Heapsort

comes under transform and conquer technique.

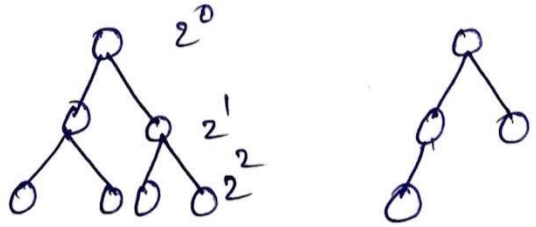
→ Idea of transform & conquer technique is to transfer the problem into some easier / similar versions using some procedure to get the better solution.

Heap

heap is a binary tree with satisfy two properties.

i) Structural property - it should be an almost complete binary tree apart from the last level, all other levels should have 2^i nodes & last level should be left filled.

Ex:-



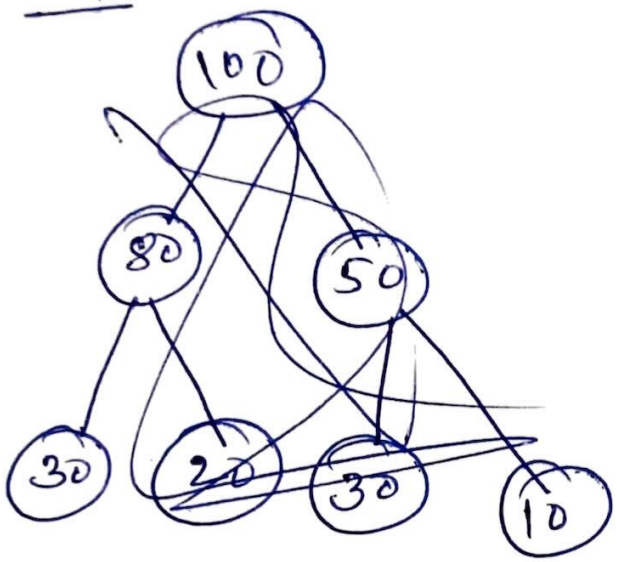
ii) Parent dominent property :- Parent should be more dominent when compared to the children.

(max/min) heap

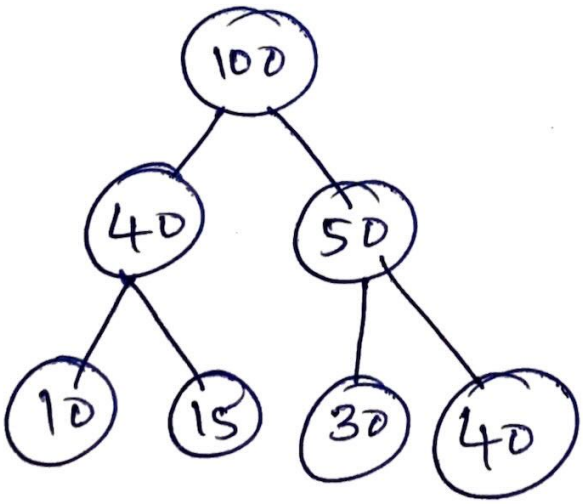
→ Priority given to the max value

" " " min "

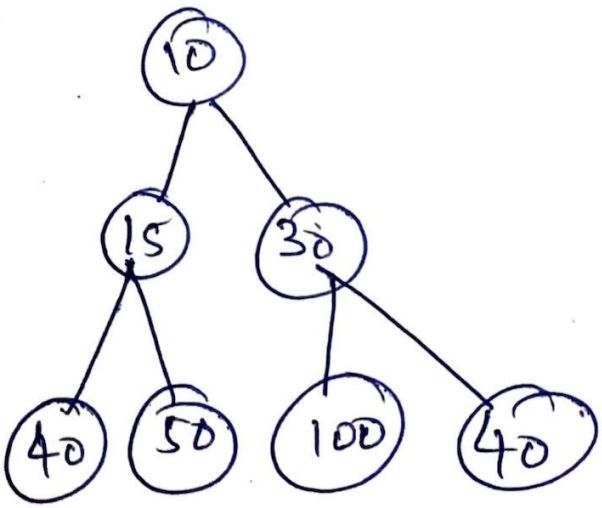
Ex



max



min



2, 7, 6, 5, 10, 8, 3

Procedure to Perform heapsort

1) Construct a max-heap for the given set of elements (n).

2) Exchange the first & last element, reduce the size of heap by 1 & construct heap for remaining (n-1) elements.

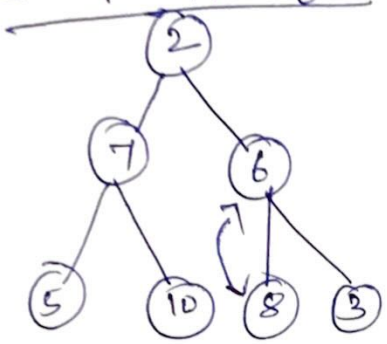
3) Repeat Step 2 until only one element is leftout.

Methods of constructing Heap

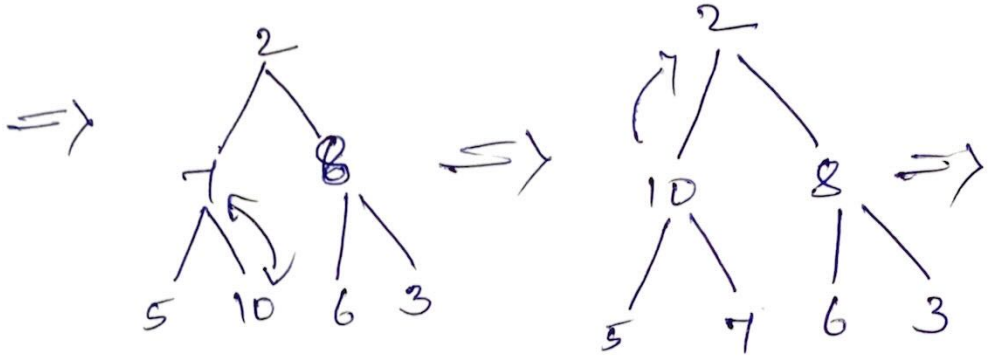
- Bottom-up approach
- Top-down approach

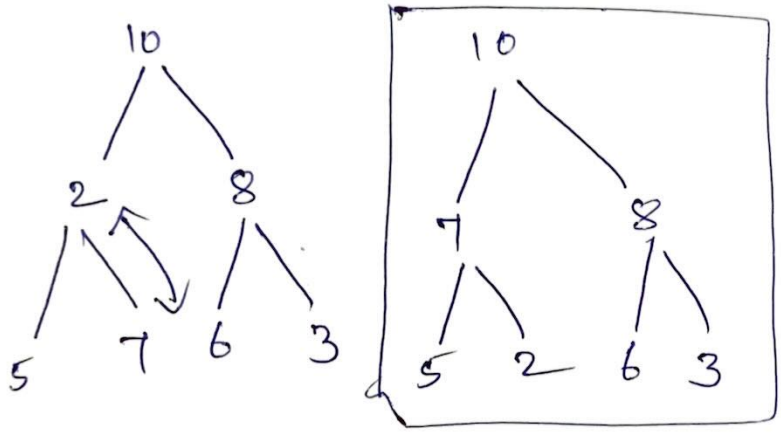
Ex :- 2, 7, 6, 5, 10, 8, 3

B-up-approach

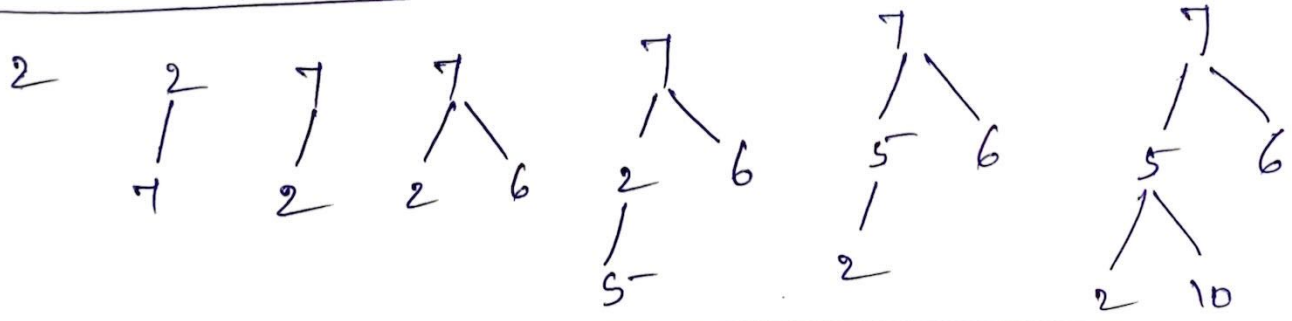


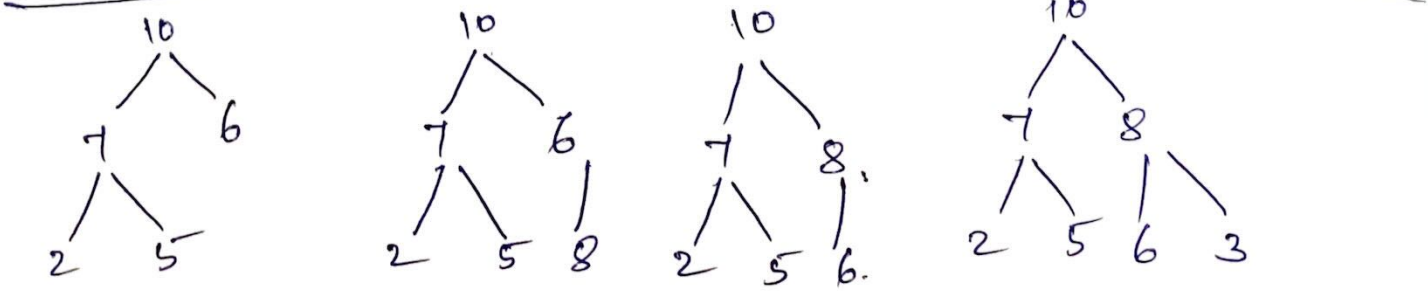
⇒ Max heap





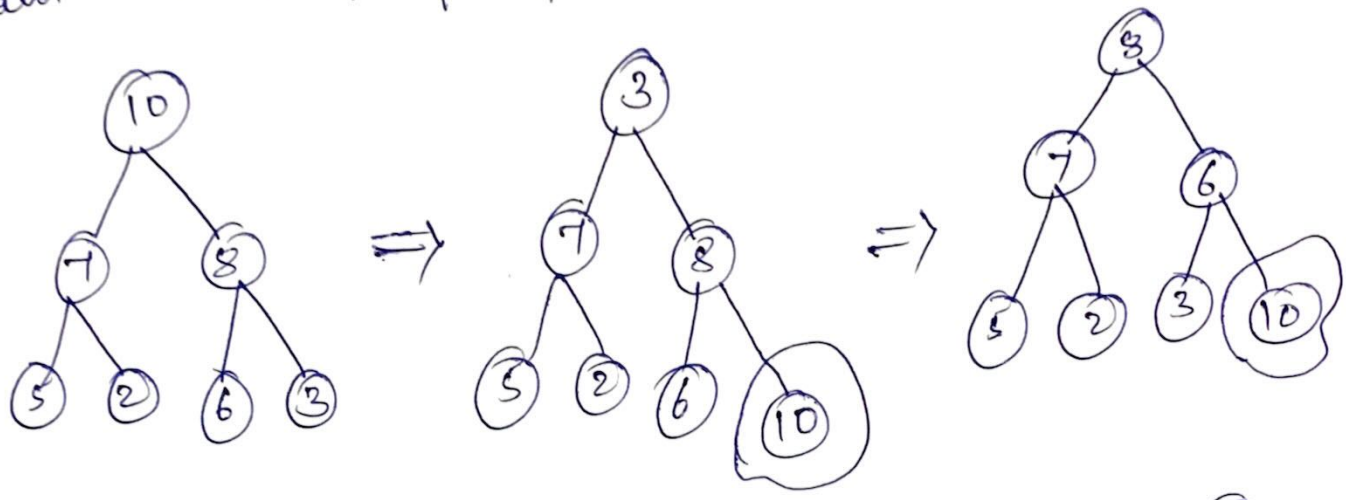
Top-down approach :- 2 7 6 5 10 8 3

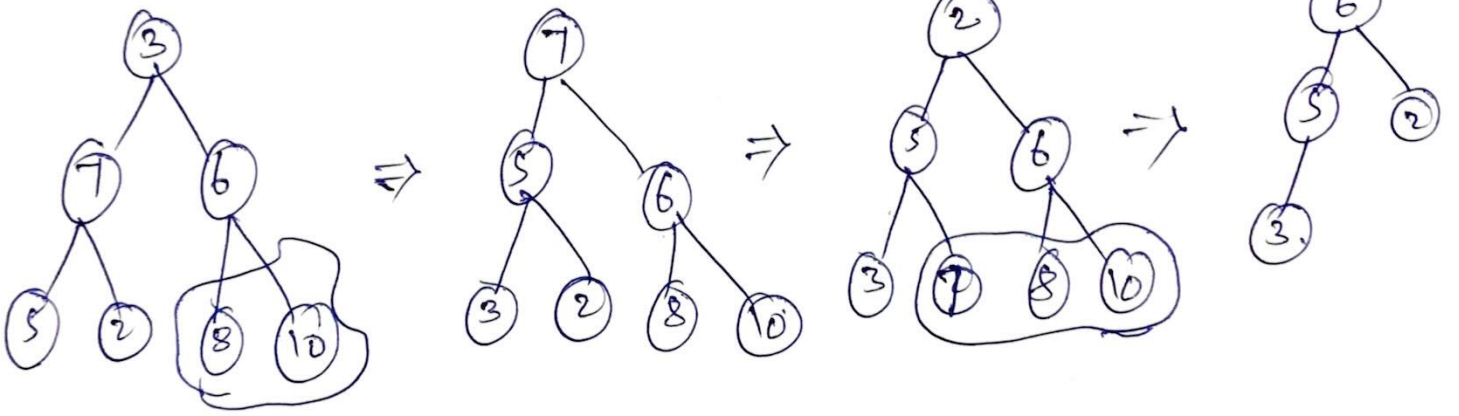


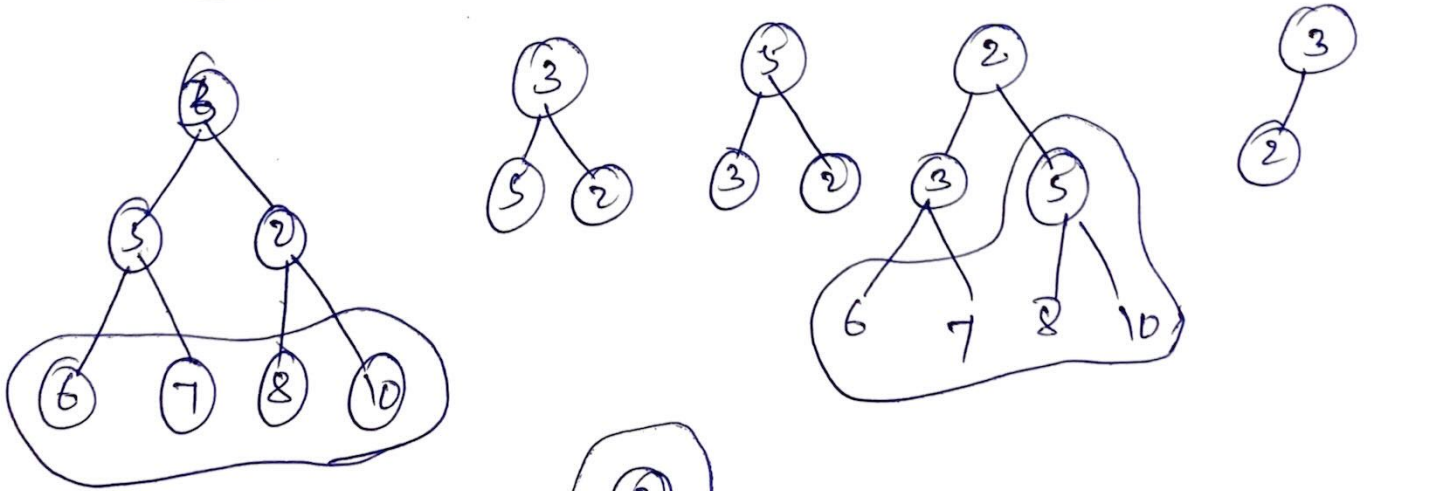


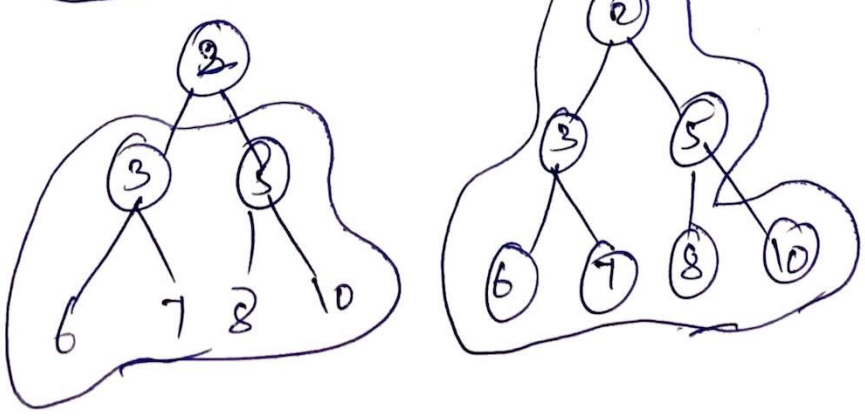
~~array tree~~

index →	1	2	3	4	5	6	7
Ex:- → data	2	7	6	5	10	8	3









Efficiency of heapsort.

construction of heap

B up - O(n)
T down - log n

construction of heap: $O(n)$ +

exchange 1st & last element: $(n-1)\log n$ ✓

highest order is

worst & Average case: $n \log n$

heapsort in inplace sorting algorithm.

Algorithm Heapify(H[1...n])

//construction of maxheap using bottom-up approach

//Input: An array H of 'n' elements

//output: Heap, represented using an array.

```
for i ← [n/2] down to 1 do
    k ← i
    v ← H[k]
    heap ← FALSE
while not heap and 2k<n do
    j ← 2k
    if j<n
if H[j] < H[j+1]
    j ← j+1
if v >= H[j]
    heap ← TRUE
else
    H[k] ← H[j]
    k ← j
H[k] ← v
```