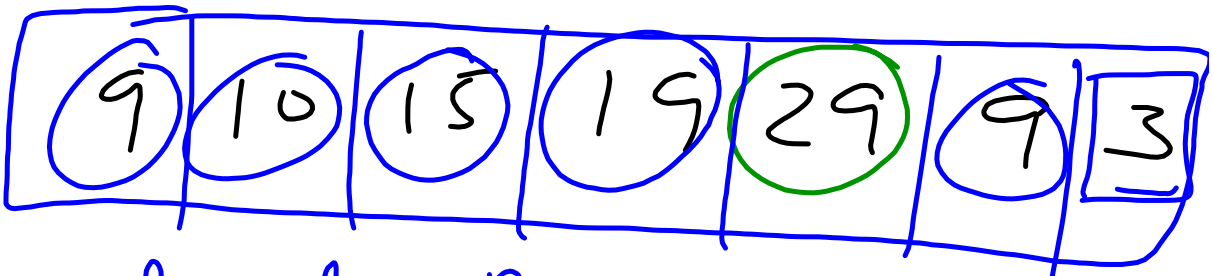
Priority Queue.

$\Longleftrightarrow$ a queue structure that allows us to serve higher priority first.

1) 

	Remove	Insert
~~Unsorted~~ Unordered array	$O(n)$	$O(1)$
Unordered Linked list	$O(n)$	$O(1)$
Ordered array	$O(1)$	$O(n)$
Ordered linked list	$O(1)$	$O(n)$
Binary heap	$O(\log n)$	$O(\log n)$

Heap Data structure

Binary Heap

Heap data structure is a type where: (a) Max heap
→ the item with highest value is always at the root
& for each parent node
The value of parent is always greater than the children.

Node
x
key ≥ keys of children

→ Binary tree where a key of parent node is greater or equal to key of children nodes.

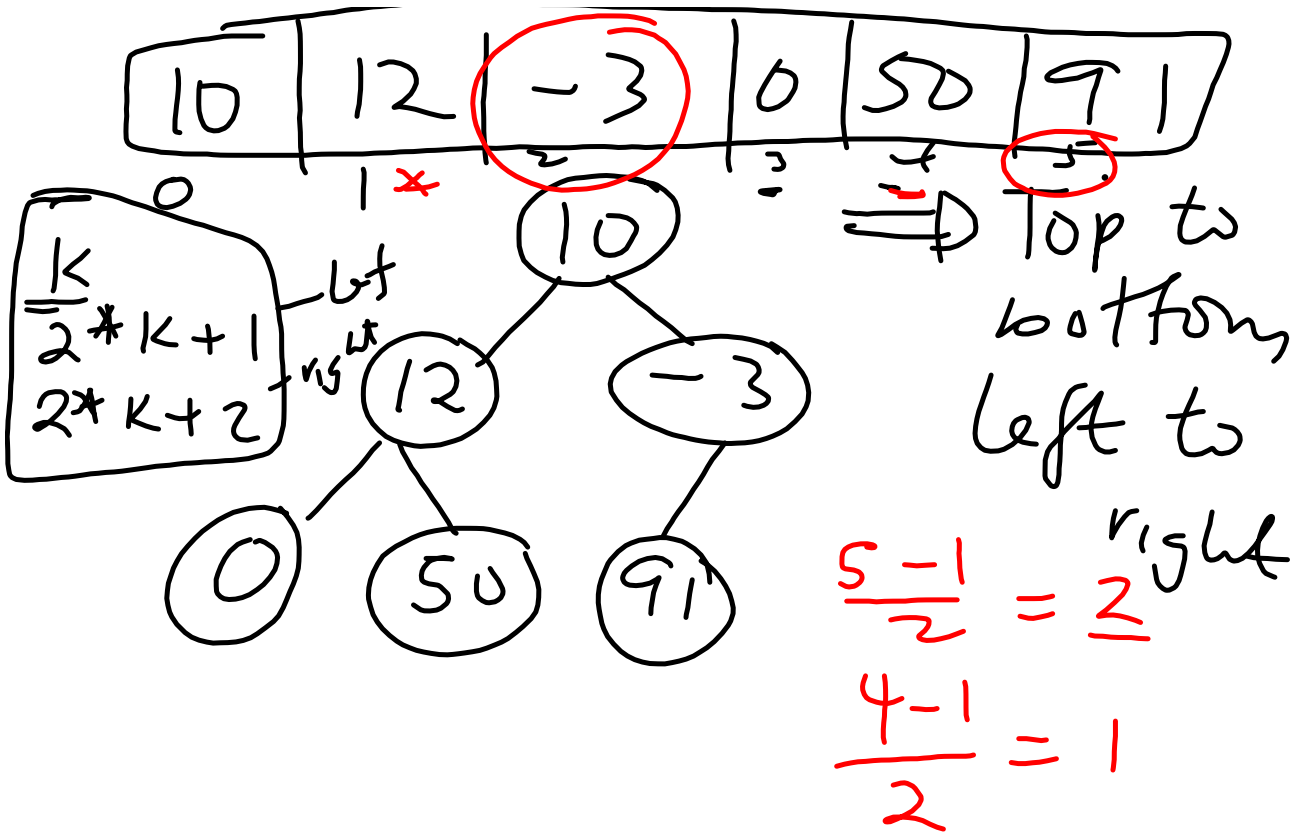
Binary Heap has two properties

(i) Ordering property

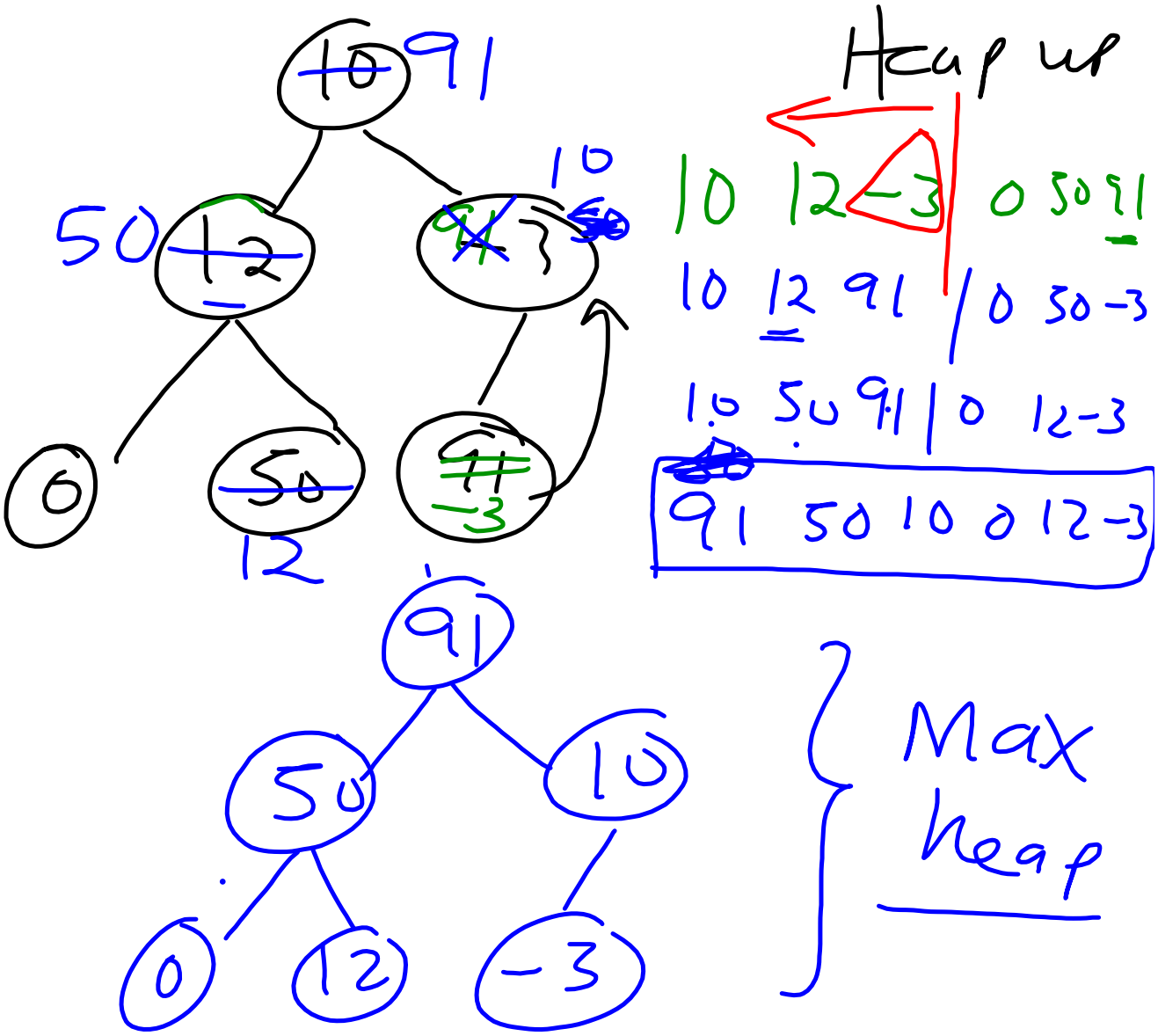
Parent Nodes Key $\geq$ Key of Child

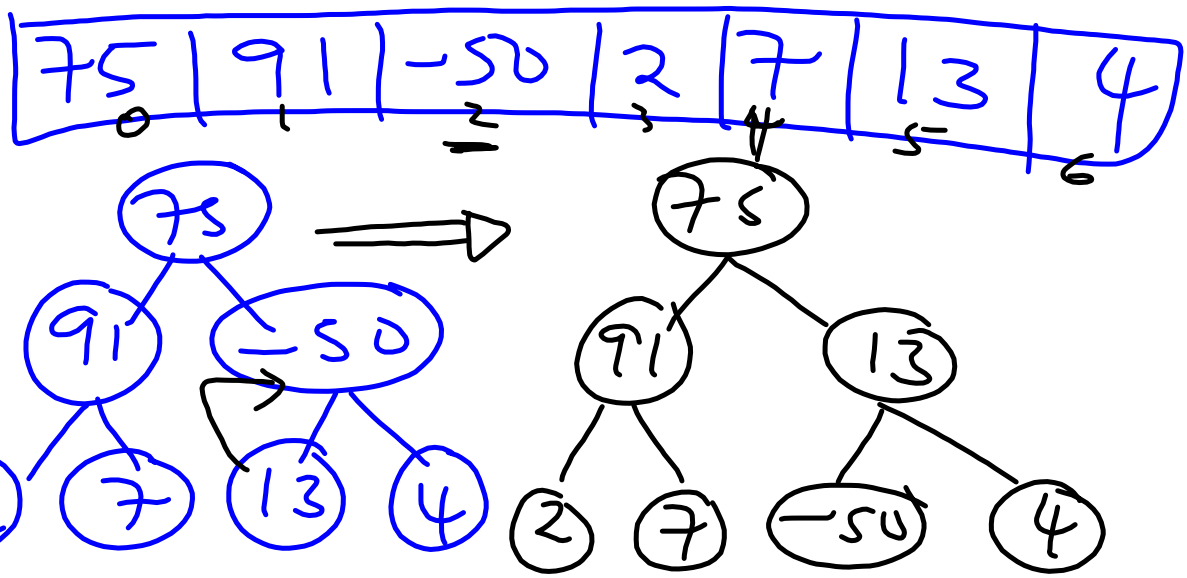
(ii) Structural property

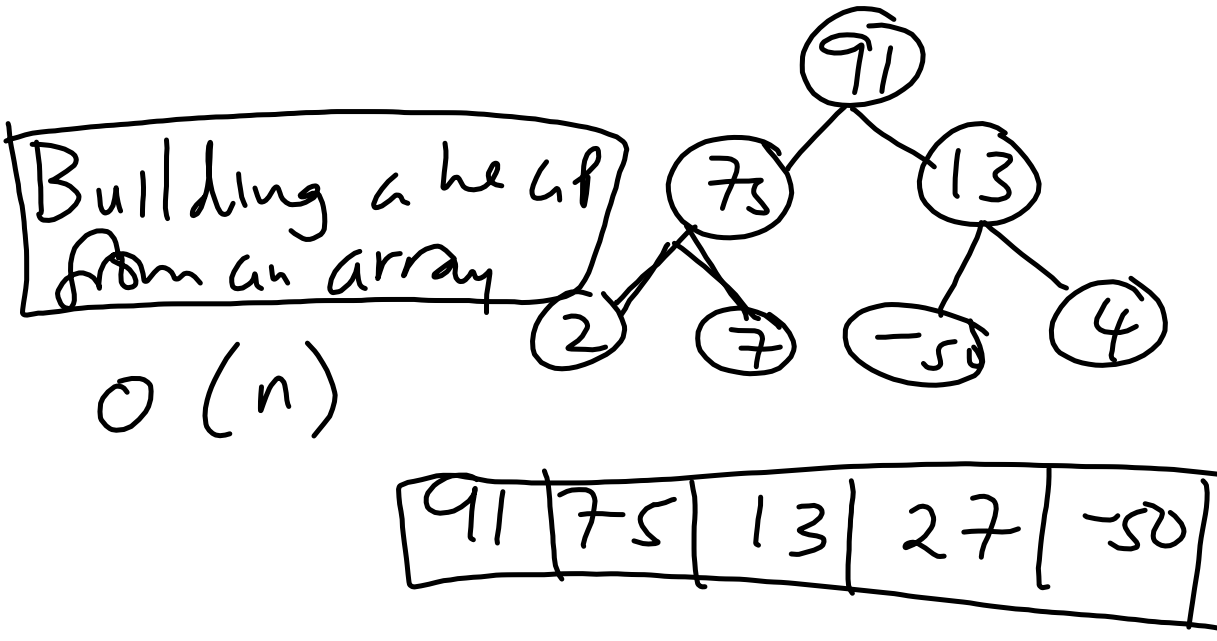
The binary tree is a complete/almost complete


10
12
-3
0
50
91
0
1
2
3
4
5
k
2*k+1
2*k+2
left
right
10
12
-3
0
50
91
Top to bottom
left to right
5-1/2 = 2
4-1/2 = 1

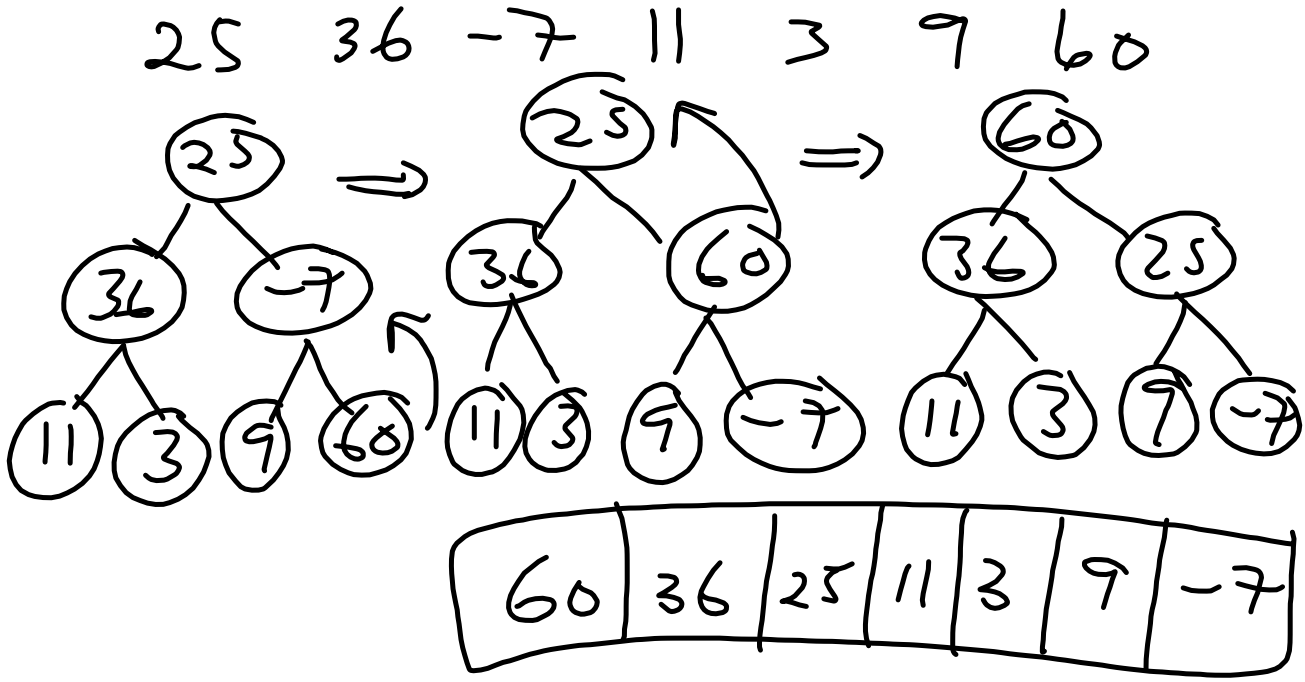
Heapify an Array to make it a binary heap:

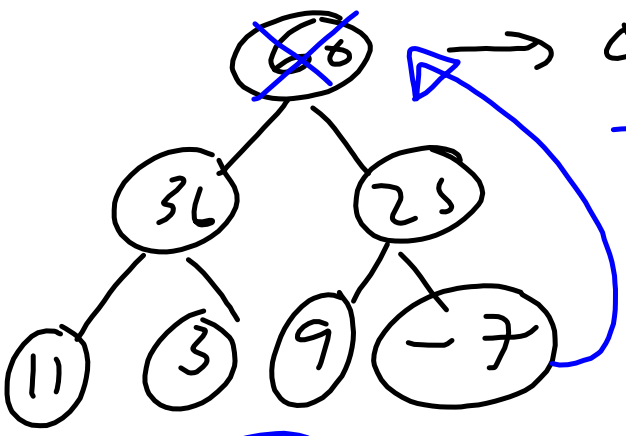

Heap up
10 12 -3 0 50 91
10 12 91 0 50 -3
10 50 91 0 12 -3
91 50 10 0 12 -3
Max heap


75 91 -50 2 7 13 4
0 1 2 3 4 5 6


Building a heap from an array
O(n)

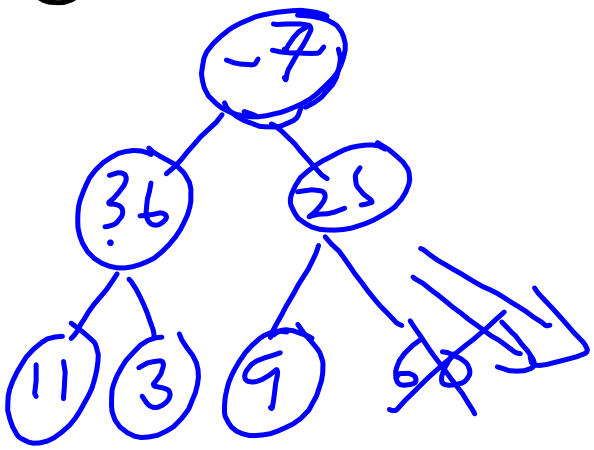
91 75 13 27 -50 4


25 36 -7 11 3 9 60
60 36 25 11 3 9 -7



O(1)

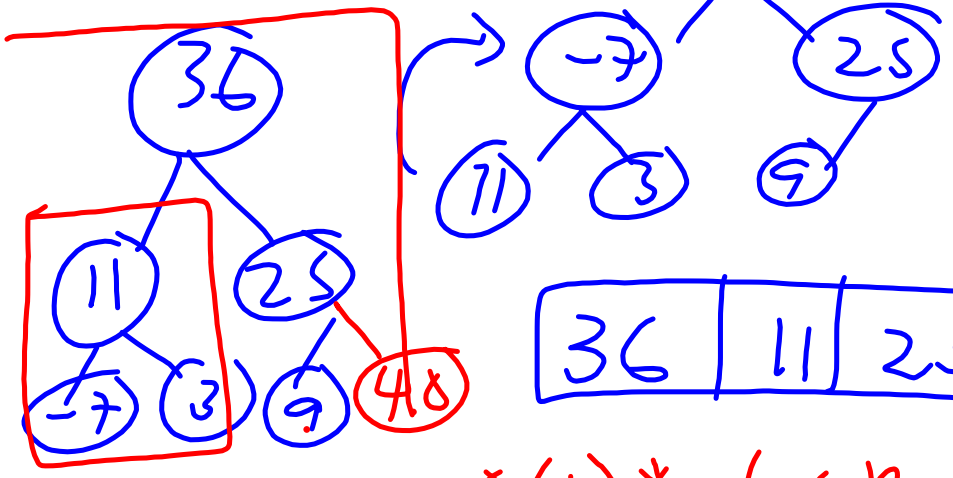
→ go to right most node & swap with root before removing root



→ We are always serving the root

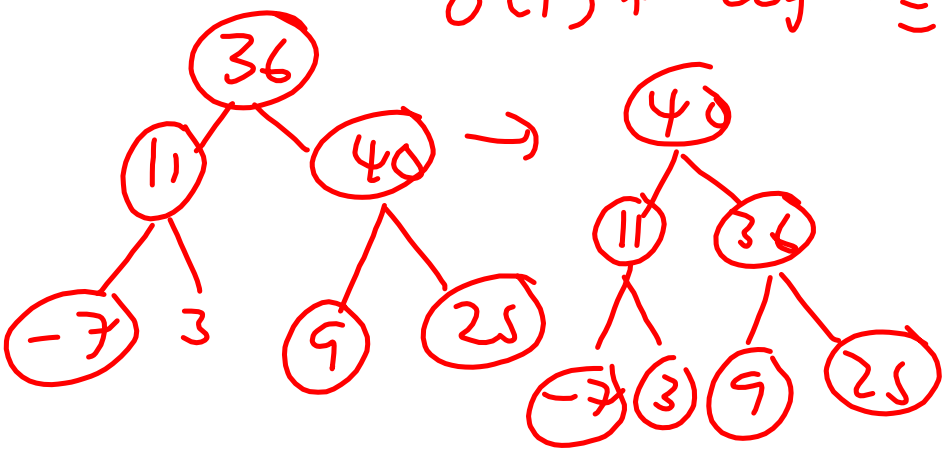
1 * log n

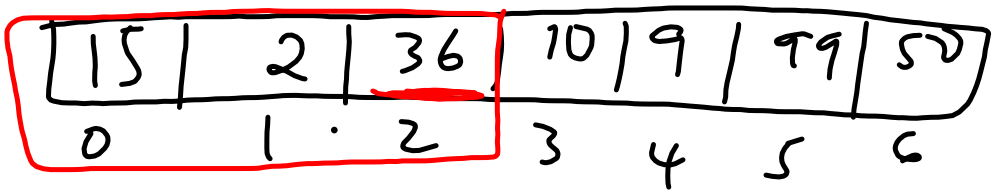
O(log n)



36	11	25	-7	3	9

O(1) * log n = O(log n)

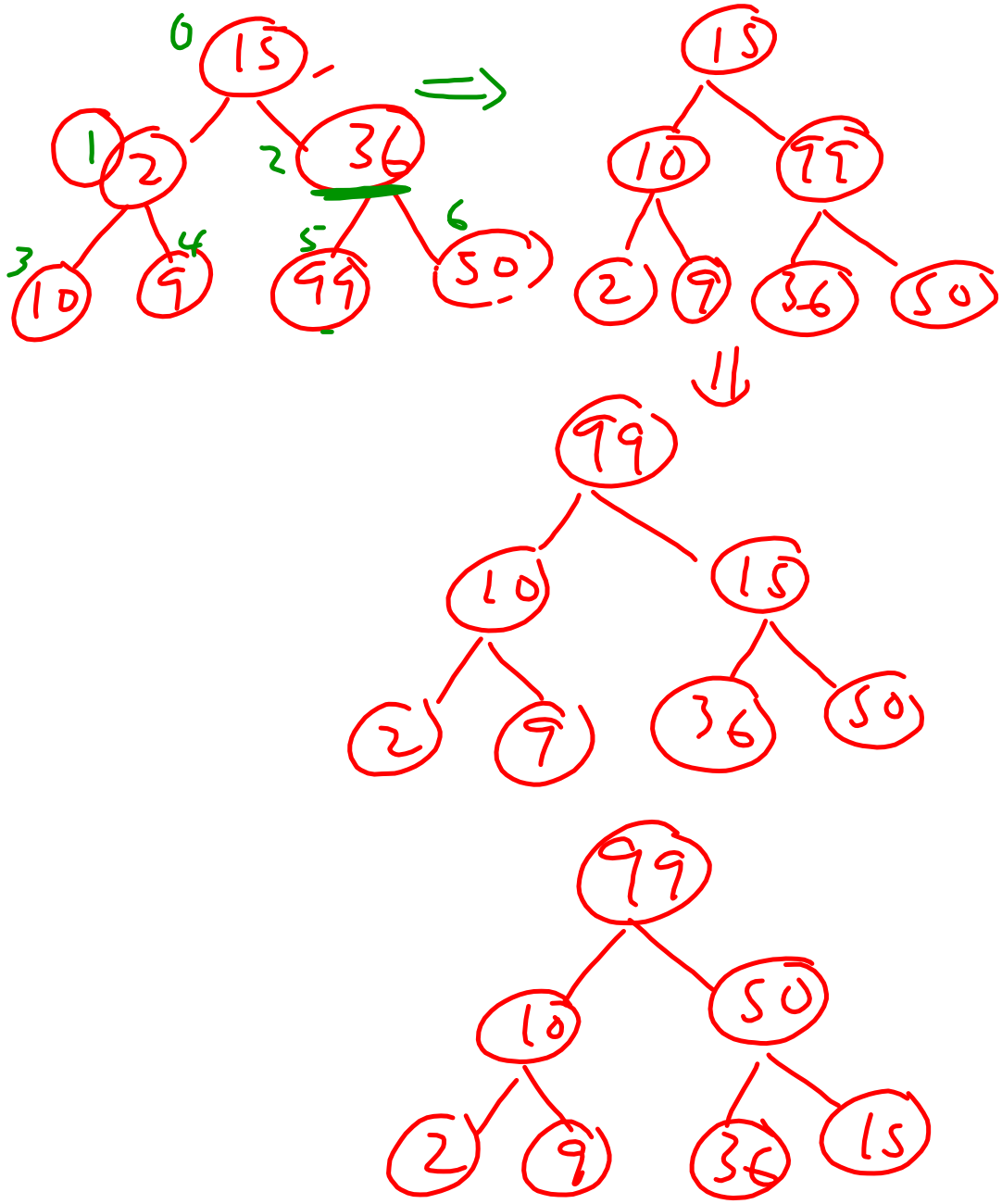




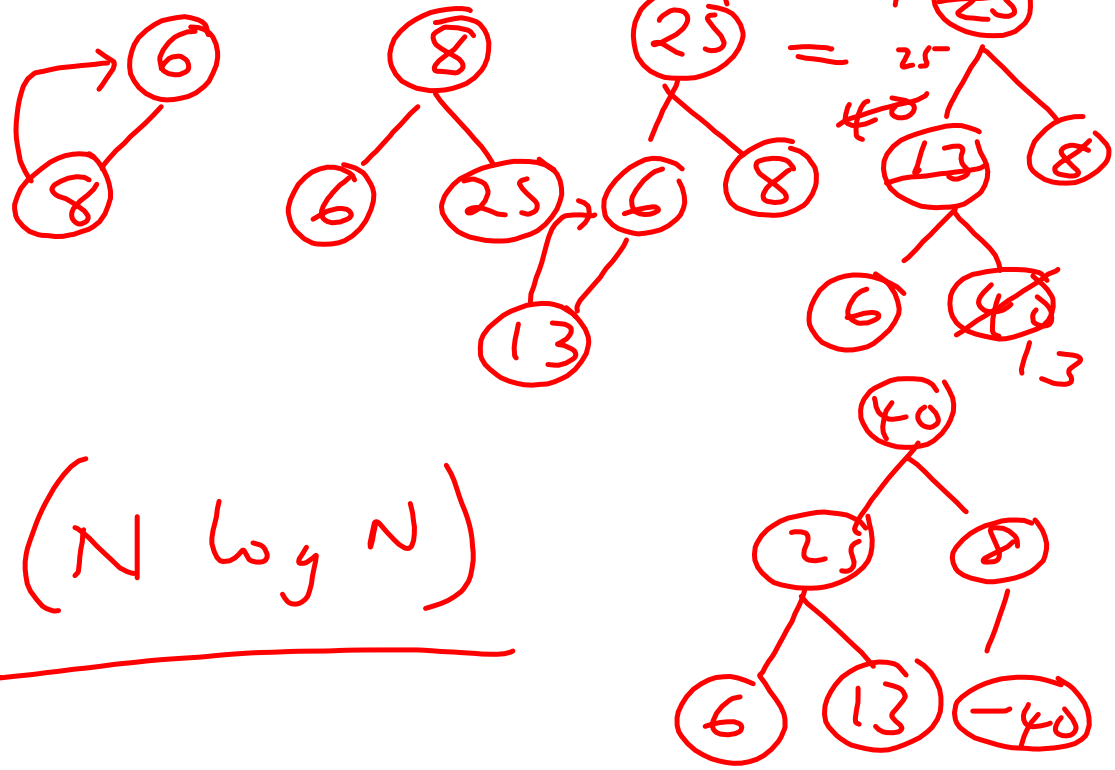
Go to the rightmost item; & get its index

$\frac{6-1}{2} = \underline{\underline{2}}$ ← index of the parent last item

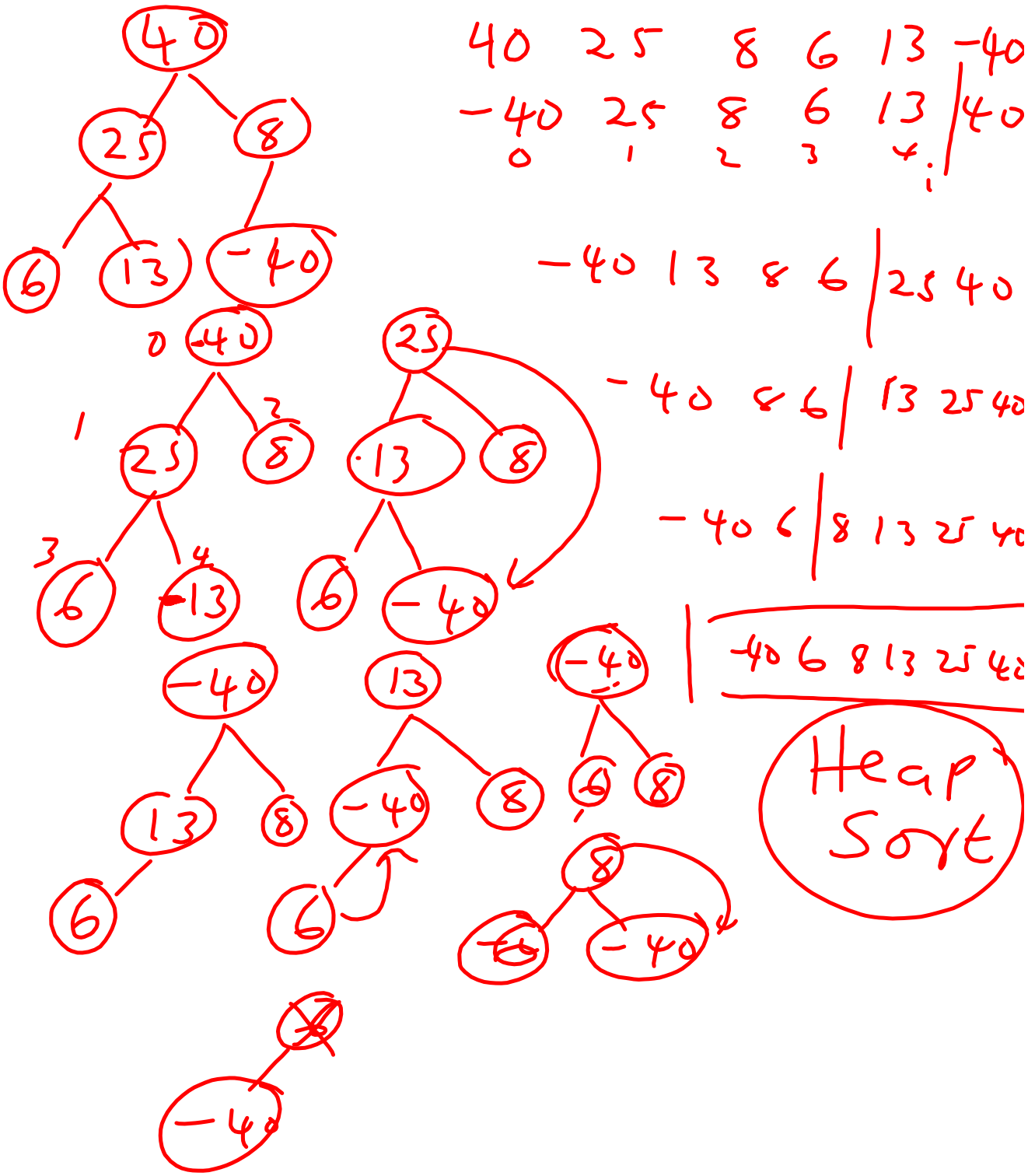
```
for ( i = (last index - 1) / 2 ; i >= 0 ; i--)
    { heapDown ( i );

}
```


0
15
1
2
2
36
3
10
4
9
5
99
6
50
15
10
99
2
9
36
50
99
10
15
2
9
36
50
99
10
50
2
9
36
15

6, 8, 25

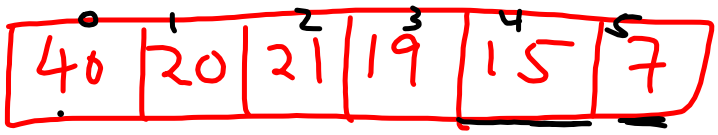


$O(N \log N)$


40 25 8 6 13 -40
-40 25 8 6 13 | 40
0 1 2 3 4 i
-40 13 8 6 | 25 40
-40 8 6 | 13 25 40
-40 6 | 8 13 25 40
-40 6 8 13 25 40
Heap Sort

Heap sort is a sorting algorith that uhlises the Binary heap where you repeated extract max/min & heapify the array

Time complexity $O(N \log N)$ $O(1) * \log n$ $O \log n$

A = 

0 1 2 3 4
7 40 20 21 15 40

```
for ( i = lastIndex; i >= 0; i--) {
    int temp = A[i];
    A[i] = A[0];
    A[0] = temp;
    HeapDown((i-2)/2);
}
```

Extract max (Remove) = $O(\log n)$

Insert — $O(\log N)$

* Build from Array $O(N)$

Build by Inserting one by one $(N \log N)$

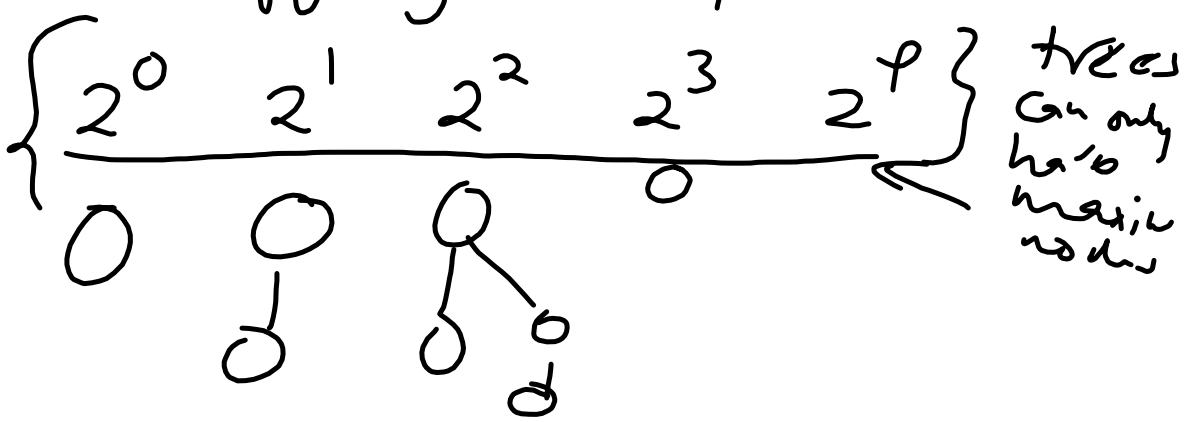
Heap sort $\Rightarrow$ $(N \log N)$

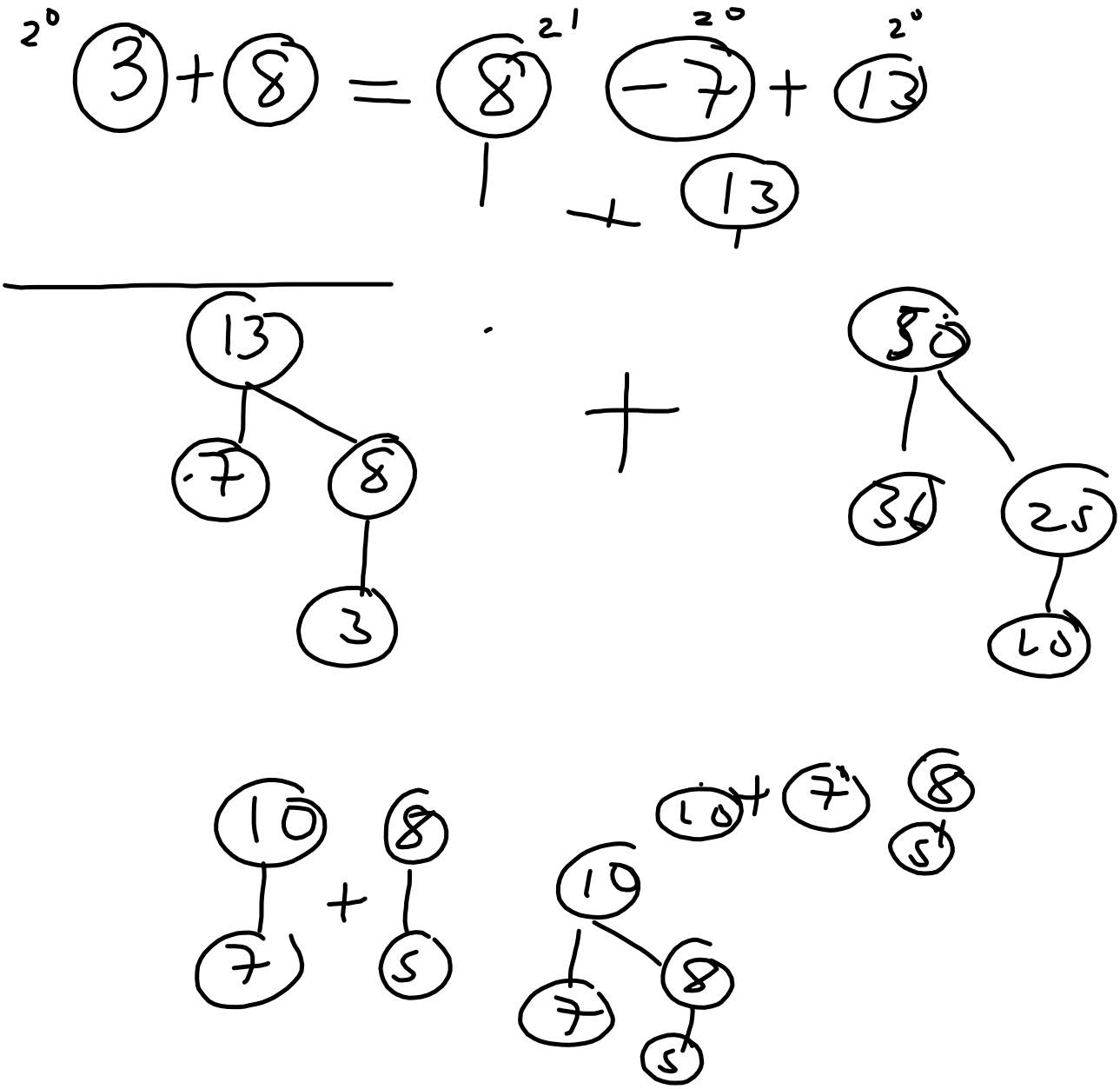
Merge two Binary Heaps.

$O(n + n) = O(n)$

Binomial Queues

⇒ We maintain a forest of Queues satisfying max/min property



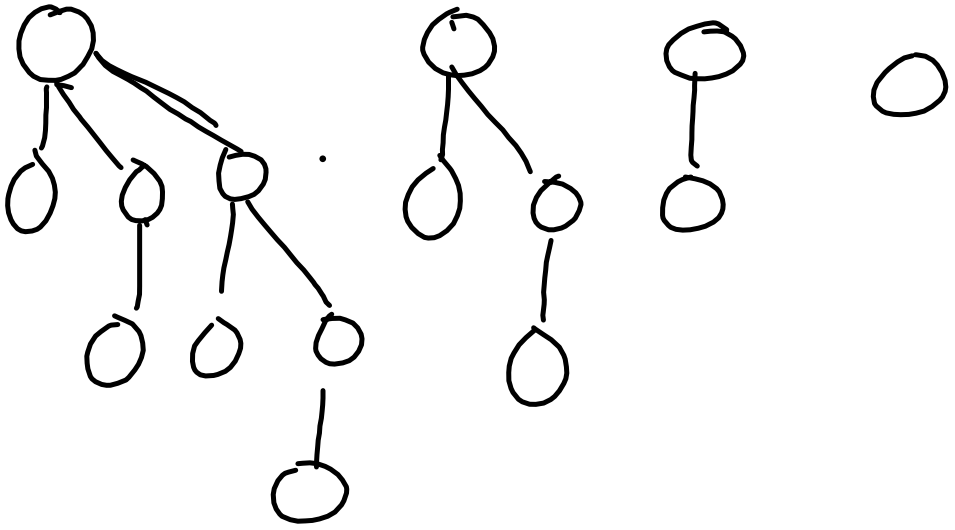

3 + 8 = 8 + 13
-7 + 13
13
7
8
3
50
30
25
10
10
7
8
5
10
7
8
7
5

Binomial queue

(1) 15 nodes

(1) convert 15 to Binary

2^4	2^3	2^2	2^1	2^0
0	1	1	1	1



1110 = 14 nodes