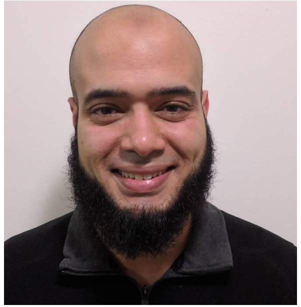
Data Structures

Heap Creation

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

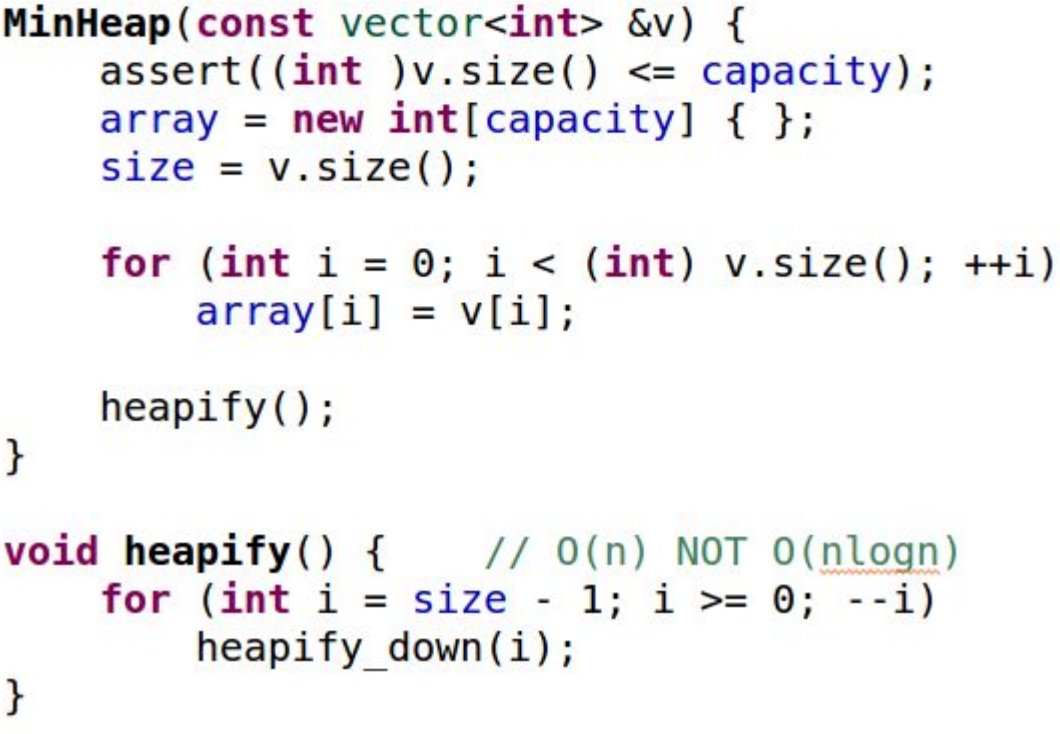


Let’s create a heap from array

- MinHeap(const vector<int> &v)
- This constructor should take this array and build a heap using it
- We know a simple way to do so:
  - Iterate on elements, keep pushing using **heapify_up**
  - Correct, but time complexity O(nlogn)
- Floyd described another simple O(n) algorithm that uses heapify_down
  - Build the tree level by level but starting from the **bottom level**
    - E.g. iterate from the last number to the first number
  - With each number’s index, just fix its sub-tree with **heapify_down**
  - This is all can/should be done **in-place**
  - Prove for the time complexity is out of our scope (estimate #comparisons)

Floyd heapfiy algorithm

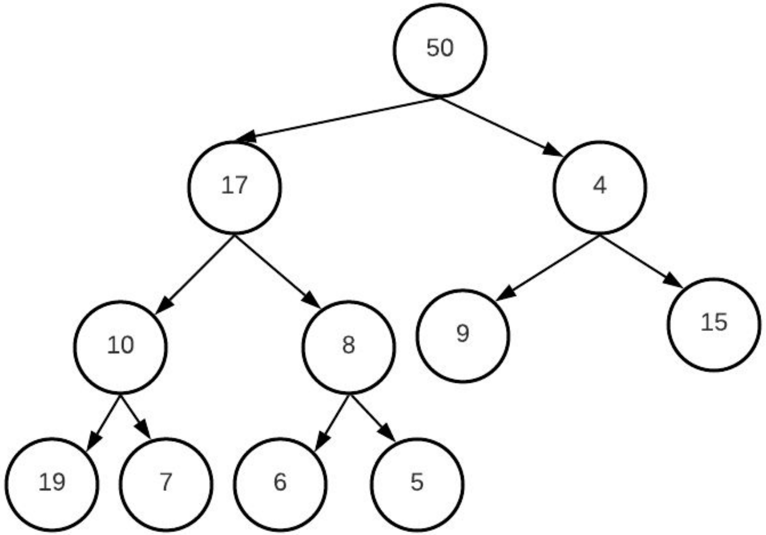
- Iterate from the end to start and keep fixing
    - This means lower sub-trees levels are always correct after each step
- Think for 10 min why correct



```
MinHeap(const vector<int> &v) {
    assert((int )v.size() <= capacity);
    array = new int[capacity] { };
    size = v.size();

    for (int i = 0; i < (int) v.size(); ++i)
        array[i] = v[i];

    heapify();
}

void heapify() {    // O(n) NOT O(nlogn)
    for (int i = size - 1; i >= 0; --i)
        heapify_down(i);
}
```

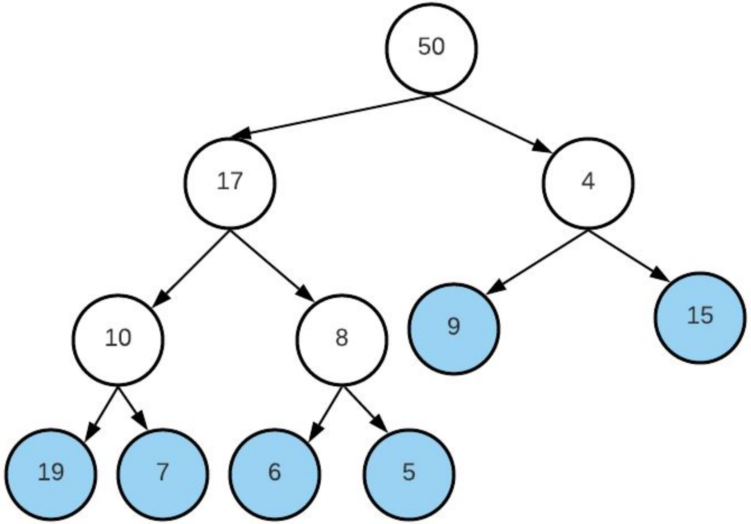
Let’s simulate

- Assume the given array as following
  - 50, 17, 4, 10, 8, 9, 15, 19, 7, 6, 5
- Using array representation, we can represent the array as binary tree
- Clearly: not binary heap



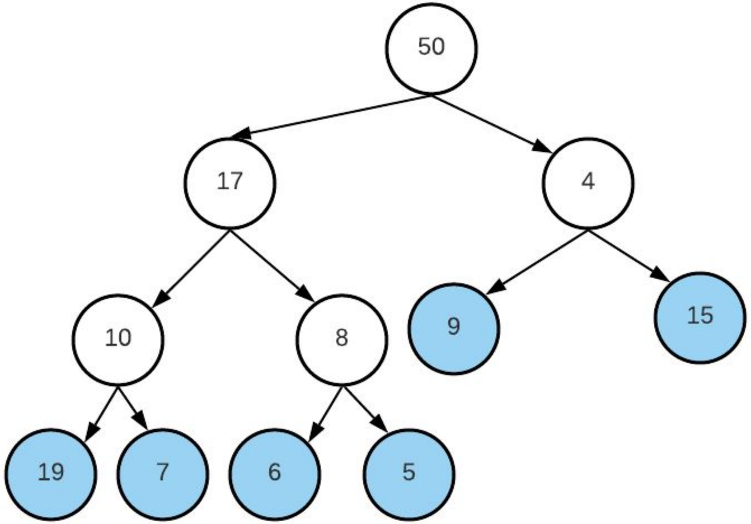
Leaf nodes are met first

- The algorithm goes backward
  - **5, 6, 7, 19, 15, 9**, 8, 10, 4, 17, 50
- The first 6 calls heapify_down(idx) do nothing, as they are leaf nodes
  - Indices: 10, 9, 8, 7, 6, 5
  - Now we have 6 scattered nodes



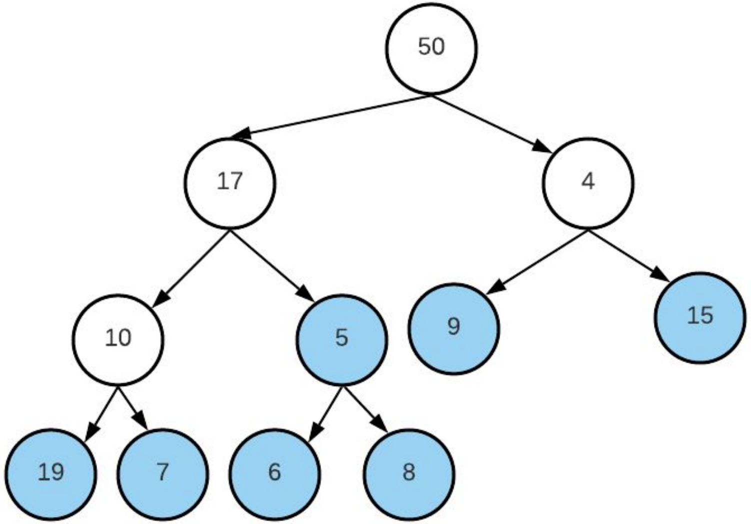
Next: value 8 - index 4

- Next position to heapify down is 4
- Parent 8 for (6, 5)
- Clearly 5 need to be swapped with 8
- 8 now is leaf, we stop



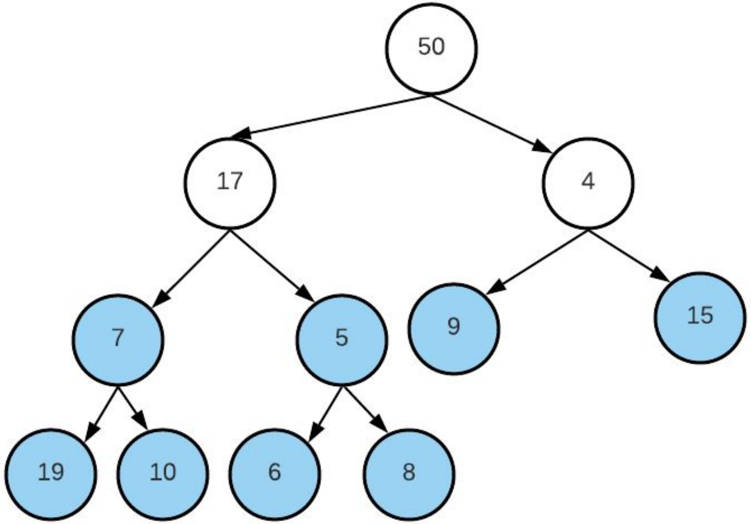
Next: value 10 - index 3

- *Observe: root (5) is a min heap*
- Next position to heapify down is 3
- Parent 10 for (19, 7)
- Clearly 10 need to be swapped with 7
- 10 now is leaf, we stop



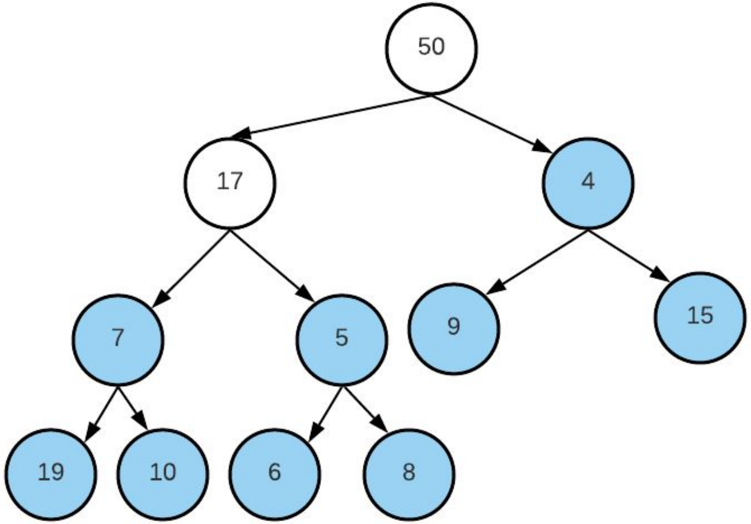
Next: value 4 - index 2

- *Observe: root (7) is a min heap*
- Next position to heapify down is 2
- Parent 4 for (9, 15)
- Perfect parent. Stop
  - In a large tree, this will happen a lot.
  - Remember, left and right subtrees are already min-heap as we are building from bottom to up



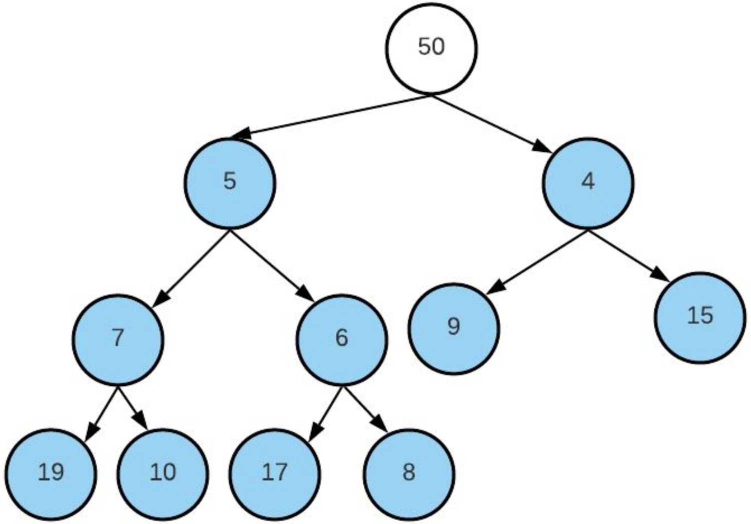
Next: value 17 - index 1

- Next position to heapify down is 1
- Parent 17 for (7, 5)
- Clearly 17 need to be swapped with 5
- Then Parent 17 for (6, 8)
- Swap 17 with 6



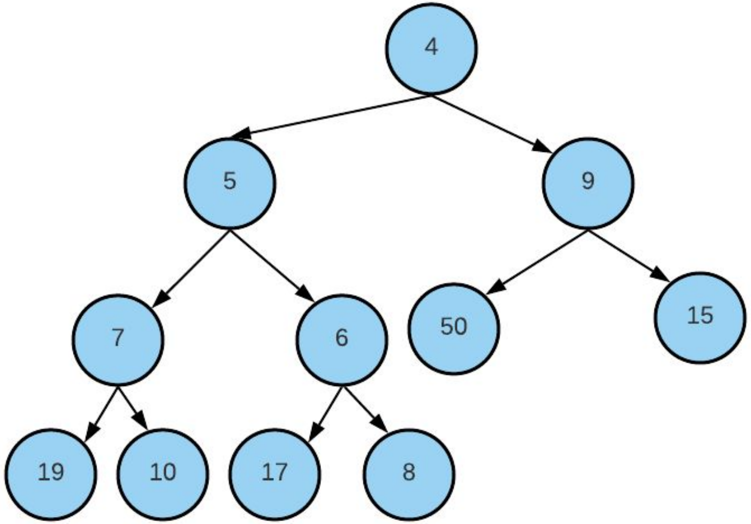
Next: value 50 - index 0

- *Observe: root (5) is a min heap*
- Next position to heapify down is 0
- Parent 50 for (5, 4)
    - Swap 50 and 4
- Parent 50 for (9, 15)
    - Swap 50 and 9
- 50 is leaf



Done

- As you see, we were building bottom-up **separate binary** trees
- With each level we go up, we create new parents for some of them
- Fix the new parent with the children subtree
- Interestingly: we were creating a tree but using a deletion procedure

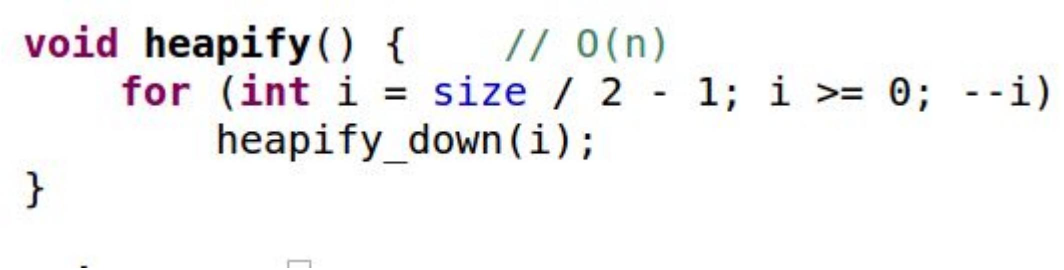


Optimization: Skipping the leaves

- We iterate on many leaf nodes.
  - By definition a leaf node has no children and hence nothing happens there
- To optimize, we can simply process **only non-leaf nodes**
- How many non-leaf nodes in complete binary tree of n nodes?
- Think for 5 minutes. Just try some trees and guess the formula

Optimization: Skipping the leaves

- How many non-leaf nodes in complete binary tree of n nodes?
  - Let’s enumerate
  - 16 ⇒ 8, 15 ⇒ 7, 14 ⇒ 7, 13 ⇒ 6, 12 ⇒ 6, 11 ⇒ 5, 10 ⇒ 5
  - Clearly floor (n/2) non-leaf nodes
- So only process backward the first n/2 nodes
  - In C++ 0-based, position is n/2 - 1 is the first non-leaf node in a complete binary tree



```
void heapify() {    // O(n)
    for (int i = size / 2 - 1; i >= 0; --i)
        heapify_down(i);
}
```

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”