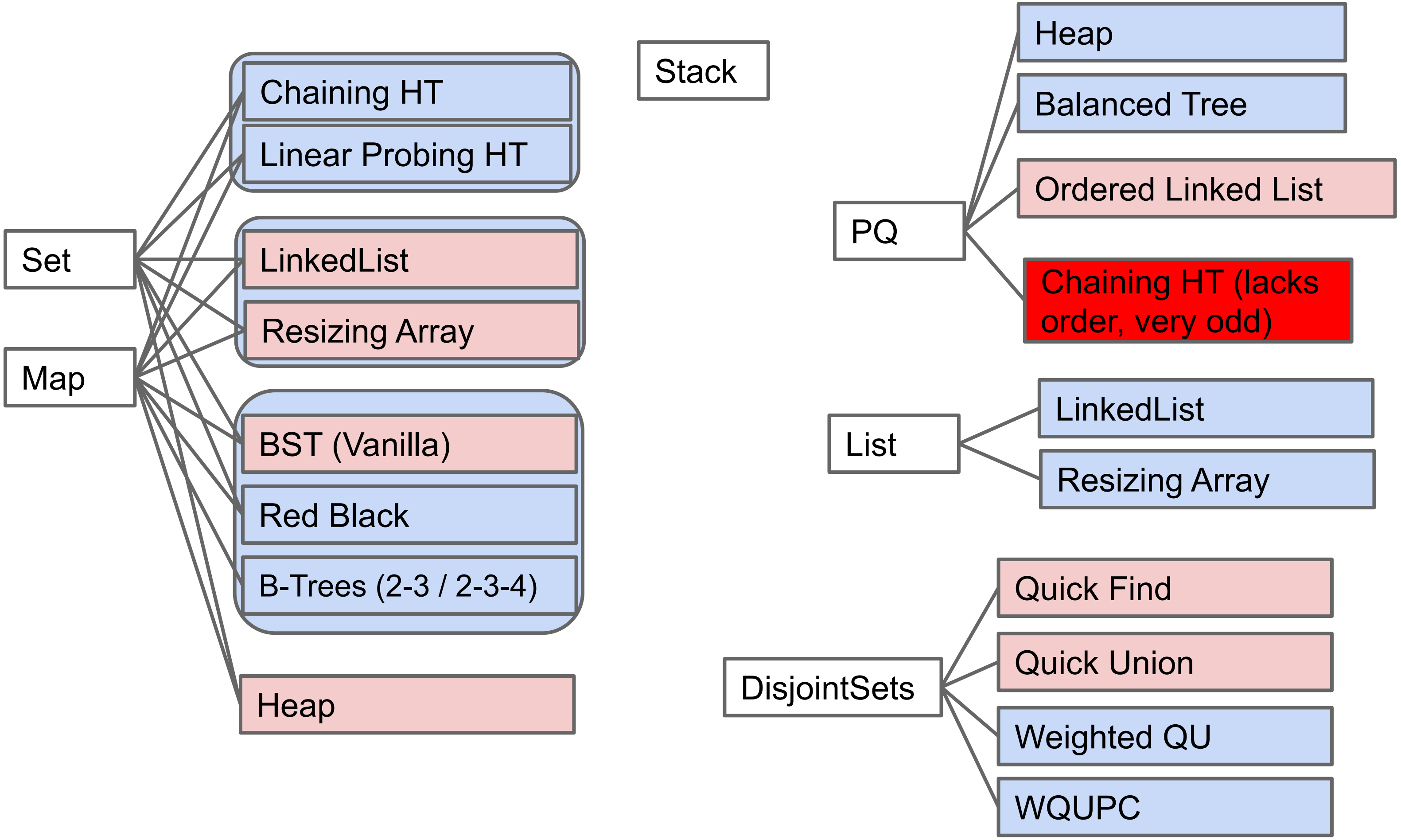
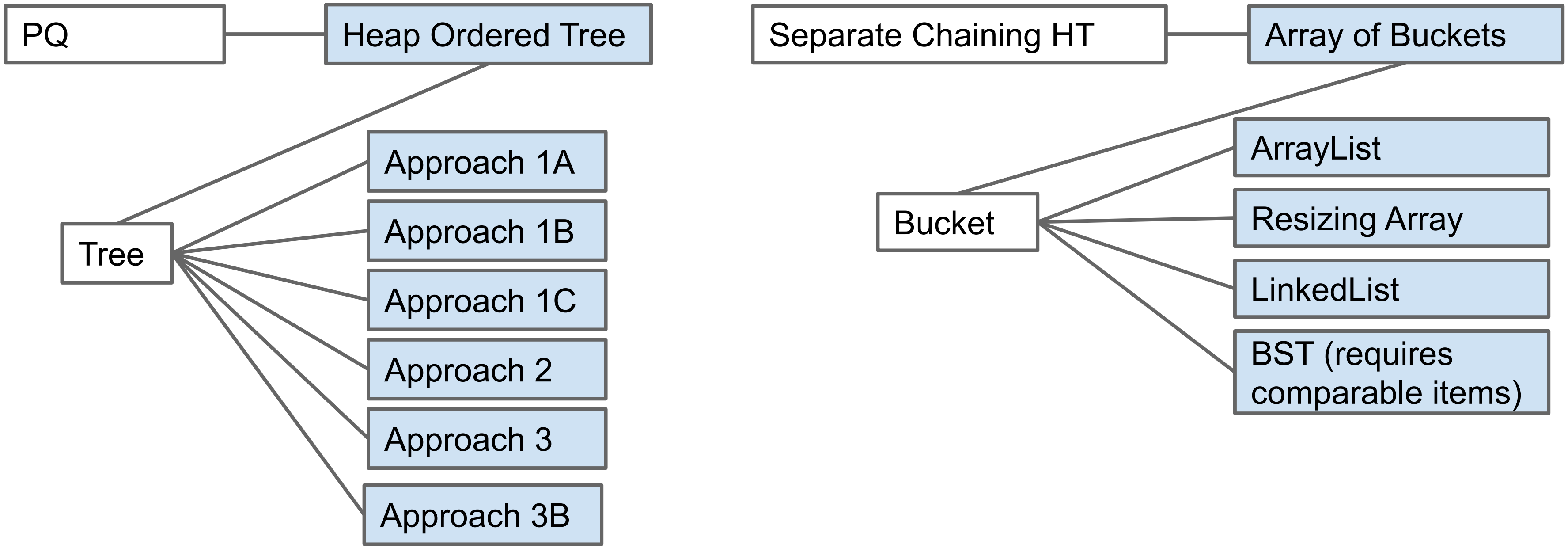
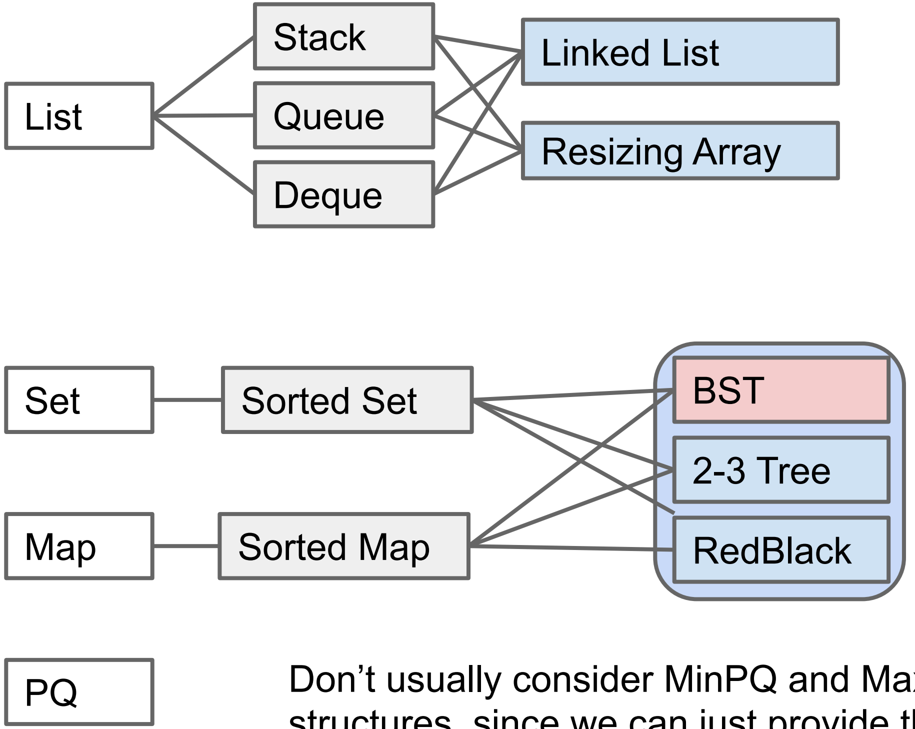


Diagram of Data Structures and ADTs

Abstraction often happens in layers!



Specialized Searching Data Structures:



In Java:
java.util.SortedSet

java.util.SortedMap

Don’t usually consider MinPQ and MaxPQ to be different data structures, since we can just provide the opposite comparator.

Data Structure: A particular way of organizing data.

- We've covered many of the most fundamental abstract data types, their common implementations, and the tradeoffs thereof.
- We'll do two more in this class:
    - Tries, graphs.

V·T·E	Data structures [hide]
Types	Collection · Container
Abstract	Associative array · Double-ended priority queue · Double-ended queue · List · Map · Multimap · Priority queue · Queue · Set (multiset) · Disjoint Sets · Stack
Arrays	Bit array · Circular buffer · Dynamic array · Hash table · Hashed array tree · Sparse array
Linked	Association list · Linked list · Skip list · Unrolled linked list · XOR linked list
Trees	B-tree · Binary search tree (AA · AVL · red-black · self-balancing · splay) · Heap (binary · binomial · Fibonacci) · R-tree (R* · R+ · Hilbert) · Trie (Hash tree)
Graphs	Binary decision diagram · Directed acyclic graph · Directed acyclic word graph