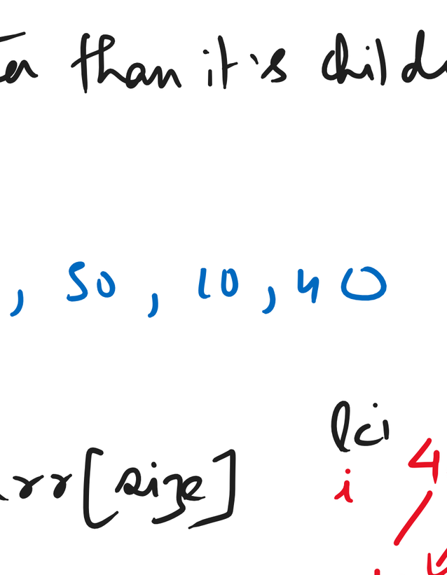
Heap Data Structure Implementation:→

arr = {2, 8, 6, 3, 4, 1}; Complete Binary Tree: Insertions always from left to right.

Max-Heap Structure:→

Heap:



-1	8	4	6	2	3	1
0	1	2	3	4	5	6

Sentinel value

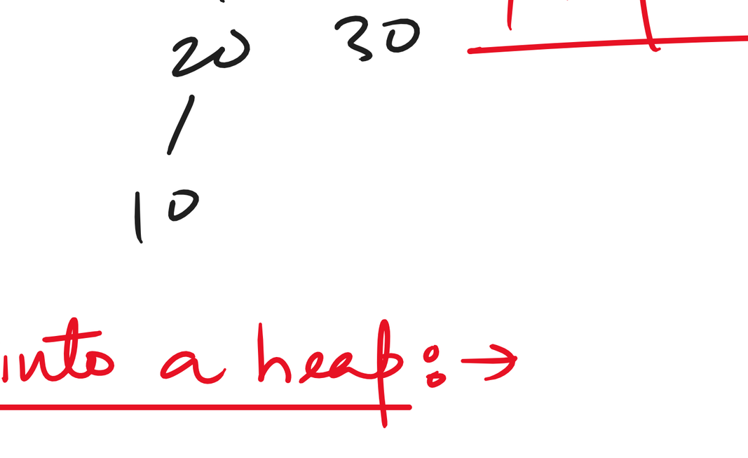
size ++
arr[size] = val;

For any index 'i'
left Child index = $2 * i$
right Child index = $2 * i + 1$

For any index 'i'
Parent Index = $\frac{i}{2}$

{For max heap:→
Everynode is greater than it's children nodes.}

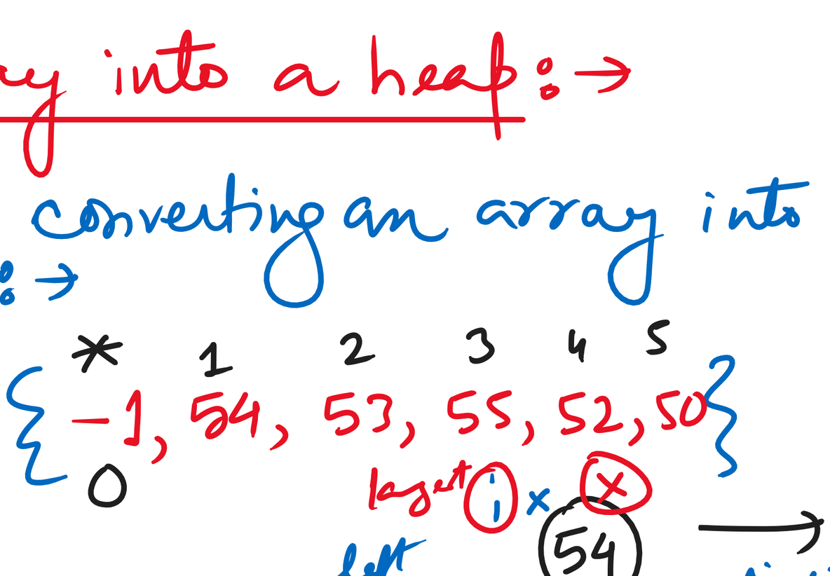
20, 30, 50, 10, 40



Step1: arr[1] = arr[size]
size --

print → 50, 40, 30, 10, 20

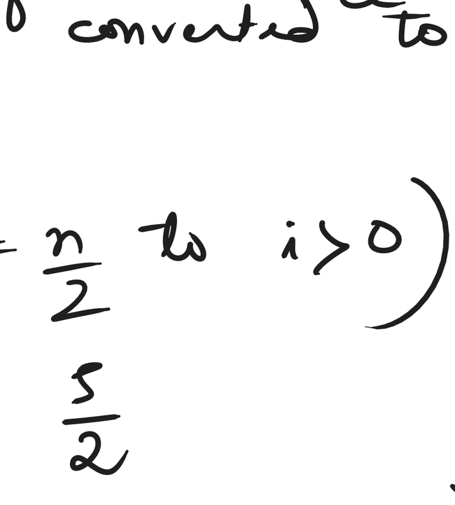
// progate the root to it's correct position



Convert an array into a heap:→

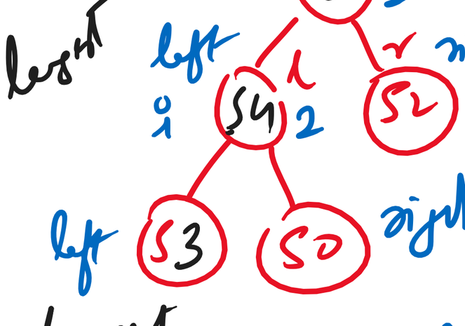
* The process of converting an array into a heap is called heapify:→

int arr = {-1, 54, 53, 55, 52, 50}
(indices: 0 for -1, then 1 2 3 4 5)
int n = 5



→ Not a max heap

No of leaf nodes = $\frac{n}{2} + 1$
$= \frac{5}{2} + 1$
$= 2 + 1 = 3$



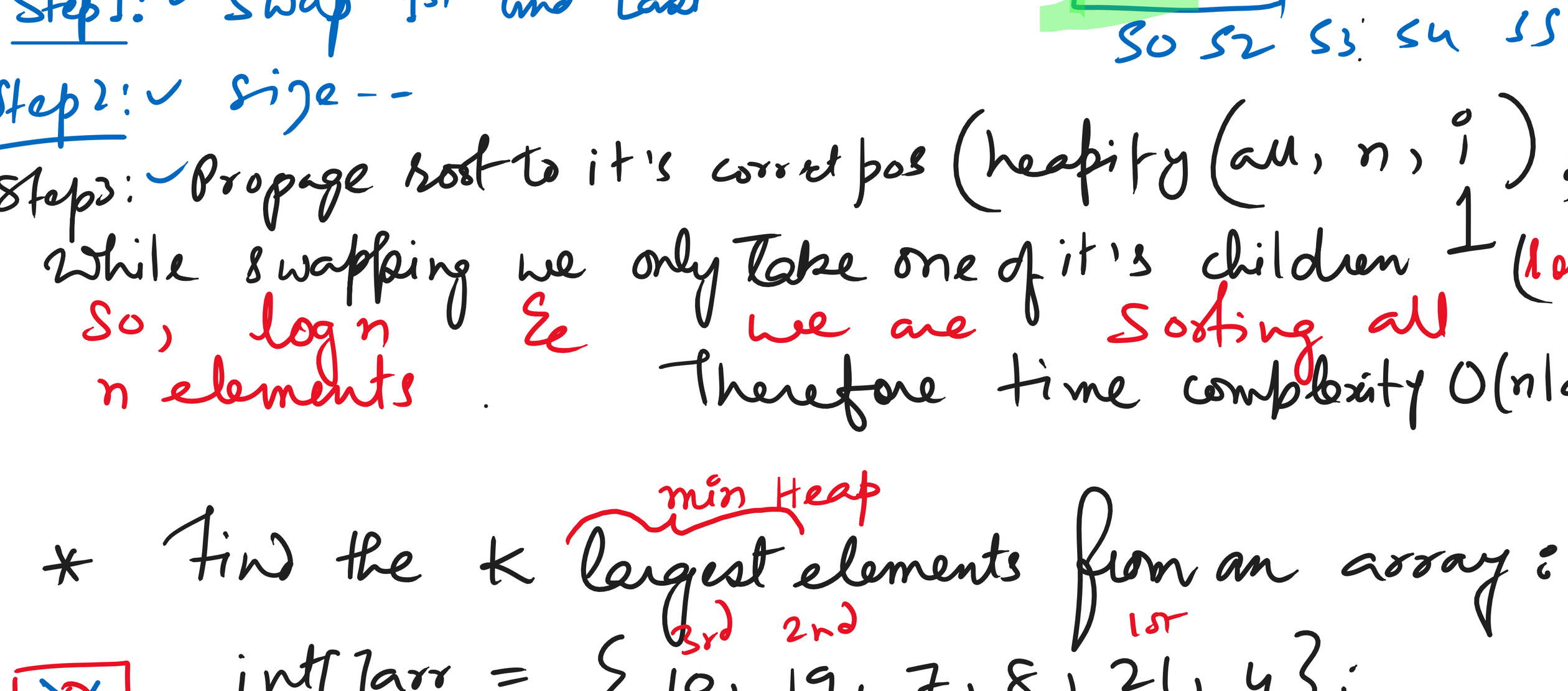
for ($i = \frac{n}{2}$ to $i > 0$)

No of non-leaf nodes = n − leaf nodes = n − 3 = 2

* We take only the non-leaf nodes & automatically the leaf nodes will be converted to heap.

($i = \frac{n}{2}$ to $i > 0$)
$\frac{5}{2}$
($i = 2$ to $i > 0$)
$i = 1$

* Heap Sort Algorithm:→



{Step1: ✓ swap 1st and Last
Step2: ✓ size --
Step3: ✓ Propage root to it's correct pos (heapify (arr, n, i));
While swapping we only Take one of it's children (log n)
So, log n & we are Sorting all n elements. Therefore time complexity O(n log n)

* Find the k largest elements from an array: (min Heap)

int[] arr = {10, 19, 7, 8, 21, 4}; (3rd: 10, 2nd: 19, 1st: 21)

k = 3

* Remove smallest elements from the min Heap till size > k.

4 ✗
7 ✗
8 ✗

size 5, 4, 3
3 == 3
(k)

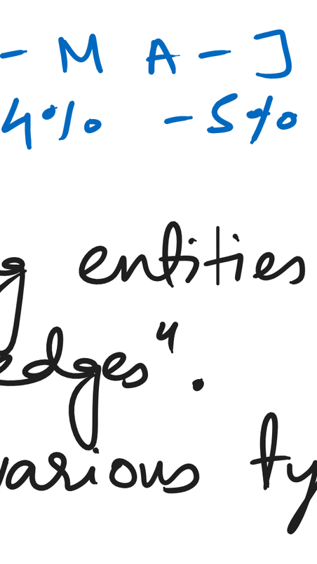
4
7
8

Find the kth (Smallest) element:

int[] arr = {7, 4, 3, 10, 20, 8}

k = 3

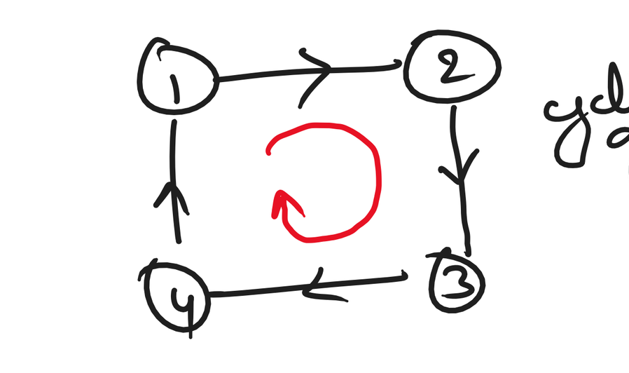
Max Heap.poll(){



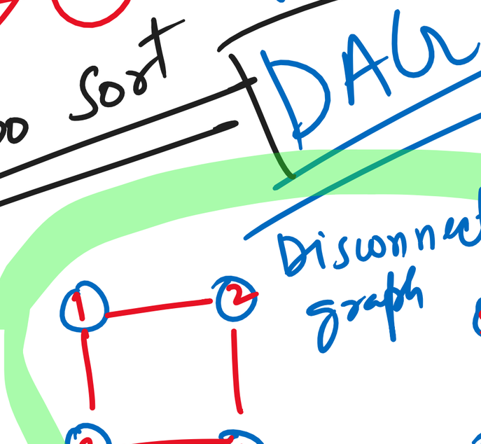
* Graph Data Structure:→

J- M A-J
4% -5%

It is a non linear data structure containing entities called "nodes" connected to each other via "edges".
depending on the connection there are various types:

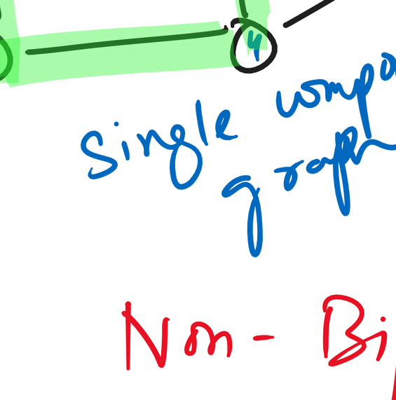


Undirected & Unweighted Graph



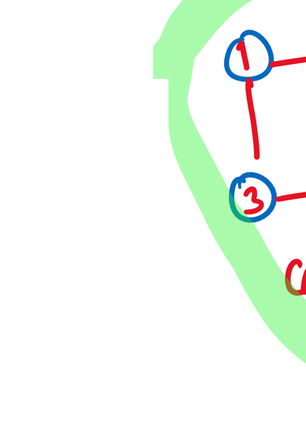
Directed Weighted Graph ←

* when can a graph have (−) weights?
(Statistical)



cyclic graph

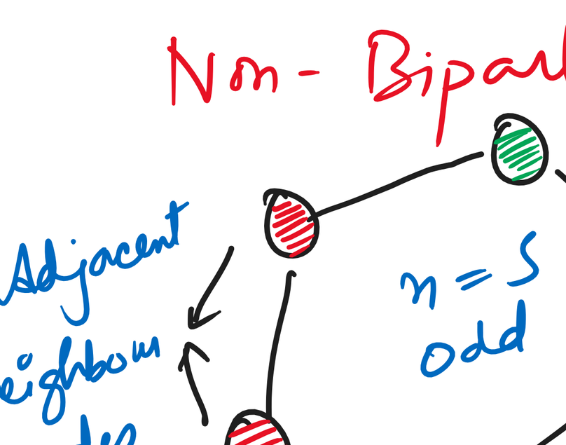
Kahn's Algo



Directed Acyclic Graph

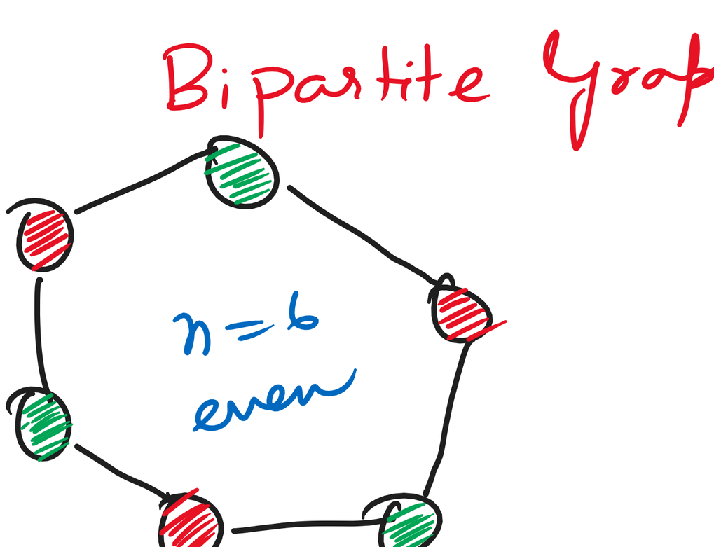
Topo Sort

DAG



Connected Graph

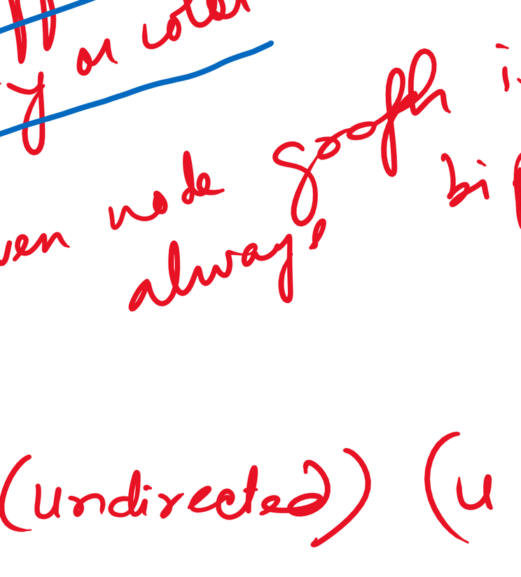
Single component graph



Non-Bipartite Graph

Adjacent neighbour nodes have same colour or property.

Bipartite Graph

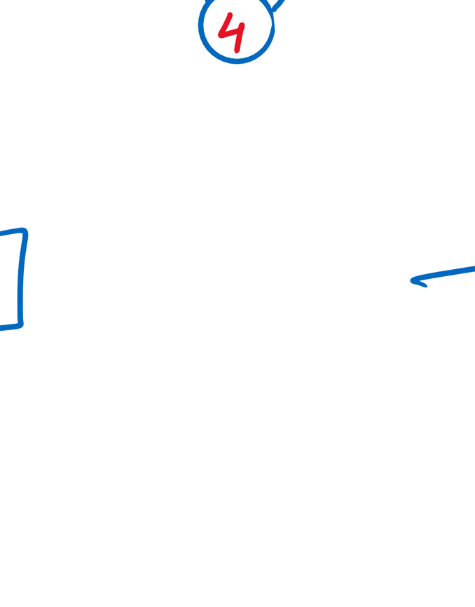


Adjacent neighbour nodes have different property or color.

Note: {Even node graph is always bipartite}

Graph Representations:→ (undirected) (u ↔ v)

Adjacency Matrix



	1	2	3	4	5
1	0	1	1	0	1
2	1	0	0	0	1
3	1	0	0	1	1
4	0	0	1	0	1
5	1	1	1	1	0

[Node to self is 0]

Adjacency List

Node : List of Neighbours
1 : 2, 3, 5
2 : 1, 5
3 : 1, 4, 5
4 : 3, 5
5 : 1, 2, 3, 4

Time Complexity: O(n)
Because we traverse all nodes.