HEAP DATA STRUCTURE AND HEAP SORT

1. Heap data structure

Definition 1. Heap is an array that visualized as a binary tree.

Remark 2. There are some interesting relations between index and nodes. Which is embedded in the tree structure.

- For any node i, it's parent is $i/2$.
- If i is even, it's left child, if i is odd, it's the right child, excluding the root node.

Now let's introduce max-heap property, which is the central idea of heap sort.

Definition 3. Max-heap property. For each node, it's value is large or equal to value of it's children.

Based on this property, there are mainly two operations of heap structure:

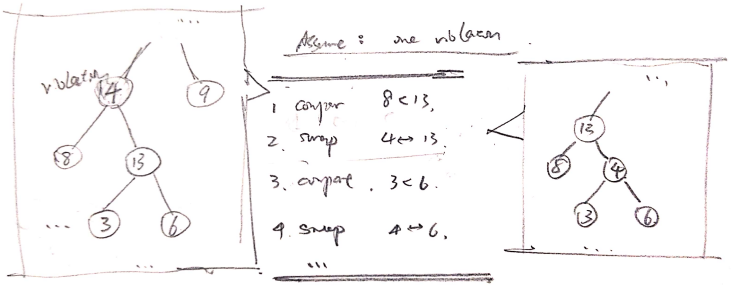
- To correct a single violation of max-heap-property, which called **max_heapify**.
- To produce a new heap structure with max-heap-property, which is **build_max_heap**.

2. Correcting a violation

To make problem narrower, we tackle with a single violation. Assumption: No violation of it's children's nodes. Notice that this assumption is important to guide us to build the whole structure.

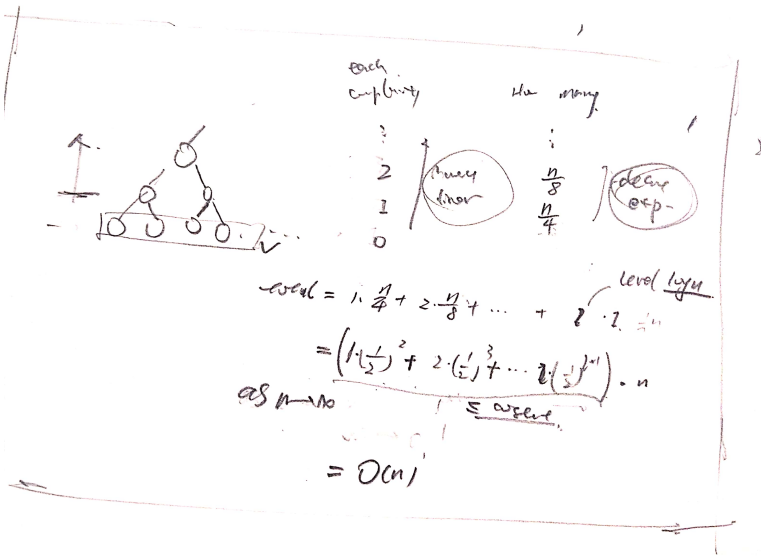
Here is the intuition: we focused node has smaller value than 'the correct answer' in its position. So we just push it downside where it belongs. Each time we also need to compare it's two children to decide which one to replace. The informal answer for the complexity is $O(log(n))$, for tree structure has $log(n)$ levels.

Now let's open the box to take a close look at it.



3. Build max_heap

Build max-heap can be solve by correcting violations. To do that, we simply correct violation from bottom to top. The informal answer for complexity is $O(nlog(n))$, which need to be inspected later.



4. Heap sort

Now, we can first build max-heap, then easily return a sorted array, this procedure is called heap sort.

(1) Build max_heap. $O(n)$
(2) Find the max(root), swap with the end node.
(3) discard that node.
(4) fix the violation. $O(log(n))$
(5) Loop step 2 - 4.

The most expensive step is step 4: $O(log(n))$, which loops n times, so the complexity is $O(nlog(n))$.