Implementation

In this lesson, we'll see how to implement a heap and involved operations.

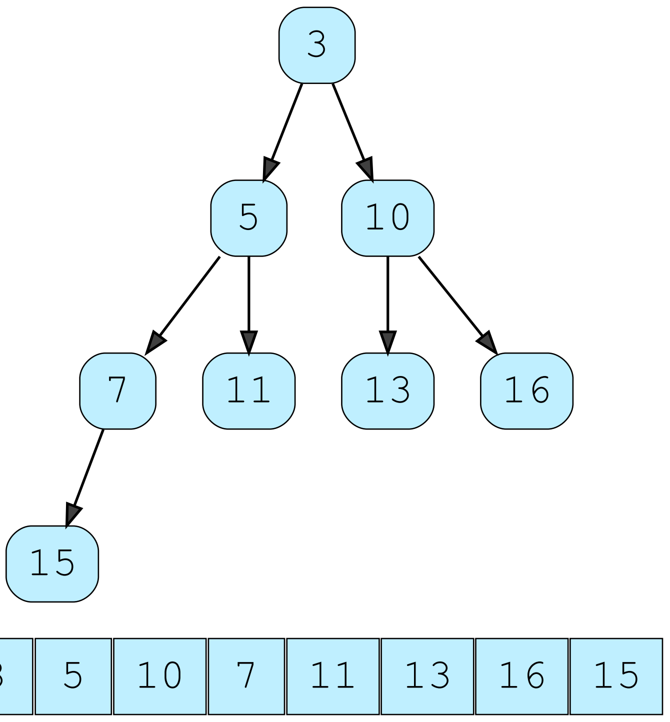
We'll cover the following

Heapify

Starting with an empty heap, let’s see how to build a heap of N elements.

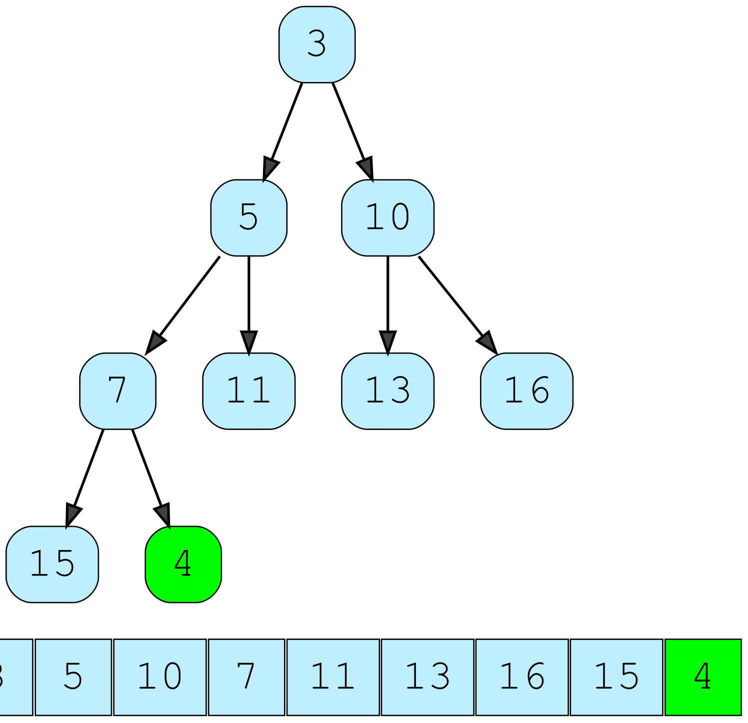
What happens when a new element is added at the end of a min-heap? Obviously, this new element can violate the heap property. An important point to note is that it violates the heap property for its parent only.

Now we can check this (parent > child) and swap the nodes, after which, the parent might be violating the min-heap property with its parent. We keep checking until we reach the root.



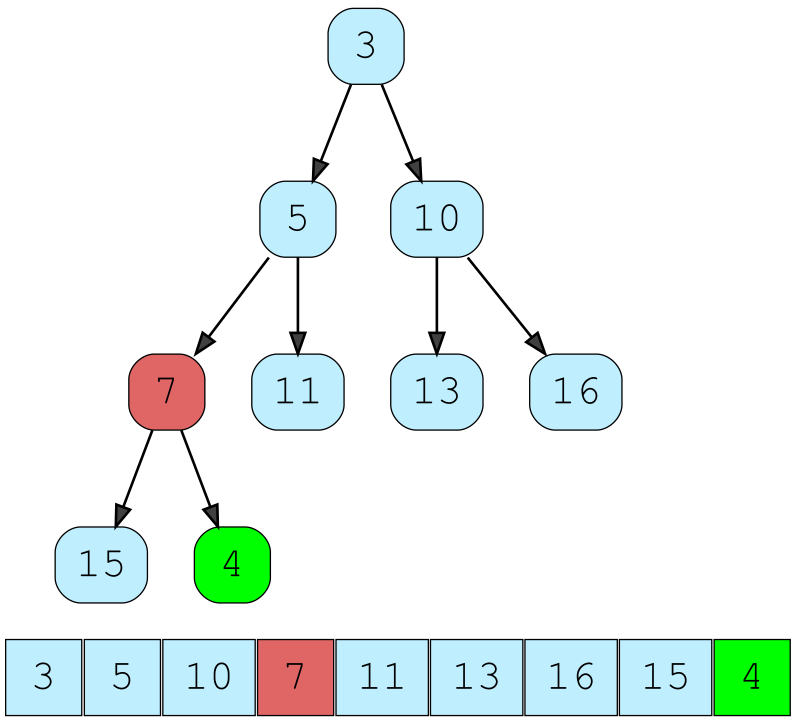
A Min Heap

1 of 9



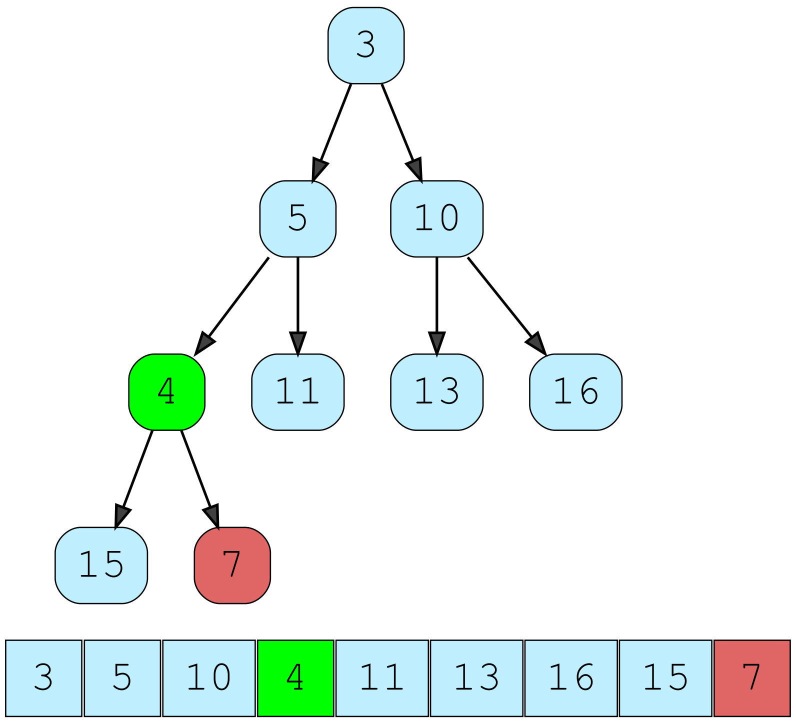
add 4 to the end

2 of 9

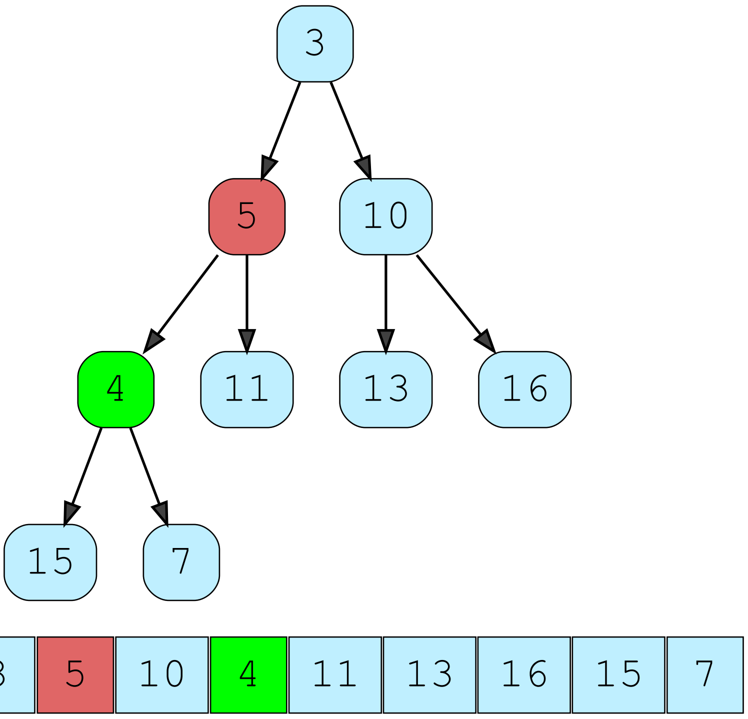


Heap property violated for 7, so we swap

3 of 9



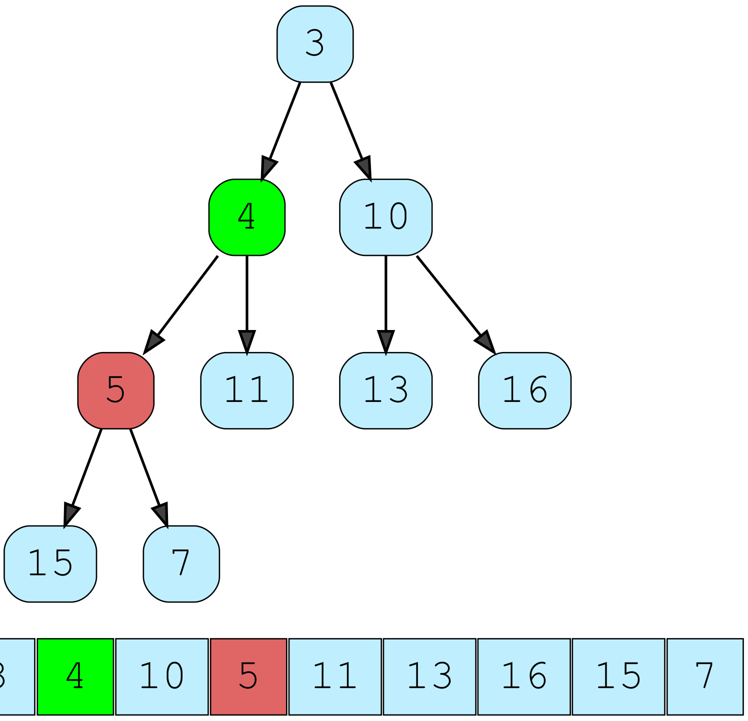
4 of 9



3	5	10	4	11	13	16	15	7

Now check with it's parent

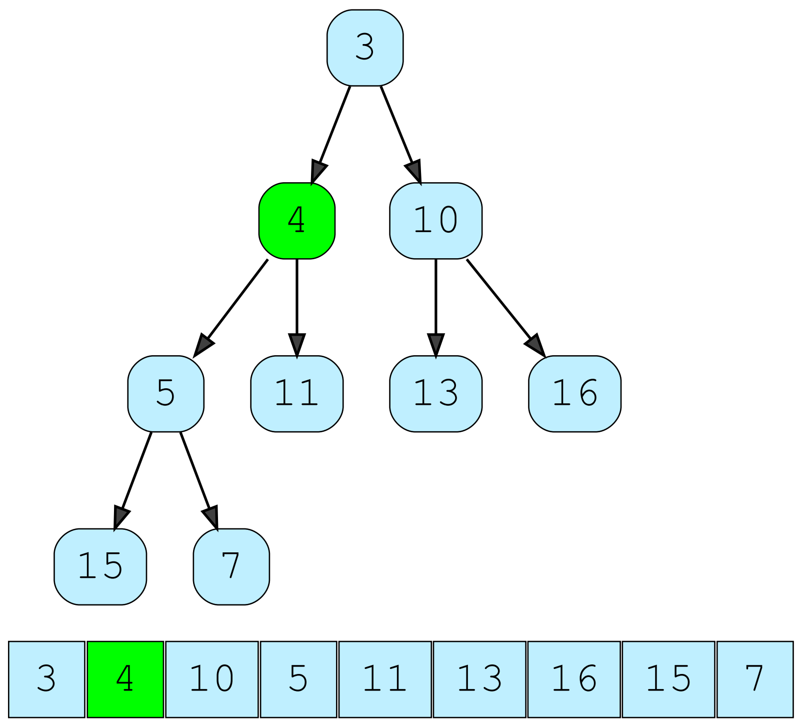
5 of 9



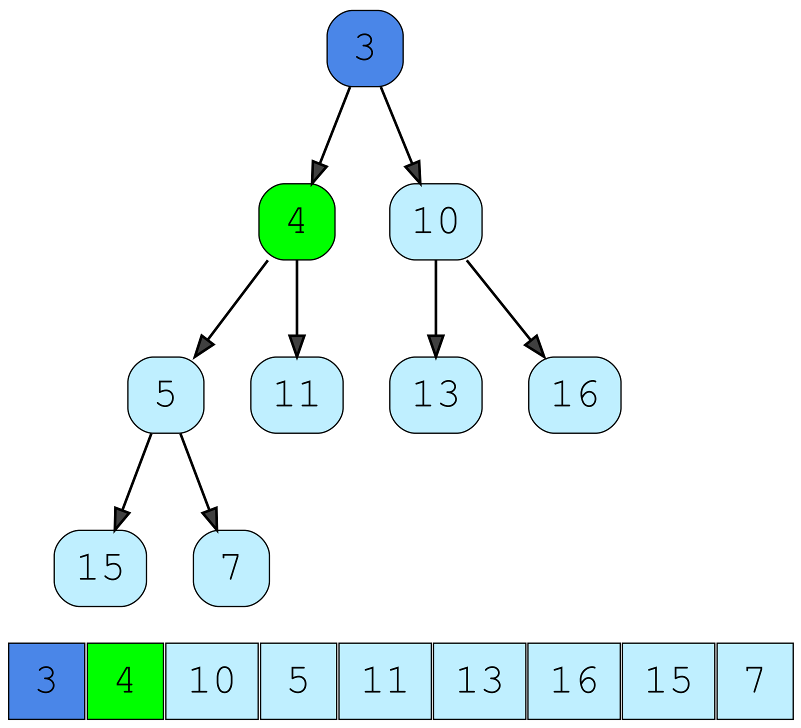
3	4	10	5	11	13	16	15	7

Now check with it's parent

6 of 9

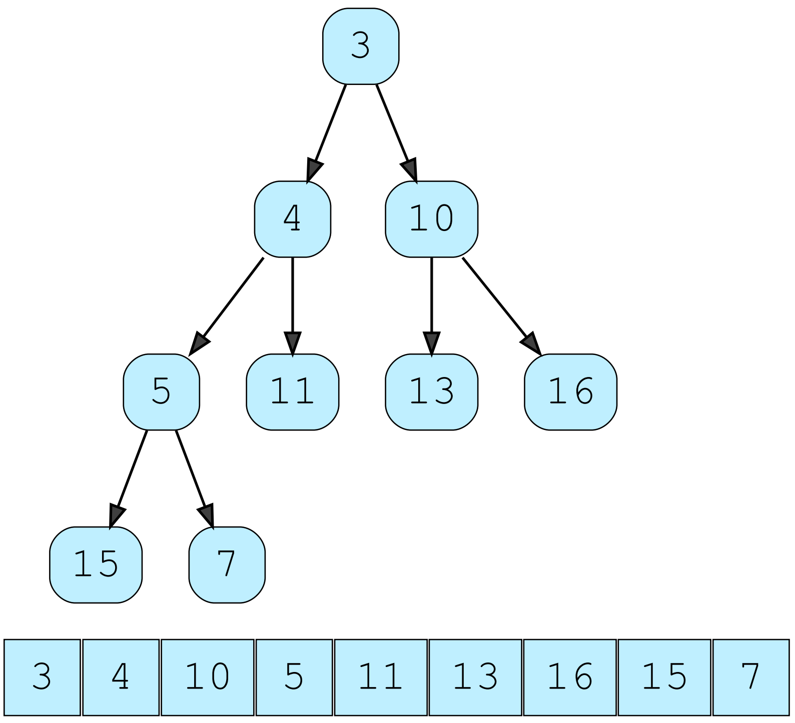


7 of 9



Again, check with parent. No need to swap here

8 of 9



We get min heap back

9 of 9

Building the heap

We can build a min-heap using an array of N integers by assuming an empty heap, inserting the integers one by one, and using the above technique to restore the heap property after every insertion.

```
#include <bits/stdc++.h>
using namespace std;

void min_heapify_bottom_up(int arr[], int N, int idx) {
  if (idx == 0)
    return;
  int parent = (idx - 1) / 2;
  if (arr[parent] > arr[idx]) {
    swap(arr[parent], arr[idx]);
  }
  min_heapify_bottom_up(arr, N, parent);
}

void build_heap(int arr[], int N) {
  for (int i = 0; i < N; i++) {
    min_heapify_bottom_up(arr, N, i);
  }
}

int main() {

  srand (time(NULL));
  int N = 10;
  int arr[N];
  for(int i = 0; i < N; i++)
    arr[i] = rand() % 32 + 1;

  for(int i = 0; i < N; i++) cout << arr[i]<<" "; cout << "\n";
  build_heap(arr, N);
  for(int i = 0; i < N; i++) cout << arr[i]<<" "; cout << "\n";
  return 0;
}
```

The time complexity of heap construction: $O(N)$

Getting minimum element (Peek)

For a min-heap, the minimum element is always the root. So getting the minimum element from a heap is a constant time operation, $O(1)$.

Popping min element

In this operation, we want to remove the root (min element) from the heap while maintaining the complete binary tree structure.

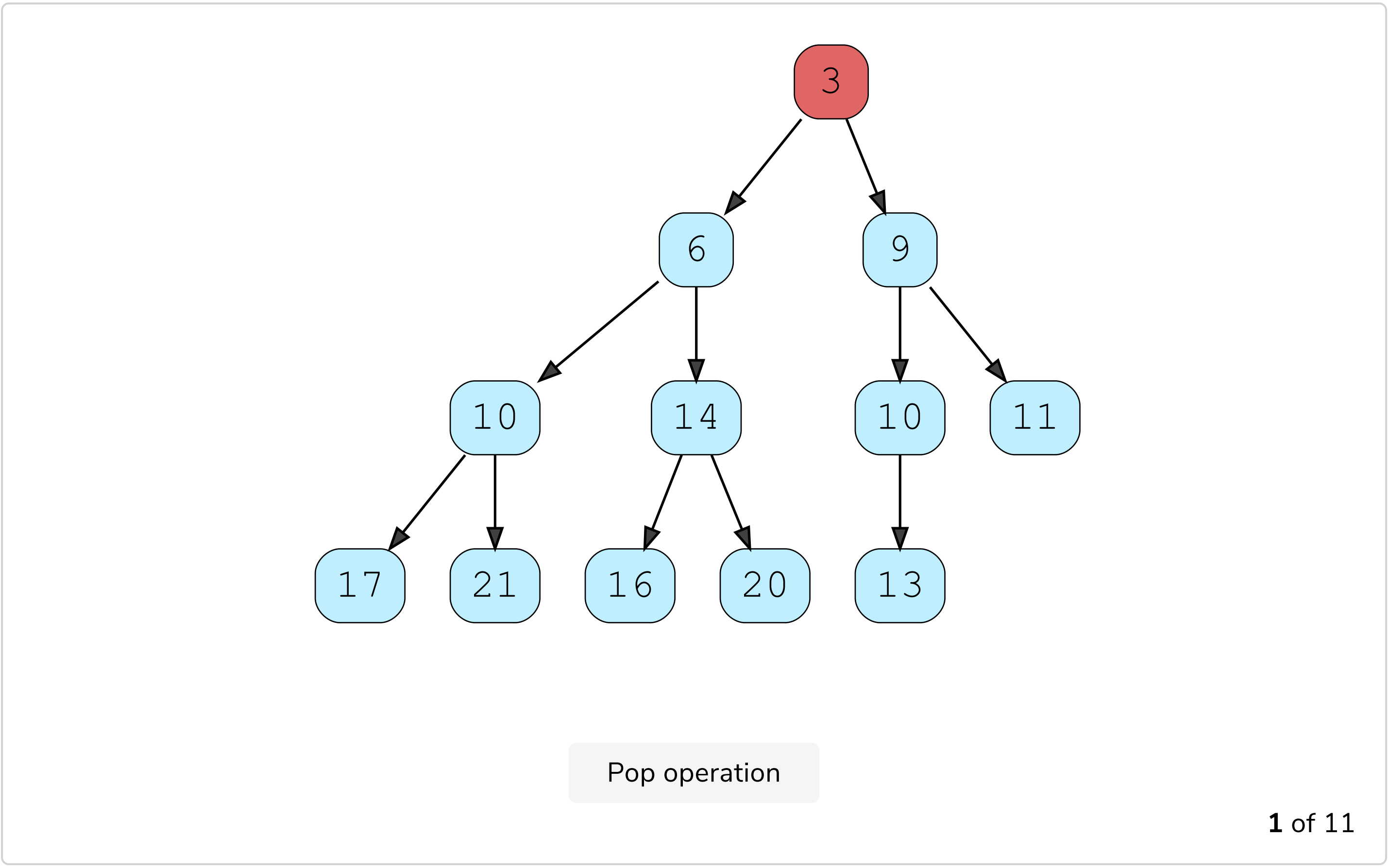
Steps

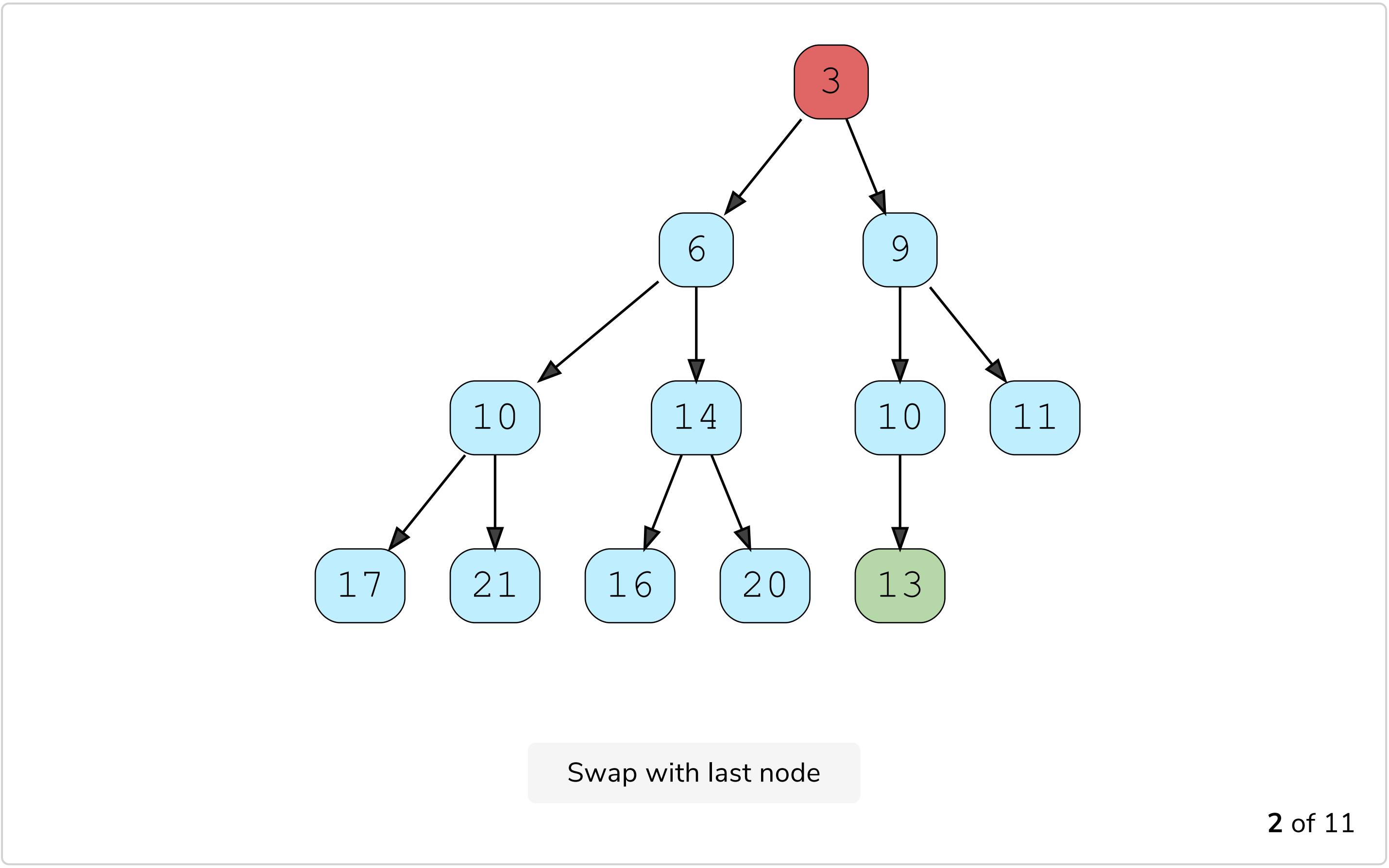
- Swap the root with the last node.
- Remove the last node.
- Min/Max Heapify from top to bottom starting from the root.

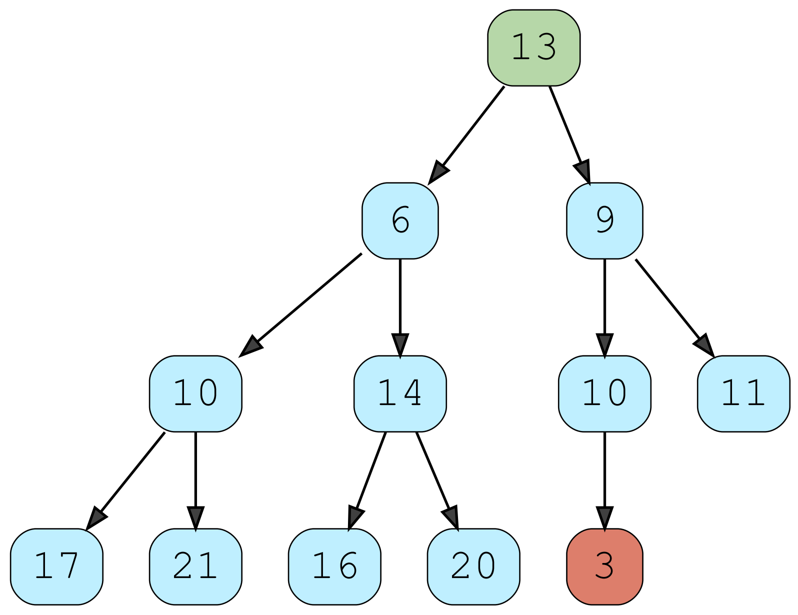
There are three cases to be handled when the root is swapped with the last node.

1. The root is smaller than both its children. In which case, the heap property is intact, and we don't have to do anything.
2. The root is greater than one of its children. In which case, swapping with this child will restore the heap property for this node. Then recursively call heapify for this child.
3. The root is greater than both of its children. In which case, swapping with the

smaller child will restore the heap property for this node. Then recursively call heapify for this child.

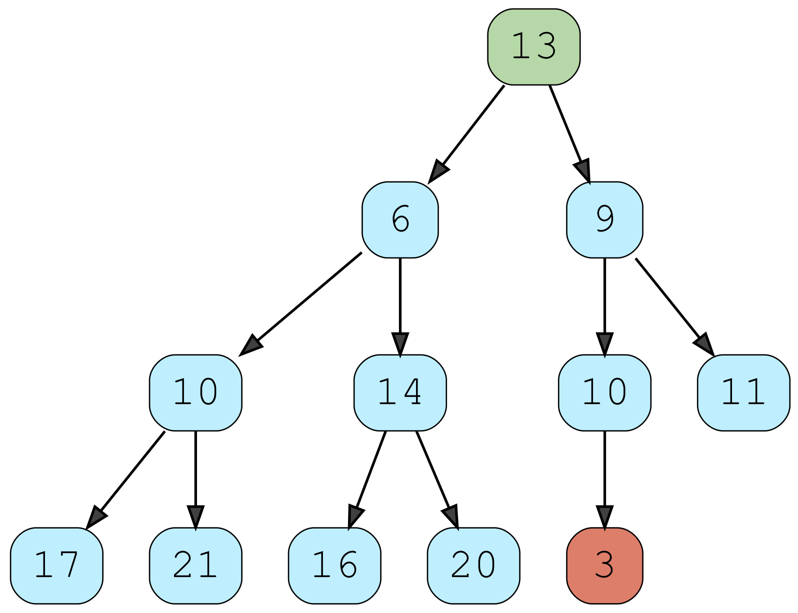






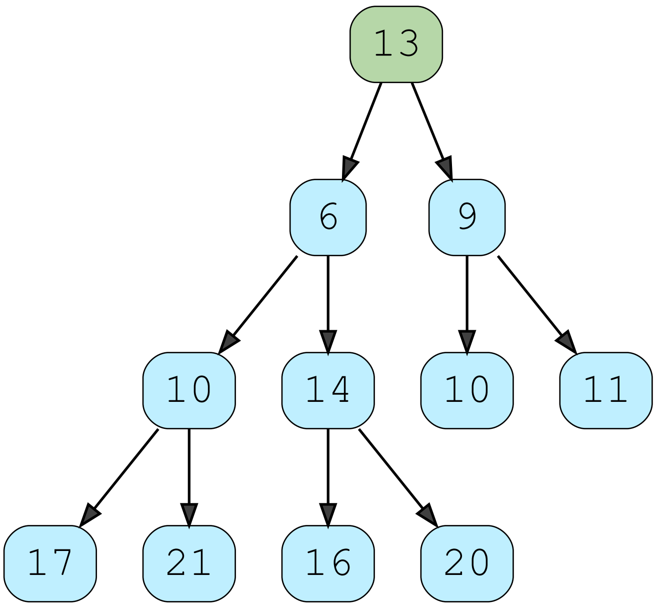
Swap with last node

3 of 11



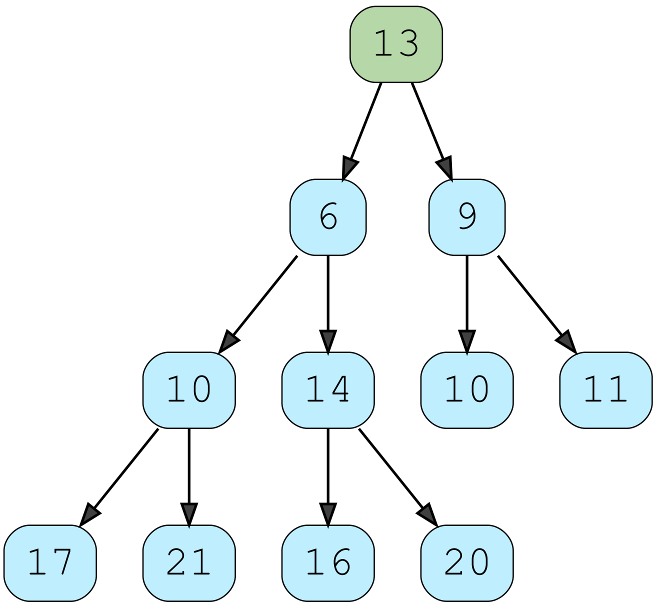
Remove last node

4 of 11



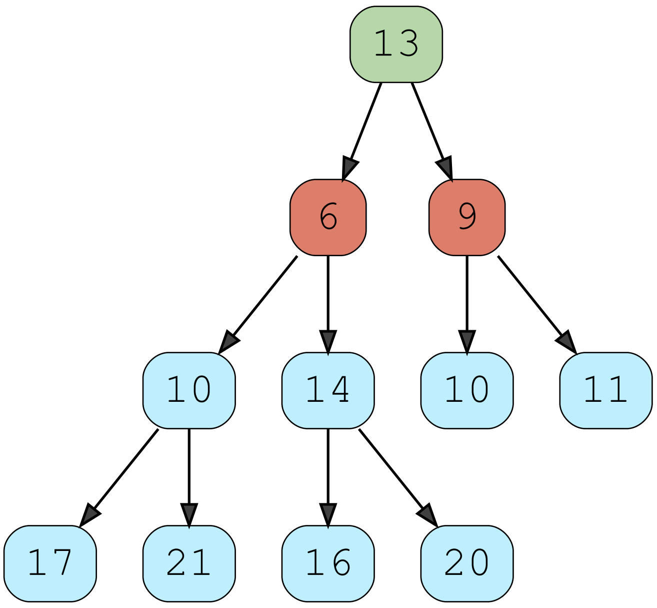
Remove last node

5 of 11



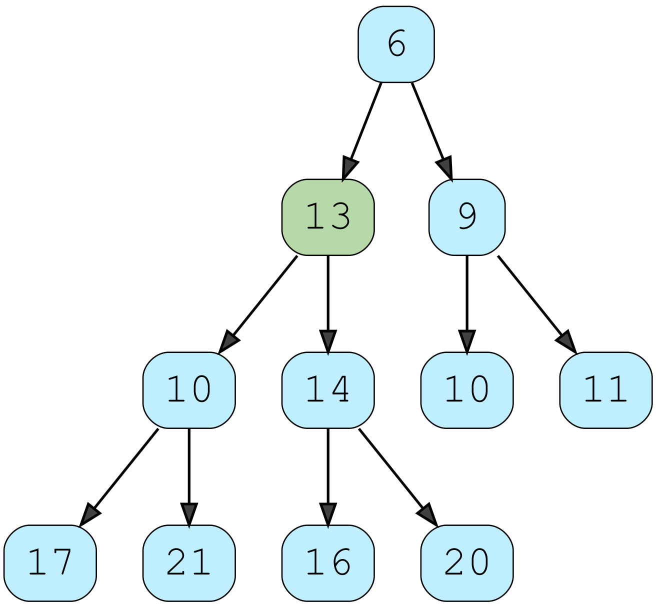
Heapify top to bottom starting from root

6 of 11

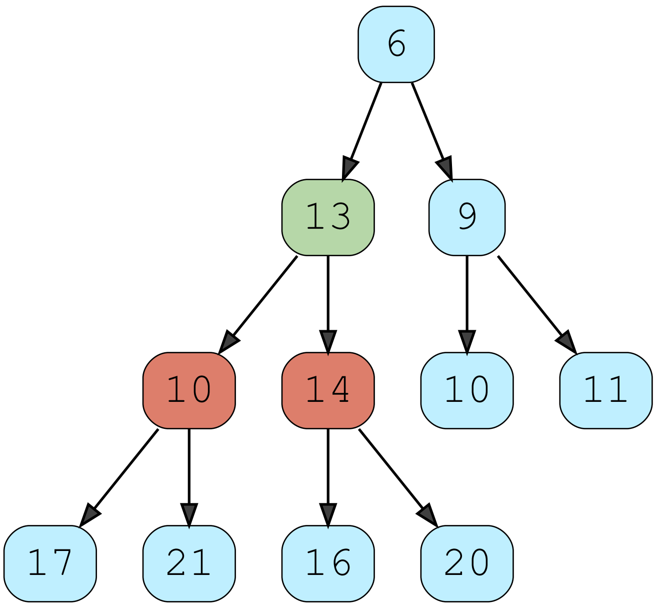


Case 3 : Swap with smaller node i.e 6

7 of 11

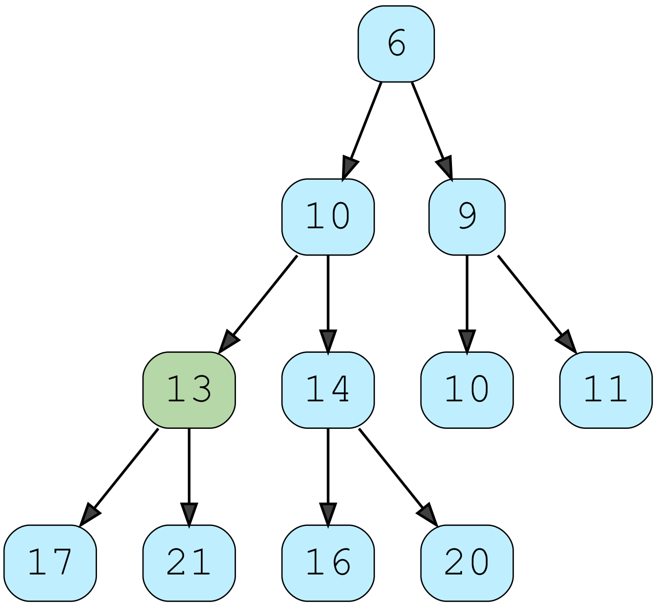


8 of 11

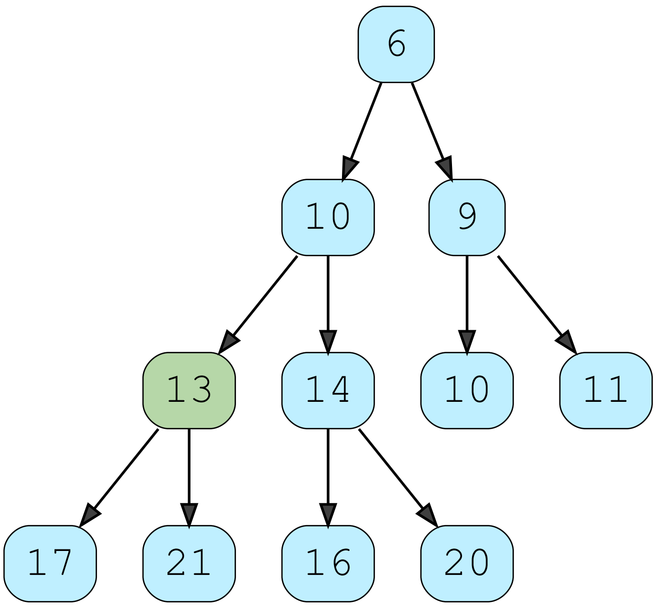


Case 2: Swap with 10

9 of 11



10 of 11



Case 1: Heap property restored

11 of 11

```
#include <bits/stdc++.h>
using namespace std;

void min_heapify(vector<int> &arr, int idx) {
  int N = arr.size();
  if (idx >= N)
    return;

  int left  = 2*idx + 1;
  int right = 2*idx + 2;
  int smallest = idx;

  if(left < N and arr[left] < arr[smallest] )
    smallest = left;
  if(right < N and arr[right] < arr[smallest] )
    smallest = right;

  if(smallest != idx) { // Case 2 and 3
    swap(arr[idx], arr[smallest]);
    min_heapify(arr, smallest);
  }
}

int pop(vector<int> &arr) {
  int root = arr[0];
  int N = arr.size();

  swap(arr[0], arr[N-1]);
  arr.pop_back();
  min_heapify(arr, 0);

  return root;

}

int main() {
  // heap
  vector<int> arr = {3, 6, 9, 10, 14, 10, 11, 17, 21, 16, 20, 13};

  cout << "Heap before popping: ";
  for (auto it : arr) cout << it << " "; cout << "\n";
  cout << "Popped integer: " << pop(arr) << "\n";
  cout << "Heap after popping: ";
  for (auto it : arr) cout << it << " "; cout << "\n";

  return 0;
}
```

In the next lesson, we'll see how to use STL priority queue for heap operations.