HEAPSORT

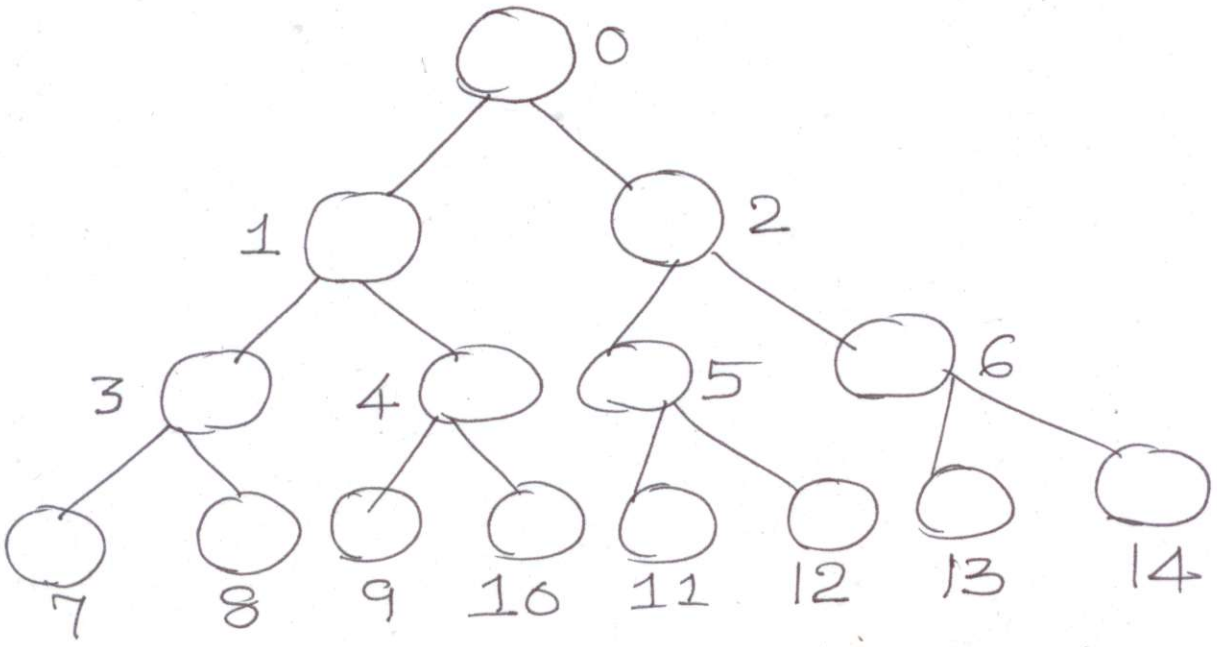
Heapsort is an extremely efficient, binary tree-based sorting method with a worst case complexity of $O(n \log n)$. It was developed by Robert Floyd and J.W.J. Williams in 1964.

A heap is an interesting data structure. It is a labelled binary tree which satisfies two properties:

(1) It is a complete binary tree (this is a structural property)

(2) The label on every node is less than (or equal to) the label on its left child and the label on its right child.

first, we focus on the structural property of a heap, namely that it is a complete binary tree. To understand the notion of a complete binary tree, let us first look at so called "level-by-level" numbering of nodes of a binary tree.

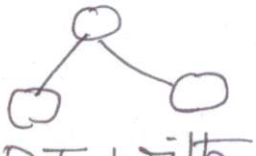


The above binary tree has 15 nodes and the nodes are numbered level-by-level and left-to-right.

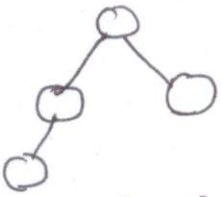
A complete binary tree with n nodes is a binary tree containing nodes $0, 1, 2, \ldots, n-1$ corresponding to level-by-level numbering. We now write down complete binary trees having 1 node, 2 nodes, etc.

CBT with 1 node

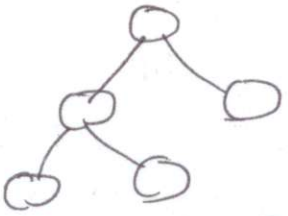
CBT with 2 nodes



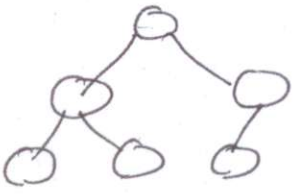
CBT with 3 nodes



CBT with 4 nodes

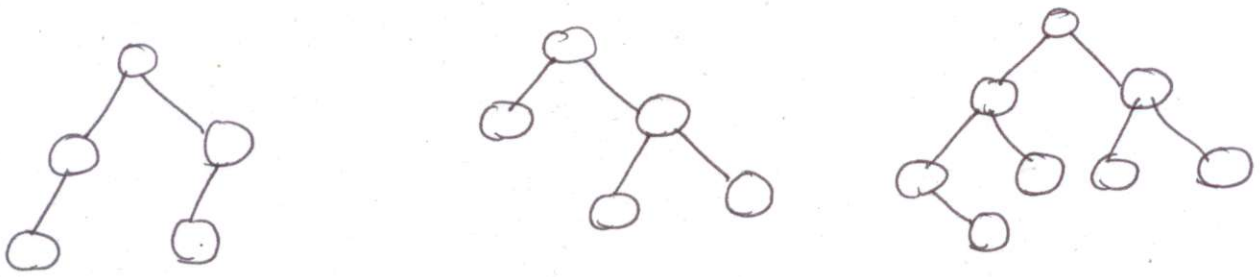


CBT with 5 nodes



CBT with 6 nodes

The following are not complete binary trees.



A complete binary tree with $2^k - 1$ nodes is called a <u>full binary tree</u>. for $k = 1, 2, 3, \ldots$

A complete binary tree with n nodes can be naturally represented using an array $A[0], A[1], \ldots, A[n-1]$.

Note that

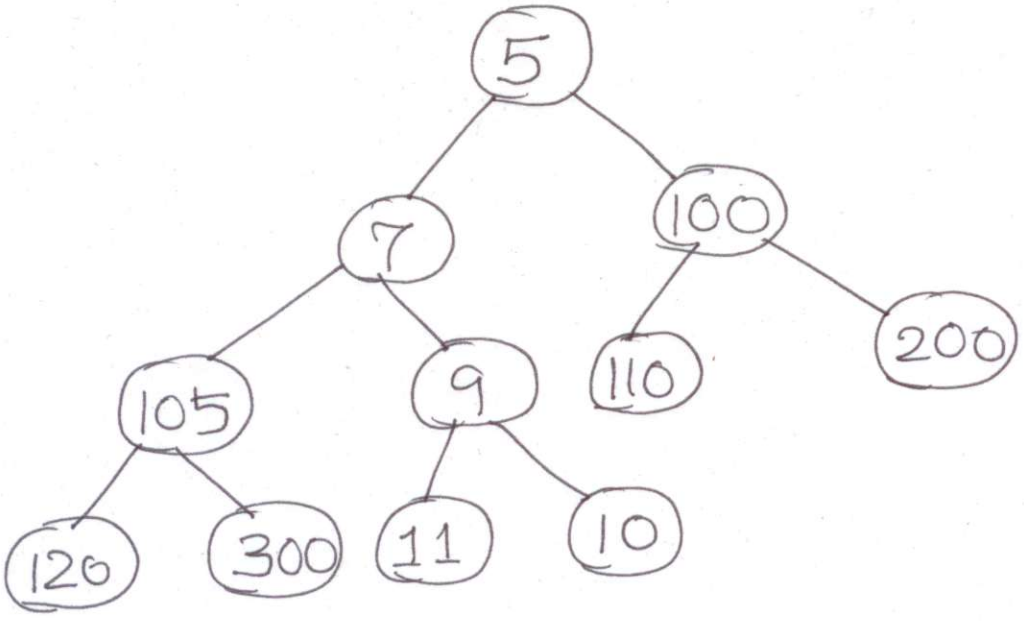
- the left child of node with index i will be $i+1$ and the right child will be $i+2$.
- Also the index of the parent of any node with index i $(\neq 0)$ will be

$$\left\lfloor \frac{(i-1)}{2} \right\rfloor$$

A major advantage of complete binary tree structure is that it is nearly balanced. In fact, a complete binary tree with n nodes has height

$$\lfloor \log_2 n \rfloor$$

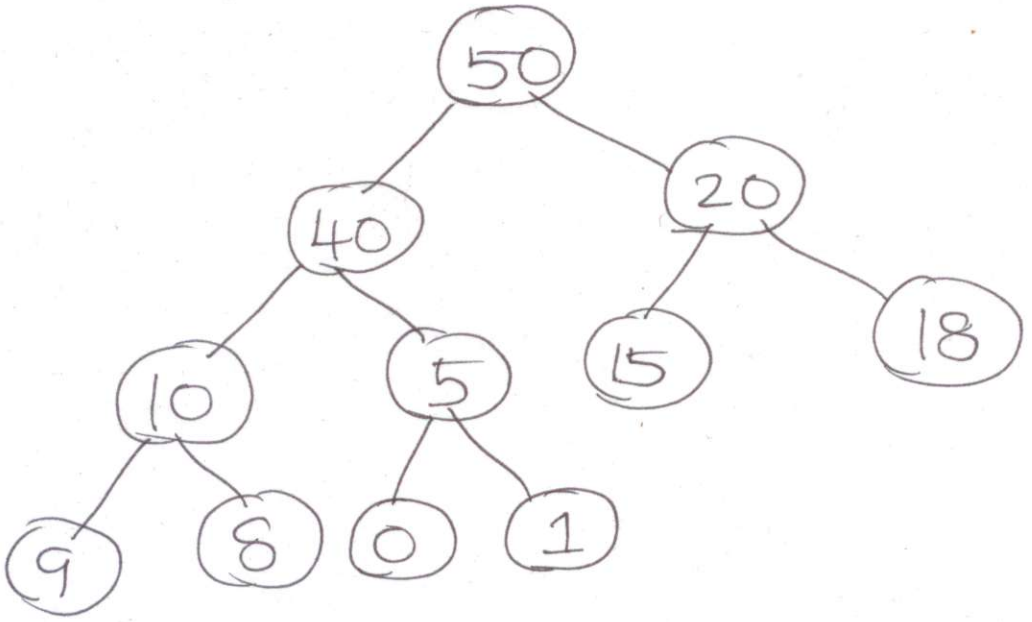
We now give an example of a heap (by including the labels inside the circles representing the nodes).



We can immediately observe that the minimum element is always present at the root. Also note that the maximum element is not that easy to find. It could be present at any of the leaves.

Max-Heap

What we have studied so far is a min-heap. A max-heap satisfies the complementary property: it is a complete binary tree in which the label at every node is greater than (or equal to) the label of its left child and the label of its right child. Here is an example:



Naturally, the maximum element will be at the root and the minimum element will be at one of the leaves.

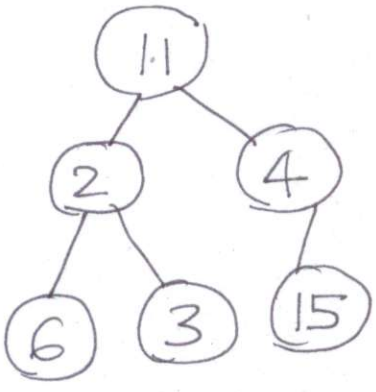
- To sort an array in descending order we use a min-heap and to sort in ~~desc~~ ascending order, we use a max-heap.

From now on, our discussion will focus on min-heap only.

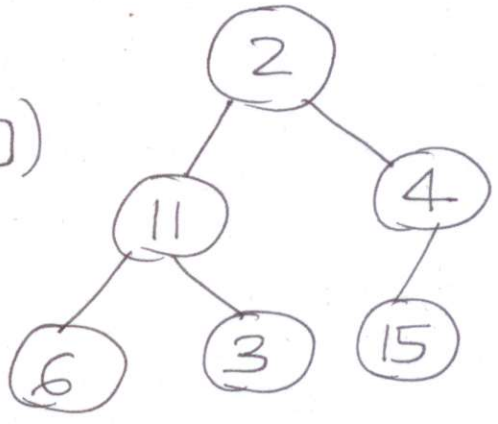
An Important operation : pushdown(i,j)

The pushdown(i,j) operation assumes that the heap property is satisfied by all elements A[i+1], ..., A[j] and that the heap property may be violated by A[i]. This operation restores the heap property by appropriately pushing down A[i]. An example follows.

pushdown(0,5)

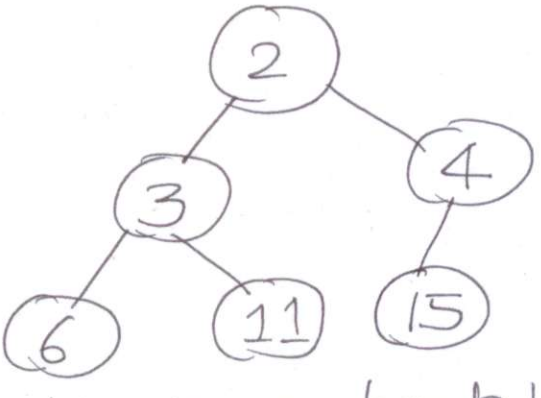


Swap A[0] with min(A[1], A[2])



Swap A[1] with min(A[3], A[4])

Worst case complexity of pushdown operation is $O(\log n)$



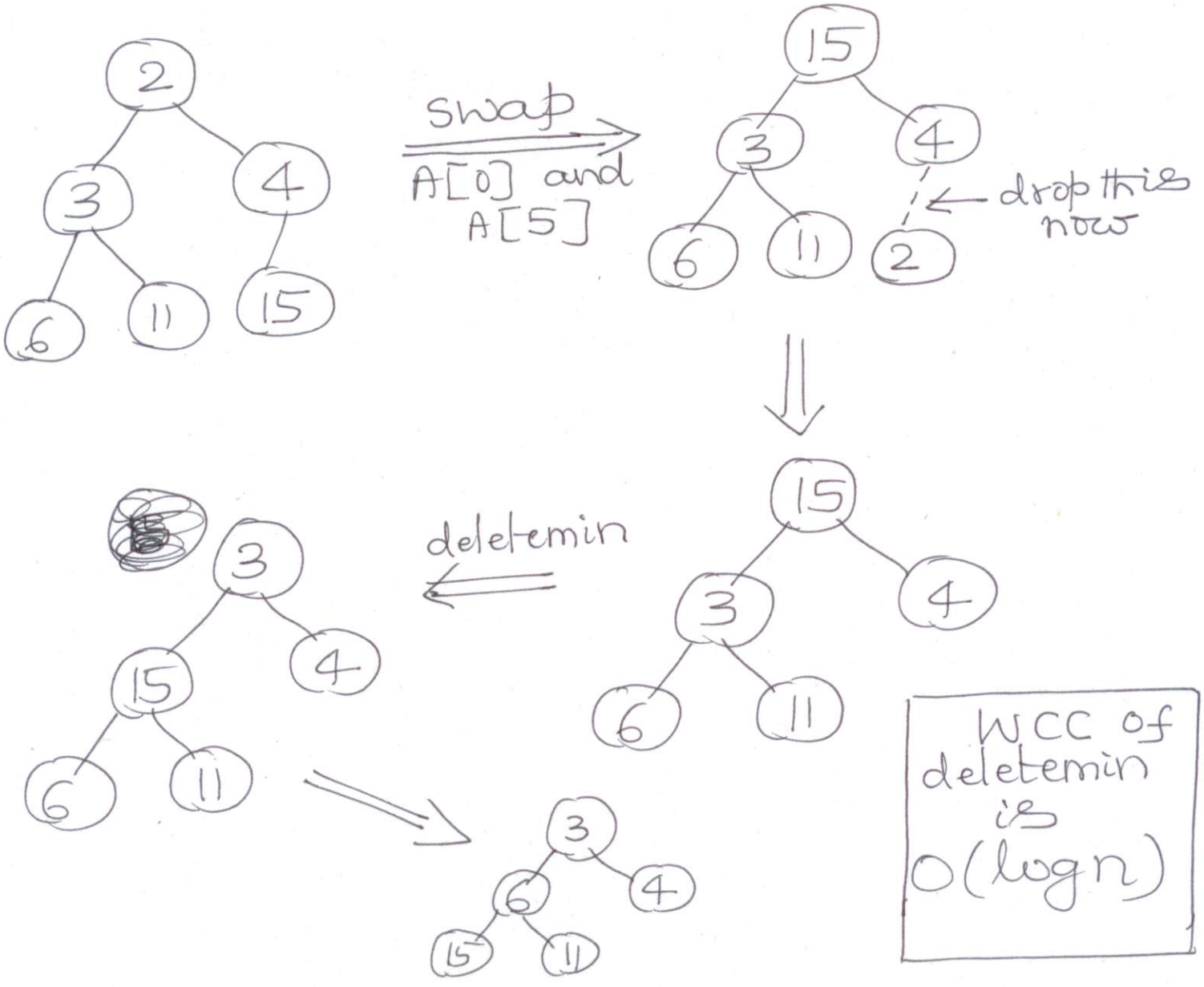
This is a heap!

Deletemin operation

This deletes the minimum element (which is present at the root) in the following way, from a heap A[0], ..., A[n-1].

1. Swap A[0] and A[n-1]
2. Delete (ignore) A[n-1]
3. pushdown (0, n-2)

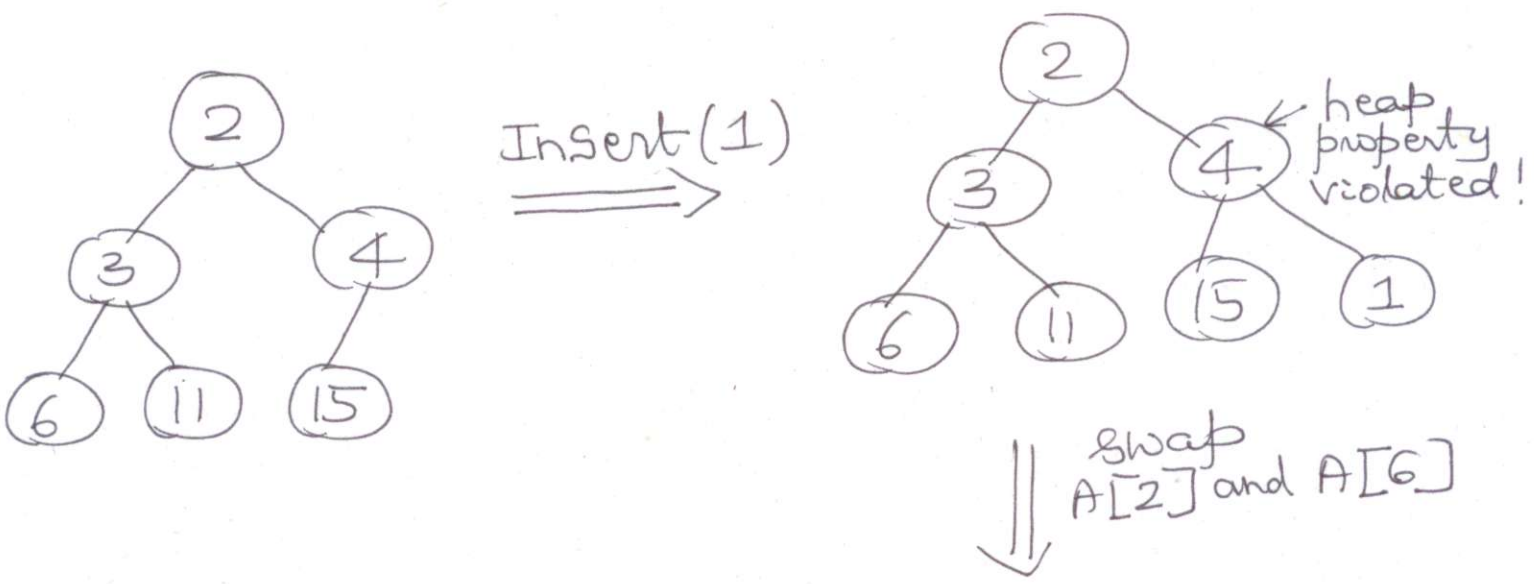
An example is provided below.



Insert operation

This is complementary to the deletemin operation.

Here, the element to be inserted is made the next element beyond the last element in the heap and then is pushed up to an appropriate position.

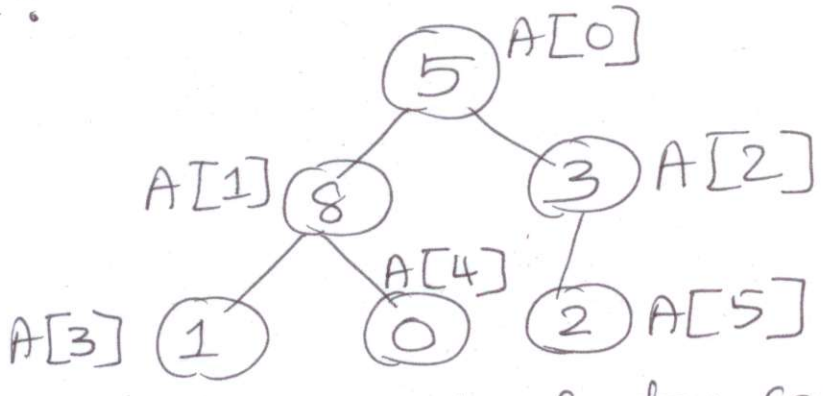


worst case complexity of insert is $O(\log n)$

Constructing a heap out of random elements

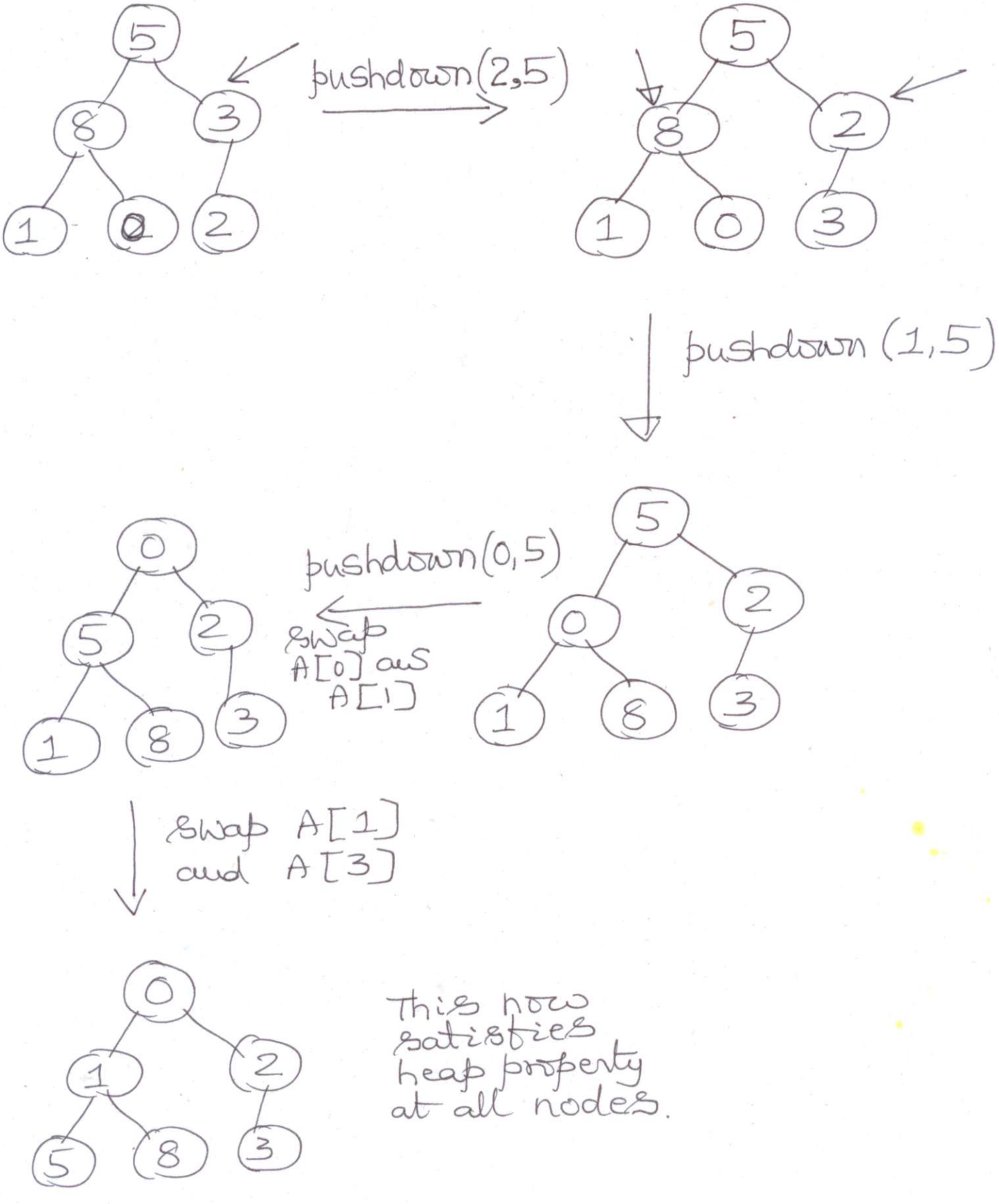
Given n random elements, we first take the elements as A[0], A[1], ..., A[n−1]. Obviously it will not form a heap. A heap can be formed by repeated use of pushdown at all positions where the heap property could be violated.

For example, given 5, 8, 3, 1, 0, 2, we get the following complete binary tree:



Note that the heap property cannot be violated at the leaf level (since leaves do not have children). We now execute pushdown(2,5), pushdown(1,5), and finally pushdown(0,5) in that order.

Interestingly, it can be shown that a heap can be constructed out of n random elements in worst case O(n) time


5
8
3
1
0
2
pushdown(2,5)
5
8
2
1
0
3
pushdown (1,5)
0
5
2
1
8
3
pushdown(0,5)
swap
A[0] as
A[1]
5
0
2
1
8
3
swap A[1]
and A[3]
0
1
2
5
8
3
This now
satisfies
heap property
at all nodes.

Heap_Sort

Given: $A[0], \ldots, A[n-1]$

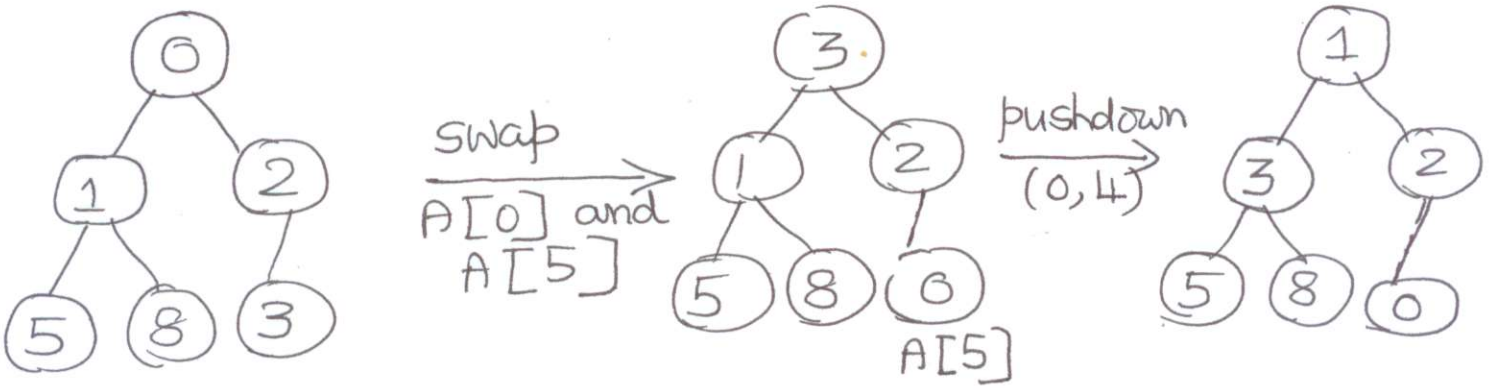
1. construct an initial heap out of $A[0], \ldots, A[n-1]$. This takes $O(n)$ time in the worstcase

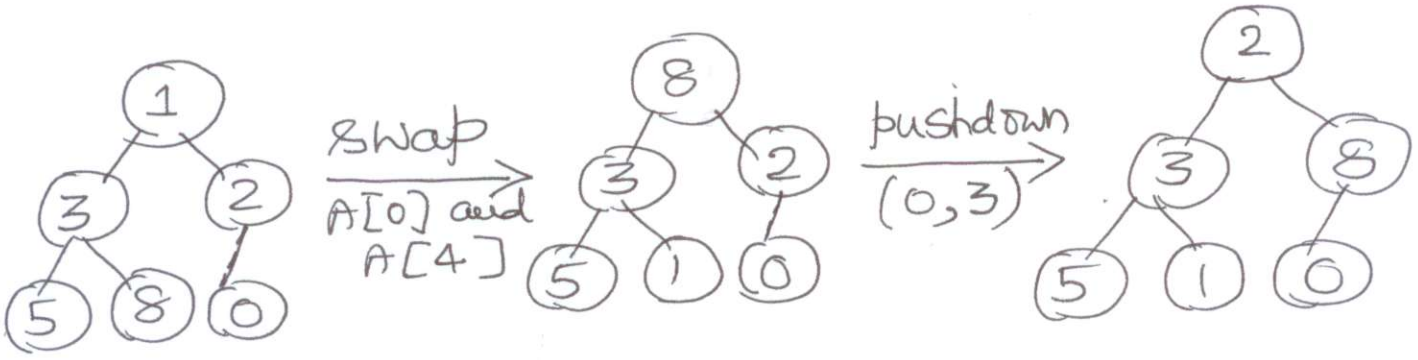
2. Repeat the following for $i = n-1, n-2, n-3, \ldots, 2$
   - swap $A[0]$ with $A[i]$
   - pushdown $(0, i-1)$

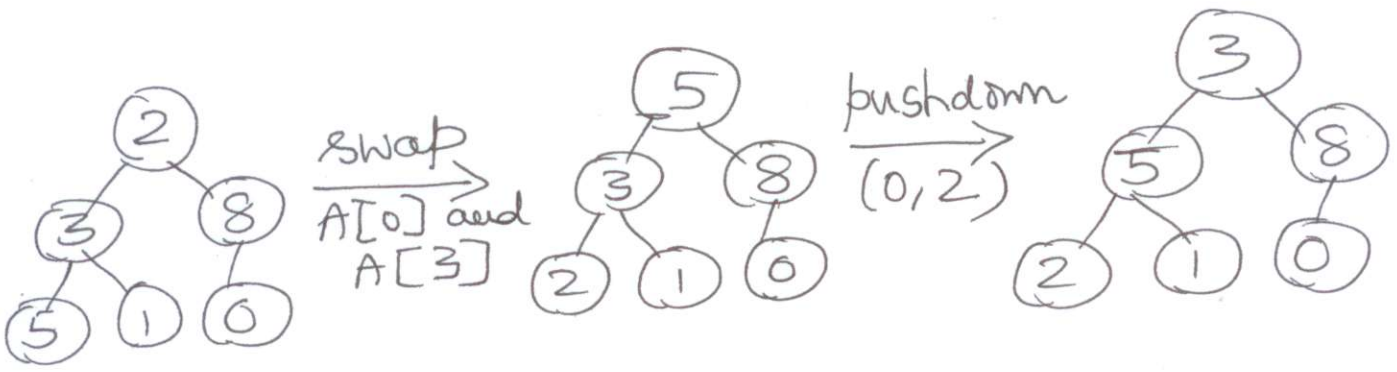
The above will ensure that the output $A[0], A[1], \ldots, A[n-1]$ will be in descending order.

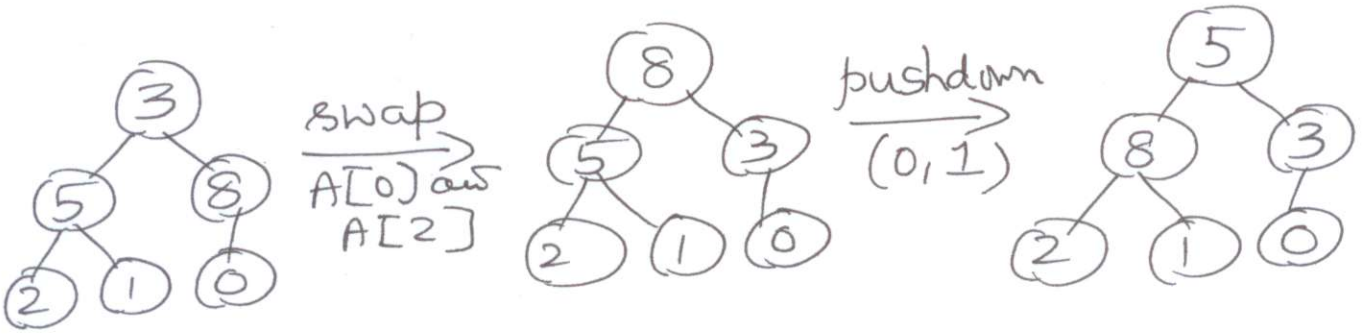
The overall complexity is

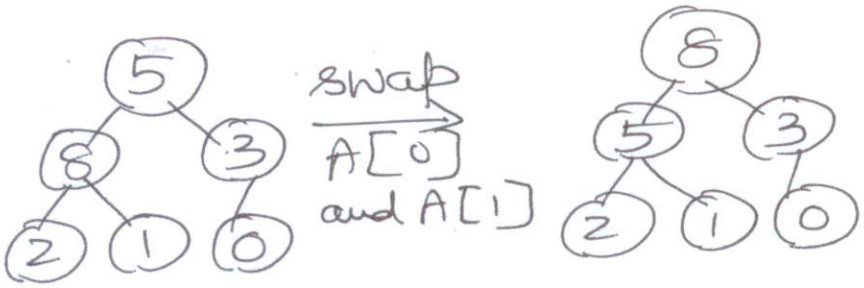
$$O(n) + O(n \log n)$$
$$= O(n \log n)$$











this structure represents sorted array