Today's Content:

- Max xr pair →
- Subarray with max xr pair → { }
- Heaps: { Fang / Peaky / Lemon Squqy }
  - → CBT
  - → Insertion / Heapify
  - → Heap vs BBST
  - → Problems if time permits

308) Given N Elements find max xor pair → (i, j) ⇒ ar[i] ^ ar[j]
max

N=4 : 0 1 2 3 } ans = 7
4 3 2 7

(4^3) = 7 (3^2) = 1
(4^2) = 6 (3^7) = 4
(4^7) = 3 (2^7) = 5

Idea1: Iterate all pairs & get max

TC: O(N²) SC: O(1)

Ex: 5 3 2 7
max = 5

Idea2: Sort & check adjacent elements: 2 3 5 7
1 2 2 : ans = 2

Hint: 10000 vs 01111
16 15

Ex: 0 1 2 3 4 5 6 7 8

// Bunch of strings, say we need to search for a prefix ?
↳ Trie

N = 9 : 22 61 38 27 21 34 42 37 43

	5	4	3	2	1	0
22 :	0	1	0	1	1	0
61 :	1	1	1	1	0	1
38 :	1	0	0	1	1	0
27 :	0	1	1	0	1	1
21 :	0	1	0	1	0	1
34 :	1	0	0	0	1	0
42 :	1	0	1	0	1	0
37 :	1	0	0	1	0	1
43 :	1	0	1	0	1	1

// find max a^b = ?

a = 22, find b such that 22^b = max

	5	4	3	2	1	0
22 :	0	1	0	1	1	0
b :	1	0	1	0	0	1
ans :	1	1	1	1	0	1

If we fix every element as a and repeat same proc we can get final ans

fix elem & get max nr

TC: N * { N }

TC: N^2

// Class Node {

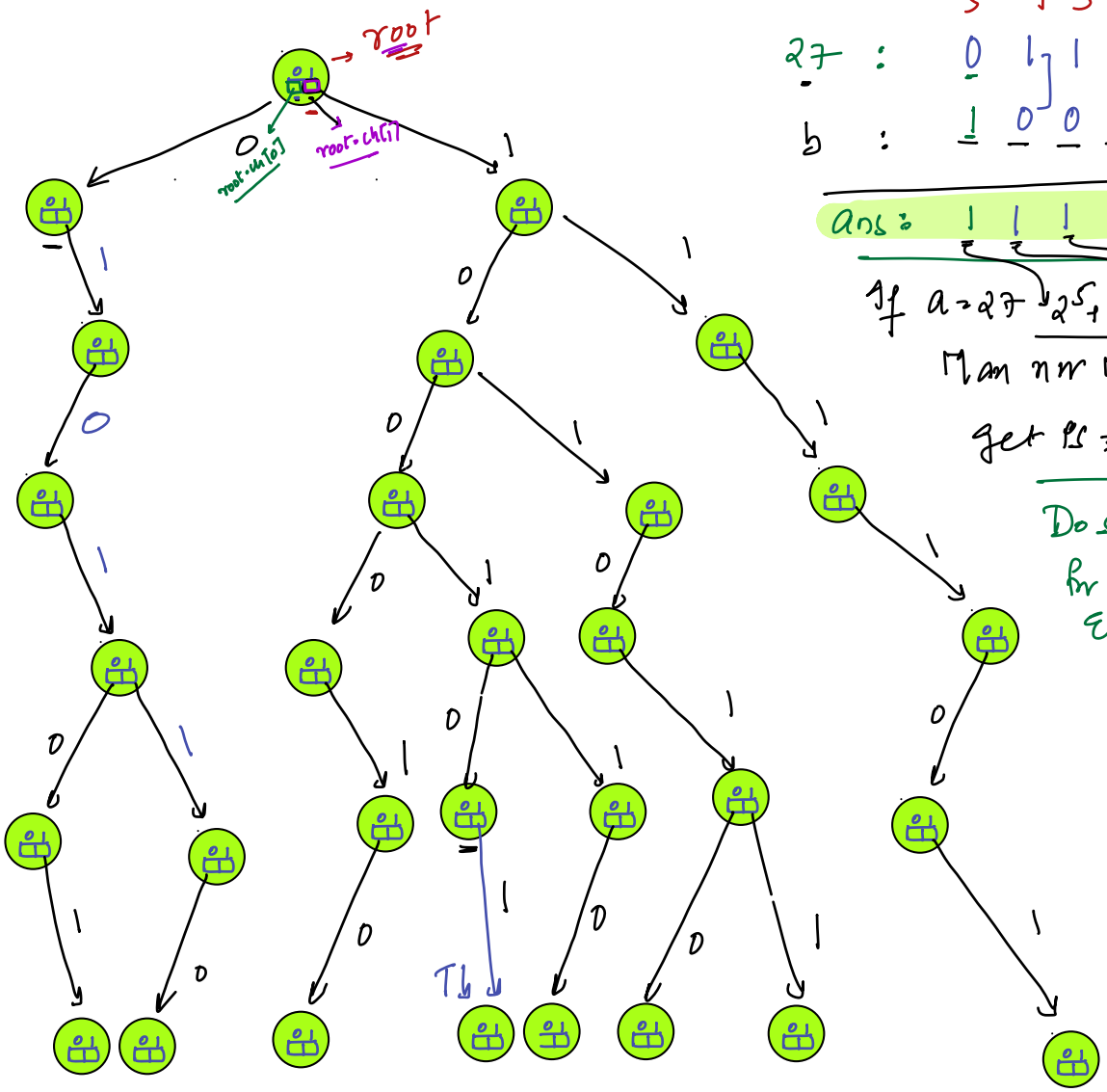
    Node c[2]

}

obs1: While inserting we insert from MSB

obs2: For every number of bits we insert should be same

Step1: Create a root



5 4 3 2 1 0

27 : 0 1 1 0 1 1

b : 1 0 0 1 0 1

ans : 1 1 1 1 1 0

If a > 27 → 2^5, 2^4, 2^3, ...

Max nr we can get is 63

Do same thing for all Elem

Idea: For every element, how many bits we need to

↳ Iterate → [0 30] → 31 bits

Entire above proc N time → O(31*N) much better N^2

32 : [0 31] → 32 bits

```
class Node {
    Node c[2]
}
```

↳ array of children

```
main() {
    int N;
    Read N array Elems
    Node r = new Node();
    int n = max(ar[1])
    // get most set bit pos of n
    int p = getMSP(n)
    for(i=0; i<N; i=i+1){
        insert(r, ar[i], p)
    }
    ans = 0;
    for(i=0; i<N; i=i+1){
        a = ar[i], for a we need
        to get max xor value?
        ans = max(ans, maxxr(r, ar[i], p))
    }
}
```

10:35 break

```
int maxXorPair(int ar[])
```

```
void insert(Node cur, int n, int m) {
    for(p = 30; p>=0; p--){        // m
        int p = checkBit(n, i)     // return 0-unset 1-set
        if(cur.ch[p] == NULL){
            cur.ch[p] = new Node()
        }
        cur = cur.ch[p]
    }
}
```

```
int maxxr(Node cur, int n, int m)
    int ans = 0;
    for(p = 30; p>=0; p--){        // m
        int p = checkBit(n, p)
        if(cur.ch[1-p] != NULL){
            // in your ans ith bit we
            can set
            ans = ans | (1<<p)
            cur = cur.ch[1-p]
        }
        else{
            cur = curr.ch[p]
        }
    }
    return ans;
}
```

TC: $N * 31$ SC: $N * 31$

28] Subarray with Max xor value ?
Sum

Eg: ar[4] = { 3 2 8 4 }

[3] [2] [8] [4]

[3 2] [2 8] [8 4]

[3 2 8] [2 8 4]

[3 2 8 4]

// Eg: a[4] = { a_0 a_1 a_2 a_3 }

// [i j] = pf[j] - pf[i-1]

[i j] = pf[j] ^ pf[i-1]

pfxr[4] = { a_0, $a_0 \wedge a_1$, $a_0 \wedge a_1 \wedge a_2$, $a_0 \wedge a_1 \wedge a_2 \wedge a_3$ }
(0, 1, 2, 3)

pf[6] = xr of all Elements [0 6] = $x_0 \wedge x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_6$

pf[2] = xr of all Elems [0 2] = $x_0 \wedge x_1 \wedge x_2$

xr [3 6]

pf[6] ^ pf[2] =

In your pfxr[]: get max xr pair, {Max Subarray XOR}

given by Trie

Edge case :

pfxr[4] = { a_0 , $a_0 \wedge a_1$, $a_0 \wedge a_1 \wedge a_2$, $a_0 \wedge a_1 \wedge a_2 \wedge a_3$, 0 }

(0 is at index 0, a_0; $a_0 \wedge a_1$ at index 1; $a_0 \wedge a_1 \wedge a_2$ at index 2; $a_0 \wedge a_1 \wedge a_2 \wedge a_3$ at index 3; 0 at index 4)

0 1 : a_1

0 2 : $a_1 \wedge a_2$

0 3 : $a_1 \wedge a_2 \wedge a_3$

1 2 : a_2

1 3 : $a_2 \wedge a_3$

2 3 : a_3

Few subarray xr are missing →

a_0

$a_0 \wedge a_1$

$a_0 \wedge a_1 \wedge a_2$

$a_0 \wedge a_1 \wedge a_2 \wedge a_3$

↳ 1 way to solve this → in pfxr[] we can get all the value of → {Iterate & get max}

// If we add extra 0 in pfxr[]

0 4 : a_0

1 4 : $a_0 \wedge a_1$

2 4 : $a_0 \wedge a_1 \wedge a_2$

3 4 : $a_0 \wedge a_1 \wedge a_2 \wedge a_3$

// pseudo code :

→ given array ar[N]

```
pfxr[0] = ar[0]
int ans = ar[0]
for(i = 1; i < N; i = i+1){
    pfxr[i] = pfxr[i-1] ^ ar[i]
    ans = max(ans, pfxr[i])
}
return max(ans, MaxXORpair(pfxr))
}
```

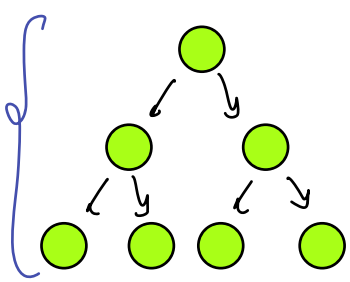
TC: O(N)

SC: O(N)

// Every square is Rectangle ✓

→ All CBT's are BBT ✓

→ All BBT's are CBT's ✗



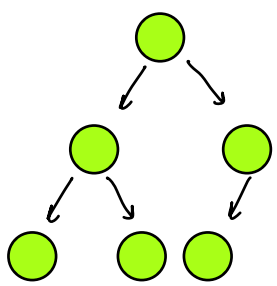
→ CBT

Complete Binary Tree:

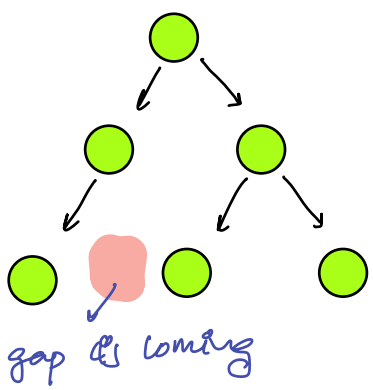
A Binary Tree is said to be (CBT) if

- Data is filled Level by Level from L-R :
- Except last level all levels have to be filled Completely

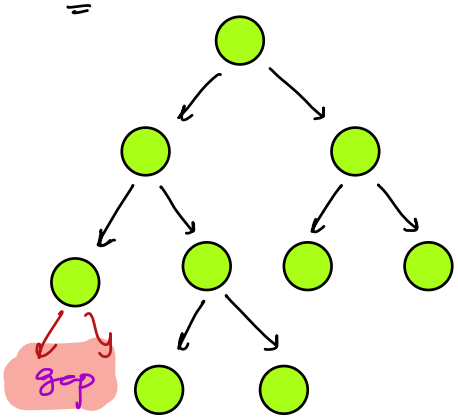
Ex1: CBT ✓



Ex2: NOT CBT



Ex3: NOT CBT

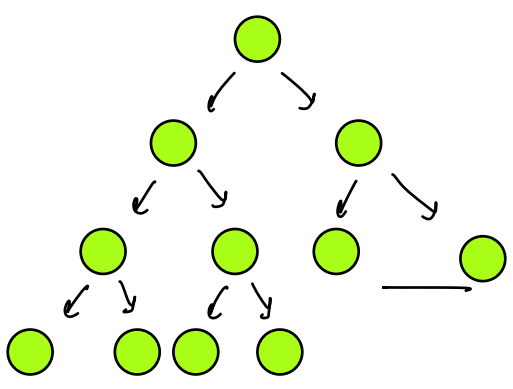


Ques1 → Height of a Balanced Tree = (log N)

Ques2 :→ Balanced Binary Tree

↓

$\forall$ fn Nodes → $|H(LST) - H(RST)| \leq 1$



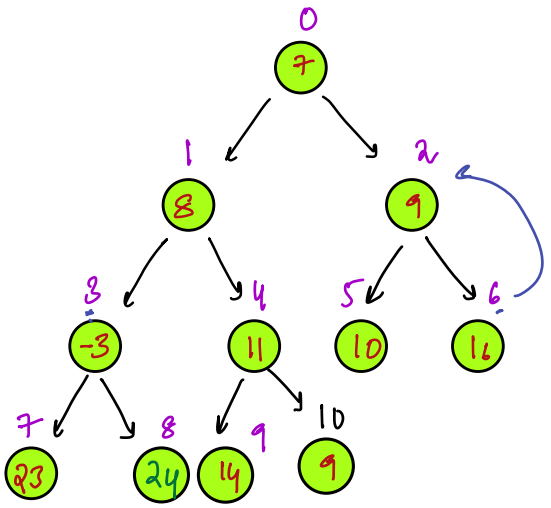
$H(LST) - H(RST)$ at most = 1

conc → Every CBT → BBT

↓

Height of Every CBT fn N Node = logN

Construction of CBT: → Complete CBT we will store in Array



Insert 10

Insert 16

Insert 23

Insert 24

Insert 14

Insert (9)

Ques 3: parent at i, children at { left $2i+1$, right $2i+2$ }

Ques 4: children at i, parents at { $(i-1)/2$ }

parent → $(i-1)/2$

$i=3 \rightarrow 1 \rightarrow (2/2)$

$i=7 \rightarrow 3 \rightarrow (6/2)$

$i=6 \rightarrow 2 \rightarrow (5/2)$

Heaps

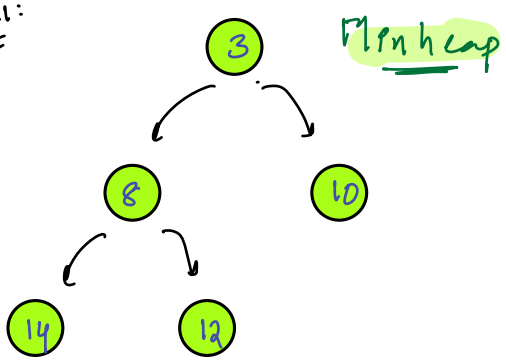
property

? : CBT

Every Node >= (It's child node): Maxheap

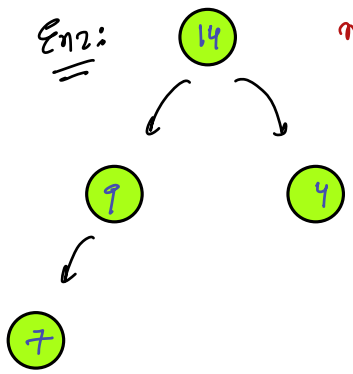
Every Node <= (It's child node): Minheap

Ex1:



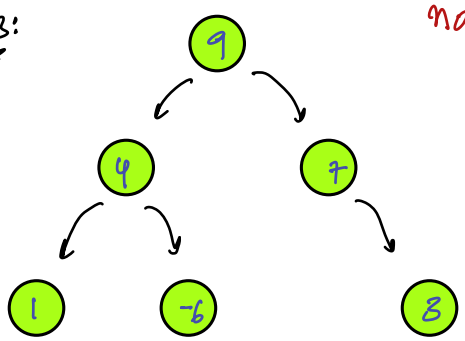
Minheap

Ex2:



maxheap

Ex3:



not CBT

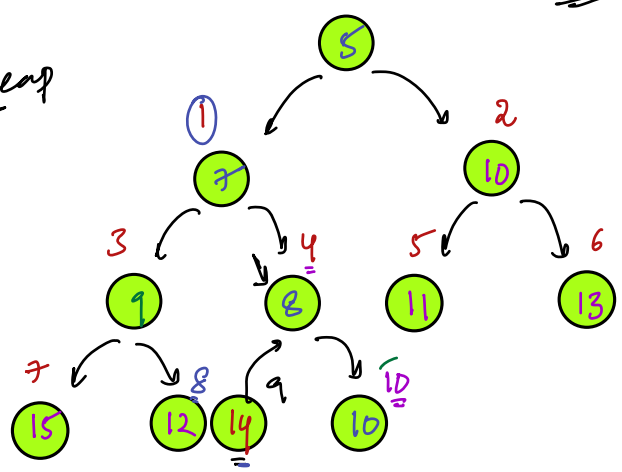
// Insert in Heap → CBT : We are given an (Ip) → so we increase size

Eg: Given Min heap, Insert new element

node x = (child)

→ no parent to compare (Break)

minheap

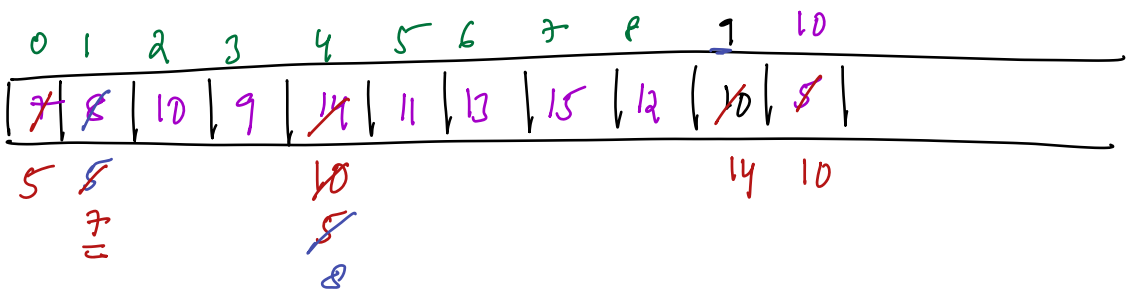


Insert 8 ✓

Insert 10

Insert 5

Insert



```
void insert (ArrayList<Int> t, int x) {          → O(log N)
    t.add(x);
    int i = t.size() - 1;
    while (i > 0) {
        int p = (i-1)/2;
        if (t[p] > t[i]) {
            swap (t[p], t[i])
            i = p;
        } else { break; }
    }
}
```

TC: O(log N)

minheap

minheap ↳

getMin() → O(1)

from smaller

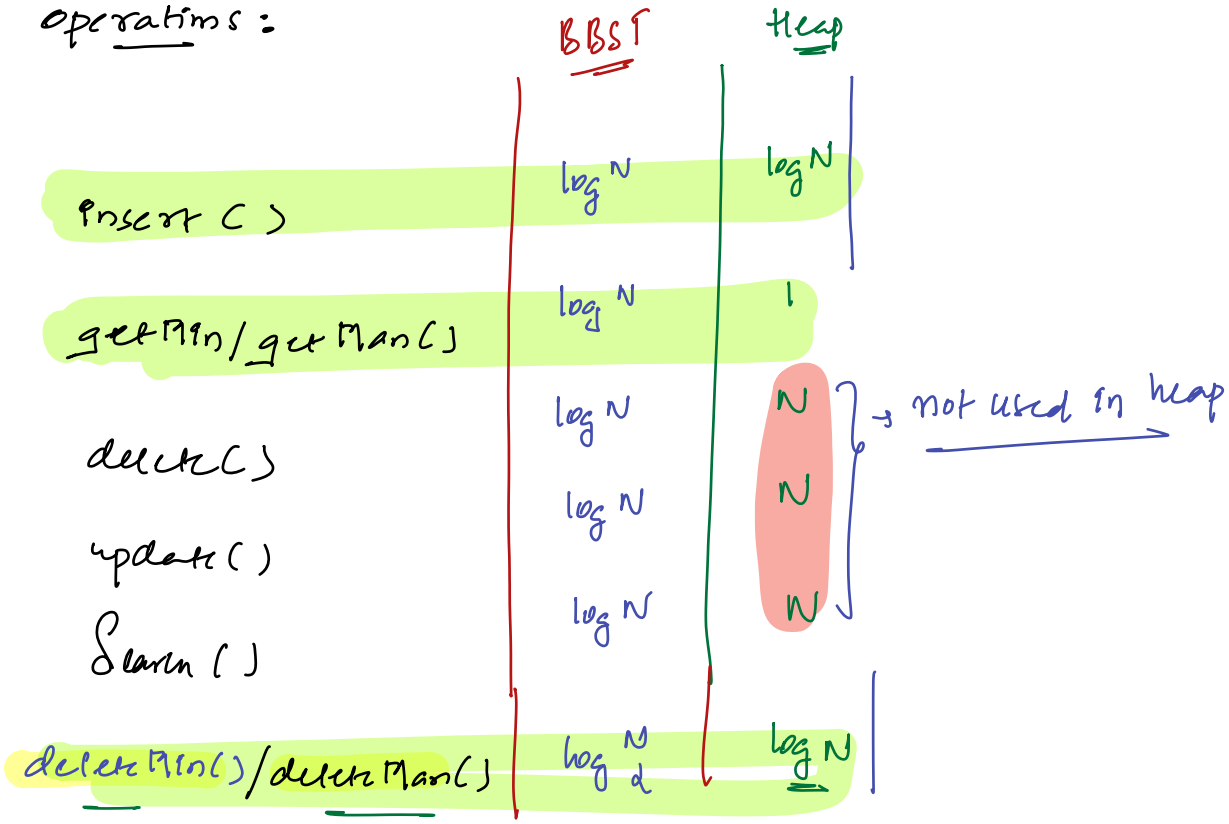
// getmin → O(1)

// delete : → O(N)

// update : → O(N)

// search : → O(N)

operations :



operations :

- insert() → O(log N)
- getMin()/getMax() → O(1)
- deleteMin()/deleteMax() → O(log N)

C++ → Java → Python

C++ : priority_queue

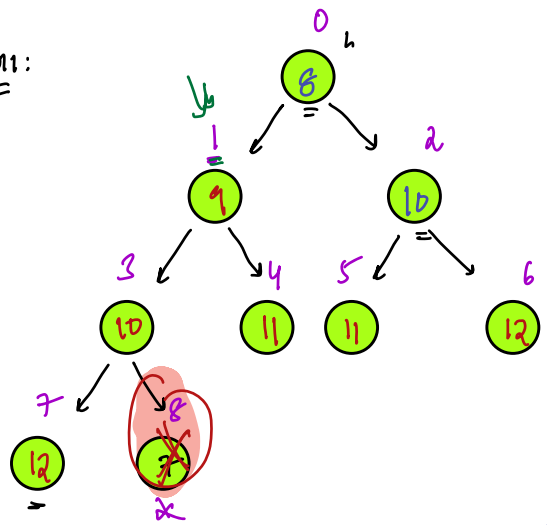
Java : PriorityQueue

Python : (heapq)

Delete in Heap: → given a list:

given min heap, delete min element

Ex1:



// deleting last element in list → O(1)

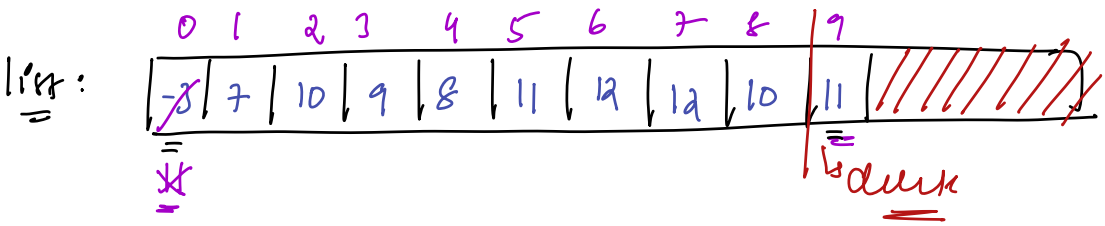
	2i+1	2i+2	min_i	Swap
i=0:	7	10	1	l[i], l[1]
i=1:	9	8	4	l[i], l[4]
i=4:	2i+1	2i+2	→ no children	

If at least break

	2i+1	2i+2	min_i	Swap
i:0:	8	10	1	l[i] & l[1], i = 1
i:1:	9	11	3	l[i] & l[3]: i = 3
i:3:	12			

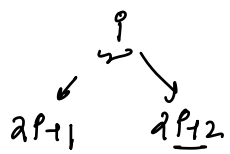
10 i = 12: { no swap, break }

list:



void deleteMin(ArrayList<Integer> l) → (log N)

```
int n = l.size();
l[0] = l[n-1] // Bringing last element to first
l.delete() // It will delete last element
n = n-1 // size reducing
i = 0
while(i<n)   → no need of any condition (true)
{
    min_i = i;
```




```
    left = 2i+1, right = 2i+2
    if(left<n && ar[min_i] > ar[left]){
        min_i = left
    }
    if(right<n && ar[min_i] > ar[right]){
        min_i = right
    }
    if(i != min_i) {   → indicating a new min. i found
        swap(ar[i], ar[min_i])
        i = min_i
    }
    else{ break }
}
```

↳ if min_i == i we anyway break

// Story:

TODO → Manheap ✓ { Plan / Problem Solving Test

Min Heap

Tree1 → All L → R

Tree2 → R → L

} S_0 S_1 S_2 S_3 S_4 S_5