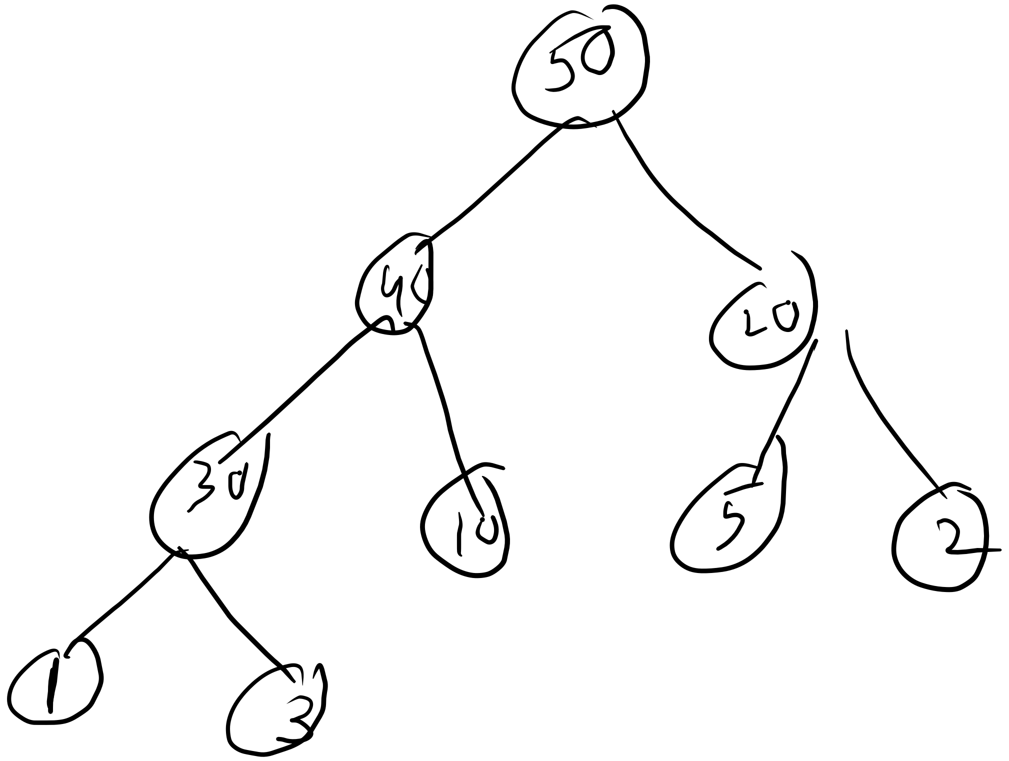
Heapsort

1> Maxheapify

2> Build-max-heap -

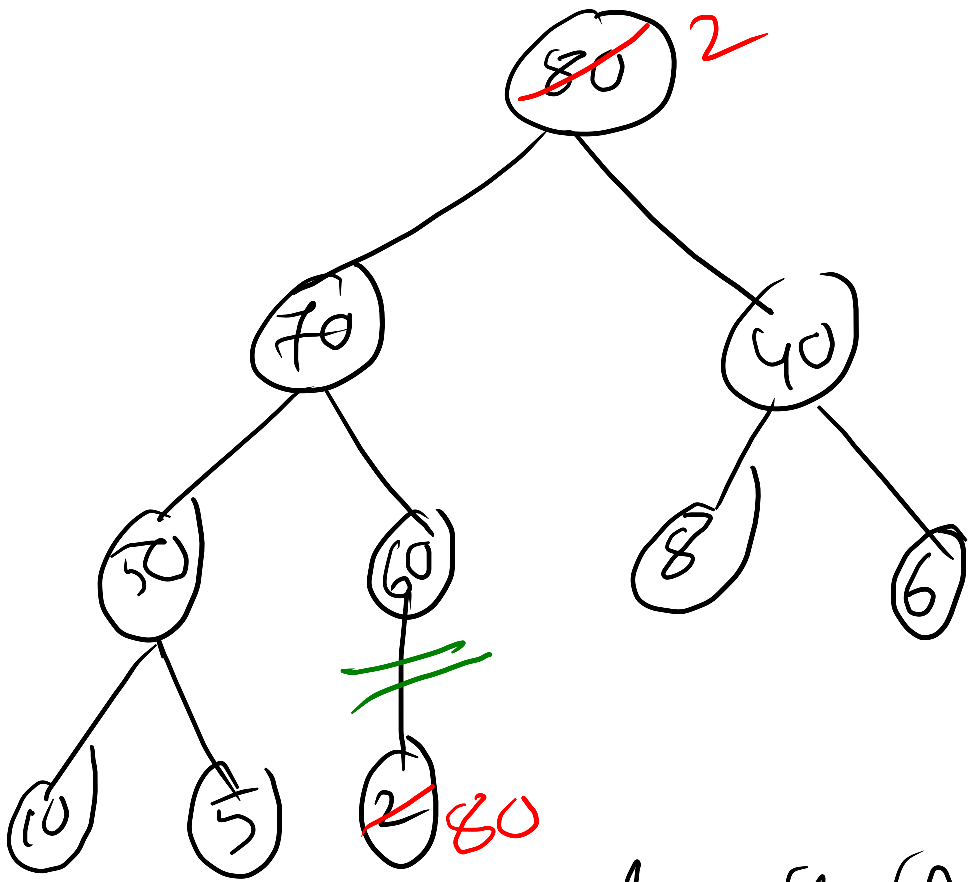


Heap Sort

w

A

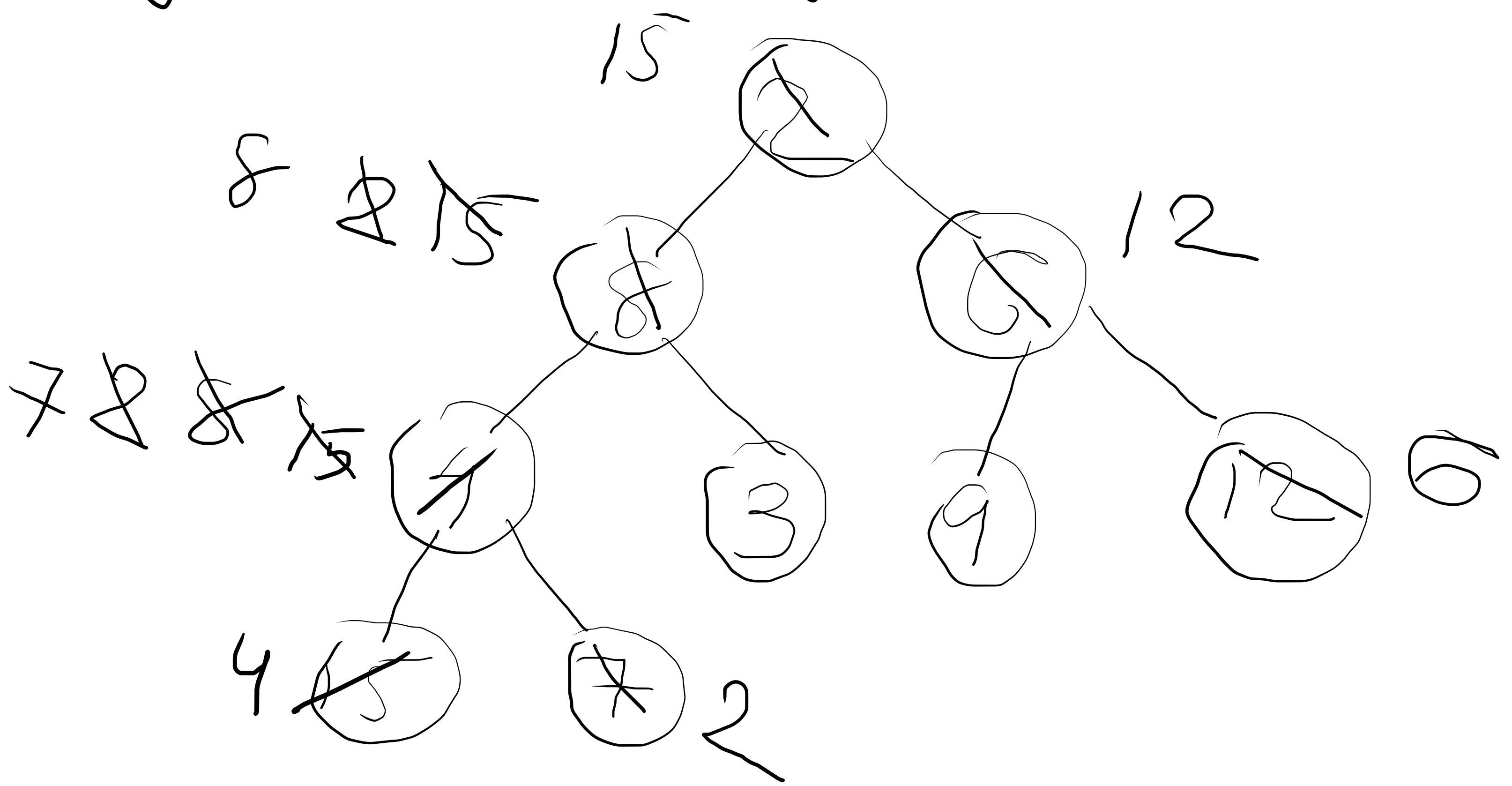
80	70	40	50	60	8	6	10	5	2

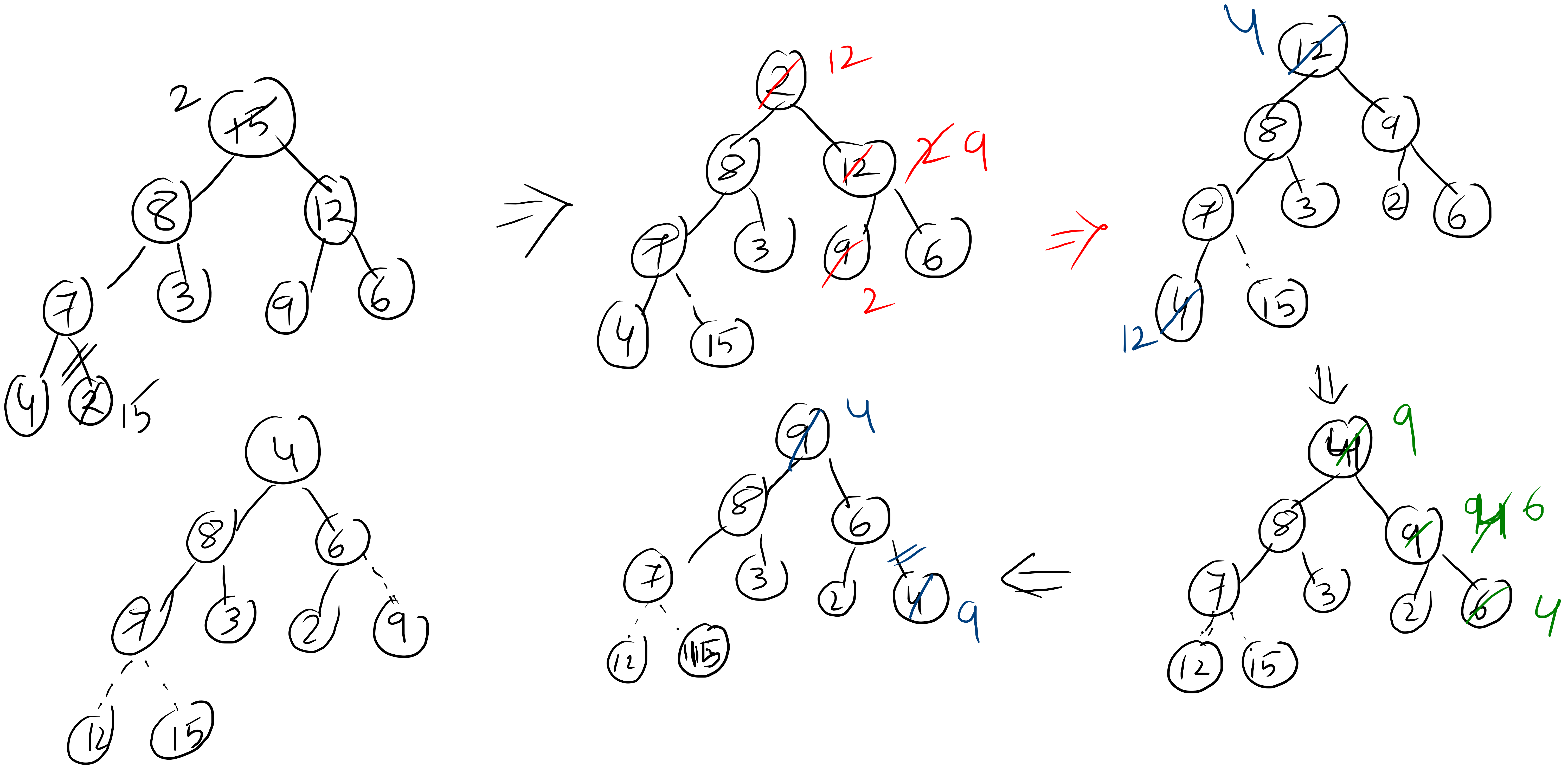


```
for i = 1 to length(A) - 1
    Swap A[1] with A[heapsize[A]]
    heapsize(A) = heapsize(A) - 1
    call maxheapify(A, 1)
```

A = 2 8 6 4 3 9 12 15 7

Sort A using heapsort algorithm.




2 15 8 12 7 3 9 6 4 2 15
2 12 8 12 2 9 7 3 9 2 6 4 15
4 12 8 9 7 3 2 6 4 12 15
4 9 8 9 9 4 6 7 3 2 6 4 12 15
9 4 8 6 7 3 2 4 9 12 15
4 8 6 7 3 2 9 12 15

```
Heapsort (A)
 Build-max-heap (A)  ———————  O(nlogn)
 for i = 1 to length(A) - 1  ———————  n times
     swap A[1] and A[heapsize(A)]
     heapsize(A) = heapsize(A) - 1
     max-heapify (A, 1)  ———  O(logn)
```

Total time: $O(n\log n)$

Priority queue

A data structure have the following operations is called a priority queue data structure.

insert.

i) Insert (S, x) : insert the element x into the set S.

minimum

ii) Maximum (S) : Returns the element of S with largest key

Extract-min

iii) Extract_max (S) : Return the element of S with largest key and remove it from S.

Decrease-key

iv) Increase_key (S, x, K) : Increase the value of x's key to a new value K.

Array as a priority queue

Insert (A, x) —— $\Theta(1)$

Maximum (A) —— $O(n)$

Extract-max (A) —— $O(n)$

Increase-key (A, x, k) —— $\Theta(1)$

Sorted array as a priority queue

2	5	8	10	15	30	45

1> $O(n)$

2> $\theta(1)$

3> $\theta(1)$

4> $O(n)$

Linked List as a priority queue