

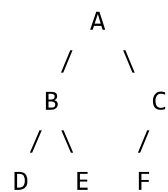
Heap Data Structure

- Treelike structure
- Represented in an array

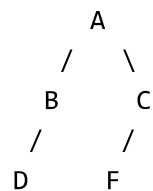
In []:

In []:

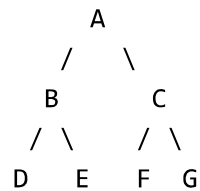
Heap is a Complete Binary tree
Filled L->R



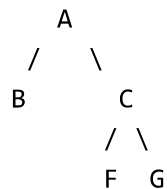
Yes



NO



Yes

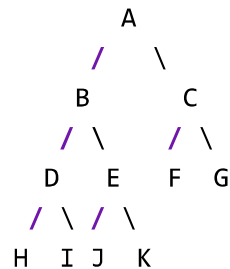


No

In []:

Array representation of complete binary tree

Child and parent



Data	A	B	C	D	E	F	G	H	I	J	K
Index	0	1	2	3	4	5	6	7	8	9	10
ParentIndex	0	0	1	1	2	2	3	3	4	4	

$$\text{leftChild} = \text{parentIndex} * 2 + 1$$

$$\text{rightChild} = \text{parentIndex} * 2 + 2$$

$$\text{parentIndex} = \text{floor}((\text{childIndex} - 1) / 2)$$

$$\text{floor}(1.5) \Rightarrow 1$$

$$\text{floor}(1.9) \Rightarrow 1$$

$$\text{floor}(1.1) \Rightarrow 1$$

$$\text{floor}(1) \Rightarrow 1$$

$$\text{floor}(0.9) \Rightarrow 0$$

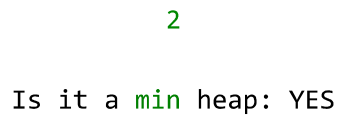
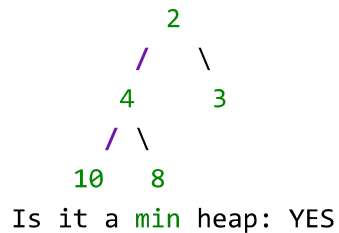
In []:

In []:

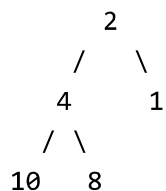
Heap property

Binary tree in which the root node is always less than(min heap) or equal to the child nodes.

Above property is true for all sub trees of the complete binary tree.



Building heap



Data	2	4	1	10	8
Index	0	1	2	3	4

Operations:

- Heapify (siftUp/Down repeatedly)
- GetMax/Min
- Del max/min
- Add new key

If you need remove a random key by value, it's an $O(N)$ operation since finding index of an element in a heap is $O(n)$. This can be reduced by keeping additional mapping of value to indexes

Internal Ops:

- SiftUp
- Sift Down

To add a new key: Append and then do siftUp

To delete max/min: Swap with last element and then do siftDown

In []:


```
In [1]: class MaxHeap {

    std::vector<int> data;

    // O(N)
    MaxHeap(std::vector<int> data) {
        this->data = data;
        buildMaxHeap();
    }

    int leftChild(int pos) {
        return pos*2 + 1;
    }

    int rightChild(int pos) {
        return pos*2 + 2;
    }

    int parent(int pos) {
        return floor( (pos - 1)/2 );
    }

    // log N
    void siftUp(int pos);

    // log N
    void siftDown(int pos) {
        if (pos < 0 || pos > data.size() - 1) {
            return;
        }

        int largestPos = pos;

        leftIdx = leftChild(pos)
        if (leftIdx < data.size() && data[largestPos] < data[leftIdx]) {
            largestPos = leftIdx;
        }

        rightIdx = rightChild(pos)
        if (rightIdx < data.size() && data[largestPos] < data[rightIdx]) {
            largestPos = rightIdx;
        }

        if (pos != largestPos) {
            swap(pos, largestPos);
            siftDown(largestPos);
        }
    }

    // O(N)
    void buildMaxHeap() {
        for (i=data.size()/2; i >= 0; i--) {
            siftDown(i);
        }
    }

    int getMax() {
```

```

        if (data.size() == 0) {
            throw "Empty"
        }
        return data[0];
    }

    // log (N)
    int removeMax() {
        if (data.size() == 0) {
            throw "Empty"
        }
        if (data.size() == 1) {
            int res = data[0];
            data.pop_back();

            return res;
        }

        int res = data[0];
        swap(0, data.size() - 1);
        data.pop_back();
        siftDown(0);

        return res;
    }

    // log N
    void insert(int key) {
        data.push_back(key);
        siftUp(data.size() - 1);
    }
};

```

File "C:\Users\LEANGA~1\AppData\Local\Temp\ipykernel_2304\2386475699.py", line 1

```

class MaxHeap {
    ^

```

SyntaxError: invalid syntax

When to use what

Static Data

- ordering: sorting
- min/max: heap/sorting

Dynamic Data

- ordering: bst
- min/max: heap/bst

In []:

Heap sort

- buildMaxHeap(data)
- repeat N times
 - removeMax(): max element is at the beginning of array -> move it to the end

Why heap sort is not good ? comparisons and swaps

In []:

<https://leetcode.com/problems/kth-largest-element-in-an-array/>
[\(https://leetcode.com/problems/kth-largest-element-in-an-array/\)](https://leetcode.com/problems/kth-largest-element-in-an-array/)

In []:

```
# SortingSolution
# TC: O(N Log N)
# SC: O(1)

class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        nums.sort()
        return nums[-k]
```

In []:

```
# TC : O(N Log k)
# SC: O(K)
import heapq

class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        h = []

        for num in nums: # O(N)
            if len(h) >= k:
                if h[0] < num:
                    heapq.heappop(h) # Log K
                    heapq.heappush(h, num)
            else:
                heapq.heappush(h, num) # Log K

        return h[0]
```