randomNo [start, end]

Using Random No as Pivot

→ pivotPos = randomNo % (end - start + 1)

0 .. (end - start)

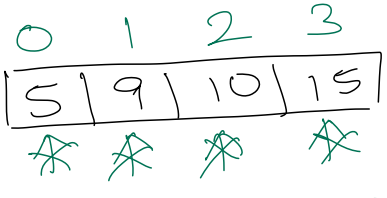
→ pivotPos = pivotPos + start

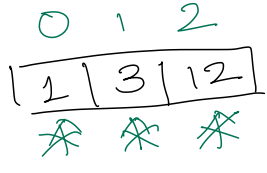
pivot ∈ [start, end]

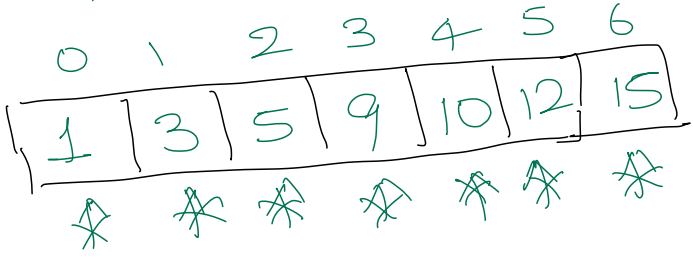
start .. en

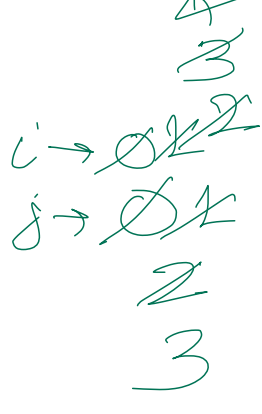
Merge Sort

→ Merge two sorted set of elements to arrange them all together & sorted.









MergeTwoSortedArrays(elements1, elements2, result)
- Set i to first element of elements1
- Set j to first element of elements2
- while (i < size of elements1) and (j < size of elements2) do
  - if (elements1's i elements < elements2's j element) then
    - append i'th element to result.
    - Move i to next element.
  Else
    - append j'th element to result.
    - Move j to next element.

X times

// Copy remaining elements from group that is still having some left to result
- while (i < size of elements1) do
  - append i'th element to result
  - Move i to next element.

N - X

OR

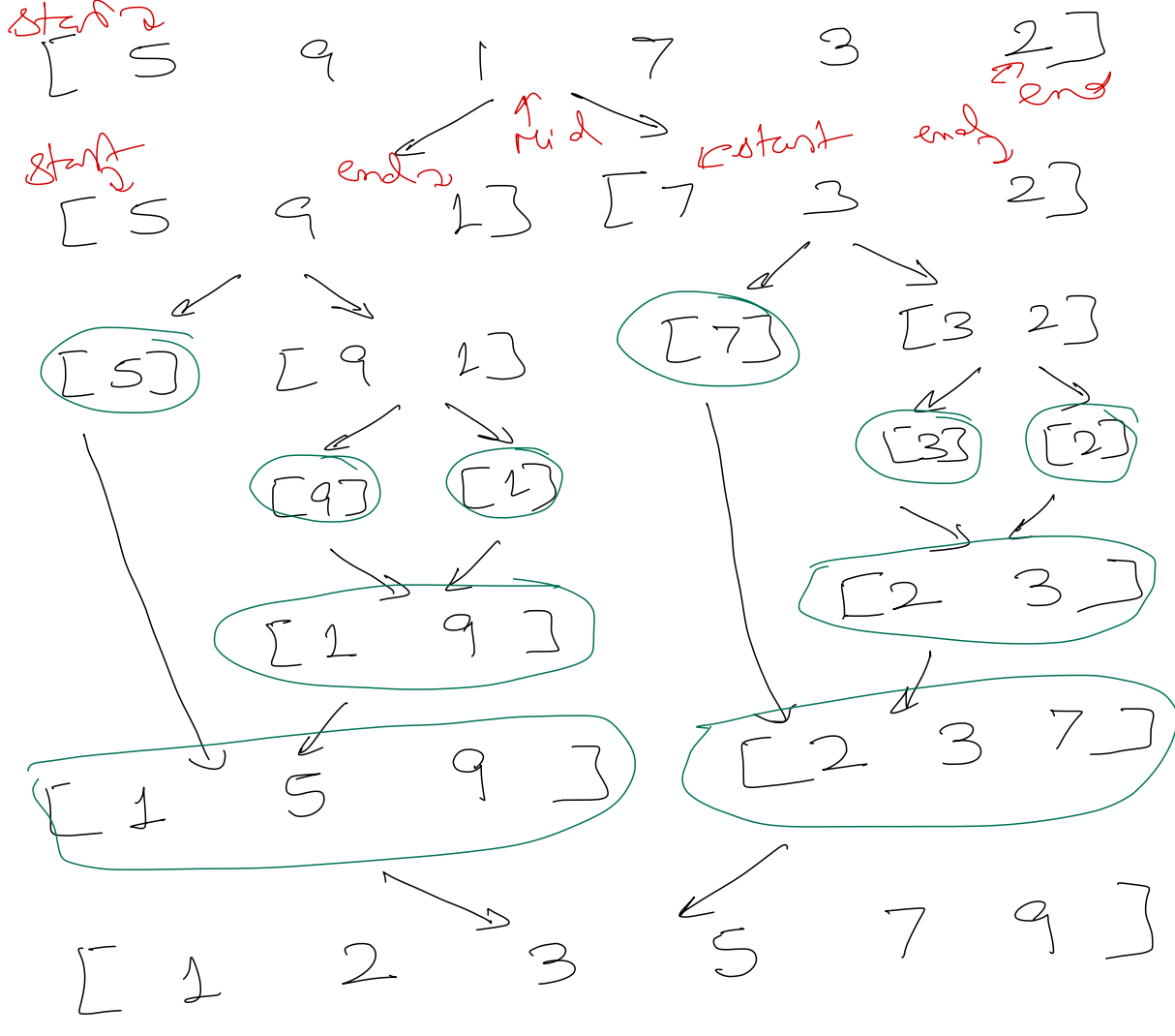
- while (j < size of elements2) do
  - Append j'th element to result
  - Move j to next element.

M - X

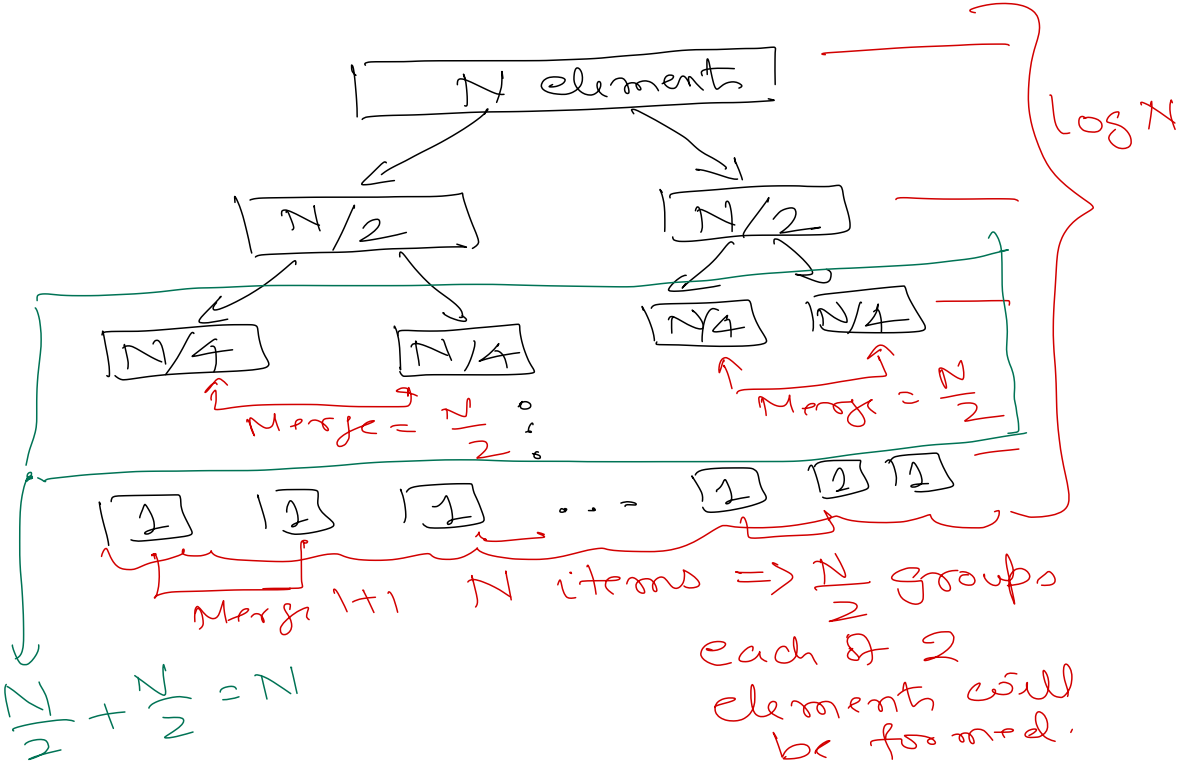
elements1 has N elements
elements2 has M elements
result has (N+M) elements

N+M

$\Rightarrow O(N+M)$

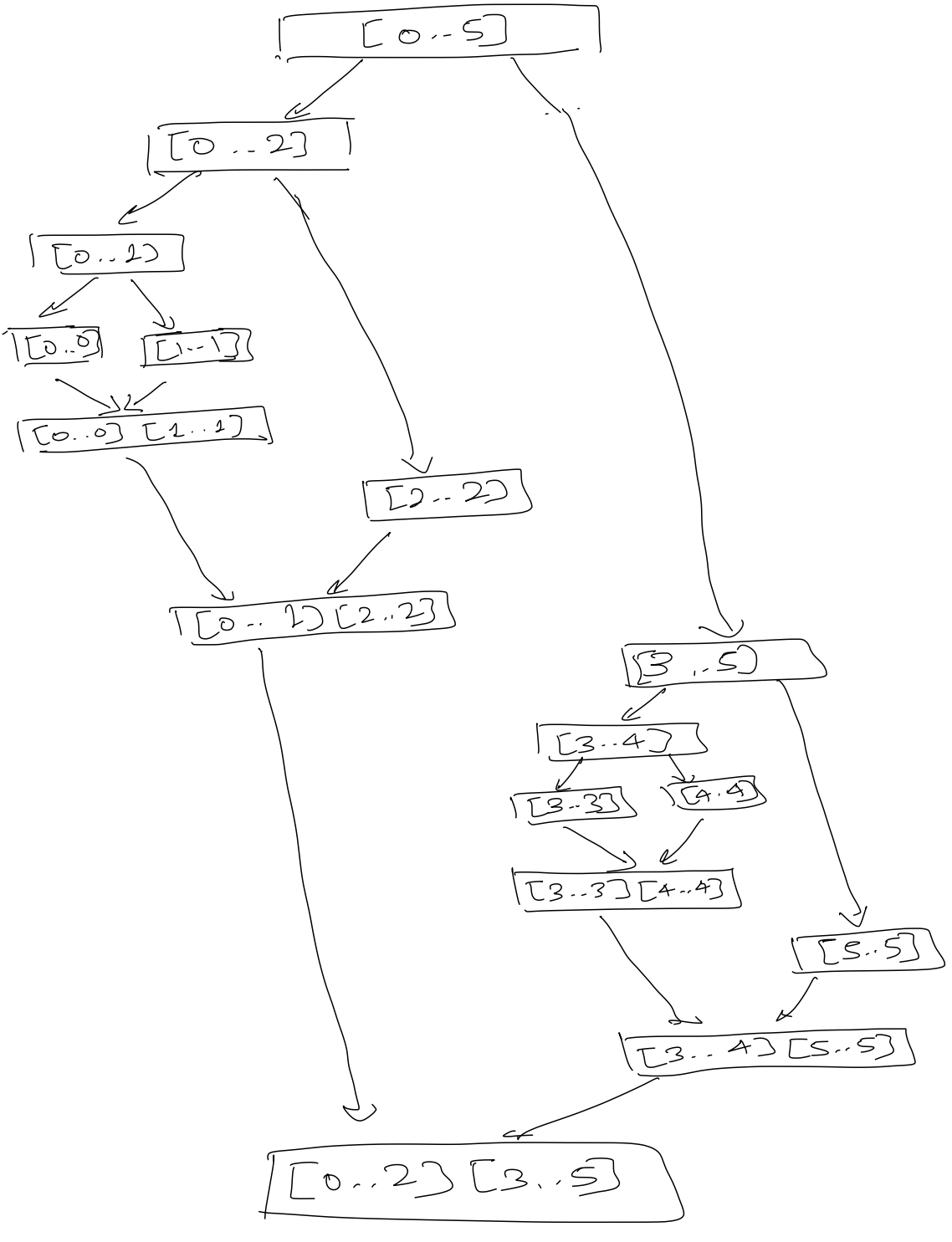

start
5 9 1 7 3 2
end
Mid
start
end
start
end
5 9 1
7 3 2
5
9 1
7
3 2
9
1
3
2
1 9
2 3
1 5 9
2 3 7
1 2 3 5 7 9

MergeSort(elements, start, end)
- Size of elements is 1 then
  - Stop.
- Mid = Size of elements / 2
- MergeSort(elements, start, Mid)
- MergeSort(elements, Mid + 1, end)
- Merge(elements, start, Mid, Mid + 1, end)



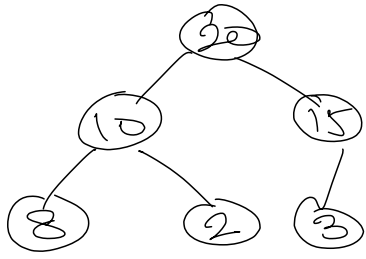
Total work
$= \underbrace{N + N + \cdots + N}_{\log N \text{ times}}$

$\Rightarrow O(N \log N)$


[0..5]
[0..2]
[0..1]
[0..0]
[1..1]
[0..0] [1..1]
[2..2]
[0..1] [2..2]
[3..5]
[3..4]
[3..3]
[4..4]
[3..3] [4..4]
[5..5]
[3..4] [5..5]
[0..2] [3..5]

Heap Sort

Heap



- Max Heap
- Min Heap

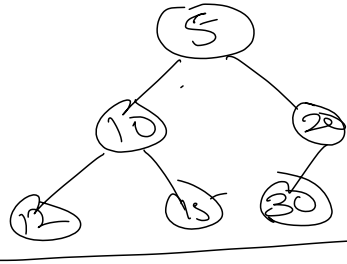
Max Heap Property

parent > (left child, right child)

Max Heap ⇒ Heap, when all nodes satisfy Heap property.

Min Heap Property

parent < (left child, right child)

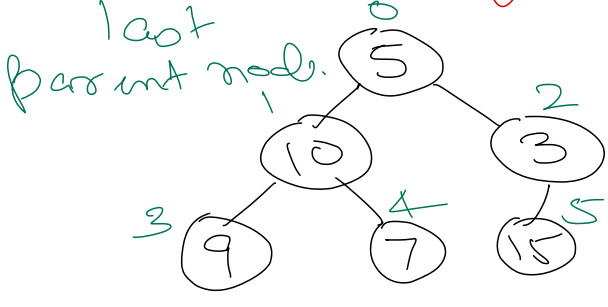


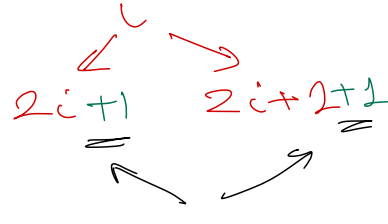
⇒ Min Heap.

0	1	2	3	4	5
5	10	3	9	7	15

last parent node.

⇓ Tree structure

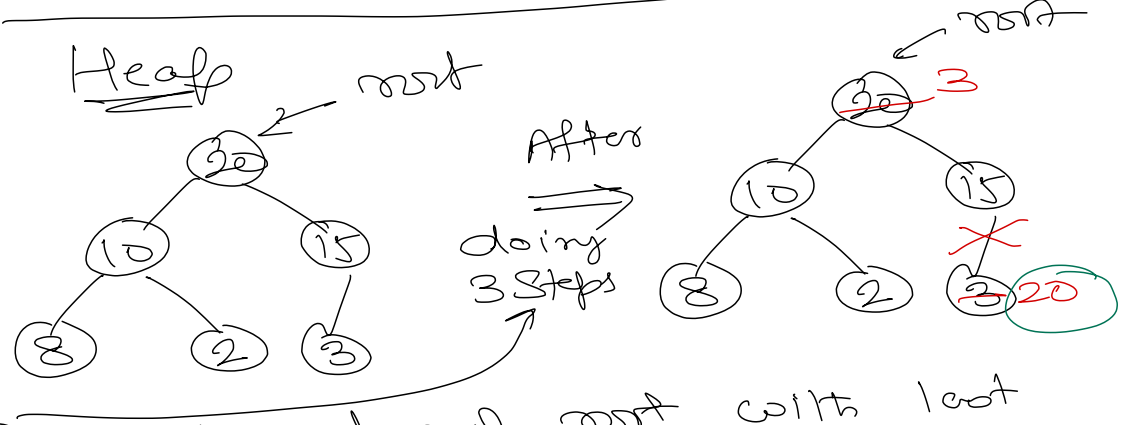




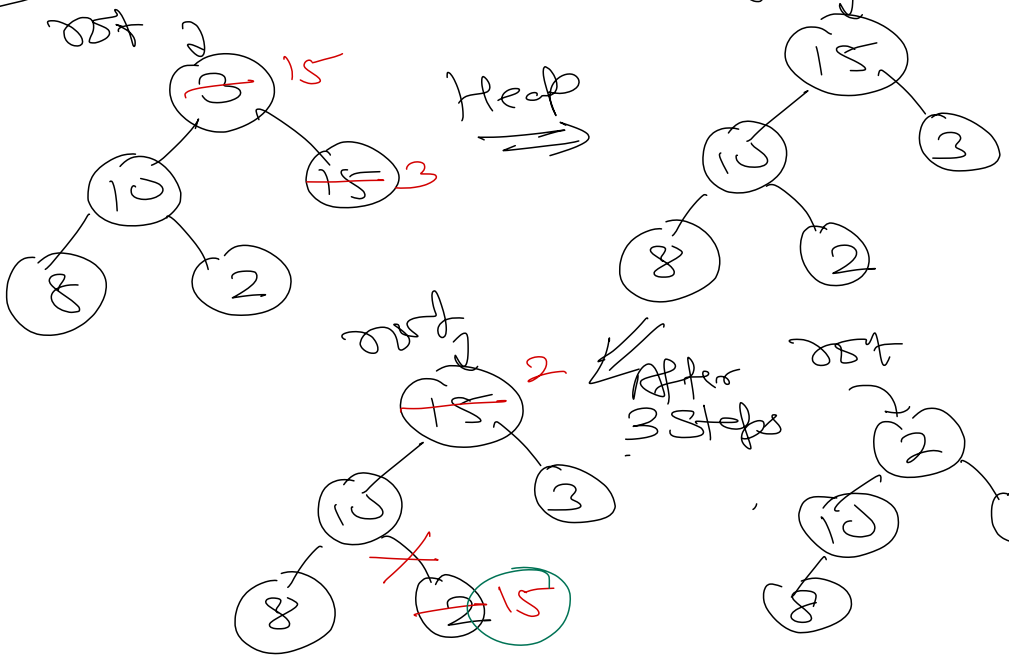
As array index starts from 0 in JAVA.

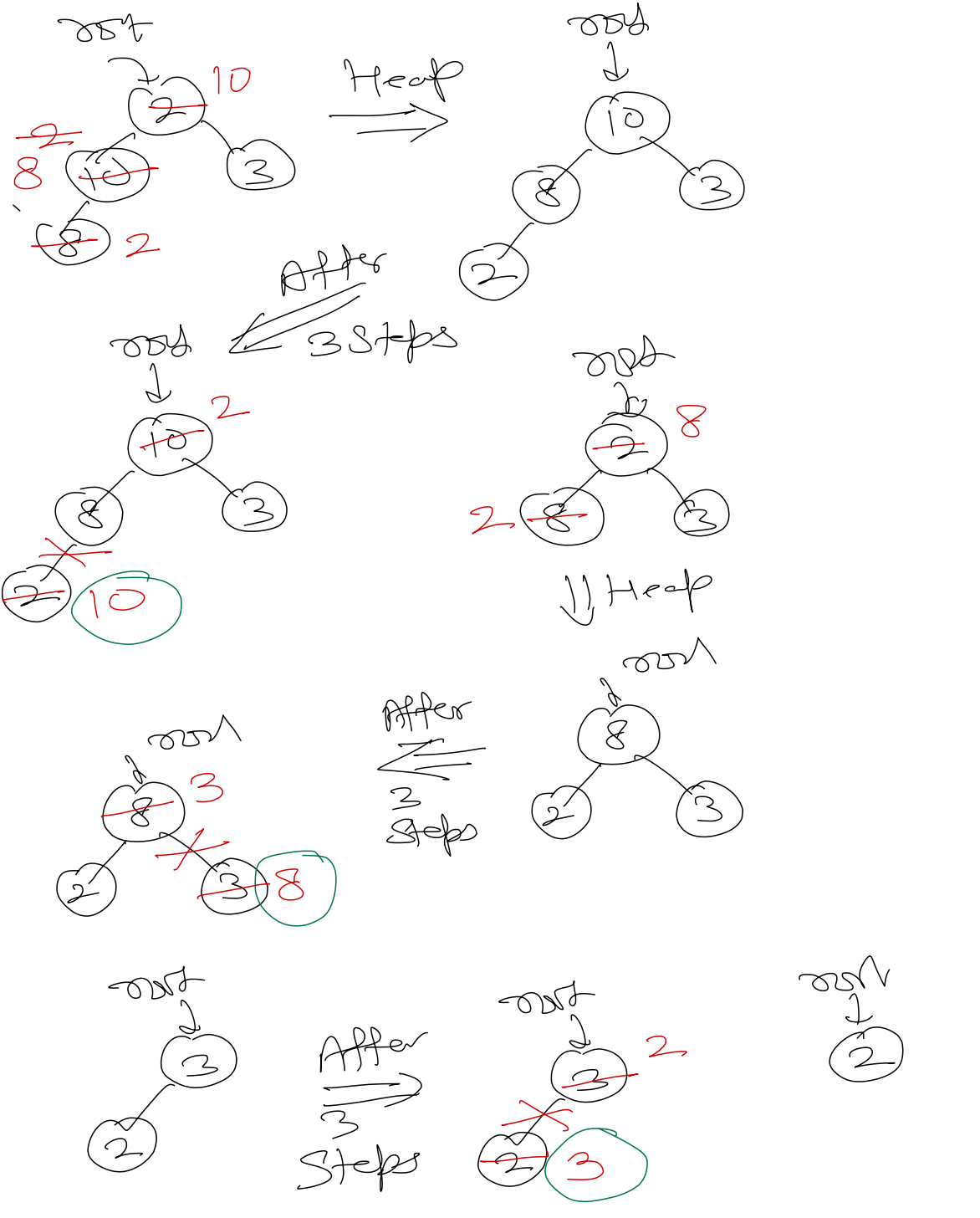
If array has N elements, where will be the last parent node?

$\Rightarrow N/2^{th}$ element.



- Swap value of root with last leaf node.
- Remove last leaf from tree
- Make tree into heap.



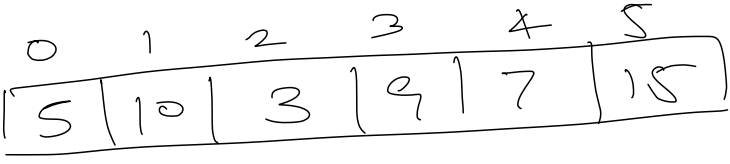


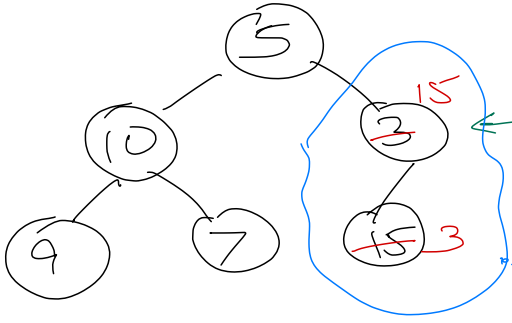
Sorted elements are: 2 3 8 10 15 20

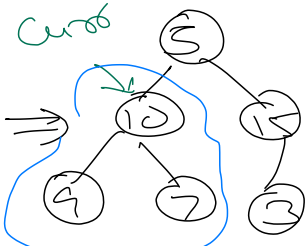
Heap'Sort(elements) $\Rightarrow O(N \log N) + O(N \log N) \Rightarrow O(N \log N)$

- ConvertToHeap(elements) $\rightarrow O(N \log N)$
- while (elements have more than 1 element) $\rightarrow N$ times
  - Swap (first element, last element). $\rightarrow 1$
  - Remove last element. $\rightarrow 1$
  - MakeHeap(elements, first position) $\rightarrow O(\log N)$

$O(N \log N)$

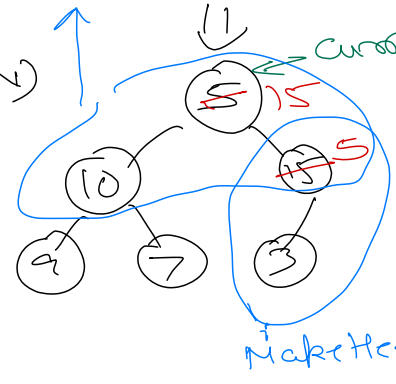






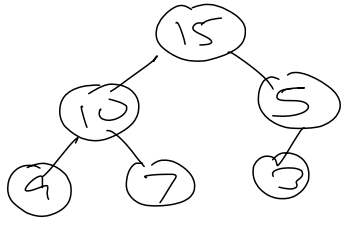
ConvertToHeap(elements) $\Rightarrow O(N \log N)$

- Set current to last parent.
- While current is a parent do
  - MakeHeap(elements, current)
  - Set current to previous parent.



$\frac{N}{2}, \left(\frac{N}{2}-1\right), \left(\frac{N}{2}-2\right) \ldots 1$

$\frac{N}{2}$ times.



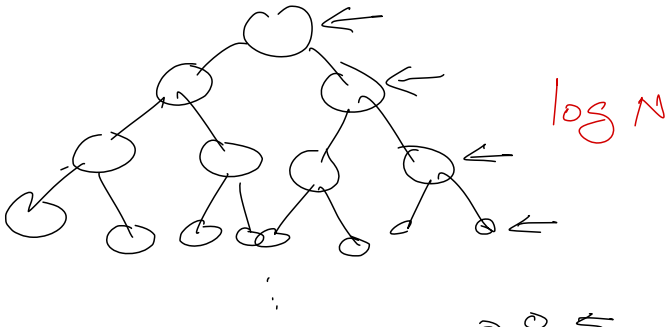
$O(N \log N)$

$\log N$ times Operations $\Rightarrow$ Total work done $= \log N + \log N + \ldots + \log N$ ($N/2$ times)

$N/2$ times

MakeHeap(elements, current)
- maxChild = child of current having largest value.
- if maxChild value > parent then
  - Swap value of maxChild and parent
  - if maxChild is not a leaf node then
    - MakeHeap(elements, maxChild)

$\Rightarrow O(\log N)$



	Time Complexity	Space Complexity
Quick Sort	$\Rightarrow O(n \log n)$	$O(\log n)$
Can be parallelized	$O(n^2)$ ← worse case	$\rightarrow O(n)$
	For merging $\rightarrow$	$O(n)$
Merge Sort	$\Rightarrow O(n \log n)$	$+ O(\log n)$
↳ Can do external sorting.		
Heap Sort	$\Rightarrow O(n \log n)$	$O(1 \log n)$

Heap { Priority Queue
Merge N sorted arrays / list

Bubble Sort $\Rightarrow O(n^2)$ $O(1)$

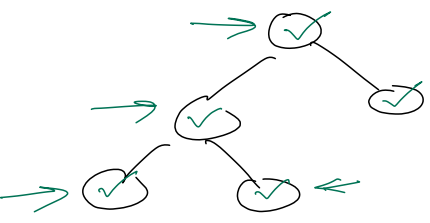
Selection Sort $\Rightarrow O(n^2)$ $O(1)$

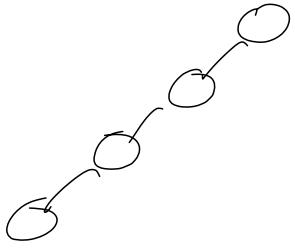
Insertion Sort $\Rightarrow O(n^2)$ $O(1)$

BST $\Longrightarrow O(\log n)$ $O(1)$

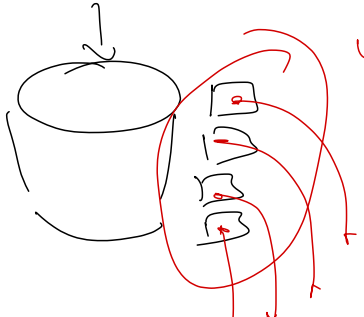
Inorder Traversal $\Rightarrow O(n)$ $O(\log n)$

worse case $\rightarrow O(n)$





100,000 records



External Merge Sort

break into smaller chunks on disk

load a chunk at a time in memory & do Quick Sort

→ Merge sort will merge sorted chunks.