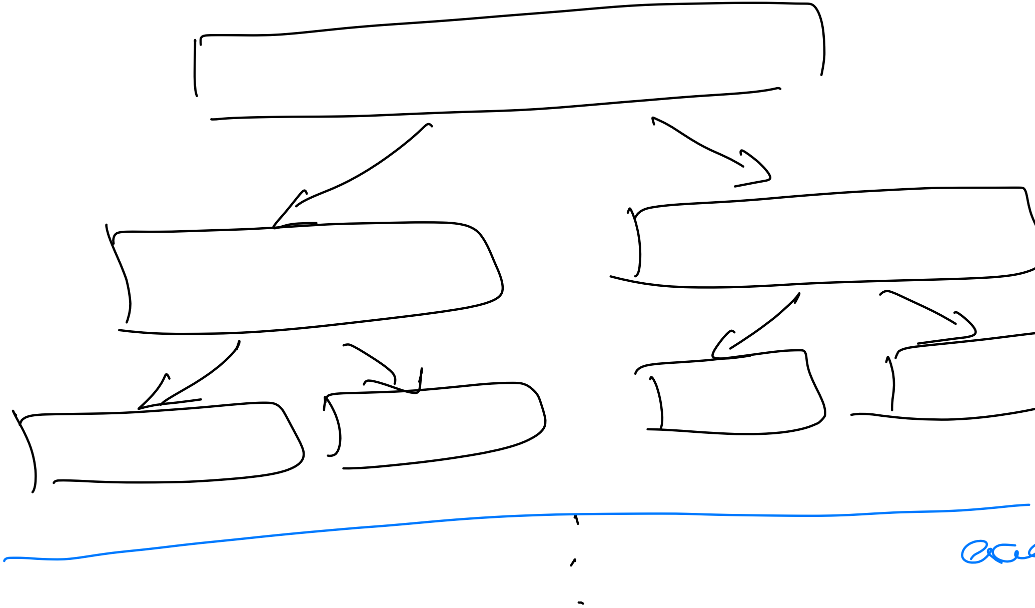


Size of each block is K

⇓

Sort each block using insertion sort

Steps of Merge

	1	2	3	4	8	10	15	20
arr	~~1~~	~~4~~	~~8~~	~~10~~	~~2~~	~~3~~	~~15~~	~~20~~
	0	1	2	3	4	5	6	7

i → ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4

start1 → 0
end1 → 3

j → ~~4~~ ~~5~~ ~~6~~ ~~7~~ 8

start2 → 4
end2 → 7

k → ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ 8

	0	1	2	3	4	5	6	7
result	1	2	3	4	8	10	15	20

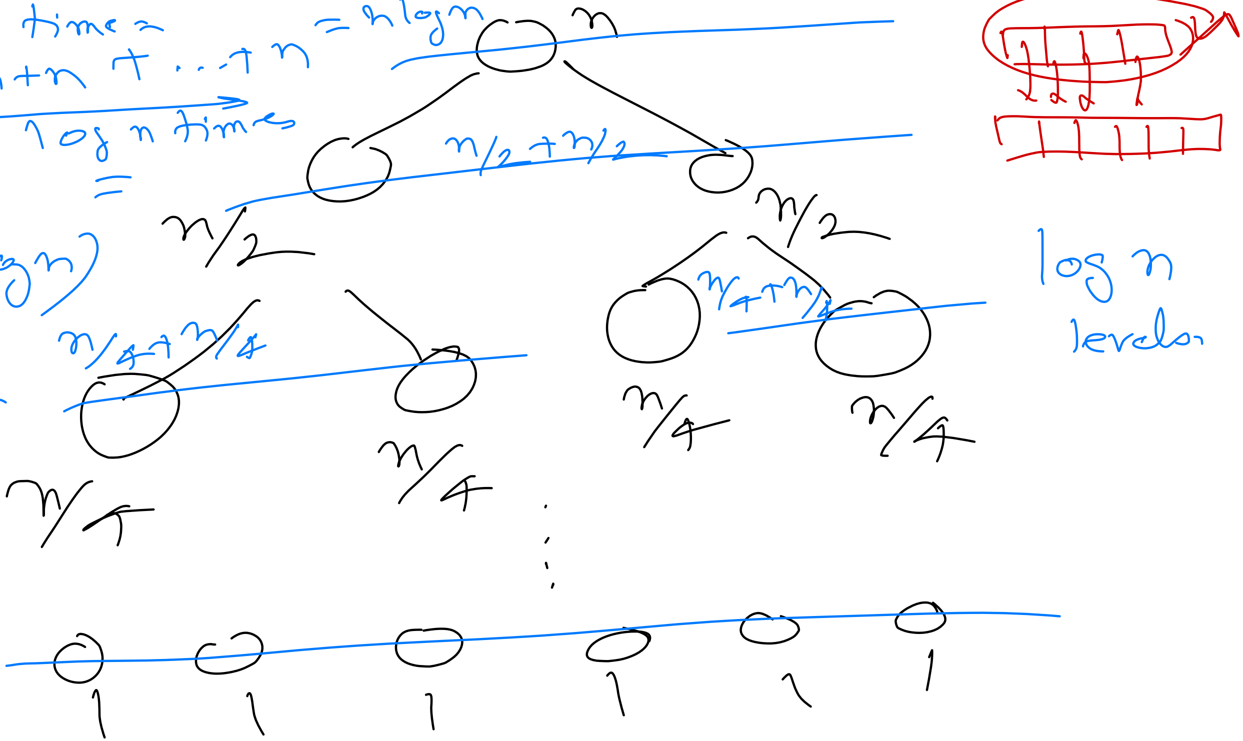
Total time = $n + n + n + \dots + n = n \log n$

$\underbrace{\qquad\qquad\qquad}_{\log n \text{ times}}$

$n =$

$O(n \log n)$

$n =$



log n levels.

Space Complexity

Selection Sort $\Rightarrow$ j, max, i $= 3$

$\approx O(1)$

Merge of two sorted arrays of size n & m $\Rightarrow$ i, j, k, result array of size
some elements

$\Downarrow$

Size $= n+m$

$O(n)$

Quick Sort $\Rightarrow$ Pivot, left, right

$\approx O(1)$

$O(\log n)$

Merge Sort → log n for system stack for recursive calls

n ← Merge.

$O(n + \log n)$

Worse case time complexity

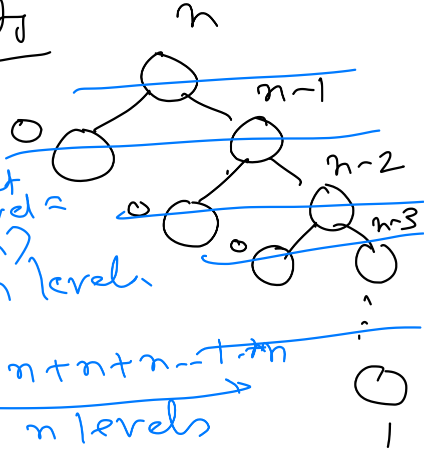
[① 2 3 4]

[] 1 [② 3 4]

1 [] 2 [3 4]

Quick Sort

$O(n^2)$ $(n-1)$ $\frac{n+n+n \dots + n}{n \text{ levels}}$

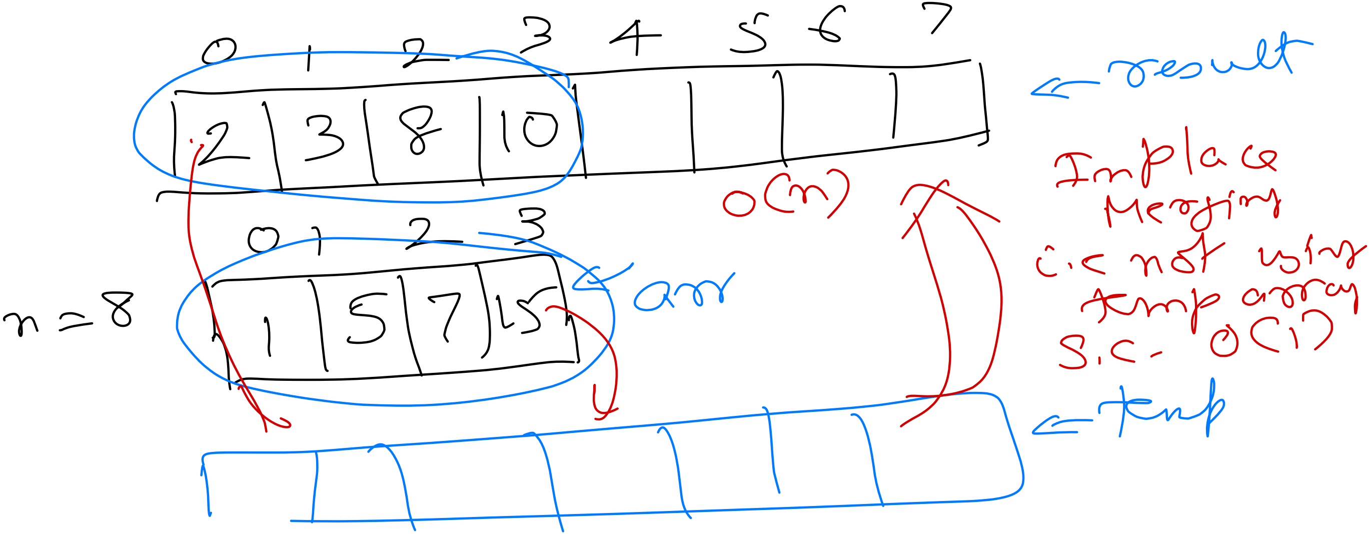


work done at each level = O(n)

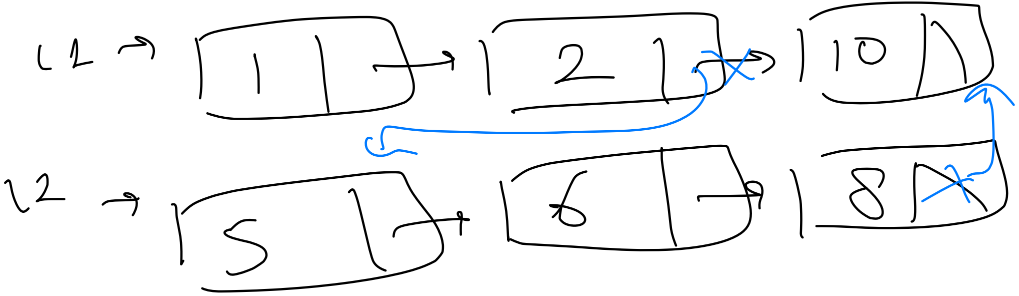
n level.

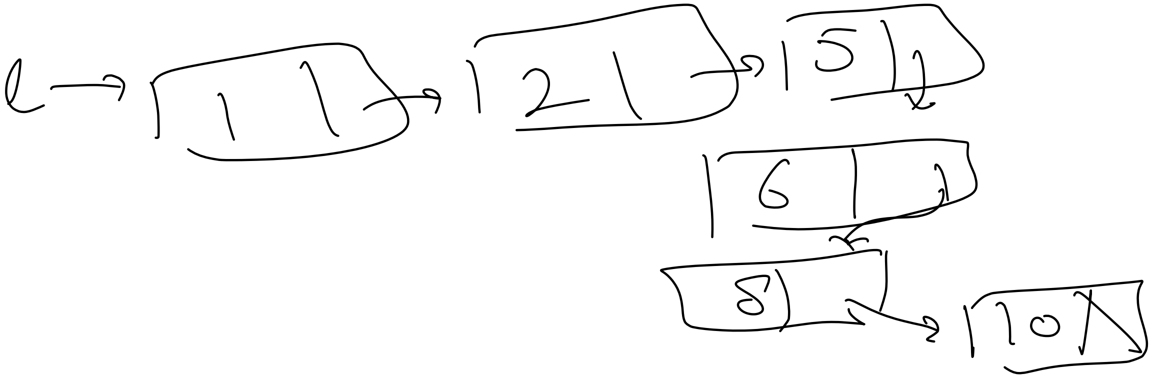
Sort a linked list ⇒ Merge Sort

External Merge Sort ⇒ Sorting data stored in file.



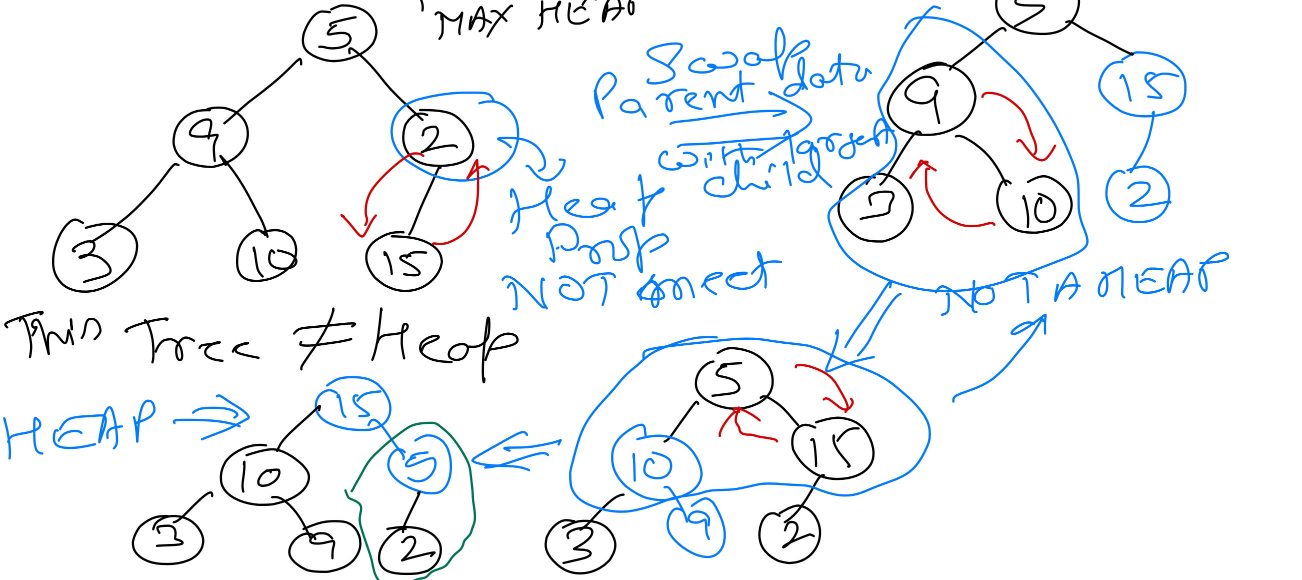
Inplace Merging
i.e not using temp array
S.C - O(1)

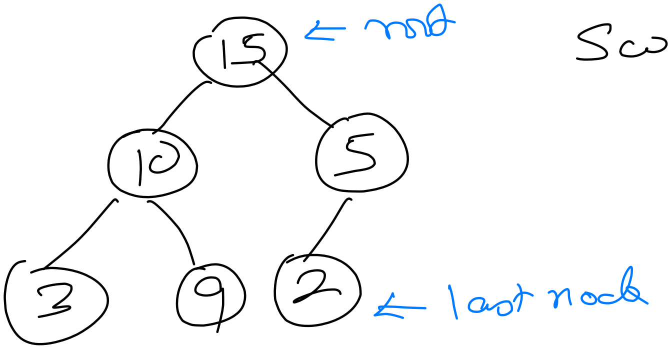

l1
1
2
10
l2
5
6
8


l
1
2
5
6
8
10

Heap Sort

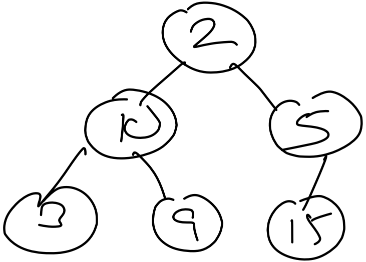
Heap Property => Parent's data > MAX(child's data)
for MAX HEAP



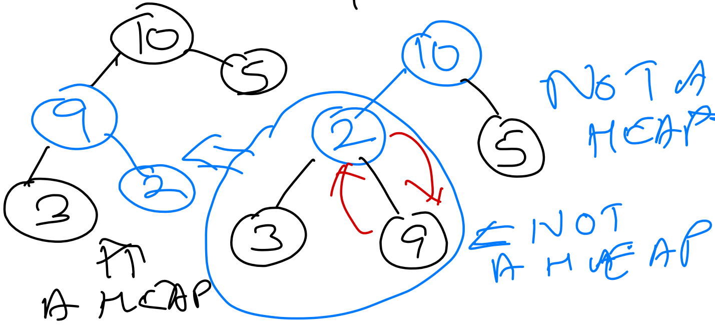


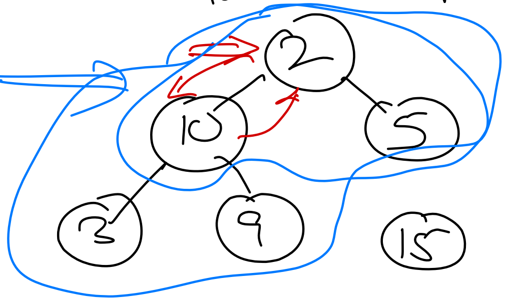
Heap

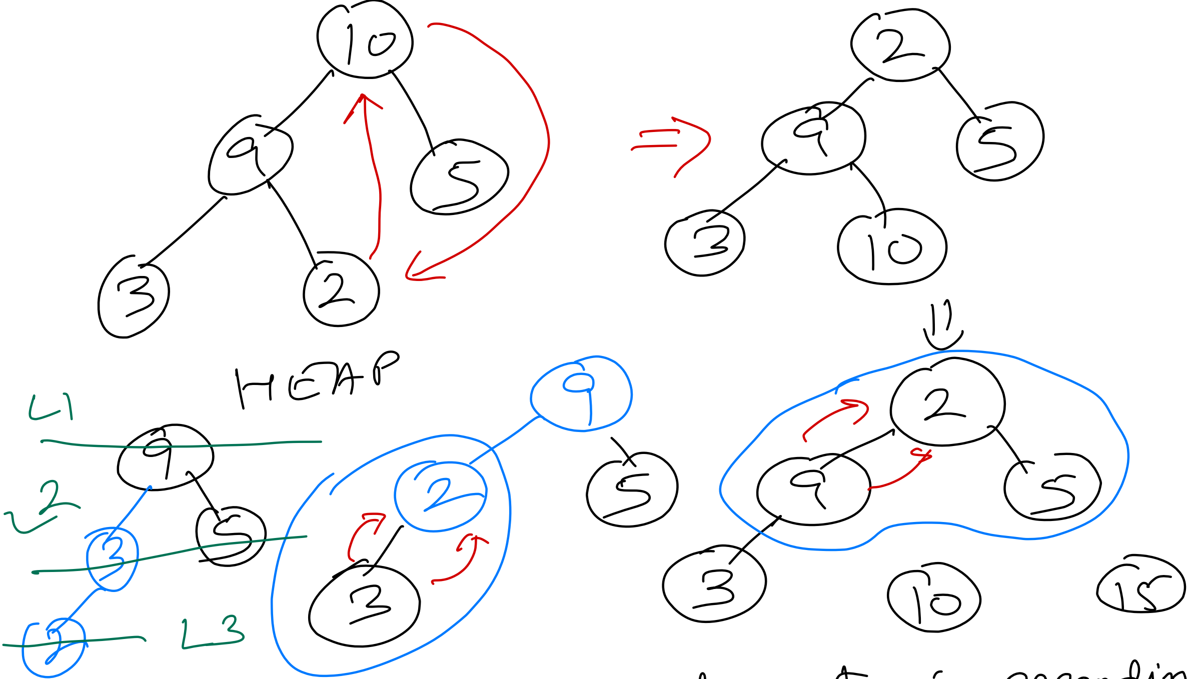
Swap root with last node.



Remove last node from heap/tree.







Max Heap → Sort elements in ascending order

Min Heap → 〃 descending order

Property for MIN HEAP

⇓

Parent's data < MIN (child's data)

---

i^{th} elem

lchild / \ rchild

$2 \times i$ $2 \times i + 1$

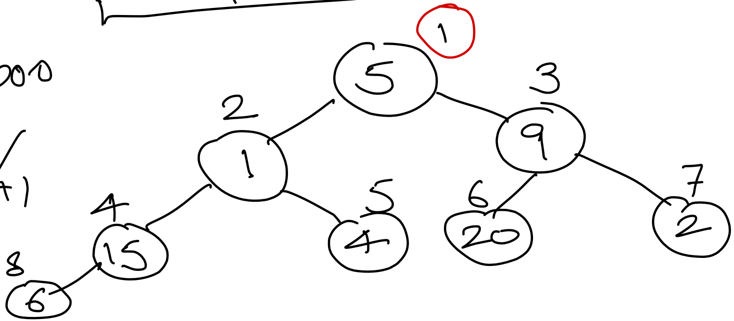
1	2	3	4	5	6	7	8
5	1	9	15	4	20	2	6

n elements

last parent node = $n/2$

$n/2$ pos

$$2 \times \frac{n}{2} \Downarrow n \qquad \cancel{2 \times \frac{n}{2} + 1 \Downarrow n+1}$$



```
HeapSort (arr, n)                ARRAY STARTS AT 1
1. for (i = n/2 ; i >= 1 ; --i)
       1.1 convertToHeap (arr, i, n)
2. for (i = n; i > 1; --i)
       2.1 Swap root element with
           last node/elem
           // Swap ( 1st elemt, i th elem )
   2.2 convertToHeap (arr, 1, i)
```

ROOT = 1st elem

```
ConvertToHeap( arr, root,  n  )
0:                                       if root
1.  maxChild = 2 * root                  is a
2.   maxChildVal = arr[maxChild]         leaf
3.  rChild = 2 * root + 1                node
4.   if ( rChild <= n )                  then
       4.1 (arr[rChild]] > maxChild      STOP
                             Val
            4.2  maxChildVal =           2*root > n
                     arr[rChild]
            4.3 maxChild = rChild
                 > arr[root])
5.  if ( maxChildVal
```

$\boxed{i^{th}}$ = Parent

$2 \times i$ & $2 \times i + 1$ = child

Max elements = n

$$2i = n \Rightarrow i = \frac{n}{2}$$

$$2 \times i \boxed{+1} = n$$

$$\boxed{i} = \frac{n-1}{2} \approx \frac{n}{2}$$

5.1 Swap(root's val, max child's val)

5.2. ConvertToHeap(arr, maxchild, n)