Insertion Sort

75, 90, 100, 95, 85, 80
0 1 2 3 4 5

key = 90

J = i - 1

i = 1

90 < 75

0 1 2 3 4 5
75, 90, ~~100~~, ~~95~~, 85, 80
95 100

i = 2

key = 100

J = 1

True

95 < 100

i = 3

key = 95

J = ~~2~~ 1

True

95 < 90

while (J >= 0 and key < arr(J)) :

{ arr(J+1) = arr(J)
  J = J - 1

arr(J+1) = key

0 1 2 3 4 5
75, ~~90~~, ~~95~~, ~~100~~, ~~85~~, 80
85 90 95 100
↑J ↑i

i = 4 key = arr(i) = 85

J = ~~3~~ ~~2~~ ~~1~~ 0

T f

85 < 75

while J >= 0 and key < arr(J)

arr(J+1) = arr(J)

J = J - 1

arr(J+1) = key

75	85	90	95	100	80
0	1	2	3	4	5

i = 5

key = 80

J = 4 3 2 1 0

80 < 75

while J >= 0 && key < arr(J):

arr(J+1) = arr(J) ✓

J = J - 1

arr(J+1) = key

75, 80, 85, 90, 95, 100

↳ Sorted array

Best case scenario → Increasing order

n = 5

10, 20, 30, 40, 50

↳ comparisons ⇒ 4

Time complexity → # comp + # swaps

⇒ (n-1) + 0

⇒ O(n)

10 20

J i

key = 20

20 < 10

* Given a scenario, array is sorted or almost sorted →

Which sorting algorithm is preferrable & why??

Insertion sort → O(n)

0	1	2	3	4	5
~~10~~	~~20~~	~~30~~	~~40~~	~~50~~	~~5~~
10	20	30	40	50	

n = 6

comparision + # swaps

(n-1) + (n-1)

↳ O(n)

10, 20

20, 30

30, 40

40, 50

50, 5

J = 4

i = 5 key = 5

5 < 50

→ Decreasing order

50, 40, 30, 20, 10

				C	S
key = 40	40	50			
50, 40	~~50~~	~~40~~		1	1
		40	50		
	~~40~~	~~50~~	~~30~~	2	2
30, 50	30				
key = 30					
	20	30	40	50	
30, 40	~~30~~	~~40~~	~~50~~	~~20~~	3	3

key = 20

20 50

20 40

20 30

					(n-1)	(n-1)
10	20	30	40	50	4	4
~~20~~	~~30~~	~~40~~	~~50~~	~~10~~		

Key = 10

10 50

10 40

10 30

10 20

$$\#\text{comp. \& \# swaps} \quad \overline{1+2+3+4+---+(n-1)}$$

$$\frac{n^2-n}{2} = \frac{(n-1)n}{2} \Rightarrow O(n^2)$$

Which Sorting algo is preferrable when the array is highly unsorted??

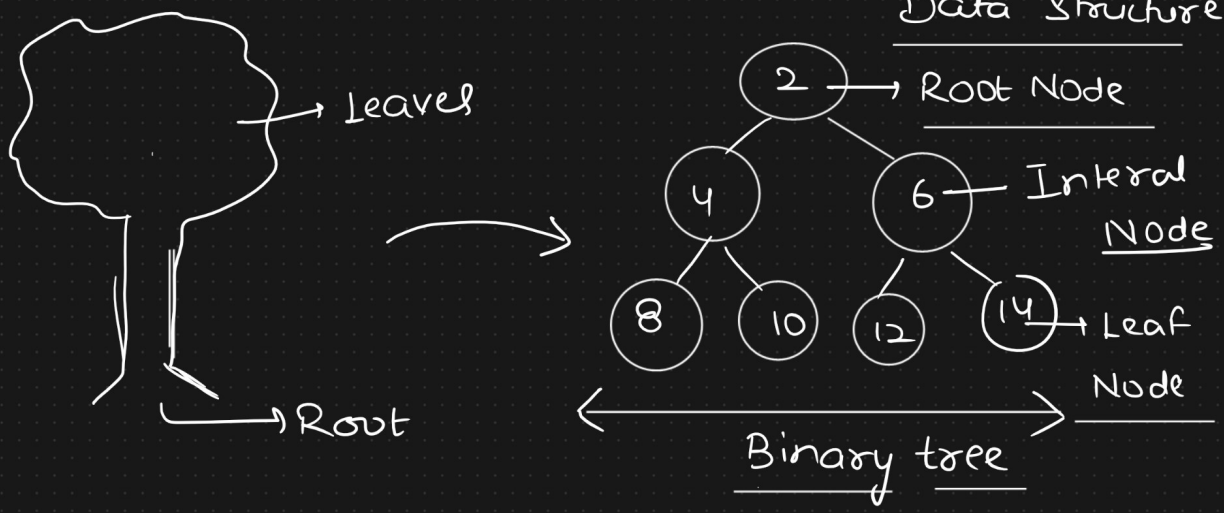
↳ QuickSort

HeapSort → heapq

↳ Heap Data Structure

Maxheap Minheap

Tree Data Structure → Non-Linear Data Structure



1) Binary Tree → 0, 1, 2

↳ Atmost 2 child nodes

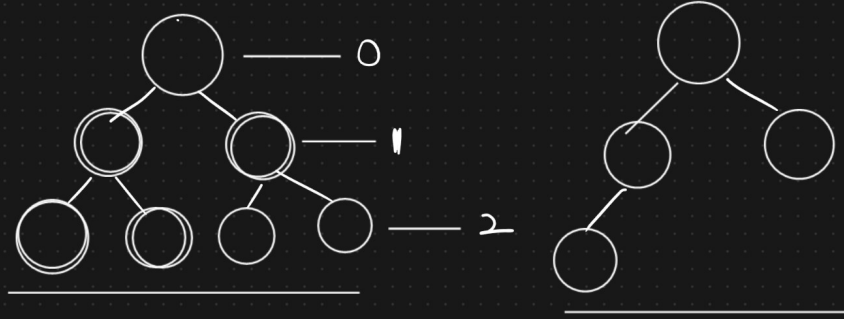
2) full Binary Tree vs Almost complete Binary tree

or

Perfect Binary Tree

Vs

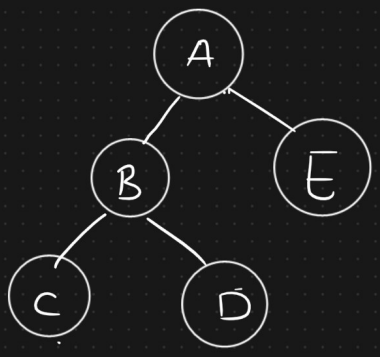
Complete Binary tree

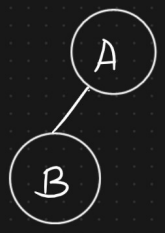


1) Insertion of nodes → (Left to right)

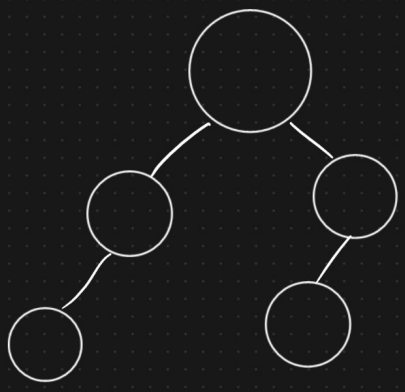
2) upper level nodes will be filled up before coming to the lower level

common for above three different binary tree





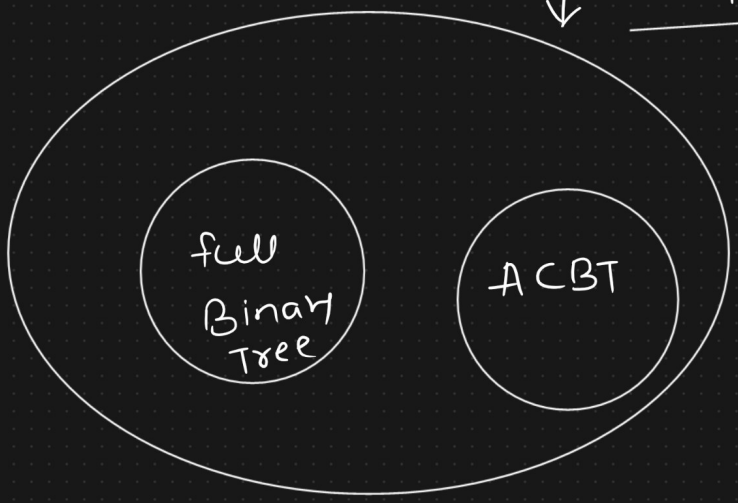
→ Valid ACBT

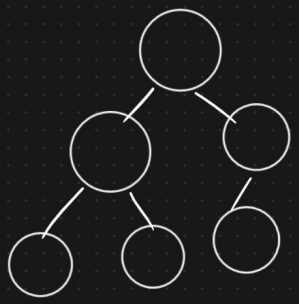


→ Not ACBT

Complete Binary Tree → full Binary Tree
&&
Almost CBT

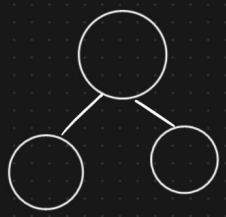
↓ complete binary tree





→

CBT → Yes
ACBT → Yes
fBT → No



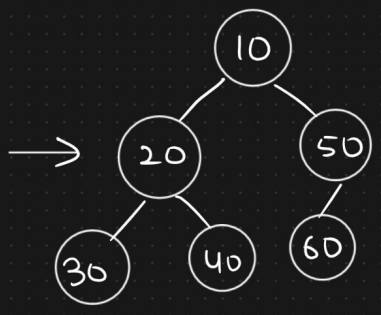
→

CBT → Yes
ACBT → No
fBT → Yes

Heap

↳ Complete Binary Tree

1) Minheap
   - ↳ CBT (1)
   - ↳ Parent node (2) data < child node data



10 < 20 & 50 — T
20 < 30 & 40 — T
50 < 60 — T

Valid Minheap

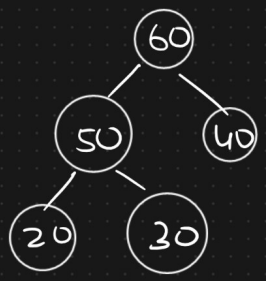
⇓

minimum element from the given array

⇓

O(1)

2) Maxheap
   - ↳ CBT (1)
   - ↳ (2) Parent node data > child node data



T — 60 > 50 & 40
T — 50 > 30 & 20

Valid Maxheap

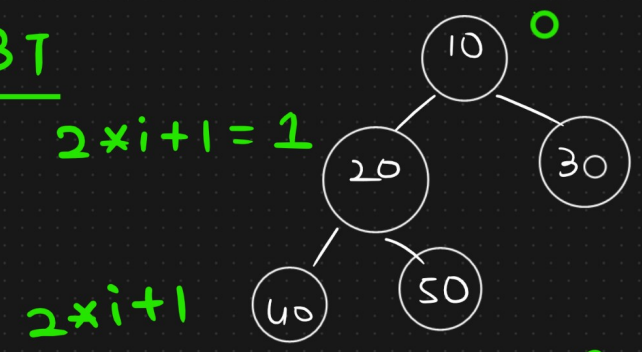
⇓

max. element from the given array

⇓

O(1)

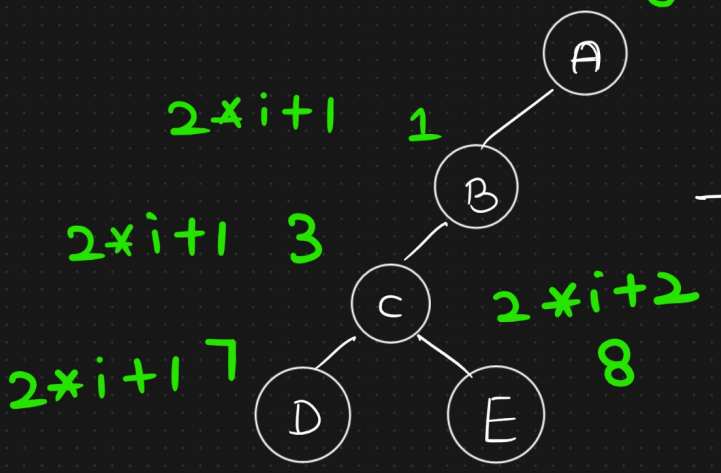
CBT



0	1	2	3	4
10	20	30	40	50

✓ Array
↳ No wastage of Space

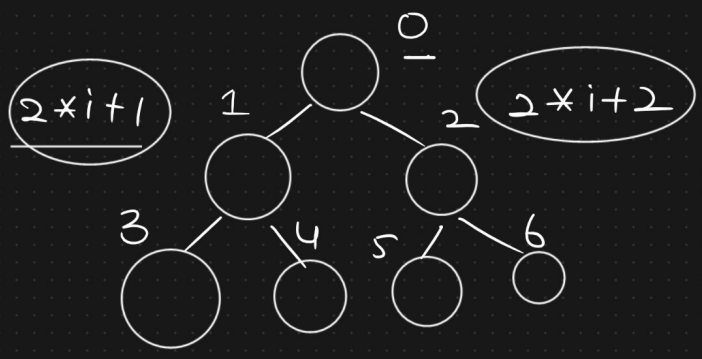
Skewed binary tree

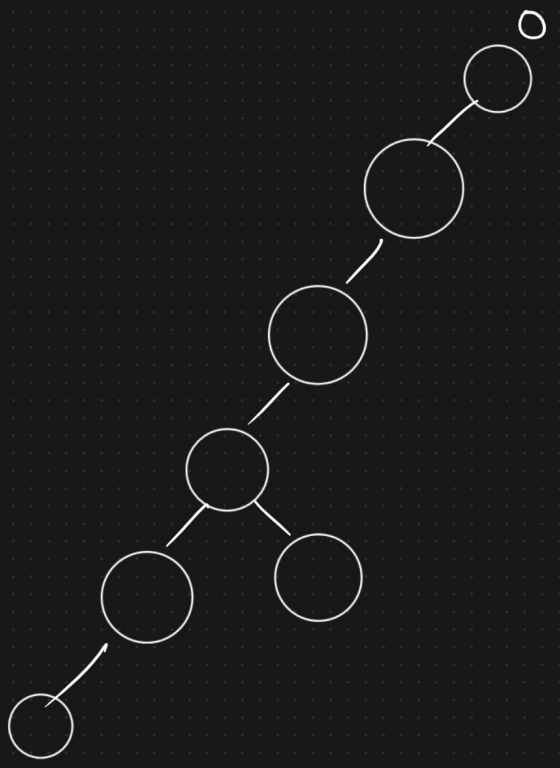


→ No
↳ Lot of Wastage of space

Linked List
↑
(Array)

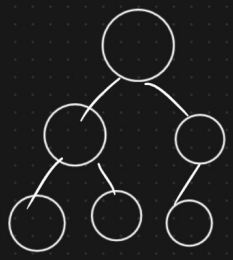
0	1	2	3	4	5	6	7	8
A	B		C				D	E



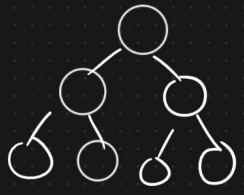


→ 2*i+1 (LI)
2*i+2 (RI)

Linked list
↳ skewed BT
⇓
Not a contiguous allocation of memory



→ CBT, ACBT



→ CBT, fBT

Operations

1) Insertion
2) Deletion
3) Build minheap

Time Complexity Analysis