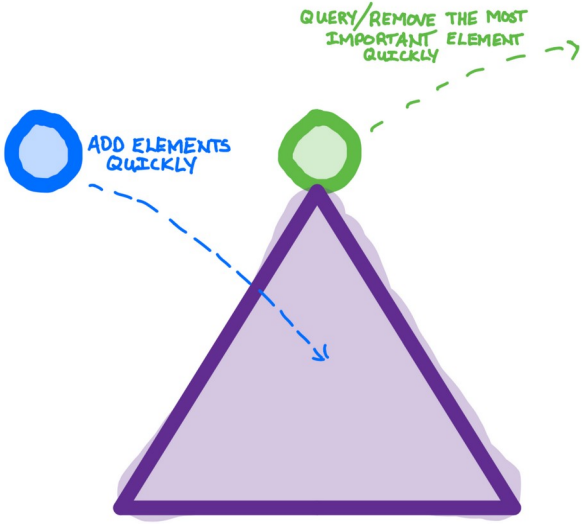
Lecture 22

Heaps



Announcements

- Office hours are up on course website
  - T: 12pm-4pm; F: 11am-1pm
  - Schedule an appointment individually or as a group

- Project II deliverable 1 feedback out today

Last time

Minimum Spanning Trees

- Prim's algo
- Kruskal's algo

Today

Heaps

- A new data structure
- Running time of operations
- Implement using an array

Use Heaps to Implement Priority Queue

Preliminaries

Storing data in a tree

A **complete binary tree** is a tree such that

i. Each vertex has at most two descendants
ii. All but the last layer have 2^i vertices
iii. Last layer is filled left to right

Preliminaries

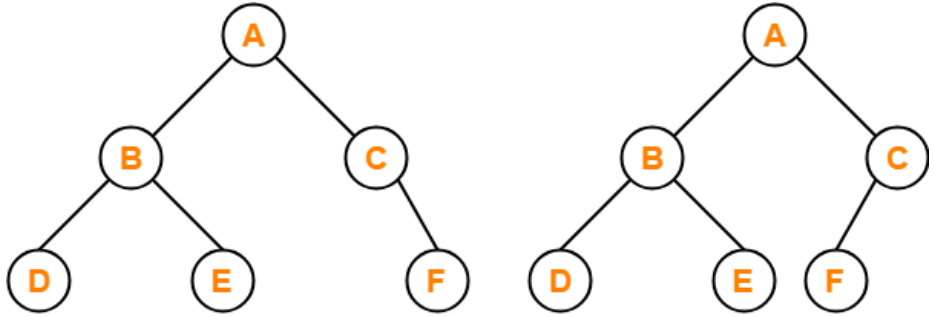
Storing data in a tree

A **complete binary tree** is a tree such that

i. Each vertex has at most two descendants

ii. All but the last layer have 2^i vertices

iii. Last layer is filled left to right



Preliminaries

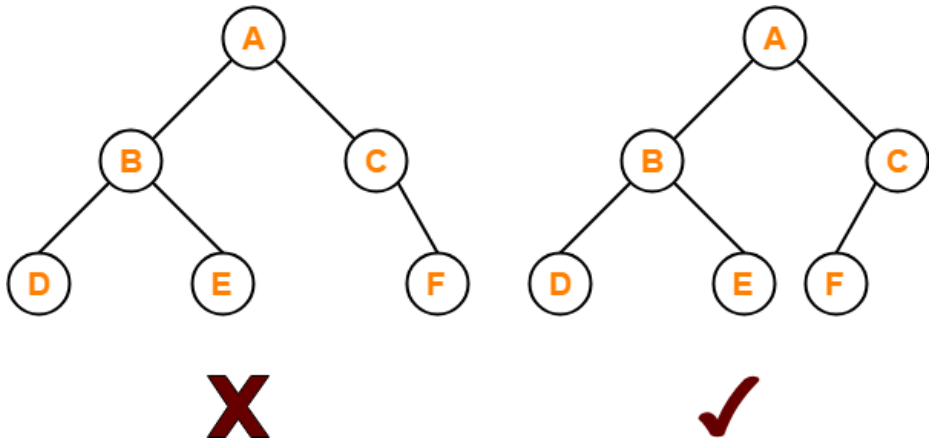
Storing data in a tree

A **complete binary tree** is a tree such that

i. Each vertex has at most two descendants

ii. All but the last layer have 2^i vertices

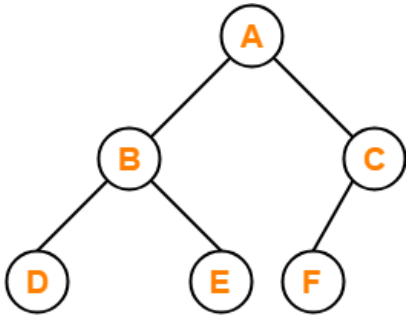
iii. Last layer is filled left to right



Preliminaries

We can represent CBT as an array

- Level `i` starts at index 2^i-1
- Level 1 starts at index 1
- Level 2 starts at index 3

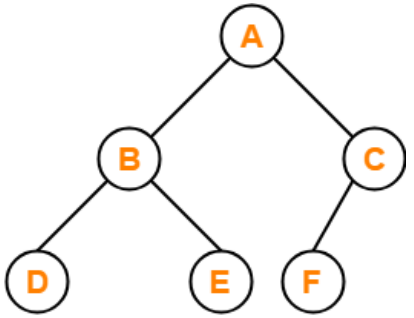


A	B	C	D	E	F

Preliminaries

We can represent CBT as an array

- Level `i` starts at index 2^i-1
- Level 1 starts at index 1
- Level 2 starts at index 3



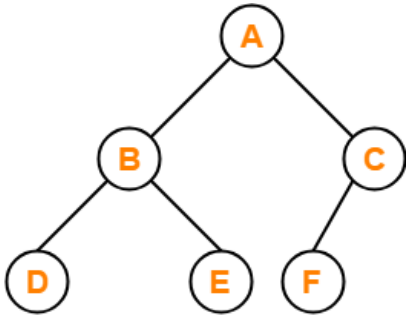
Why do we need our binary tree to be complete to represent it as an array?

A	B	C	D	E	F

Preliminaries

We can represent CBT as an array

- Level `i` starts at index 2^i-1
- Level 1 starts at index 1
- Level 2 starts at index 3



Why do we need our binary tree to be complete to represent it as an array?

- Otherwise, we would have to code for empty spaces

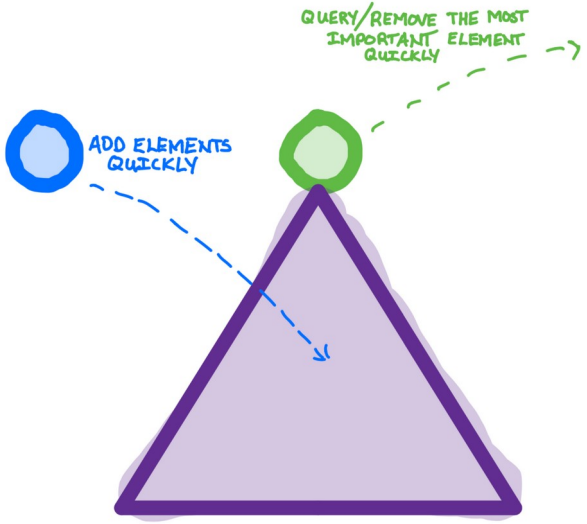
A	B	C	D	E	F

Heap

Data structure that allows efficient access to the maximum element

Operations

- **Peek**: look at maximum element
- **Insert**: add element to heap
- **Extract**: remove max element from heap



Implementing a Heap

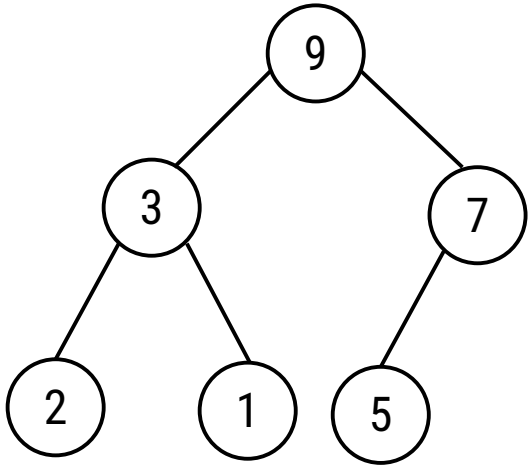
Use a CBT with the following rule

- Children are always smaller than parents

Implementing a Heap

Use a CBT with the following rule

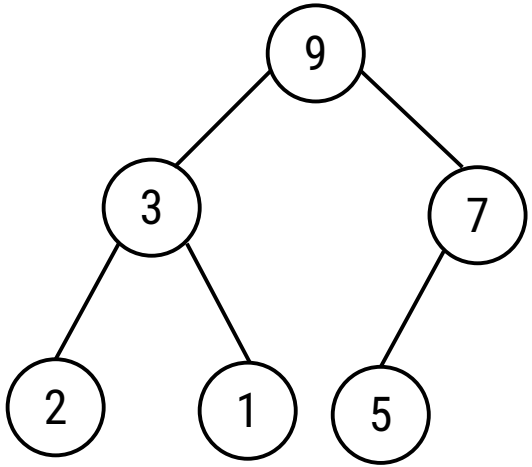
- Children are always smaller than parents



Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents



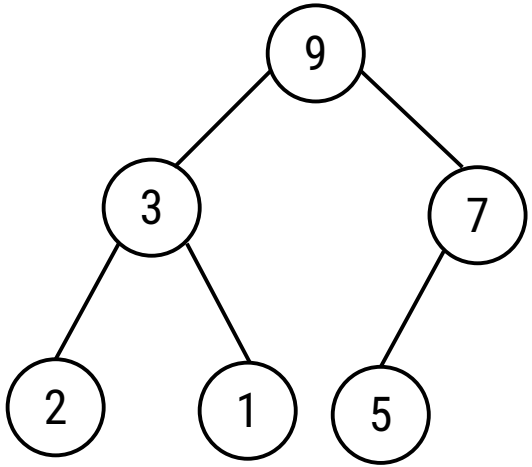
Peek Operation

- Look at the root value of CBT
- or index 0 of the CBT array
- Running time:

Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents



Peek Operation

- Look at the root value of CBT
- or index 0 of the CBT array
- Running time: `O(1)`

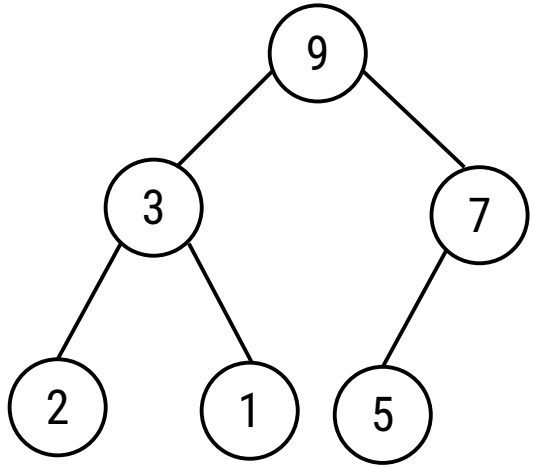
Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents

Insert Operation

- Add new element in next open position



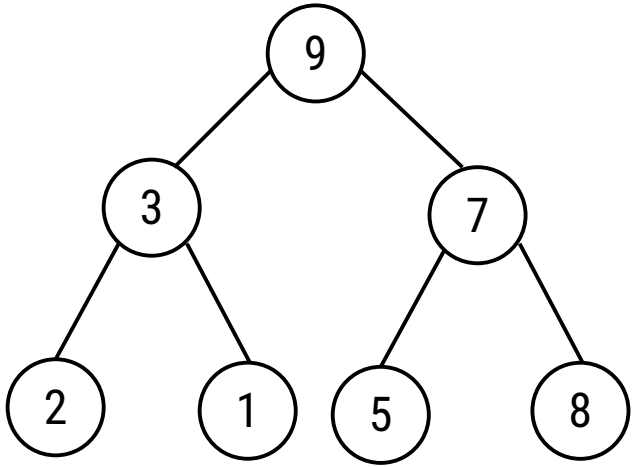
Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents

Insert Operation

- Add new element in next open position



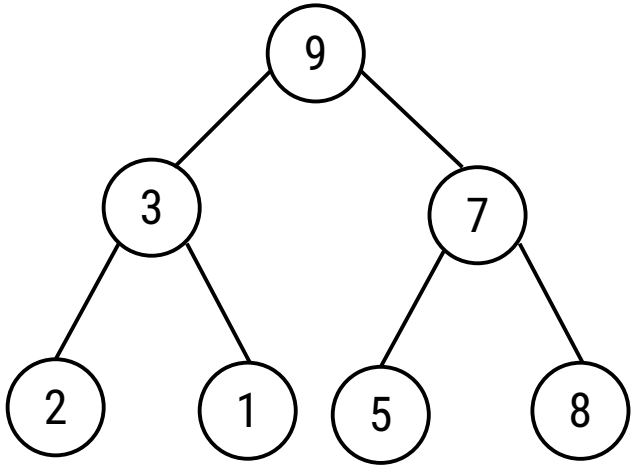
Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents

Insert Operation

- Add new element in next open position
- If rule is violated, swap with parent



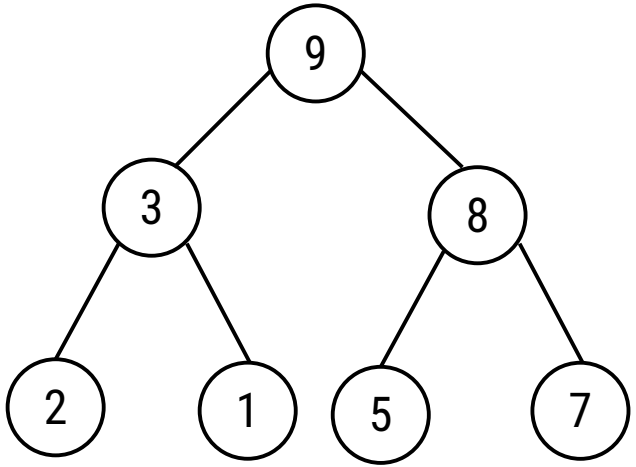
Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents

Insert Operation

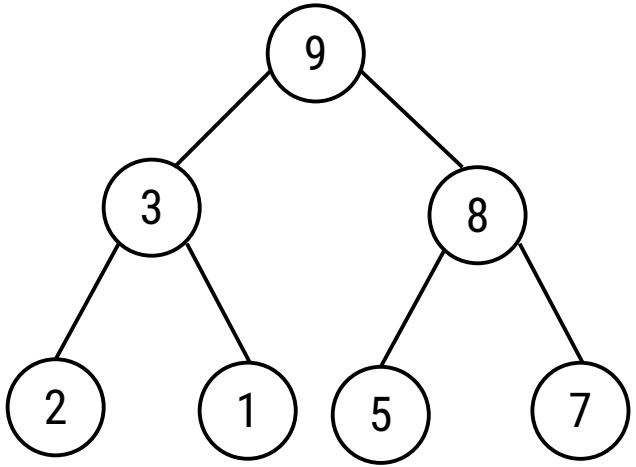
- Add new element in next open position
- If rule is violated, swap with parent



Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents



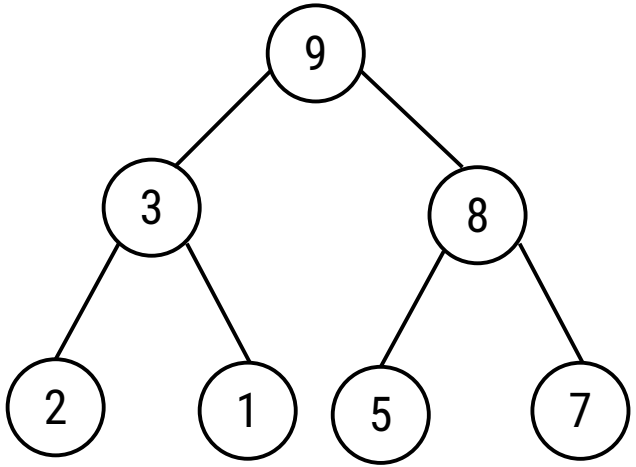
Insert Operation

- Add new element in next open position
- If rule is violated, swap with parent
- Running time:

Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents



Insert Operation

- Add new element in next open position
- If rule is violated, swap with parent
- Running time: `O(log n)`

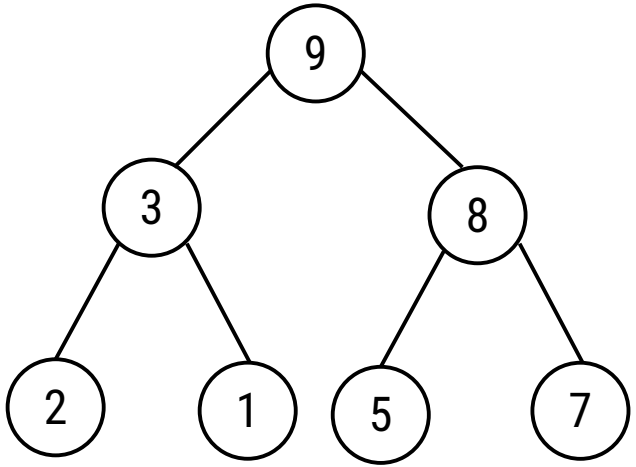
Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents

Extract Operation

- Remove the root



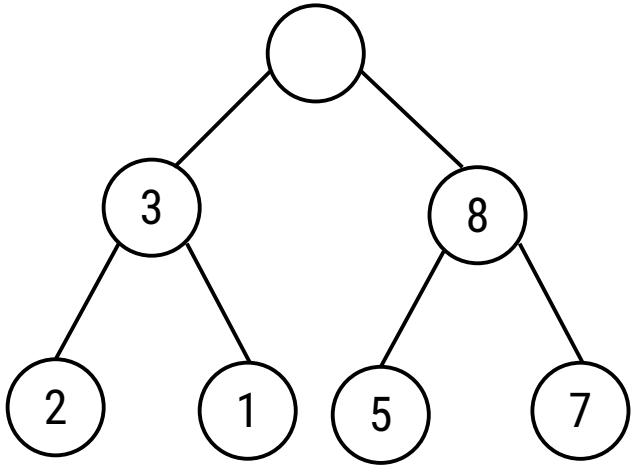
Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents

Extract Operation

- Remove the root



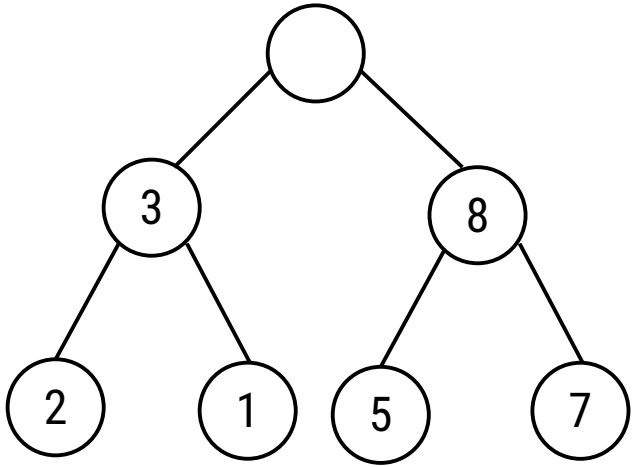
Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents

Extract Operation

- Remove the root
- Replace empty space with max of children and repeat



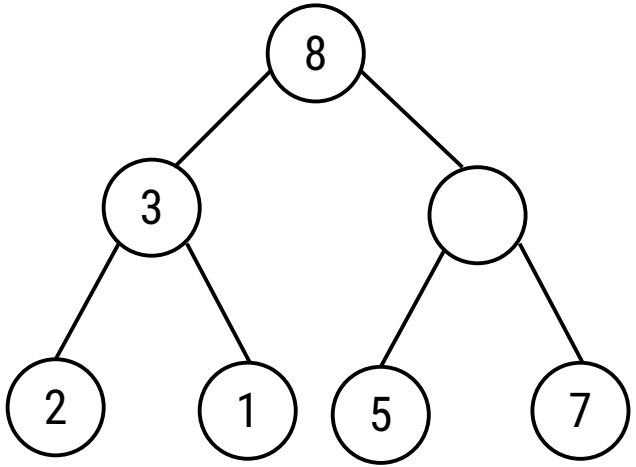
Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents

Extract Operation

- Remove the root
- Replace empty space with max of children and repeat



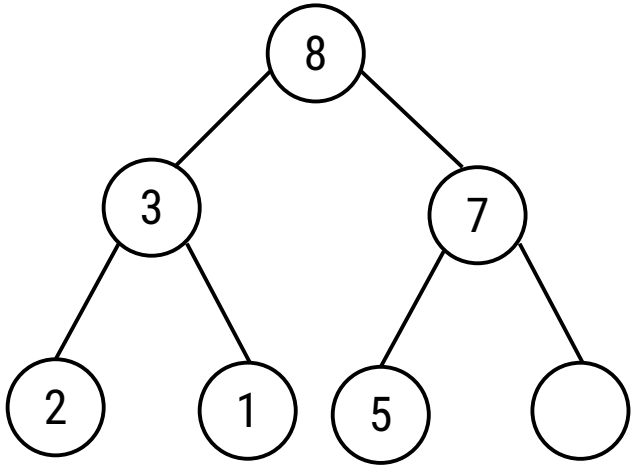
Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents

Extract Operation

- Remove the root
- Replace empty space with max of children and repeat



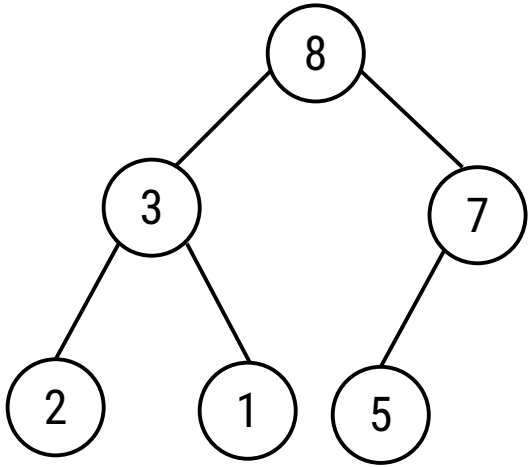
Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents

Extract Operation

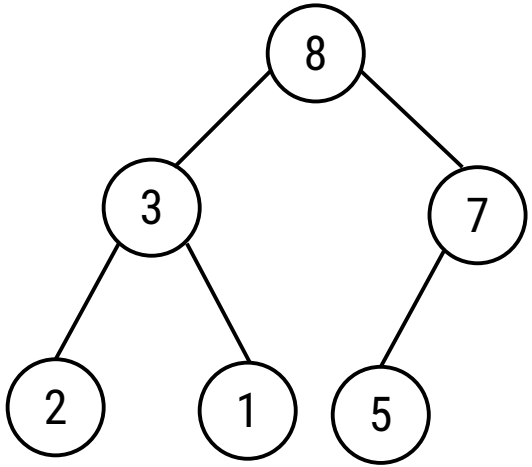
- Remove the root
- Replace empty space with max of children and repeat



Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents



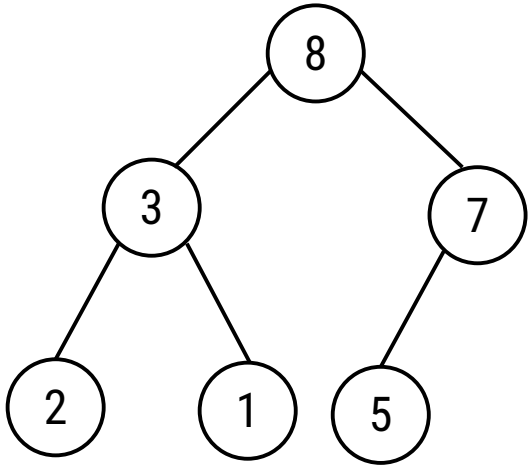
Extract Operation

- Remove the root
- Replace empty space with max of children and repeat
- Running time:

Implementing a Heap

Use a CBT with the following rule

- Children are always smaller than parents



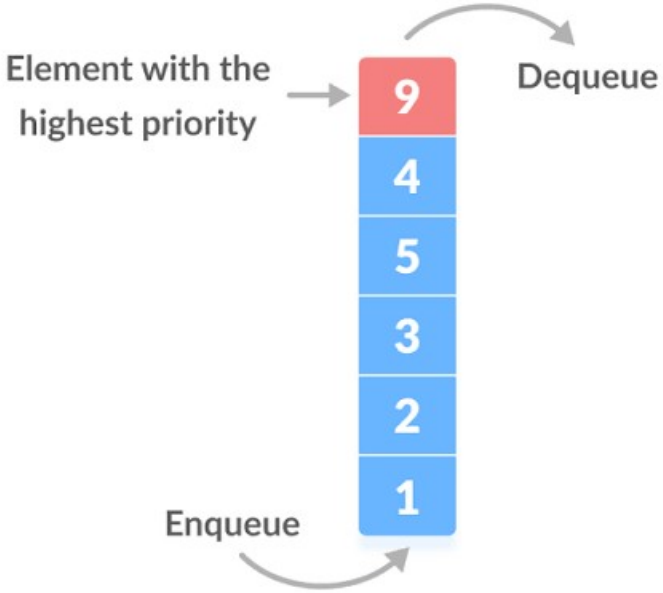
Extract Operation

- Remove the root
- Replace empty space with max of children and repeat
- Running time: `O(log n)`

Priority Queues

New data structure

- Like a queue (enqueue, dequeue, peek)
- Ordered by a priority score rather than which element entered the queue first

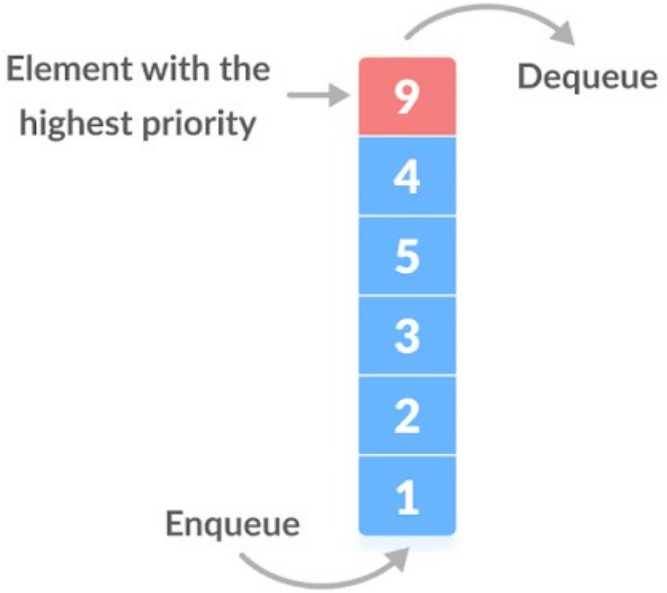


Priority Queues

New data structure

- Like a queue (enqueue, dequeue, peek)
- Ordered by a priority score rather than which element entered the queue first

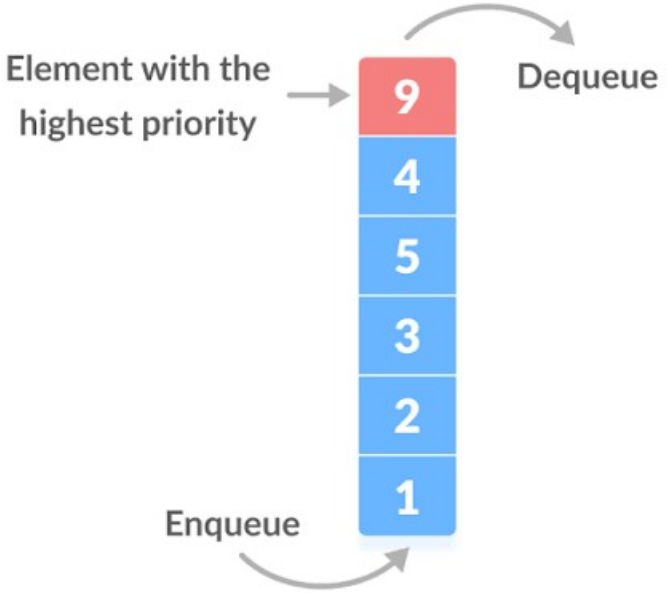
Implement a priority queue using a heap!



Exercise I

Dequeue the highest priority element (9)

- Trace operation across data structures
- Priority Queue → heap → CBT → array

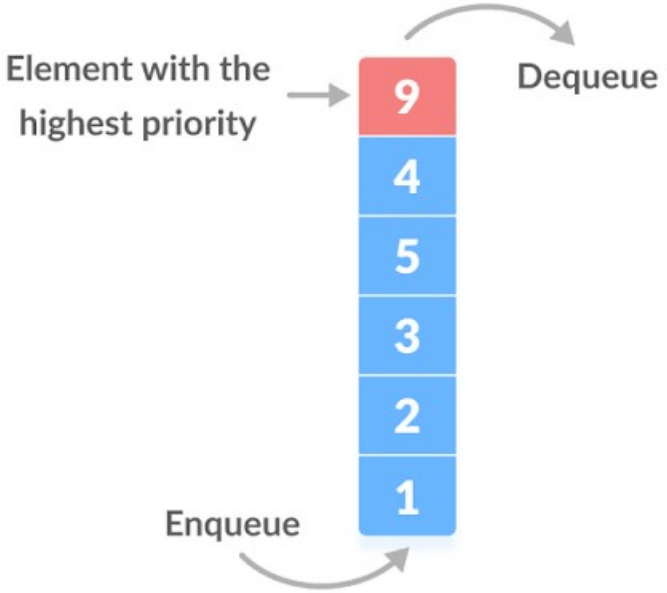


Exercise II

Dequeue the highest priority element (9)

- Trace operation across data structures
- Priority Queue → heap → CBT → array

Enqueue 6 and repeat



Next Time

- Maps