Data Structures

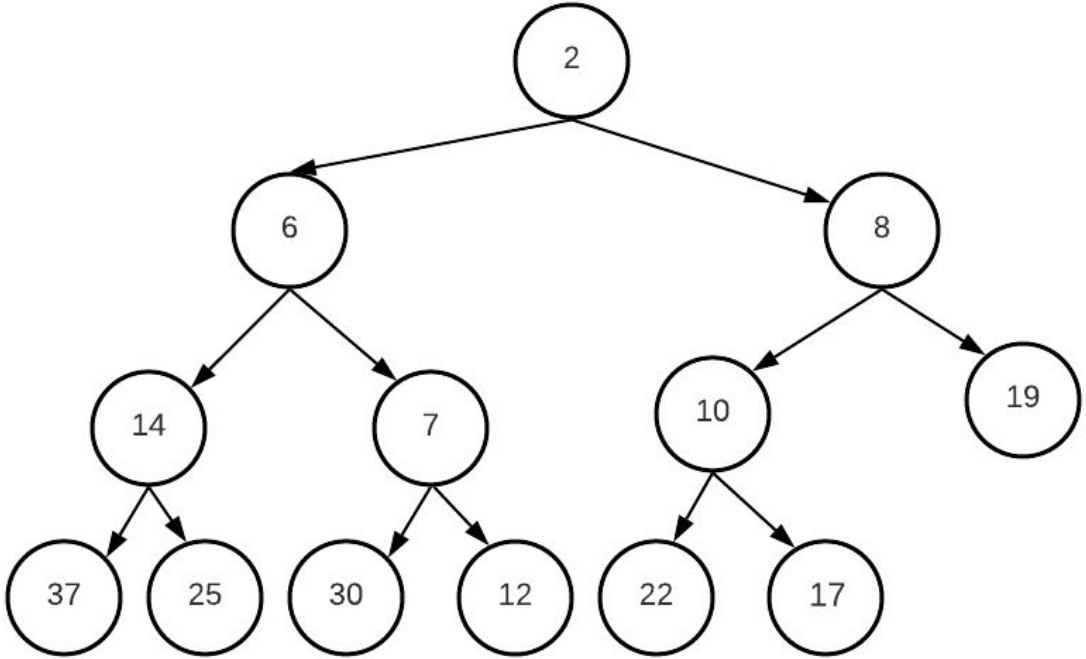
Heap Insertion

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 insert 5

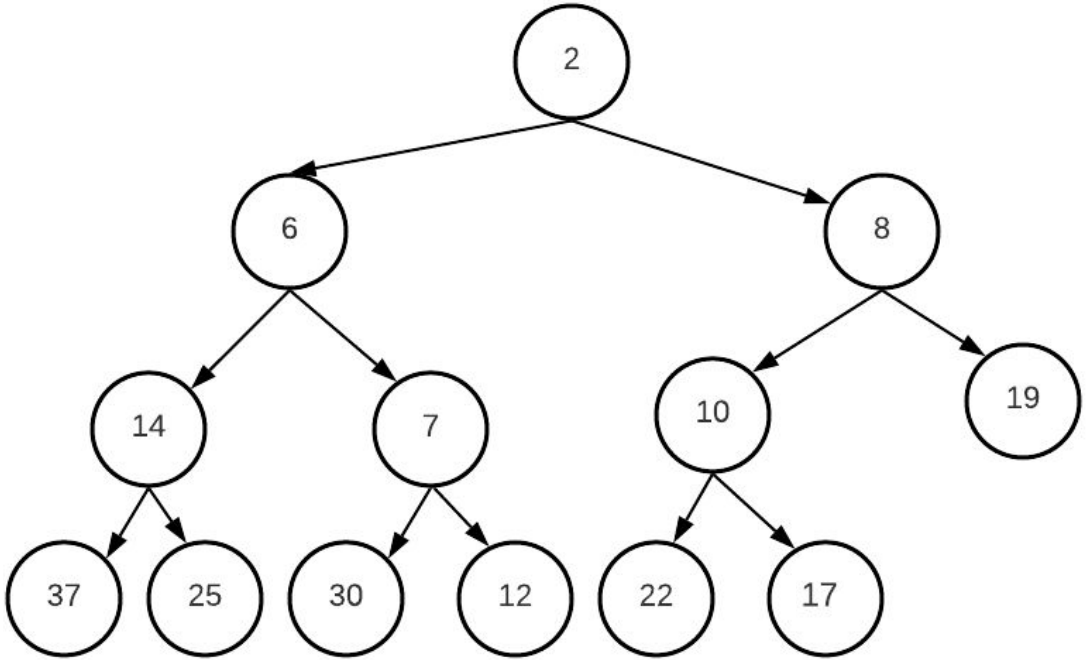
- To insert a value in heap, we depend on a trick a nice idea
- We **add** to the tree the item, and then **fix** the corruption
  - A smart approach, but sadly not widely applicable



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	6	8	14	7	10	19	37	25	30	12	22	17		

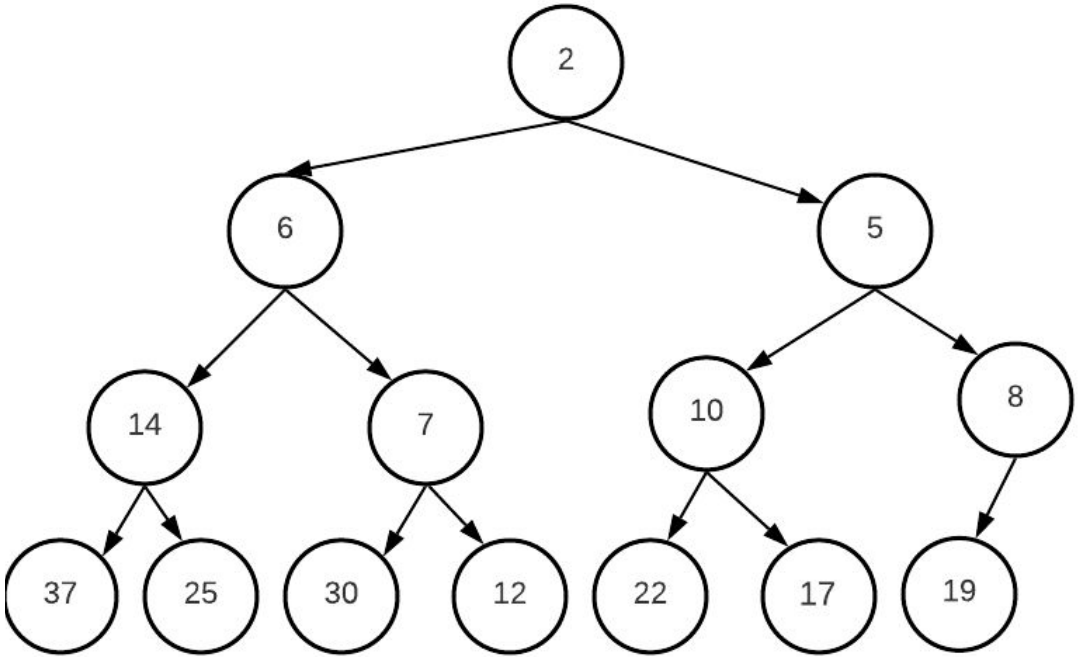
Let’s insert 5

- What is the first available node in this tree?
  - Left of 19
- Add 5
- What is the chain of its parents
  - [5, 19, 8, 2]
- Shift up 5 to be in its **right location** (decreasing seq)
  - [19, 8, 5, 2]



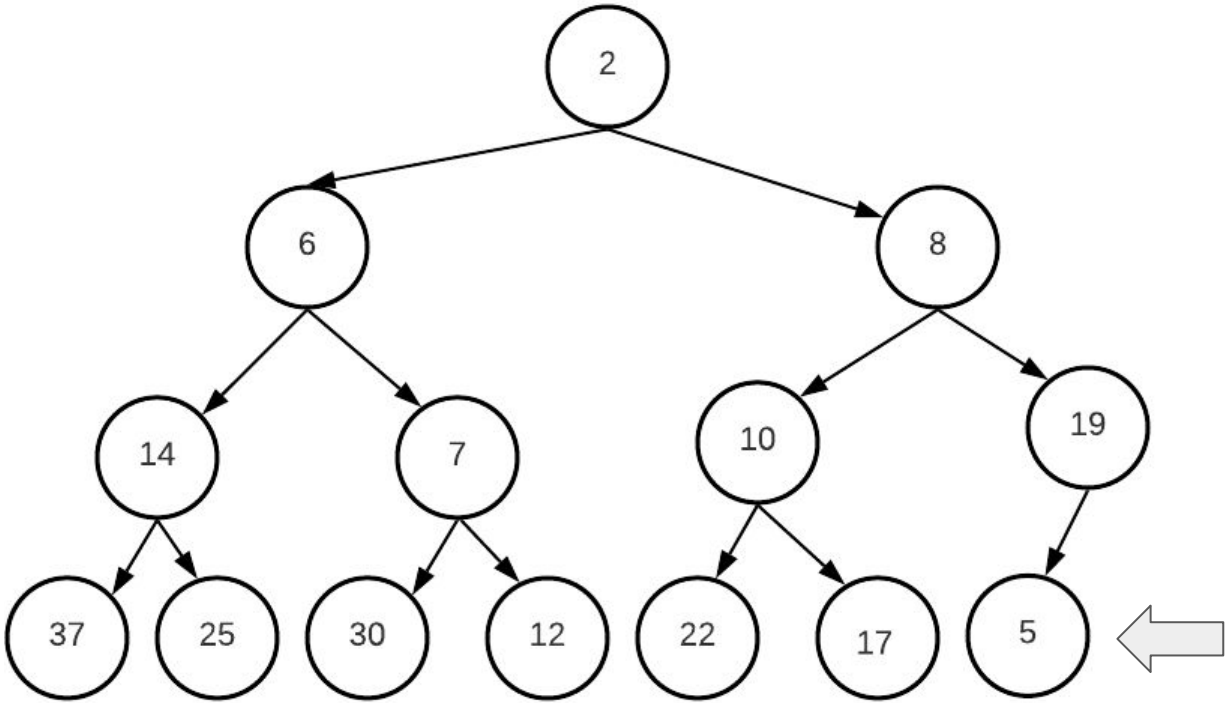
Heapify Up

- How to simply implement that?
- Start from the new added node index
- As long as its parent is greater, shift the parent to the current child node and move up
    - Observe: 19 and 8 are moved down



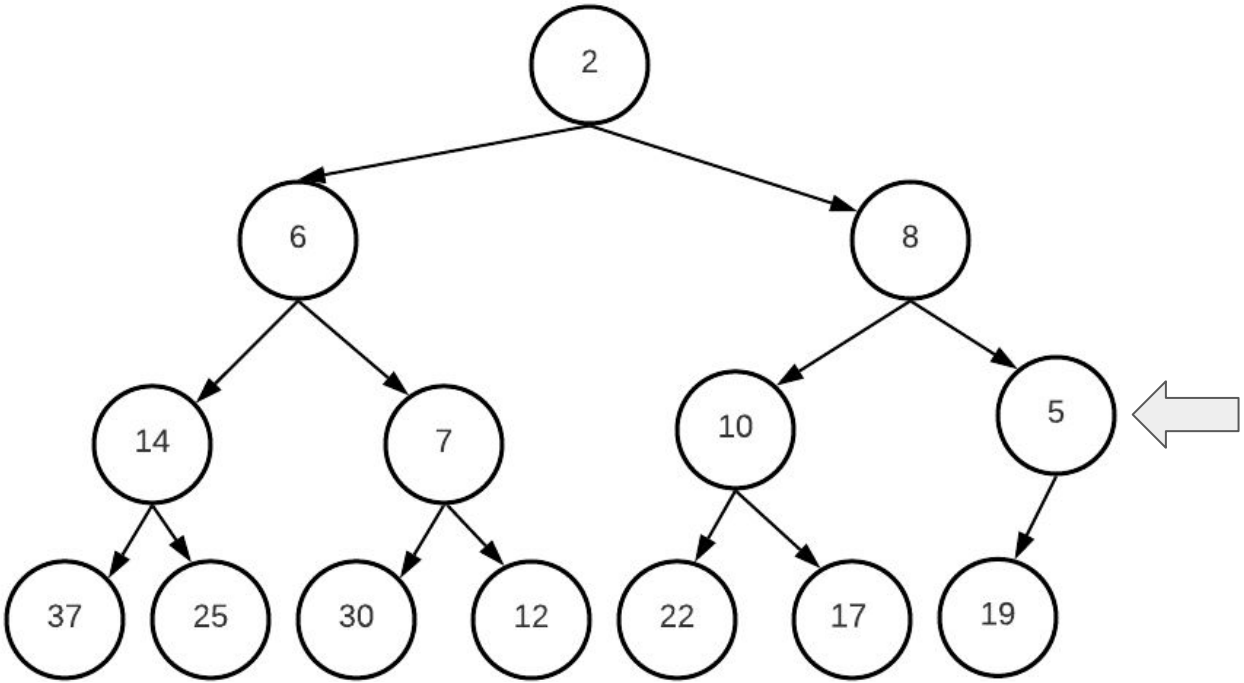
Simulation: Add 5 at available position

- 5 index = 13
  - Parent idx = 6
  - Parent value 19
- 5 < 19
  - Push 19 down



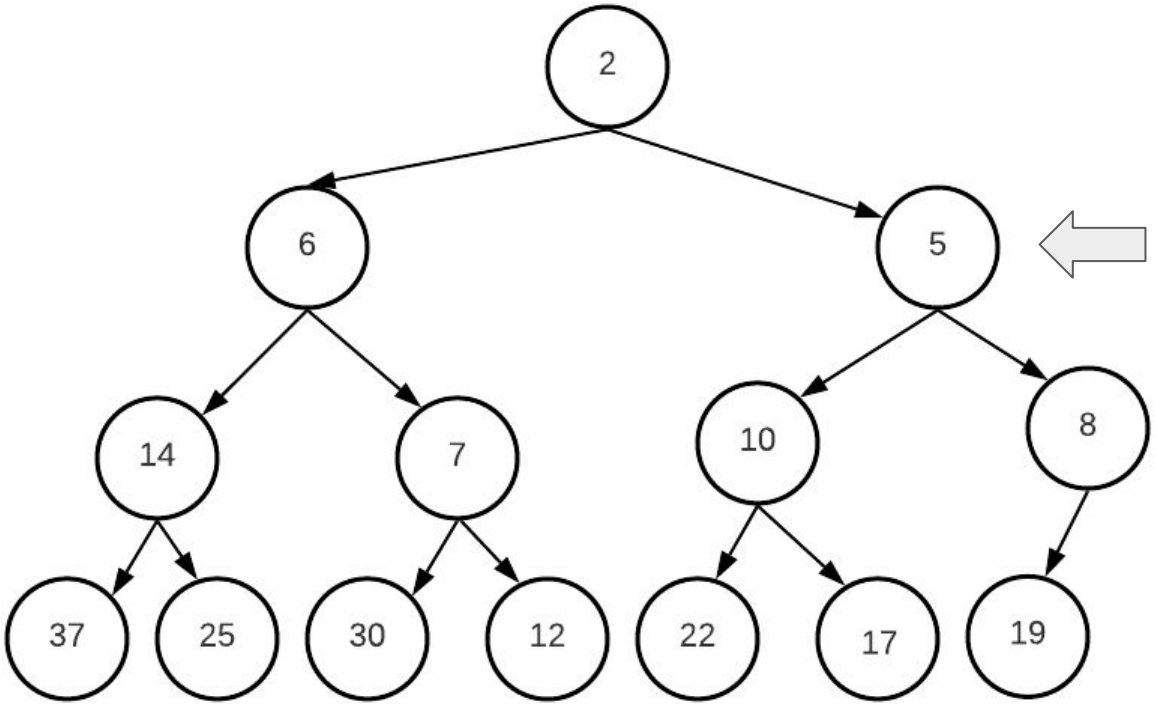
Simulation: Compare with parent

- 5 index = 6
  - Parent idx = 2
  - Parent value 8
- 5 < 8
  - Push 8 down



Simulation: Compare with parent

- 5 index = 2
  - Parent idx = 0
  - Parent value 2
- 5 > 2
  - Perfect heap
  - Stop
- Take 10 min to code it



Min Heap Class

- Let’s create a class
- Assume for simplicity some internal capacity
- left, right, parent functions added from last time

```
 6  class MinHeap {
 7  private:
 8      int *array { };
 9      int size { };
10      int capacity { 1000 };
11
12  public:
13      MinHeap() {
14          array = new int[capacity] { };
15          size = 0;
16      }
17
18      ~MinHeap() {
19          delete[] array;
20          array = nullptr;
21      }
```

Insertion Implementation

- Add element in the end of array, then refix this last position
- As tree is complete, its height is O(log(n))
- Given n elements to insert in a heap, we need O(nlog(n))

```
void heapify_up(int child_pos) {
    // stop when parent is smaller (or no parent)
    int par_pos = parent(child_pos);
    if (child_pos == 0 || array[par_pos] < array[child_pos])
        return;

    swap(array[child_pos], array[par_pos]);
    heapify_up(par_pos);
}

void push(int key) {
    assert(size + 1 <= capacity);
    array[size++] = key;
    heapify_up(size - 1);
}

int top() {
    assert(!isempty());
    return array[0];
}
```

You turn

- What if we want to remove the smallest element?
- It is the root
- How can we fix the tree? Think in a similar top-down procedure

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."