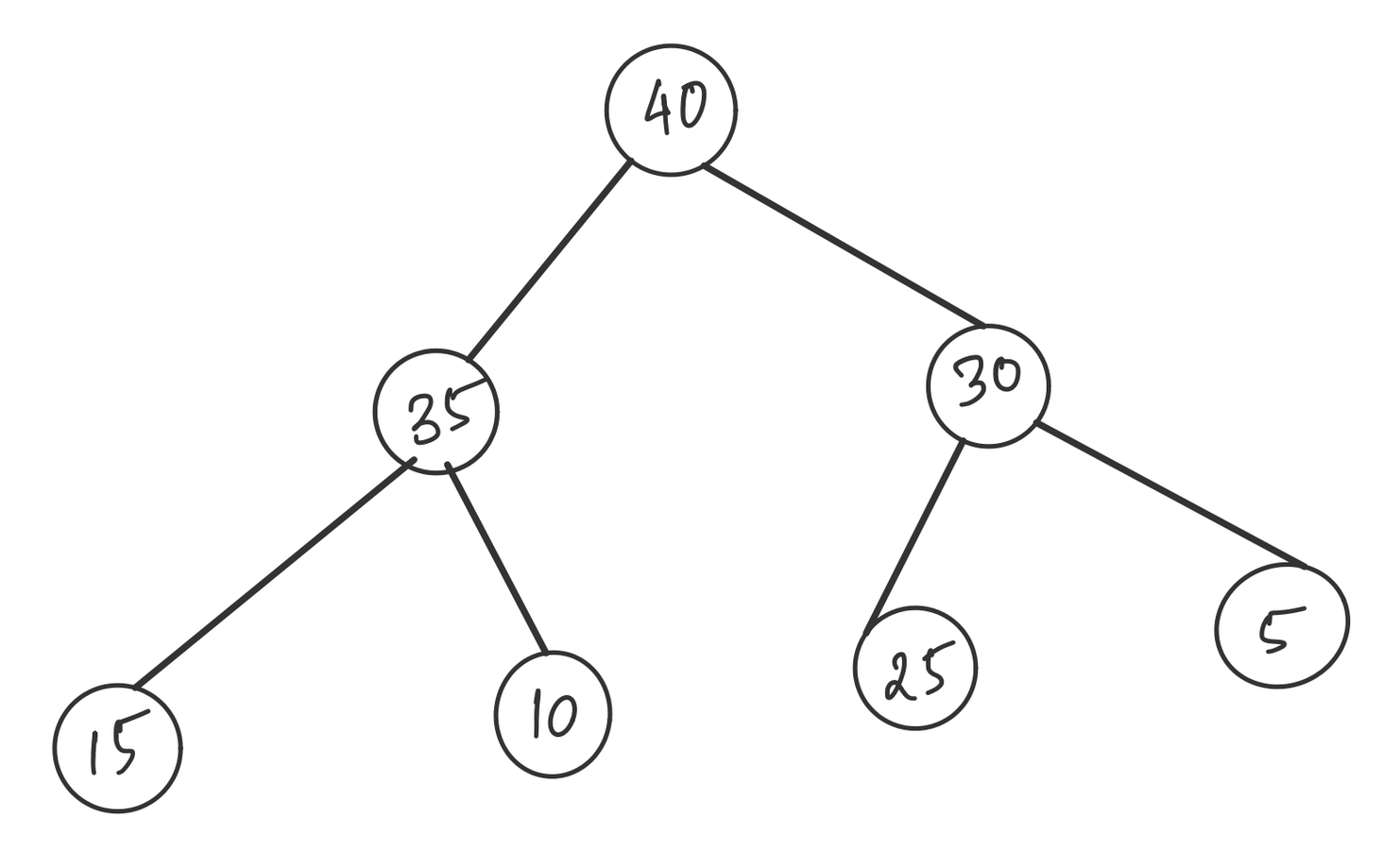
Deletion from heap and heap sort

05/06/22 2:19 AM



A

40	35	30	15	10	25	5
1	2	3	4	5	6	7

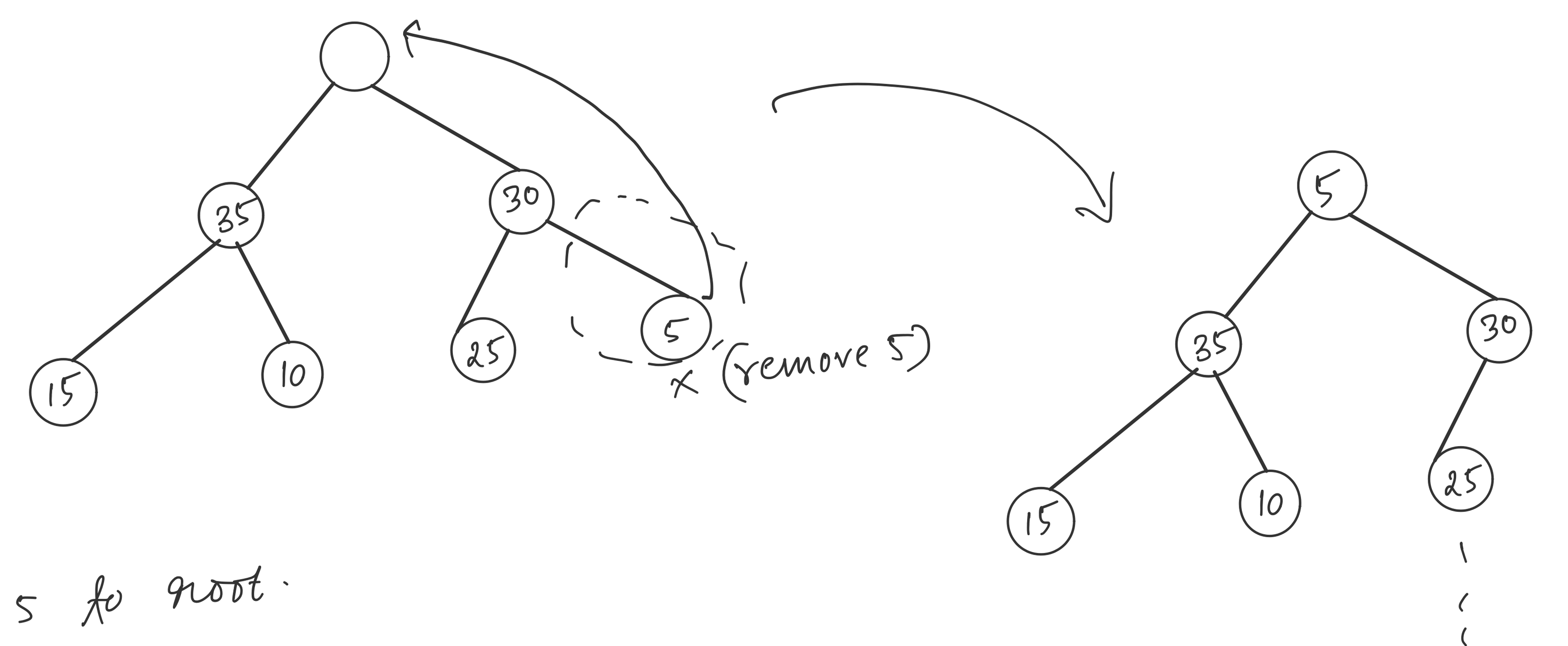
Note ⇒ from heap we can delete only root (i.e the largest element in case of max heap).

It means we can delete highest priority element.

delete 40 ⇒

Step 1 ⇒ store 40 in a temporary variable and delete 40 from root.

Step 2 ⇒ 35 should take its place but this will make the binary tree incomplete.

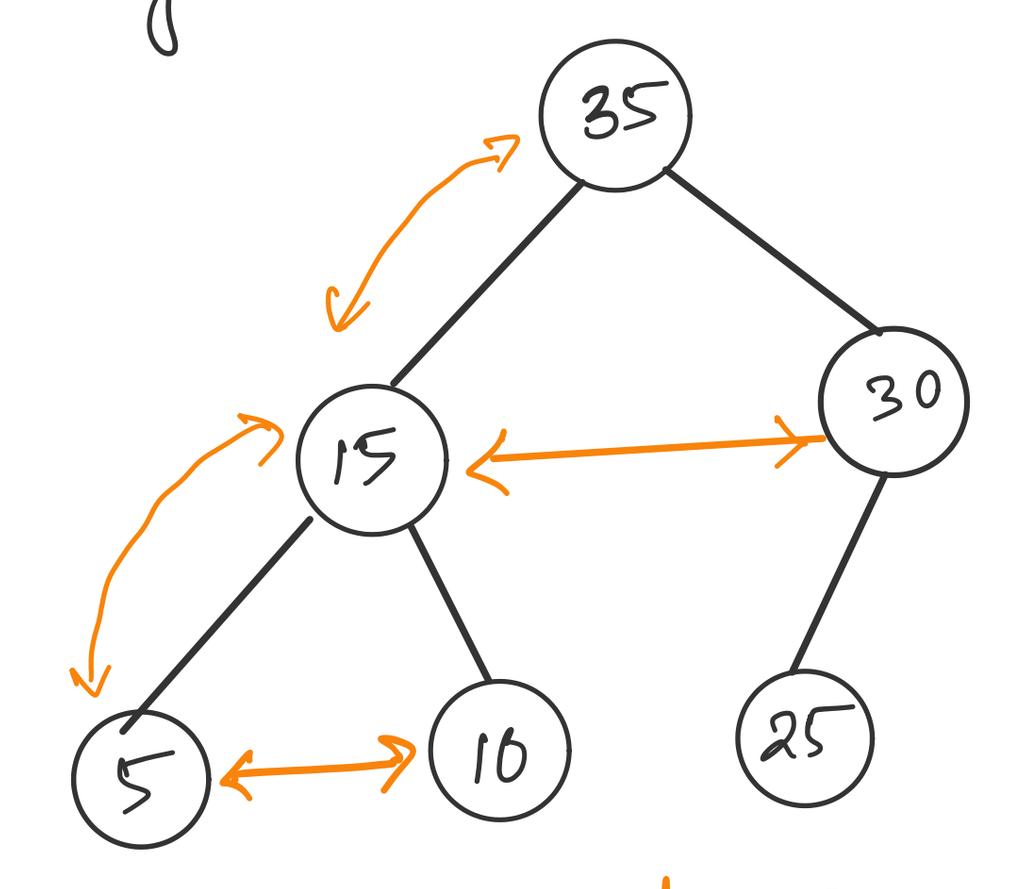


Step 2 ⇒ bring 5 to root.

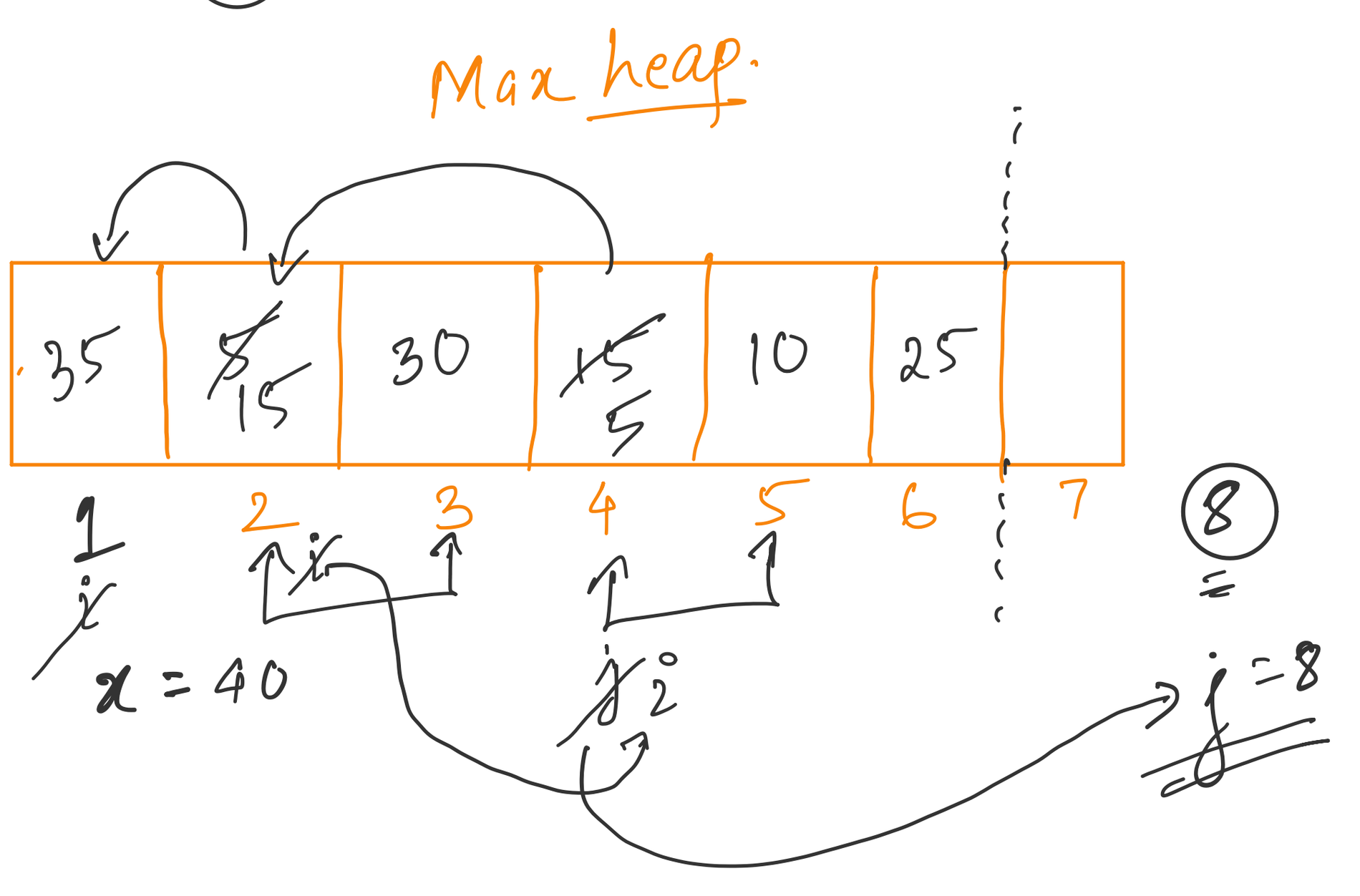
A

5	35	30	15	10	25	
1	2	3	4	5	6	7

Step 3 ⇒ to rearrange and make it a max heap compare the children of new root i.e 5. (35 and 30).
Now 35 > 30 ; compare 35 with 5 interchange 5 and 35. Now again compare children of 5 i.e 15 and 10
15 > 10 compare 15 and 5 and interchange them as 5 < 15.



Max heap.




```
void Delete (int A[]; int n)
{
    int x, i, j;
     x = A[n];
     A[1] = A[n];
     i = 1; j = 2 * i;
    while( j < n-1)
  {
    if (A[j+1] > A[j])
        j = j+1;
    if (A[i] < A[j])
       Swap(A[i], A[j]);
         i = j;
         j = 2*j; }
       else
         break; }
```

Heap sort ⇒ whenever we delete an element from the heap we get a free space at the end now if we go on storing the deleted element in this free space than our array becomes sorted and this is called heap sort.

Implementation ⇒

```
void Delete (int A[]; int n)
{
    int x, i, j;
     x = A[n];
     A[1] = A[n];
     i = 1; j = 2 * i;
    while( j < n-1)
  {
    if (A[j+1] > A[j])
        j = j+1;
    if (A[i] < A[j])
       Swap(A[i], A[j]);
         i = j;
         j = 2*j; }
       else
         break; }
    A[n] = x; }
```

Heap sort ⇒

1. creat Heap of n elements
2. delete n elements one by one

Time complexity ⇒ $O(n \log n)$
$(2n \log n)$