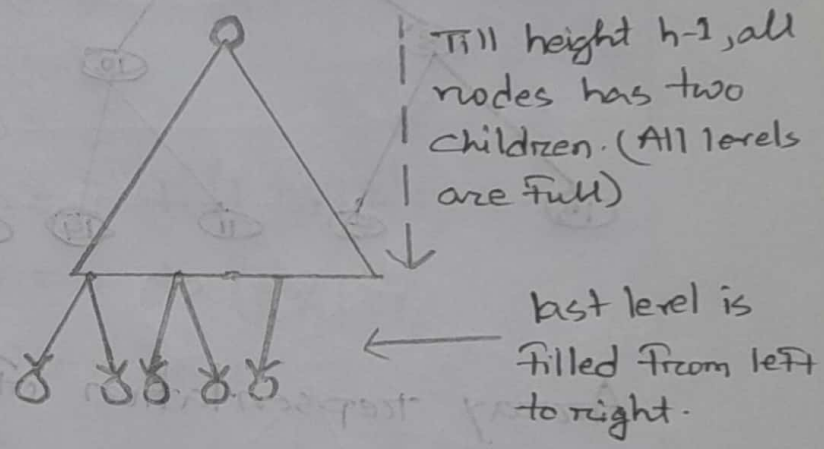
Heap

What is heap in Data Structure?

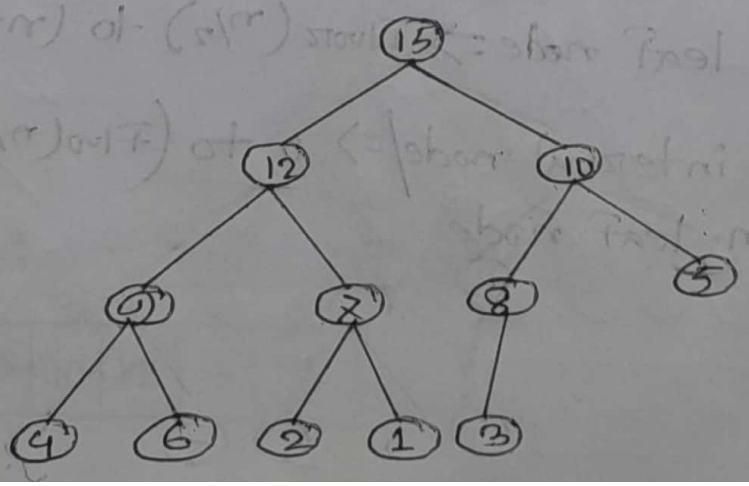
A heap is a complete binary tree or almost complete binary tree structure where each element satisfy a heap property.

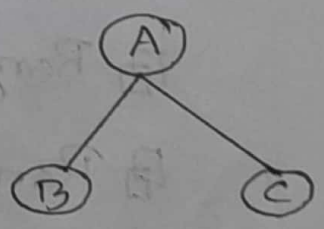
Complete binary tree:



There are two types of heap data structures: Max heap and Min heap.

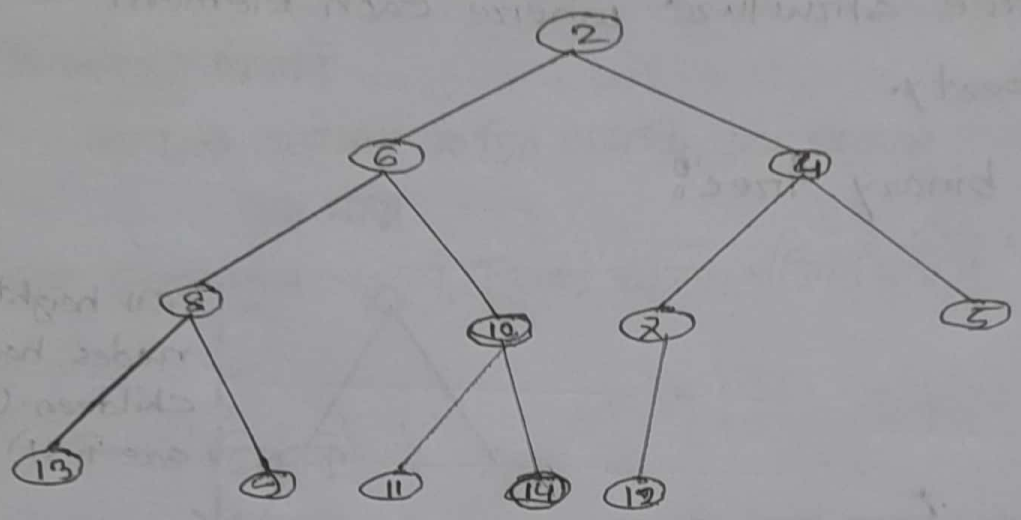
Max heap: Root node should be greater than all the left and right subtree nodes and it is recursively true all the subtree.

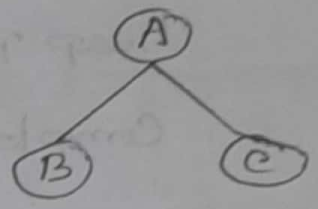




A >= B, A >= C

Min Heap: Root node should be smaller than all the left & right node subtree nodes and it is recursively true all the subtree.





$A <= B, A <= C$

Array repsentation of heap

- Root node of the binary heap is stored at A[0]
- Given element A[i]

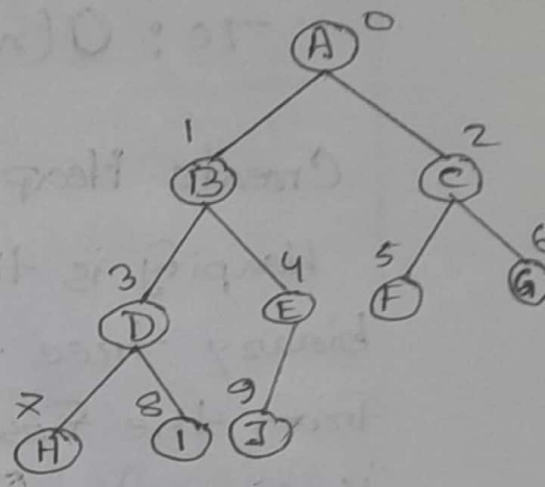
  left child of 'i' left(i) ⇒ A[2*i+1]; if 2i+1 < n

  Right child of 'i' right(i) ⇒ A[2*i+2]; if 2i+2 < n
- Parent of A[i] ⇒ A[Floor(i/2)]
- Rang of leaf node ⇒ Floor(n/2) to (n-1)
- Range of internal node / non-leaf node ⇒ 0 to (Flo(n/2) - 1)

$n = 10$

Node = 'D'

Index = 3 = 'i'



left child $= 2*i+1 = 2*3+1 = 7$

Right child $= 2*i+2 = 2*3+2 = 8$

Parent $= \text{Floor}(i/2) = \lfloor i/2 \rfloor = \lfloor 3/2 \rfloor = 1$

Range of leaf node $= \lfloor n/2 \rfloor$ to $(n-1)$

$= \lfloor 10/2 \rfloor$ to $(10-1)$

$= 5$ to 9

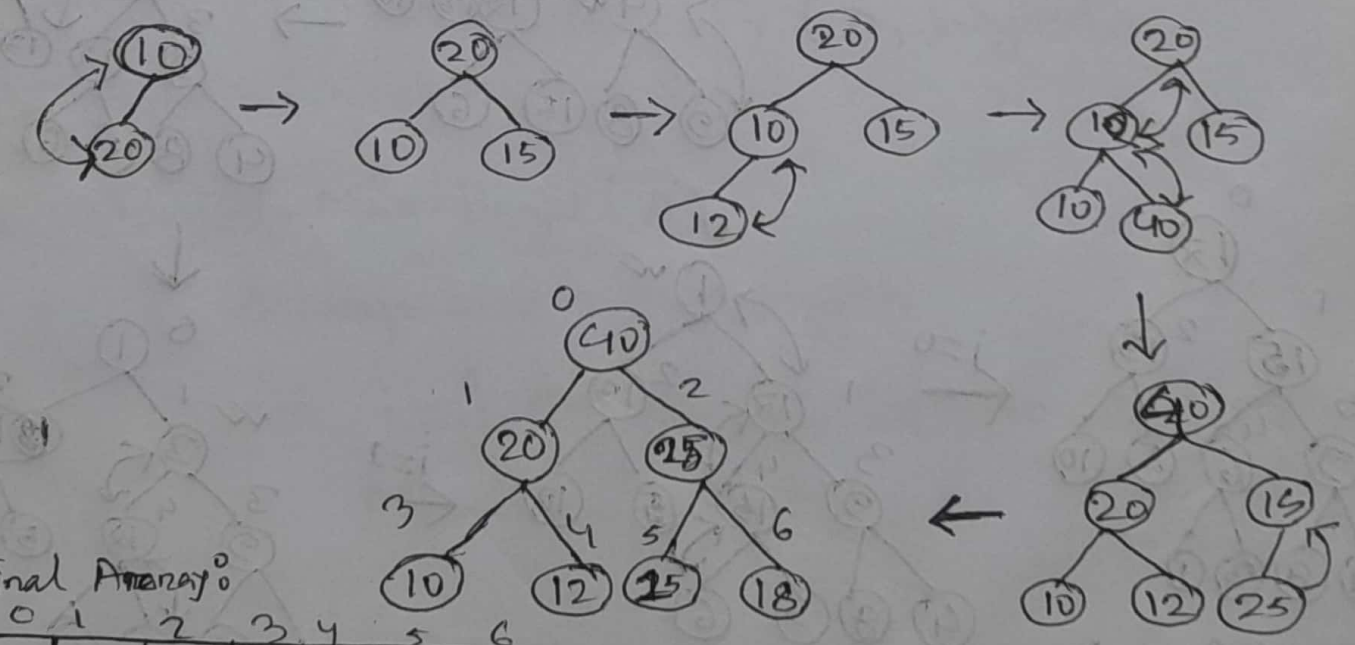
Range of ~~right~~ Internal node $= 0$ to $(\lfloor n/2 \rfloor - 1)$

$= 0$ to $(\lfloor 10/2 \rfloor - 1)$

$= 0$ to 4

Create Heap (Max heap)

10	20	15	12	40	25	18



Final Array:

0	1	2	3	4	5	6
40	20	25	10	12	15	18

TC: O(nlogn) [~~Tree~~ Tree length]

Create Heap by using Heapify:

Heapify is the process of creating a heap DS from a binary tree represented using an array. His start from the first index of the non-leaf node whose index is given by ~~(n/2 + 1)~~ (floor(n/2) - 1). Hepify uses recursion.

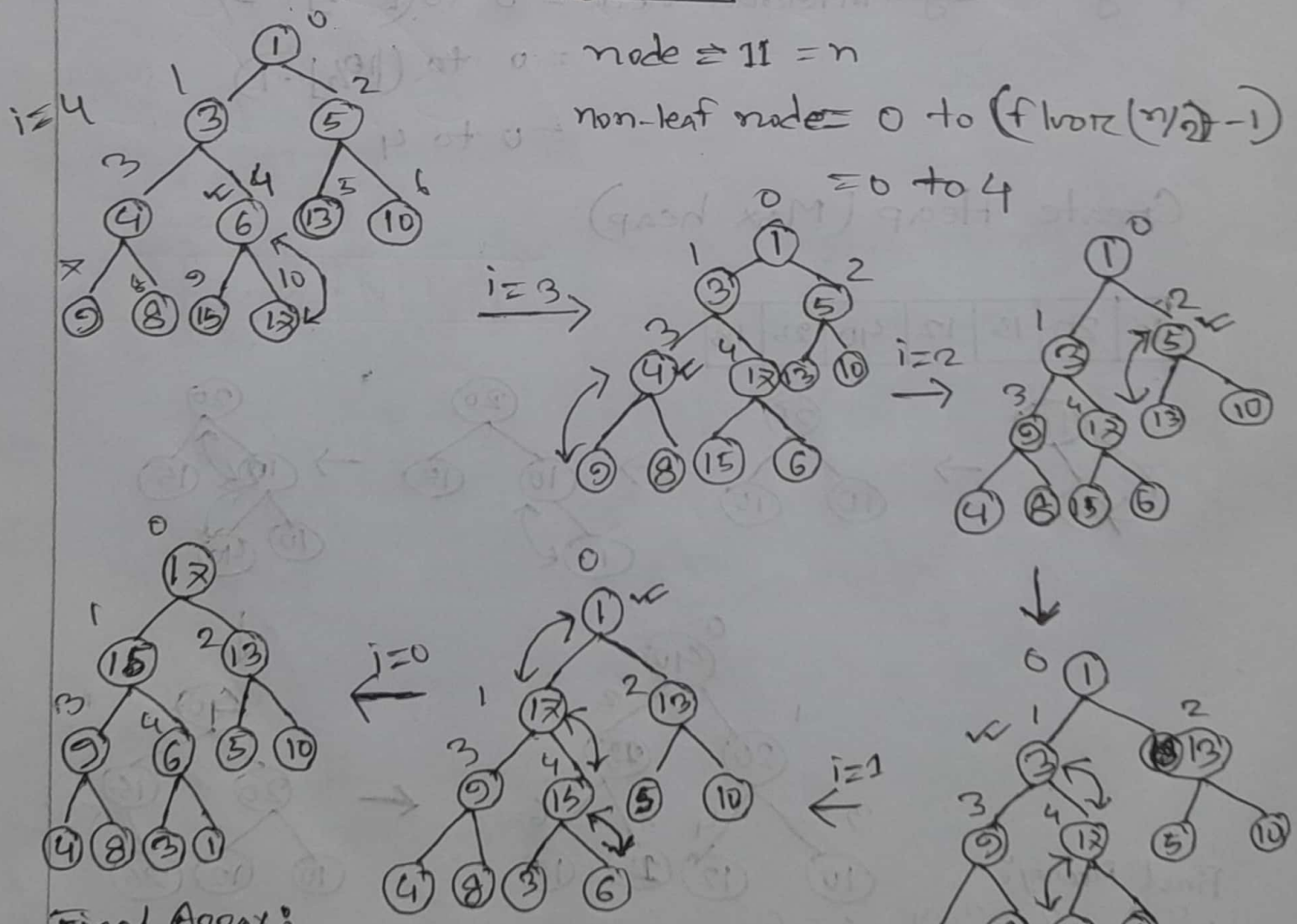
0	1	2	3	4	5	6	7	8	9	10
1	3	5	4	6	13	10	9	8	15	17

~~creat~~ Build Max heap

node = 11 = n

non-leaf node = 0 to (floor(n/2) - 1)

= 0 to 4



Final Array:

17	15	13	9	6	5	10	4	8	3	1

TC : $O(n)$

Algorithm :

```
Max-Heapify (A, i)
    l = 2*i + 1
    r = 2*i + 2
    if l ≤ A.heap-size() and A[l] > A[i]
        largest = l
    else
        largest = i
    if r ≤ A.heap-size() and A[r] > A[largest]
        largest = r
    if largest != i
        swap A[i] with A[largest]
        Max-Heapify (A, largest)
```

```
Build-Max-Heap (A)
    A.heap-size() = A.length
    for i = (⌊A.length/2⌋ - 1) to 0
        Max-Heapify (A, i)
```

∴ TC = Cost of Max-Heapify * Cost of Build-Max-Heap

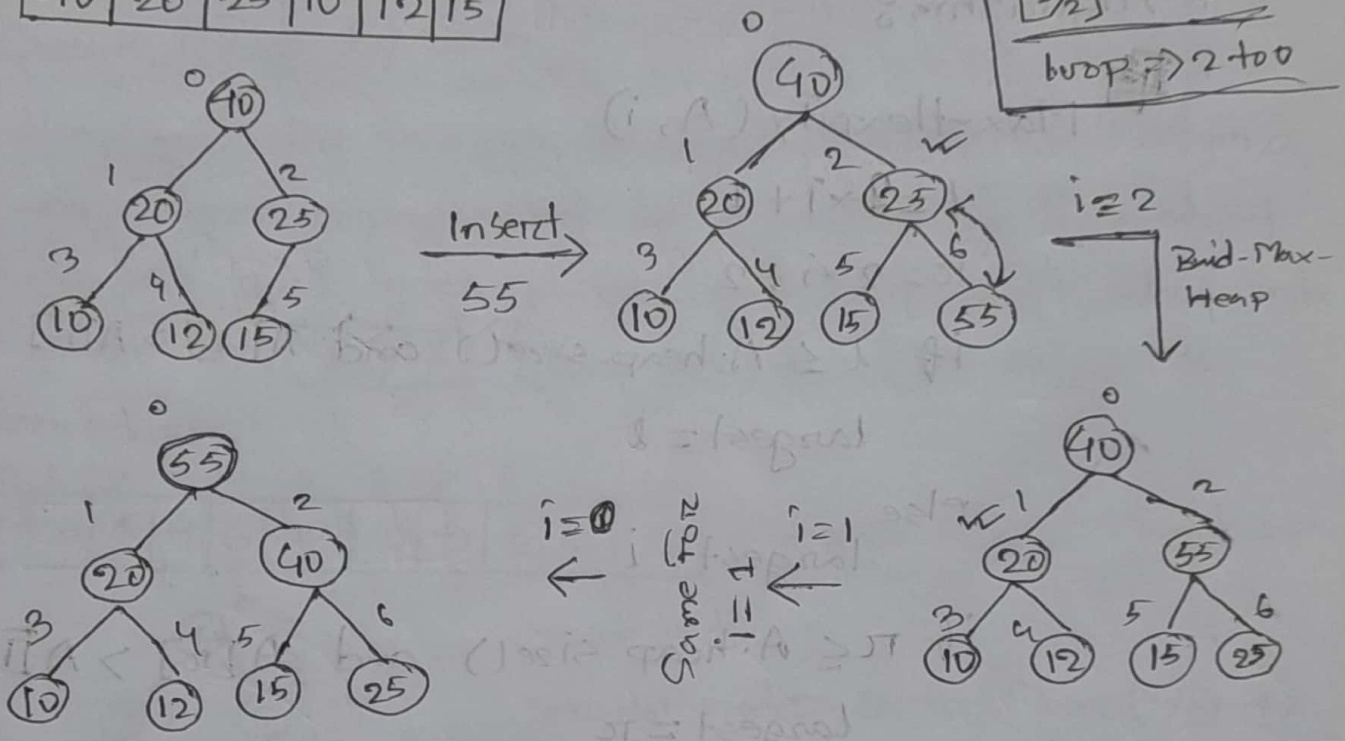
$= \log n * n/2 \approx O(n \log n)$

Insert Element in a Heap:

0	1	2	3	4	5
40	20	25	10	12	15

$\lfloor 6/2 \rfloor - 1 = 2$

loop ⇒ 2 to 0



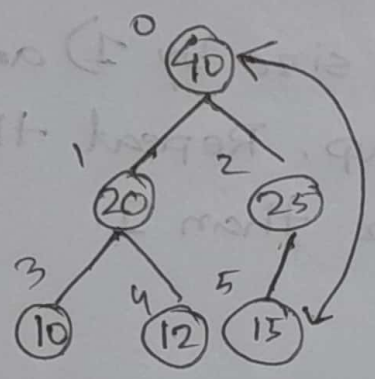
Final Array:

0	1	2	3	4	5	6
55	20	40	10	12	15	25

Delete Element in a Heap:

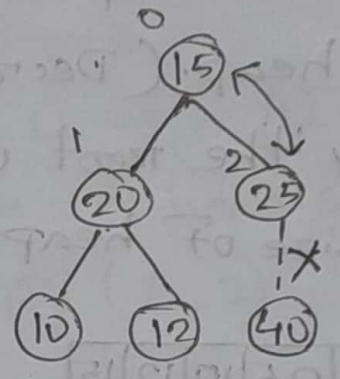
** Only Delete the root element

0	1	2	3	4	5
40	20	25	10	12	15



array size = 6

Delete '40'



array size = 5

$(\lfloor 5/2 \rfloor - 1)$ to 0

$\Rightarrow 1$ to 0

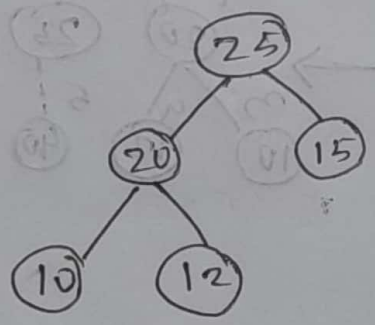
i = 1

Build-Max-Heap

Same for

i = 1

i = 0



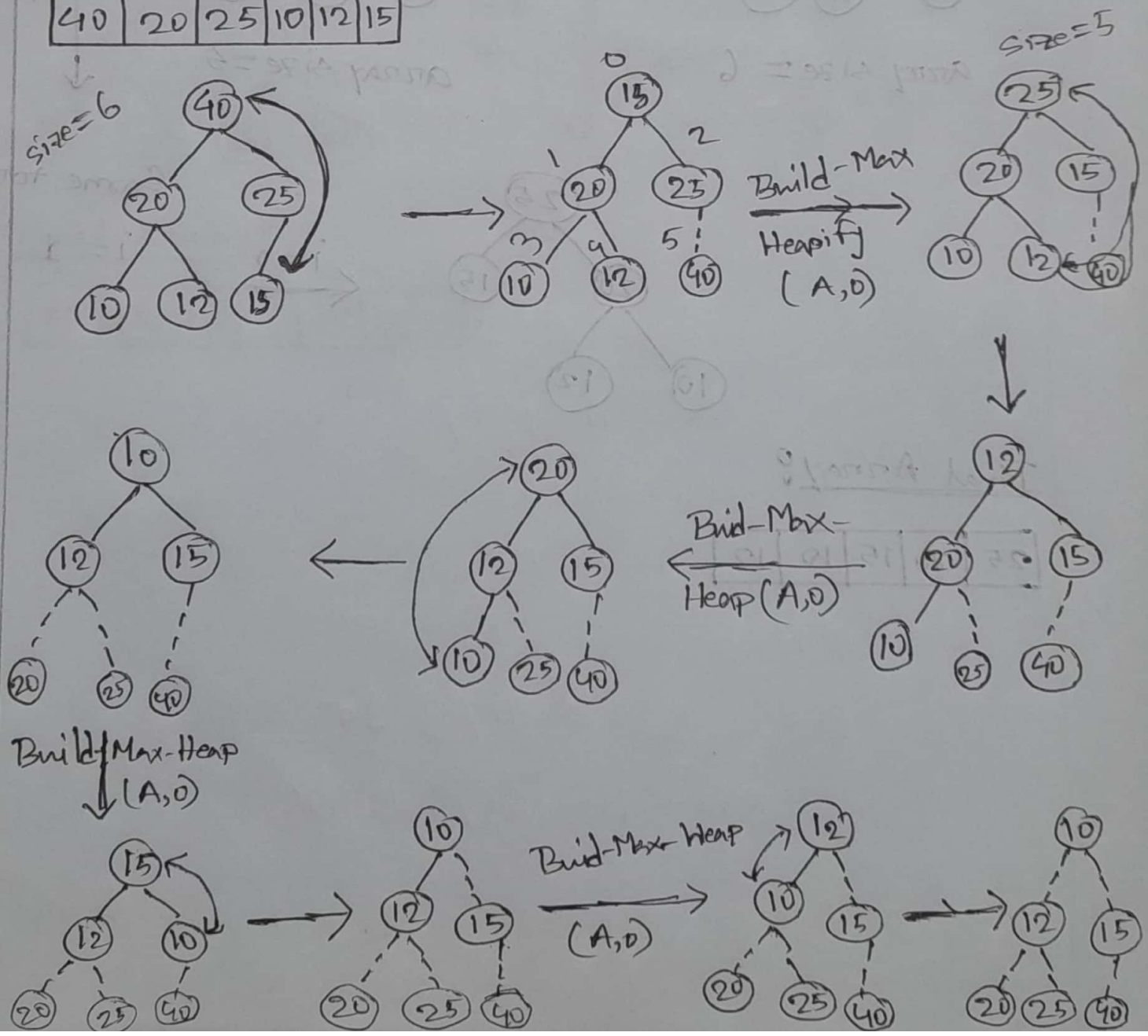
Final Array:

25	20	15	10	12

Heap Sort:

First convert the array into heap DS using heapify, then one by one delete the root node of the max heap and replace it with the last node in the heap. (Decrease heap size by 1) and then heapify the root of the heap. Repeat the process until size of heap is greater than 1.

40	20	25	10	12	15



Final Array :

10	12	15	20	25	40

Algorithm :

```
HeapSort(A)
    For = A.length downto 2
        ~~swap {A[1] with~~
        swap A[1] with A[i]
        A.heap-size = A.heap-size - 1
        Max-Heapify (A, 1)
```