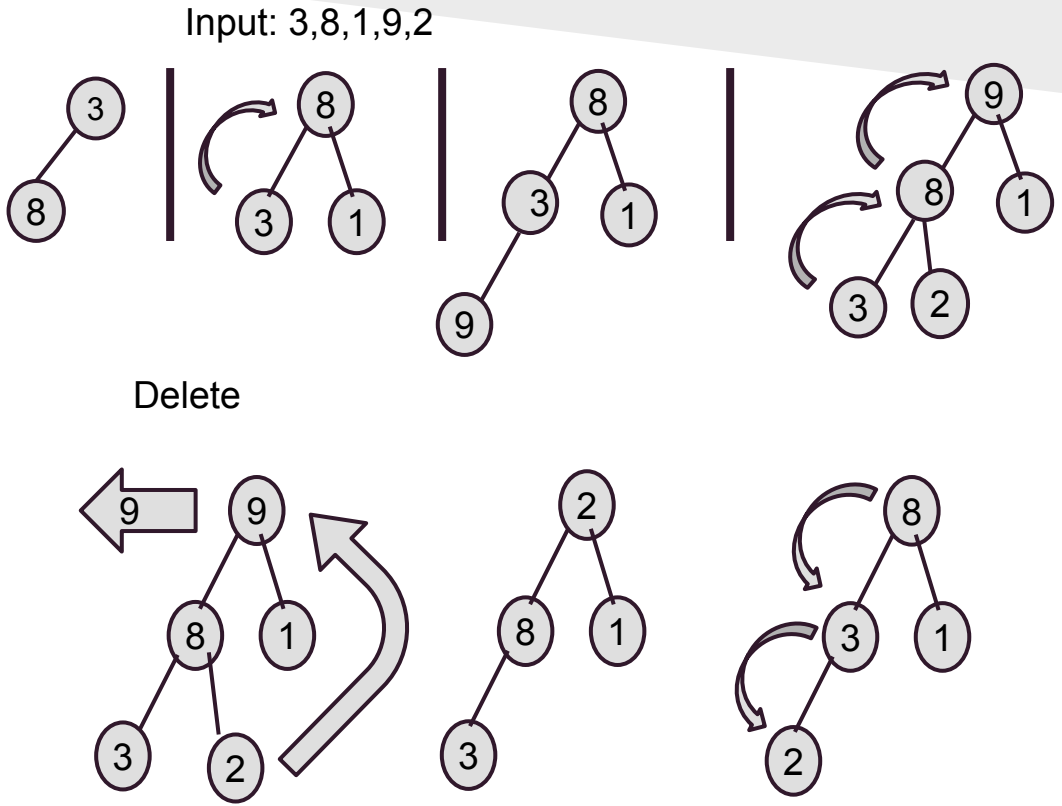
Stable-Duplicate Priority Queue

Aliza Levinger, Ashley Haiger, Rahul Sood, Rosanna Corvino

Brief Review: What's a heap?

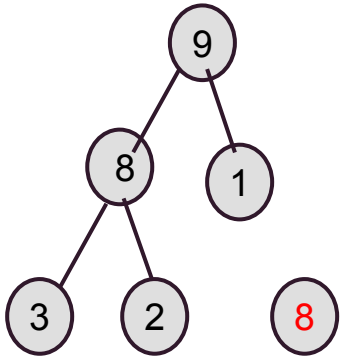
- Implement priority queue
- Partially sorted binary tree
- Breadth first



Regular Heap is UNSTABLE

Input: 3,8,1,9,2, 8

Regular implementation of a heap will not keep track of which 8 came first.



Regular Heap is UNSTABLE

What If…preserving order of duplicates is important?

1. **Boarding airplane**
2. **VIP Pass/Preferred Customer**
3. **Rental Car**
4. **Distributed Computer Processing**
5. **Bandwidth Management**

Prior Research

- Ted Herman
- Nick Paelante
- Lukas Vokrinek
- Satoshige Ukena
- Victor Arad
- Michelle Hugue

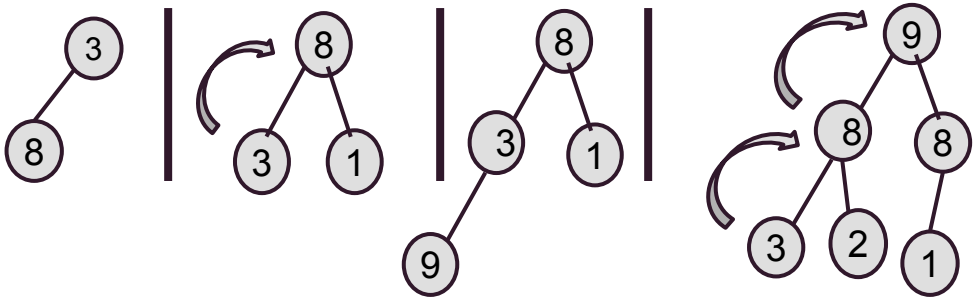
Possible Solutions

1. **Static -** implement heap as 2-d array and apply memoization using hashtable

2. **Dynamic -** implement heap as binary search tree (BST) and use linked lists to track duplicates

Static: Build a Heap

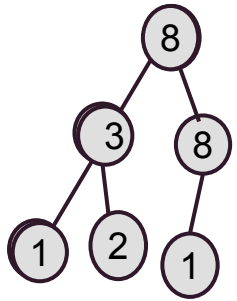
Input: 8,3,1,9,2,8



Array Index (i)	0	1	2	3	4	5
A(i)	9	8	8	3	2	1
# Seq	1	1	1	1	1	2

Hash Index	Max Seq #	Next Item To go	3 (array position of element 1)	4 (array position of duplicate 2)	5 (array position of duplicate 3)	6 (array position of duplicate 4)
1	1	1	5			
2	1	1	4			
3	1	1	3			
4						
5						
6						
7						
8	2	1	1	2		
9	1	1	0			

Static: Delete Max

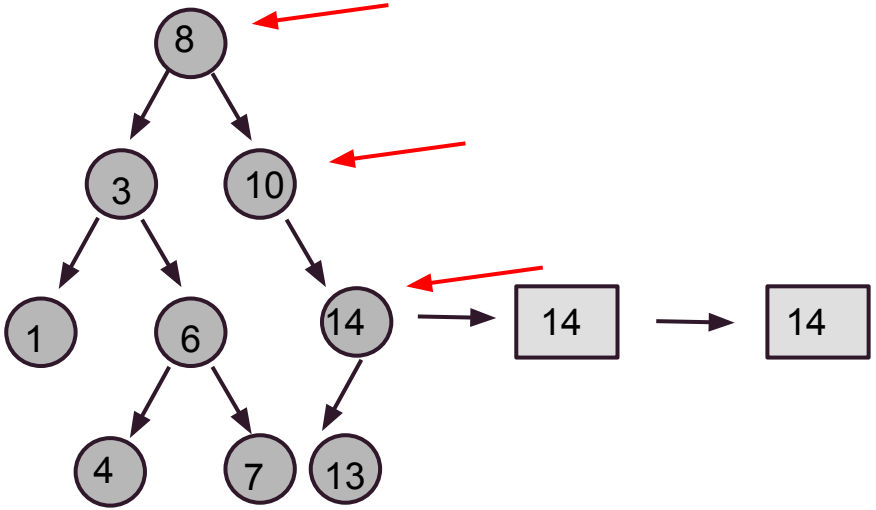


Array Index (i)	0	1	2	3	4	5
A(i)	9	8	8	3	2	1
# Seq	1	1	1	1	1	2

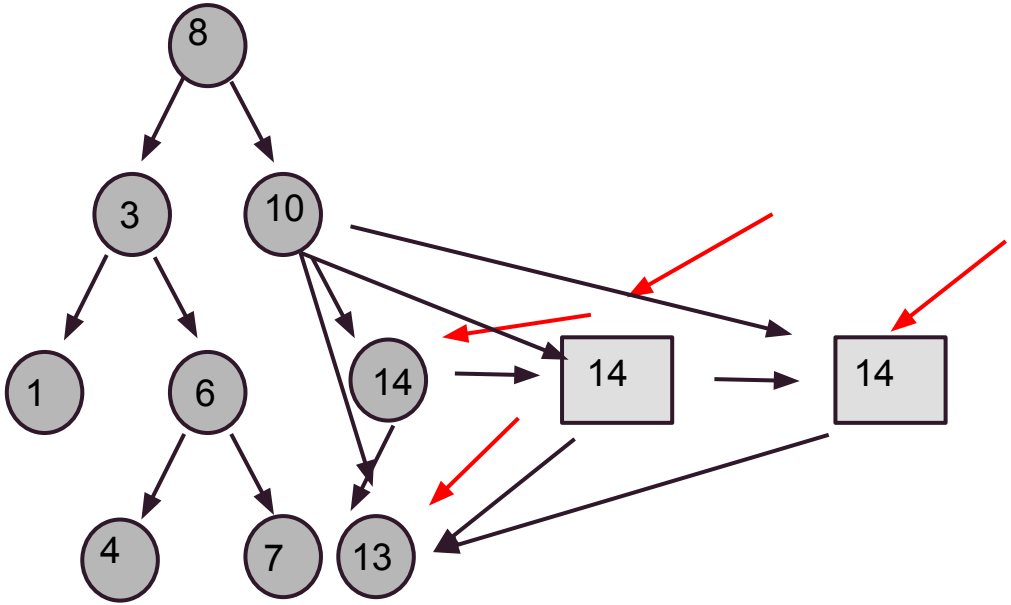
Hash Index	Max Seq #	Next Item To go	3 (array position of element 1)	4 (array position of duplicate 2)	5 (array position of duplicate 3)	6 (array position of duplicate 4)
1	1	1	5			
2	1	1	4			
3	1	1	3			
4						
5						
6						
7						
8	2	1	1	2		
9	1	1	0			

Dynamic: Build heap

8 3 10 1 6 4 7 14 13 **14** **14**



Dynamic: Delete Max



Correctness

By loop Invariant:

1. Initialization: at start count the loop variables

eg.Prove assertion P true at start of loop

2. Maintenance: after running the loop check the state of the variables

eg.Prove P is true at end of loop

3. Termination: when the loop terminates check the state again

eg.Prove that loop terminates

Complexity

	STATIC	DYNAMIC
Time (Build)	O(nlogn * 2) >>> O(nlogn)	O(logn *n) >>> O(nlogn)
Time (Delete)	O(logn * 2) >>> O(logn)	O(3) >>> O(1)
Space	O(n) + MaxValue(n) * MaxValue(k)	O(n) + n * MaxValue(k)