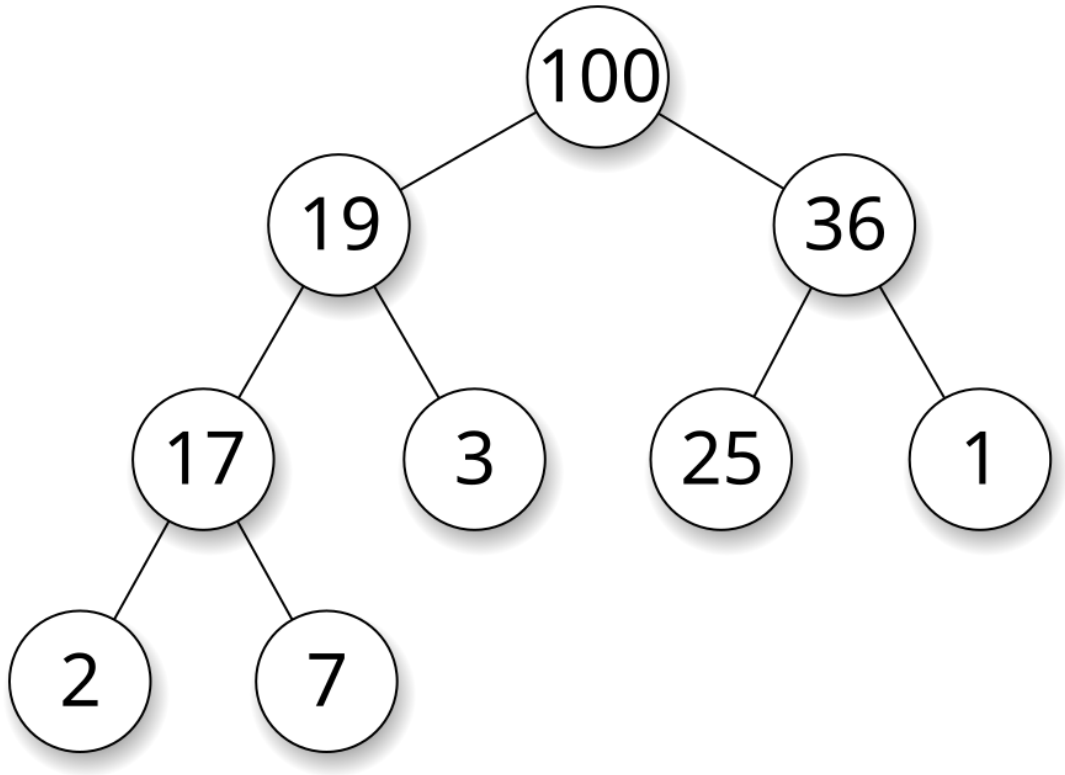
Heap

What is Heap?

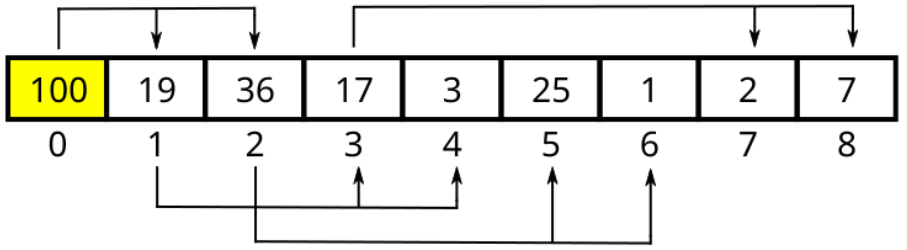
- First Look of an Heap:

- Its a special **tree** that looks like a pyramid (normally formed inside the array).

Tree representation

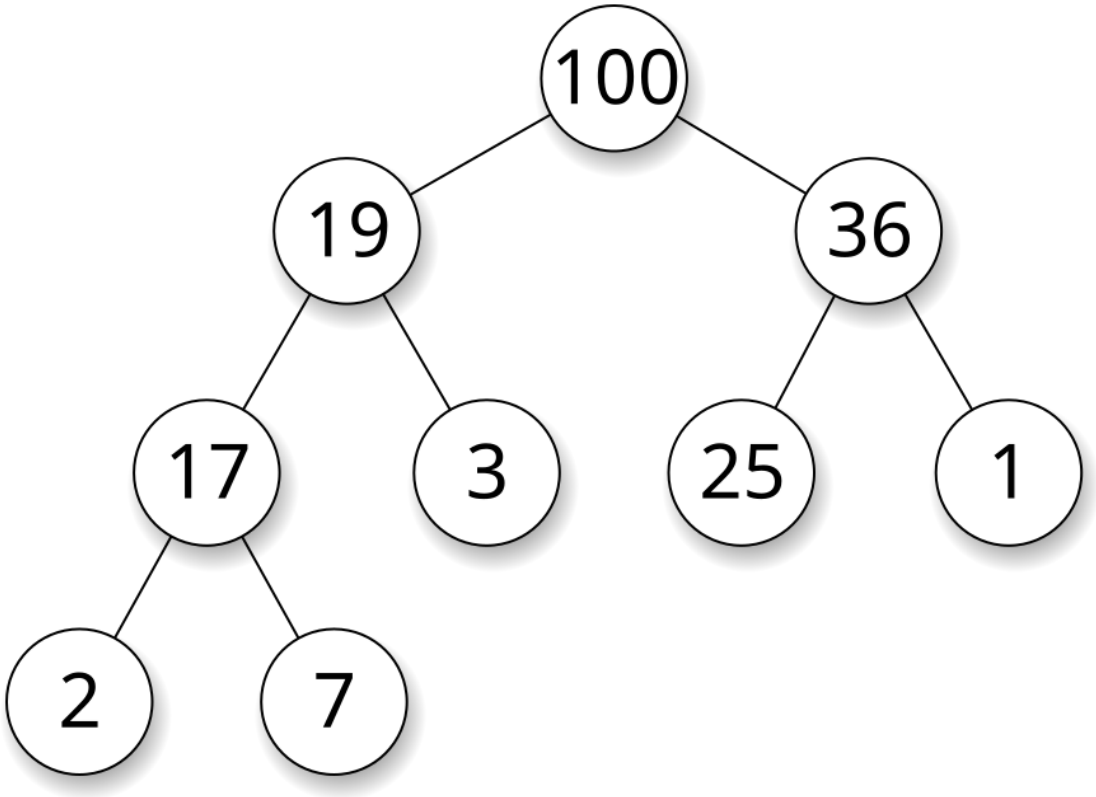


Array representation



What is Heap?

- It's a **complete binary tree**, means:
  - Every level is full (except maybe the last one).
  - Nodes fill from **left to right**.
- Heaps used to quickly find **the biggest or smallest** number!



What is Heap?

Imagine heap like a **triangle of boxes**, where:

- **top** node is → **root**.
- Each node can have **up to 2 child boxes** (left and right).
- Store this whole tree in a **simple array**.

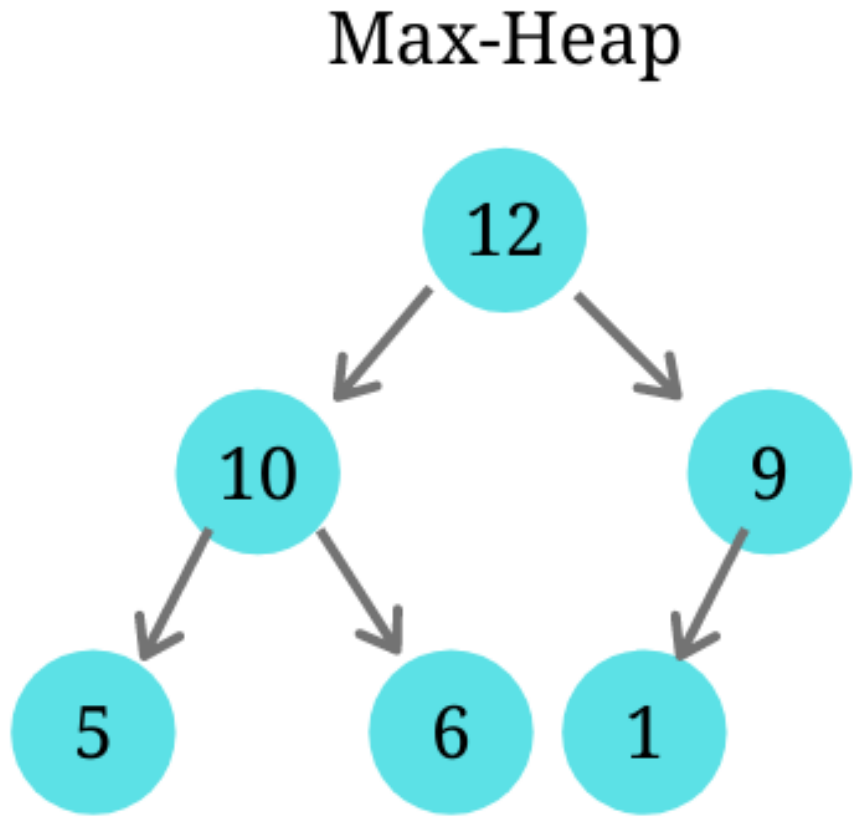
Operations of Heap

- Heapify → Make a messy array into a heap.
- Insert → Add a new number into the heap.
- Delete → Remove the top number (max or min).
- Peek → Look at the top number without removing it.

Two Types of Heaps

1. Max Heap

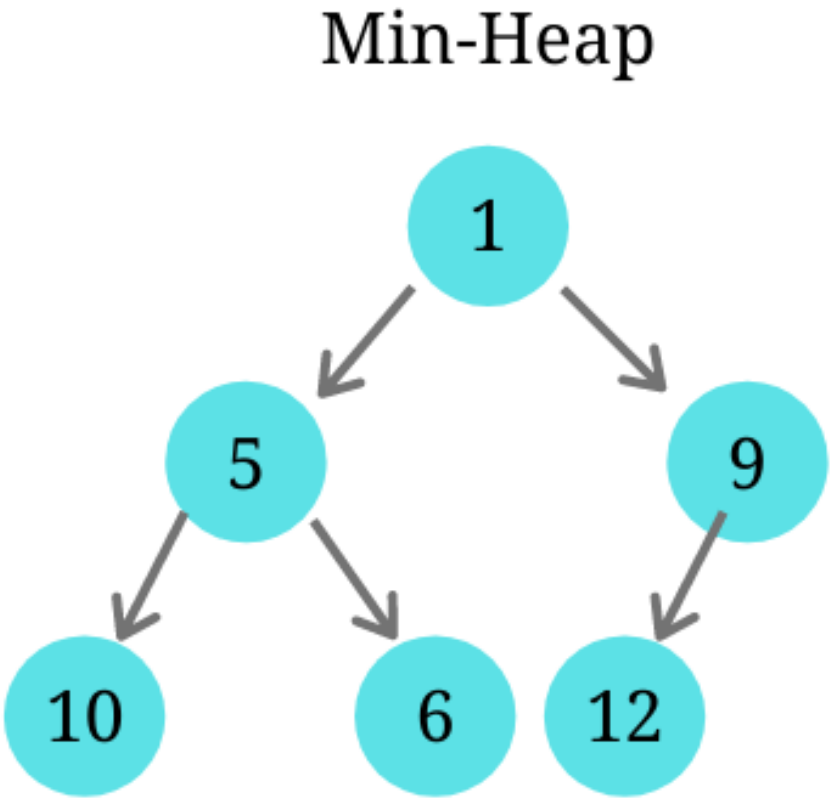
- The biggest number is always on top (root).
- Example: [50, 30, 40, 10, 20] → root = 50
- The parent is always bigger than its children.



Two Types of Heaps

2. Min Heap

- The smallest number is always on top.
- Example: [10, 30, 15, 40, 50] → root = 10
- The parent is always smaller than its children.



Heap Properties

- Must follow 2 properties
- Structural Property → Complete Tree (or Near About Complete)
- Ordering Property → Either Max (Parent is Highest) or Min (Parent is lowest)
- Take input from left to right always.
- And because of taking input always from left to right side, the rule Left < Root < Right, is not applicable here.

Heap Insertion

- Two methods:
- Insert One by one O(n log n) – linear
- Heapify O(n)

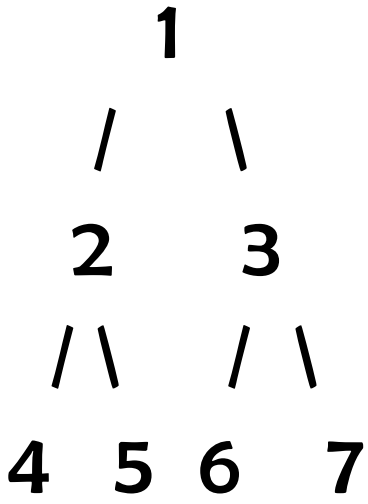
Heap Insertion

- 1st method example:
- Insert One by one O(n log n) – linear approach
- Example: 14,24,12,11,25,8,35

Heap Insertion

- 2nd method example:
- Heapify O(n)
- 145,40,25,65,12,48,18,1,100,27,7,3,45,9,30 (min heap)
- 1st → random tree (without order)
- 2nd → leaf node (half n/2 elements) needs no comparison (saves time)
- Checking start from second last level rightmost.

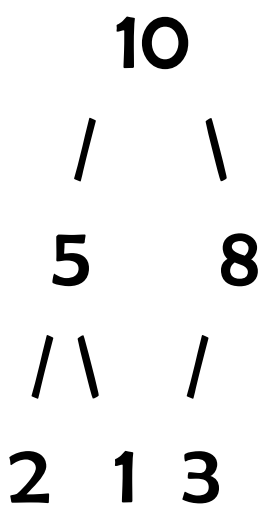
Deletion



No direct deletion is allowed (except for root node || right most element in tree), reason is if direction deletion is allowed then, property of Complete Binary Tree will be ended (min scenario).

- Take the biggest number (root).
- Move the last number to the top.
- Let it “fall” [max heap] down by swapping with the bigger child until the rules are okay.

Deletion



No direct deletion is allowed (except for root node || right most element in tree), reason is if direction deletion is allowed then, property of Complete Binary Tree will be ended.

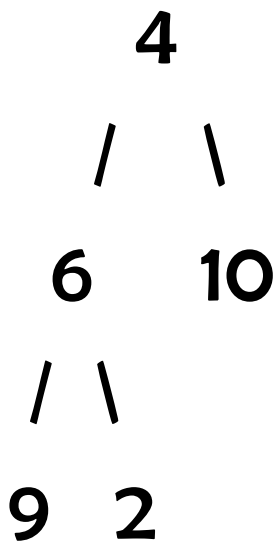
- Take the biggest number (root).
- Move the last number to the top.
- Let it “fall” [max heap] down by swapping with the bigger child until the rules are okay.

Heap

- Insertion → Bottom to top data
- Delete → Top to bottom
- For both:
- Best case → O(1) [for 1 element]
- Worst Case → O(logn) [for 1 element]

Heap sort

- Heapify (build heap) is used to sort.
- 4,6,10,9,2 (min heap) – O(n)



- After min, delete using heapify method – nlogn
- Overall O(nlogn)

Heap sort

- Heap sort is unstable because the heap-building and heap-adjusting operations can move equal elements past each other, reversing their original order.
- Heap sort is "in-place" algorithm sorts (or processes) the data without using extra memory that grows with the input size.
- In-place = uses only O(1) extra space, It may use a few temporary variables, but it does NOT create extra arrays, vectors, or large storage.

Generalize Format:

- In-place = you move books around on the same shelf, no new shelf needed.
- Not in-place = you bring a second shelf to help reorganize.

Priority Queue

- priority_queue is usually a Max-Heap.
- The largest element is always on top.

```
priority_queue<int> pq;
pq.push(10);
pq.push(5);
pq.push(20);
```

Top = 20

Priority Queue (important function)

- Pq.Push
- Pq.Top
- Pq.Pop
- Pq.Empty
- Pq.size

Priority Queue (important function)

- Pq.Push
- Pq.Top
- Pq.Pop
- Pq.Empty
- Pq.size

```
Top = 40
Now top = 20
```

```cpp
#include <iostream>
#include <queue>
using namespace std;

int main() {
    priority_queue<int> pq;

    pq.push(10);
    pq.push(40);
    pq.push(20);

    cout << "Top = " << pq.top() << endl;  // 40

    pq.pop();   // removes 40

    cout << "Now top = " << pq.top() << endl; // 20
}
```

Priority Queue

- By default priority_queue gives max.
- To make a min-heap:
- priority_queue<int, vector<int>, greater<int>> minpq;

```
minpq.push(10);
minpq.push(40);
minpq.push(5);
cout << minpq.top(); // 5
```