Heaps

CS 225 - Fall 2025

Mattox Beckman

Objectives

Objectives

Explain the operations of a Priority Queue

Describe a complete binary tree

Implement the heap data structure

Motivation

How does a regular queue work?

Motivation

How does a regular queue work?

- First In First Out

But what if you want a ‘fast lane’?

Motivation

How does a regular queue work?

- First In First Out

But what if you want a ‘fast lane’?

Priority Queue

- Data inserted gets a **priority**
- Data is removed in priority order

False Start

Implementation Attempt

- Use a `vector` to store the data
- Read in all the data and call `sort`
- Remove elements by travesing from first element

Why won't this work?

False Start

Implementation Attempt

- Use a `vector` to store the data
- Read in all the data and call `sort`
- Remove elements by travesing from first element

Why won't this work?

- It's fine only if you have all the data up front
- Inserting new data is $\mathcal{O}(n)$

Introduction

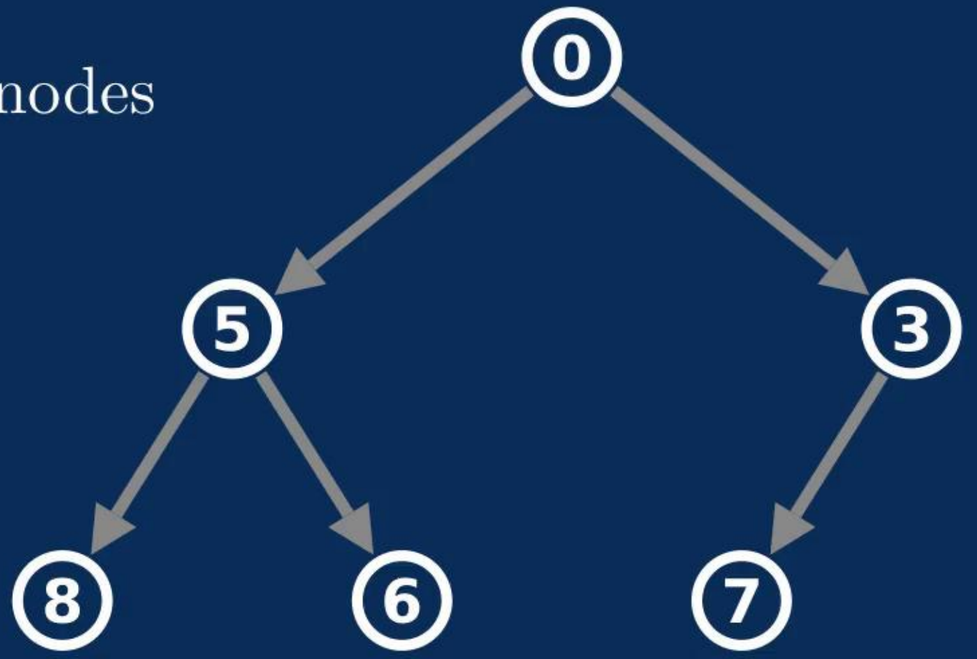
Introducing Heaps

Use a *binary tree* to store the data

Has the *Heap Property*:

- Parent nodes have *higher priority* than child nodes
- Tree is *complete*
- Lowest level filled *left to right*
- Let lower key = higher priority

We don't care about sibling relations!



Insertion

Place key in lowest left position

If root: we are done.

While key < parent:

swap(key,parent)

Insert 8

Insertion

Place key in lowest left position

If root: we are done.

While key < parent:

swap(key,parent)

Insert 6

Insertion

Place key in lowest left position

If root: we are done.

While key < parent:

swap(key,parent)

Insert 7

Insertion

Place key in lowest left position

If root: we are done.

While key < parent:

swap(key,parent)

Insert 5

Insertion

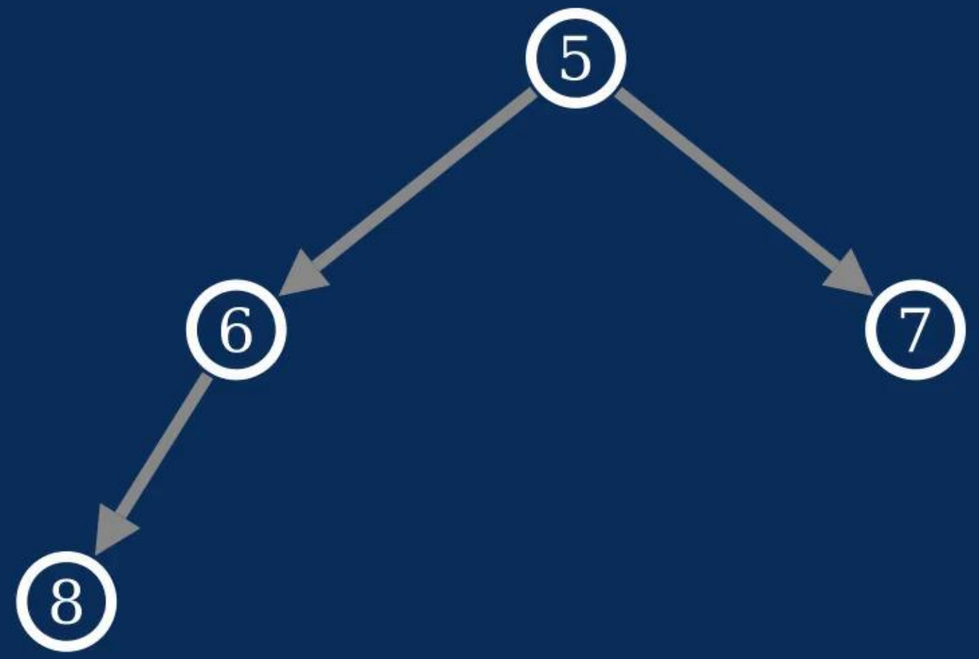
Place key in lowest left position

If root: we are done.

While key < parent:

swap(key,parent)

Insert 3



Insertion

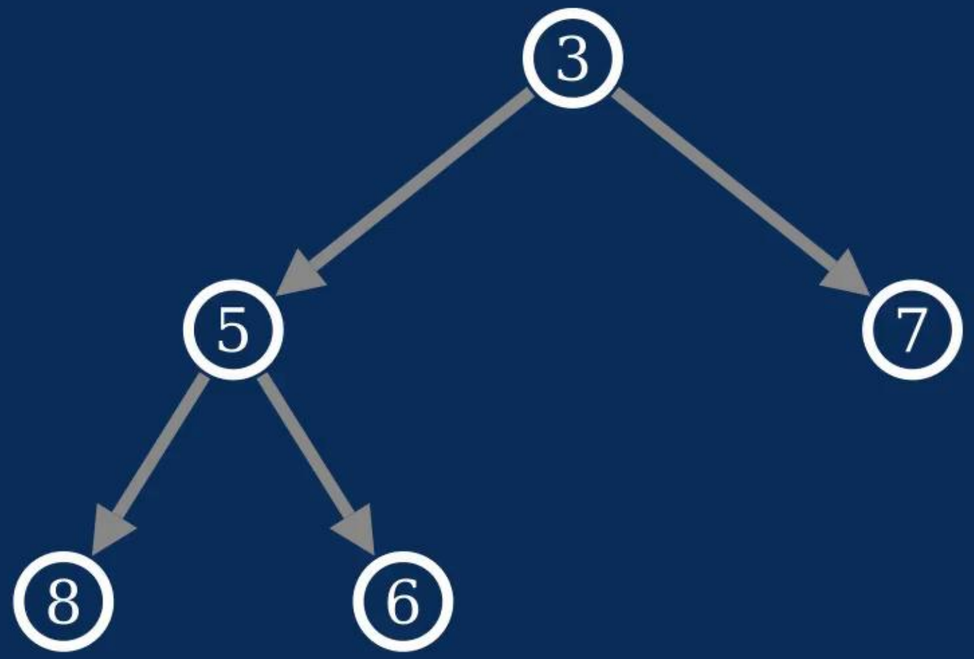
Place key in lowest left position

If root: we are done.

While key < parent:

swap(key,parent)

Insert 0



Insertion

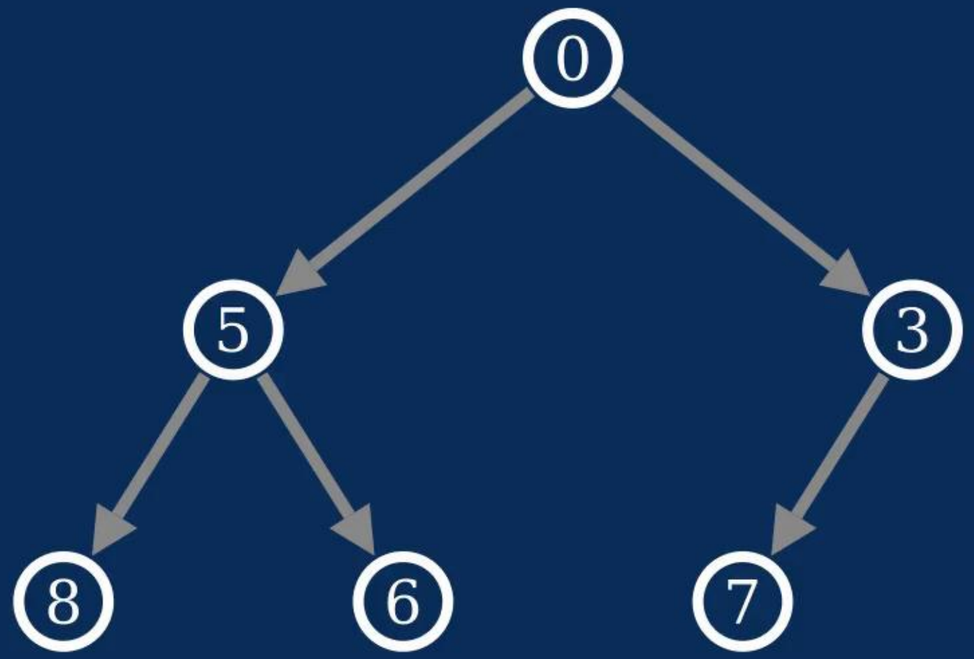
Place key in lowest left position

If root: we are done.

While key < parent:

swap(key,parent)

Insert 9



Insertion

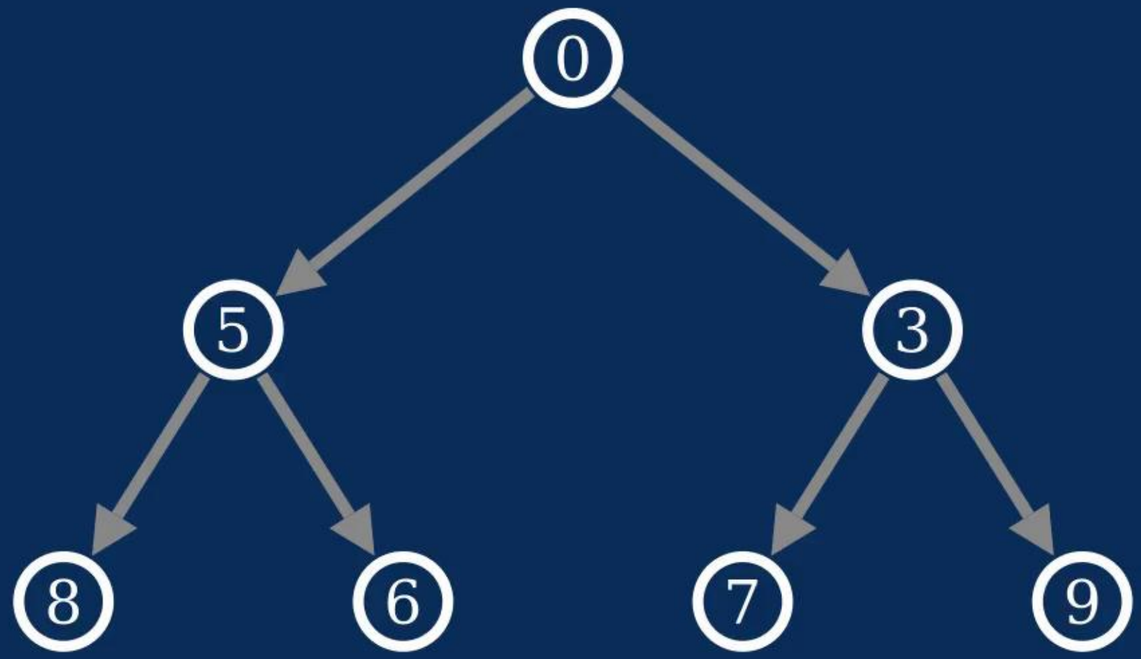
Place key in lowest left position

If root: we are done.

While key < parent:

swap(key,parent)

Insert 9



Remove

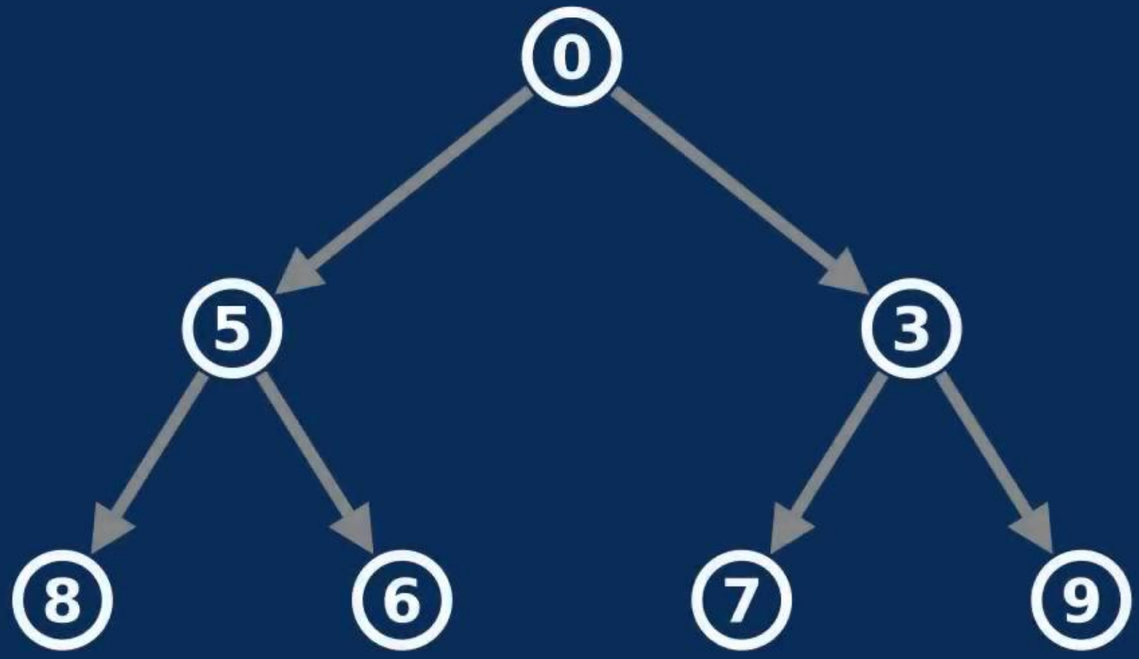
The root is the smallest element.

How to remove it?

Move last element k to root

While smallest child $< k$:

- swap(smallest child,k)



Remove

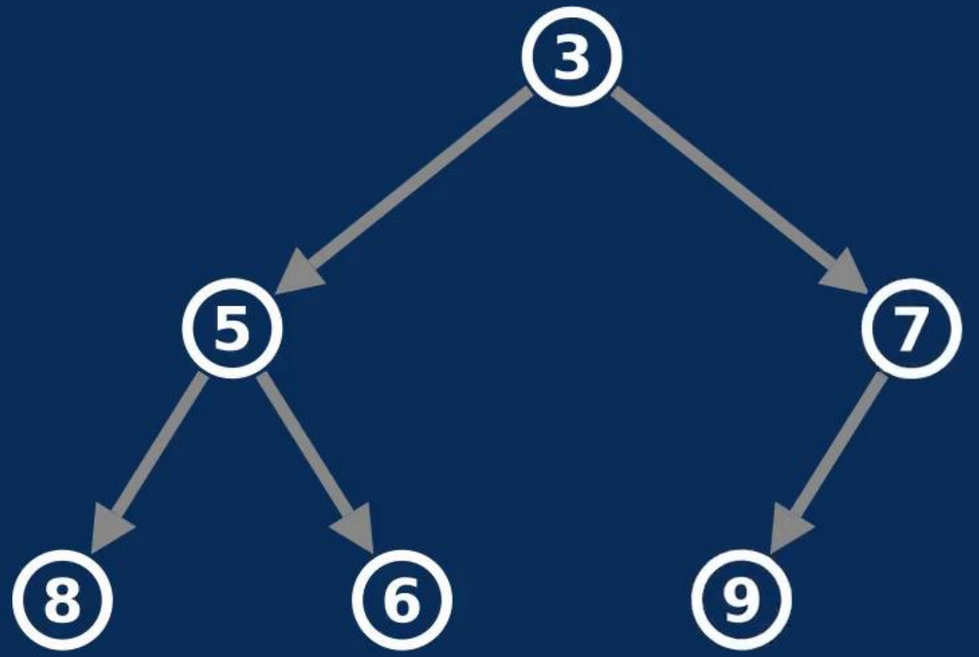
The root is the smallest element.

How to remove it?

Move last element k to root

While smallest child $< k$:

- swap(smallest child,k)



Remove

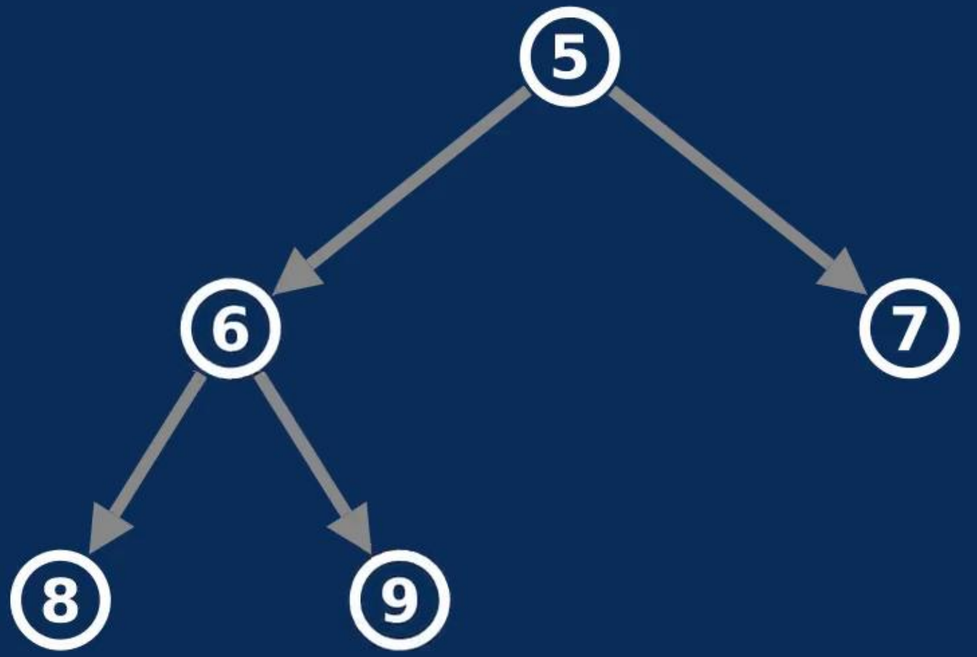
The root is the smallest element.

How to remove it?

Move last element k to root

While smallest child $< k$:

- swap(smallest child,k)



Remove

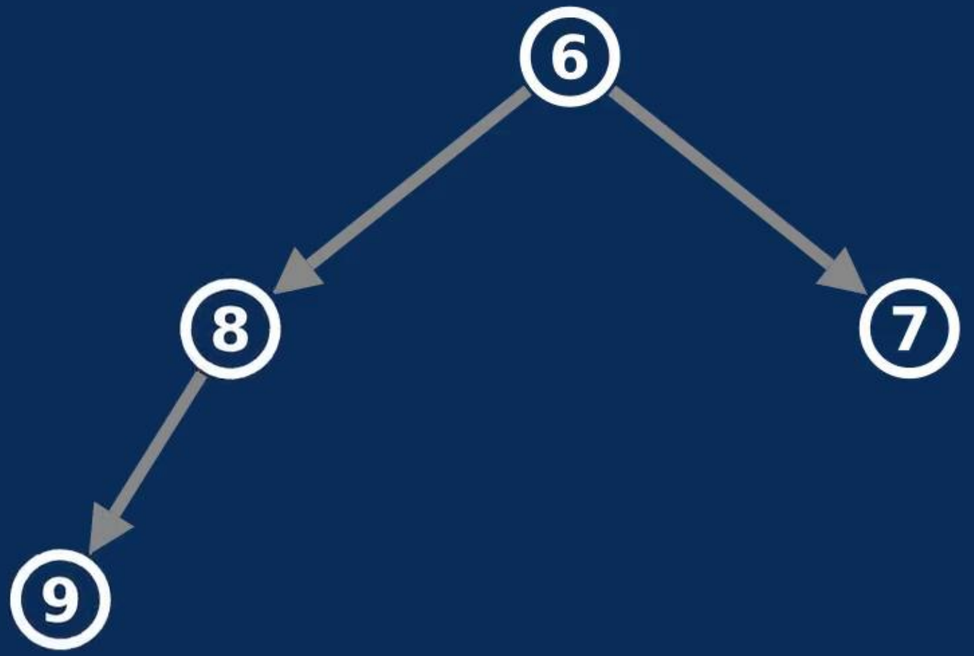
The root is the smallest element.

How to remove it?

Move last element k to root

While smallest child $< k$:

- swap(smallest child,k)



Remove

The root is the smallest element.

How to remove it?

Move last element k to root

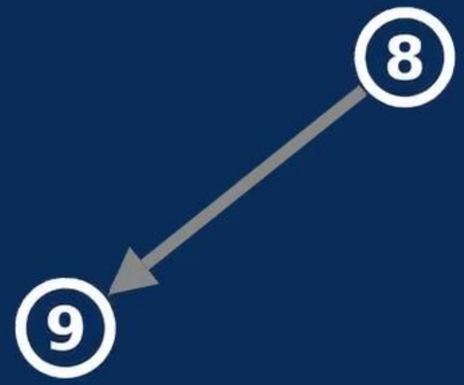
While smallest child $< k$:

- swap(smallest child,k)

Remove

The root is the smallest element.

How to remove it?



Move last element k to root

While smallest child $< k$:

- swap(smallest child,k)

Remove

The root is the smallest element.

How to remove it?

Move last element k to root

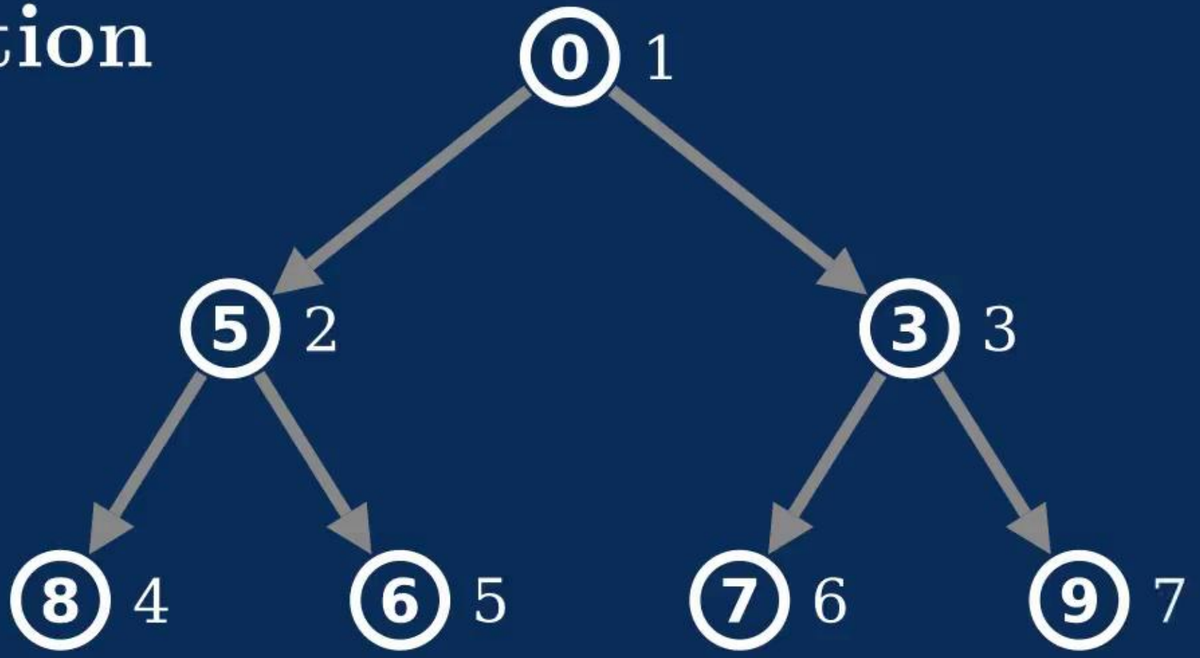
While smallest child $< k$:

- swap(smallest child,k)

Representation

An efficient representation

Let's number the nodes.



Representation

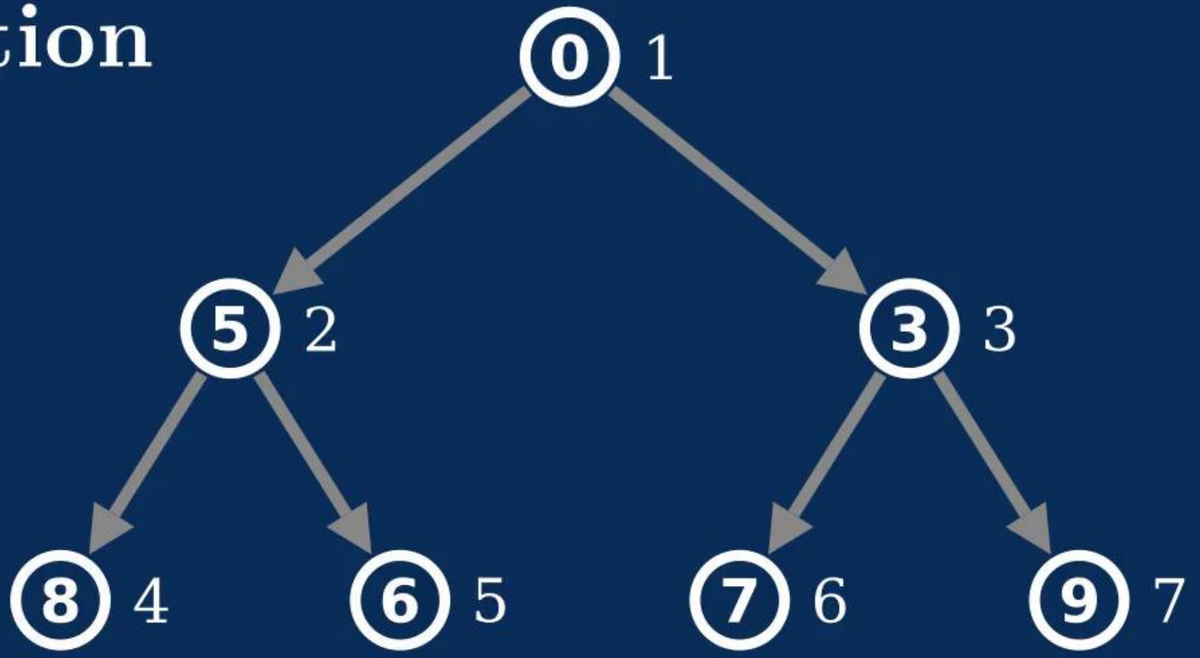
An efficient representation

Let's number the nodes.

$up(i) = \lfloor \frac{i}{2} \rfloor$

$left(i) = 2i$

$right(i) = 2i + 1$



Representation

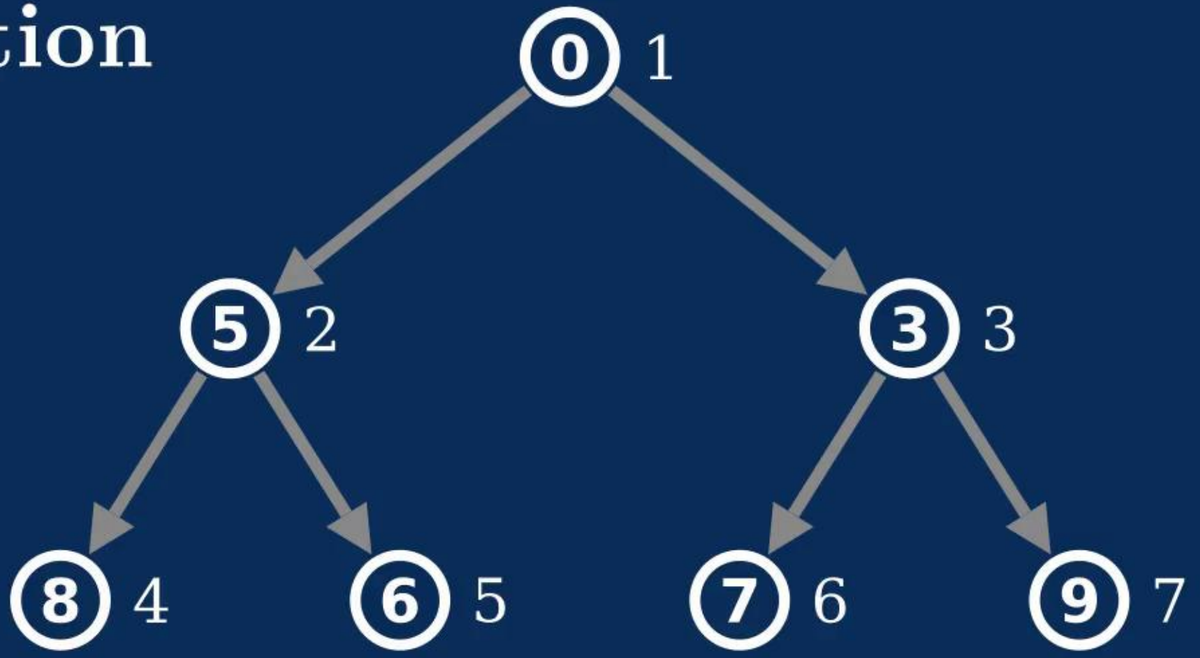
An efficient representation

Let's number the nodes.

$up(i) = \lfloor \frac{i}{2} \rfloor$

$left(i) = 2i$

$right(i) = 2i + 1$



0	1	2	3	4	5	6	7
S	0	5	3	8	6	7	9

Insert

```
 1 int up(int i) { return i/2; }
 2 void insert(vector<int> heap, int v) {
 3     heap.push_back(v);
 4     int idx = heap.size() - 1;
 5     int parent = up(idx);
 6     while (idx > 1 && heap[parent] > v) {
 7         heap[idx] = heap[parent];
 8         heap[parent] = v;
 9         idx = parent;
10         parent = up(parent);
11     }
12 }
```

Insert

```
 1 int up(int i) { return i/2; }
 2 void insert(vector<int> heap, int v) {
 3     heap.push_back(v);
 4     int idx = heap.size() - 1;
 5     int parent = up(idx);
 6     while (idx > 1 && heap[parent] > v) {
 7         heap[idx] = heap[parent];
 8         heap[parent] = v;
 9         idx = parent;
10         parent = up(parent);
11     }
12 }
```

Place element at end

Insert


```
 1 int up(int i) { return i/2; }
 2 void insert(vector<int> heap, int v) {
 3     heap.push_back(v);
 4     int idx = heap.size() - 1;
 5     int parent = up(idx);
 6     while (idx > 1 && heap[parent] > v) {
 7         heap[idx] = heap[parent];
 8         heap[parent] = v;
 9         idx = parent;
10         parent = up(parent);
11     }
12 }
```

Place element at end
Get index and parent

Insert


```
 1 int up(int i) { return i/2; }
 2 void insert(vector<int> heap, int v) {
 3     heap.push_back(v);
 4     int idx = heap.size() - 1;
 5     int parent = up(idx);
 6     while (idx > 1 && heap[parent] > v) {
 7         heap[idx] = heap[parent];
 8         heap[parent] = v;
 9         idx = parent;
10         parent = up(parent);
11     }
12 }
```

Place element at end
Get index and parent

While not done...

Insert


```
 1 int up(int i) { return i/2; }
 2 void insert(vector<int> heap, int v) {
 3     heap.push_back(v);
 4     int idx = heap.size() - 1;
 5     int parent = up(idx);
 6     while (idx > 1 && heap[parent] > v) {
 7         heap[idx] = heap[parent];
 8         heap[parent] = v;
 9         idx = parent;
10         parent = up(parent);
11     }
12 }
```

Place element at end
Get index and parent

While not done...
Swap with parent

Insert

```
 1 int up(int i) { return i/2; }
 2 void insert(vector<int> heap, int v) {
 3     heap.push_back(v);
 4     int idx = heap.size() - 1;
 5     int parent = up(idx);
 6     while (idx > 1 && heap[parent] > v) {
 7         heap[idx] = heap[parent];
 8         heap[parent] = v;
 9         idx = parent;
10         parent = up(parent);
11     }
12 }
```

Place element at end
Get index and parent

While not done...
Swap with parent

Update idx and parent

Insert

```
 1 int up(int i) { return i/2; }
 2 void insert(vector<int> heap, int v) {
 3     heap.push_back(v);
 4     int idx = heap.size() - 1;
 5     int parent = up(idx);
 6     while (idx > 1 && heap[parent] > v) {
 7         heap[idx] = heap[parent];
 8         heap[parent] = v;
 9         idx = parent;
10         parent = up(parent);
11     }
12 }
```

Place element at end
Get index and parent

While not done...
Swap with parent

Update idx and parent

Pop


```
 1 void pop(vector<int> heap) {
 2     // Assuming heap is not empty!
 3     heap[1] = heap.back();
 4     heap.pop_back();
 5     int idx = 1;
 6     while (true) {
 7         int left = idx * 2;
 8         int right = idx * 2 + 1;
 9         int smallest = idx;
10         if (left < heap.size() &&
11             heap[left] < heap[idx]) smallest = left;
12         if (right < heap.size() &&
13             heap[right] < heap[smallest]) smallest = right;
14         if (smallest == idx) break;
15         swap(heap[idx],heap[smallet]);
16         idx = smallest;
17     }
18 }
```

Pop


```
 1 void pop(vector<int> heap) {
 2     // Assuming heap is not empty!
 3     heap[1] = heap.back();
 4     heap.pop_back();
 5     int idx = 1;
 6     while (true) {
 7         int left = idx * 2;
 8         int right = idx * 2 + 1;
 9         int smallest = idx;
10         if (left < heap.size() &&
11             heap[left] < heap[idx]) smallest = left;
12         if (right < heap.size() &&
13             heap[right] < heap[smallest]) smallest = right;
14         if (smallest == idx) break;
15         swap(heap[idx],heap[smallet]);
16         idx = smallest;
17     }
18 }
```

Replace root with last

Pop


```
 1 void pop(vector<int> heap) {
 2     // Assuming heap is not empty!
 3     heap[1] = heap.back();
 4     heap.pop_back();
 5     int idx = 1;
 6     while (true) {
 7         int left = idx * 2;
 8         int right = idx * 2 + 1;
 9         int smallest = idx;
10         if (left < heap.size() &&
11             heap[left] < heap[idx]) smallest = left;
12         if (right < heap.size() &&
13             heap[right] < heap[smallest]) smallest = right;
14         if (smallest == idx) break;
15         swap(heap[idx],heap[smallet]);
16         idx = smallest;
17     }
18 }
```

Replace root with last
Remove last element

Pop


```
 1 void pop(vector<int> heap) {
 2     // Assuming heap is not empty!
 3     heap[1] = heap.back();
 4     heap.pop_back();
 5     int idx = 1;
 6     while (true) {
 7         int left = idx * 2;
 8         int right = idx * 2 + 1;
 9         int smallest = idx;
10         if (left < heap.size() &&
11             heap[left] < heap[idx]) smallest = left;
12         if (right < heap.size() &&
13             heap[right] < heap[smallest]) smallest = right;
14         if (smallest == idx) break;
15         swap(heap[idx],heap[smallet]);
16         idx = smallest;
17     }
18 }
```

Replace root with last
Remove last element

Set left, right children

Pop


```
 1 void pop(vector<int> heap) {
 2     // Assuming heap is not empty!
 3     heap[1] = heap.back();
 4     heap.pop_back();
 5     int idx = 1;
 6     while (true) {
 7         int left = idx * 2;
 8         int right = idx * 2 + 1;
 9         int smallest = idx;
10         if (left < heap.size() &&
11             heap[left] < heap[idx]) smallest = left;
12         if (right < heap.size() &&
13             heap[right] < heap[smallest]) smallest = right;
14         if (smallest == idx) break;
15         swap(heap[idx],heap[smallet]);
16         idx = smallest;
17     }
18 }
```

Replace root with last
Remove last element

Set left, right children

Who is smallest?

Pop


```
 1 void pop(vector<int> heap) {
 2     // Assuming heap is not empty!
 3     heap[1] = heap.back();
 4     heap.pop_back();
 5     int idx = 1;
 6     while (true) {
 7         int left = idx * 2;
 8         int right = idx * 2 + 1;
 9         int smallest = idx;
10         if (left < heap.size() &&
11             heap[left] < heap[idx]) smallest = left;
12         if (right < heap.size() &&
13             heap[right] < heap[smallest]) smallest = right;
14         if (smallest == idx) break;
15         swap(heap[idx],heap[smallet]);
16         idx = smallest;
17     }
18 }
```

Replace root with last
Remove last element

Set left, right children

Who is smallest?
Check left

Pop


```
 1 void pop(vector<int> heap) {
 2     // Assuming heap is not empty!
 3     heap[1] = heap.back();
 4     heap.pop_back();
 5     int idx = 1;
 6     while (true) {
 7         int left = idx * 2;
 8         int right = idx * 2 + 1;
 9         int smallest = idx;
10         if (left < heap.size() &&
11             heap[left] < heap[idx]) smallest = left;
12         if (right < heap.size() &&
13             heap[right] < heap[smallest]) smallest = right;
14         if (smallest == idx) break;
15         swap(heap[idx],heap[smallet]);
16         idx = smallest;
17     }
18 }
```

Replace root with last
Remove last element

Set left, right children

Who is smallest?
Check left

Check right

Pop

```
 1 void pop(vector<int> heap) {
 2     // Assuming heap is not empty!
 3     heap[1] = heap.back();
 4     heap.pop_back();
 5     int idx = 1;
 6     while (true) {
 7         int left = idx * 2;
 8         int right = idx * 2 + 1;
 9         int smallest = idx;
10         if (left < heap.size() &&
11             heap[left] < heap[idx]) smallest = left;
12         if (right < heap.size() &&
13             heap[right] < heap[smallest]) smallest = right;
14         if (smallest == idx) break;
15         swap(heap[idx],heap[smallet]);
16         idx = smallest;
17     }
18 }
```

Replace root with last
Remove last element

Set left, right children

Who is smallest?
Check left

Check right

Bail out when done

Pop

```
 1 void pop(vector<int> heap) {
 2     // Assuming heap is not empty!
 3     heap[1] = heap.back();
 4     heap.pop_back();
 5     int idx = 1;
 6     while (true) {
 7         int left = idx * 2;
 8         int right = idx * 2 + 1;
 9         int smallest = idx;
10         if (left < heap.size() &&
11             heap[left] < heap[idx]) smallest = left;
12         if (right < heap.size() &&
13             heap[right] < heap[smallest]) smallest = right;
14         if (smallest == idx) break;
15         swap(heap[idx],heap[smallet]);
16         idx = smallest;
17     }
18 }
```

Replace root with last
Remove last element

Set left, right children

Who is smallest?
Check left

Check right

Bail out when done
Swap with smallest

Pop


```
 1 void pop(vector<int> heap) {
 2     // Assuming heap is not empty!
 3     heap[1] = heap.back();
 4     heap.pop_back();
 5     int idx = 1;
 6     while (true) {
 7         int left = idx * 2;
 8         int right = idx * 2 + 1;
 9         int smallest = idx;
10         if (left < heap.size() &&
11             heap[left] < heap[idx]) smallest = left;
12         if (right < heap.size() &&
13             heap[right] < heap[smallest]) smallest = right;
14         if (smallest == idx) break;
15         swap(heap[idx],heap[smallet]);
16         idx = smallest;
17     }
18 }
```

Replace root with last
Remove last element

Set left, right children

Who is smallest?
Check left

Check right

Bail out when done
Swap with smallest
Move up

Next Time

Analyze the time complexity of this structure

How to convert a random array into a heap quickly!

Disclaimers….

- Code didn't handle some edge cases
- Min Heap vs. Max Heap
- Sentinel vs. no Sentinel
- "Your mileage may vary"