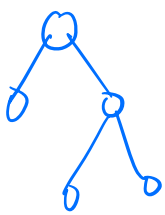
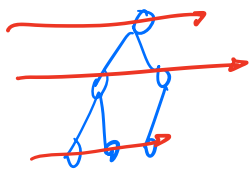
Heap → 1

→ Complete Binary Tree (CBT)

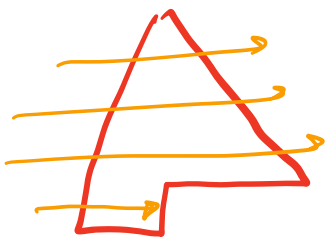
↳ fully filled upto 2nd last level.
& filled from L → R in last level.



X



✓



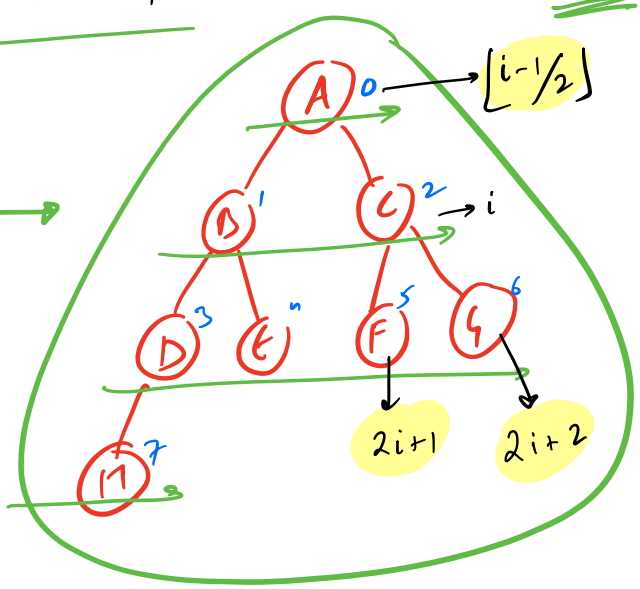
→ Structurally hierarchical.

→ Implementation is Linear.

H : [0 A, 1 B, 2 C, 3 D, 4 E, 5 F, 6 G, 7 H]

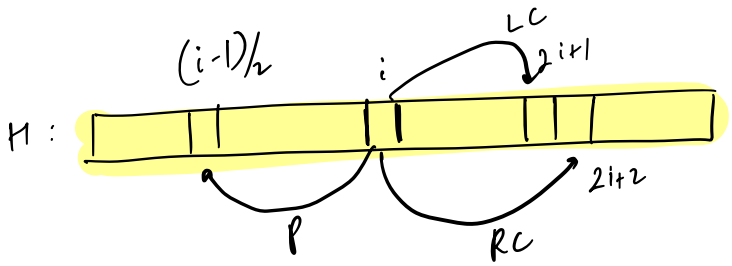
LOT

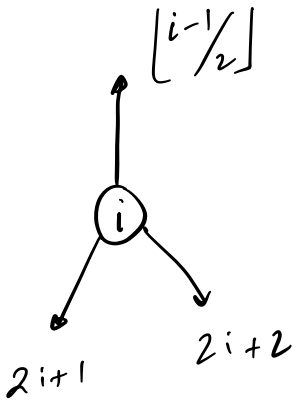
LOT



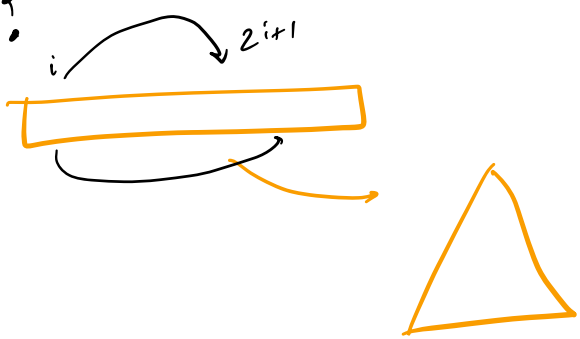
Heap enforces structure

CBT





Q H[], N . Inorder ?

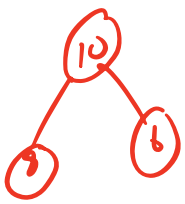



```
void inorder ( int idx ) {
    if (idx >= N) ret;
    inorder ( 2 x idx + 1);
    print ( H[idx]);
    inorder ( 2 x idx + 2);
}
```

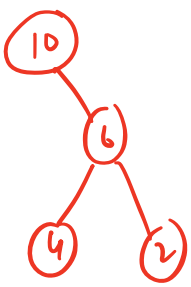
MAX heap

→ CBT

→ Every node should be greater than it's children.



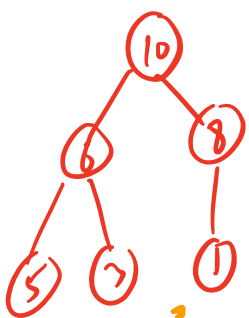
✓ ✓ ✓



X CBT

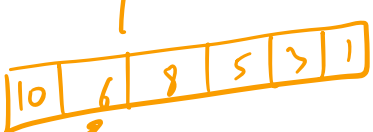
✓

X



✓ ✓ ✓

H:

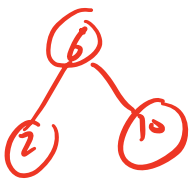


MIN heap

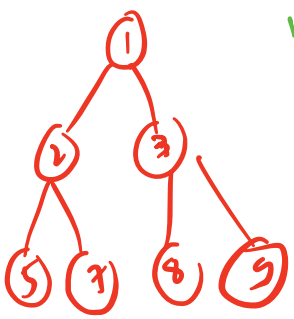
→ CBT

→ ~~~~~~~~~~~~~~~~

smaller ~~~~~~~~~~~~~~~~



✓ X X



✓ ✓ ✓

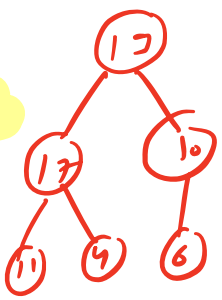
Q Given an arr. check if it is a MIN heap!


```
for(i: 0 ⟶ N-1) {
    LCi = 2i+1;
    RCi = 2i+2;
    if( LCi < N && A[i] > A[LCi]) {
        ret false;
    }
    if( RCi < N && A[i] > A[RCi]) {
        ret false;
    }
    ⟶ early break;
}
ret true;
```

→ Given a MAX heap

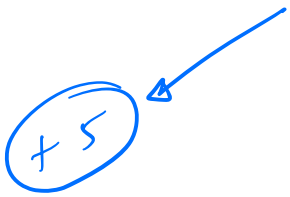
find the MAX : A[0] O(1)

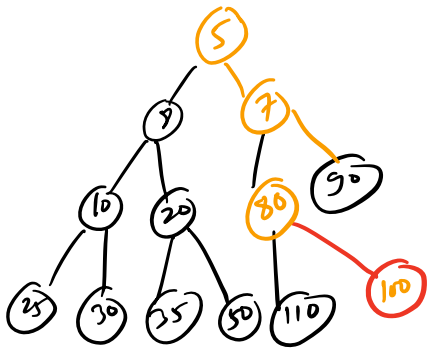
——— 2nd MAX : max(A[1], A[2]) O(1)



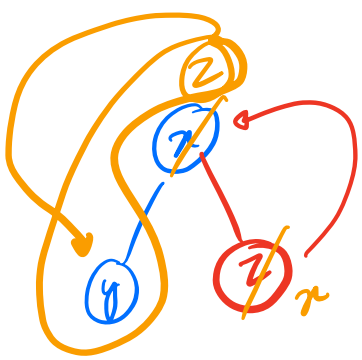
(9) Insert new elements in a heap

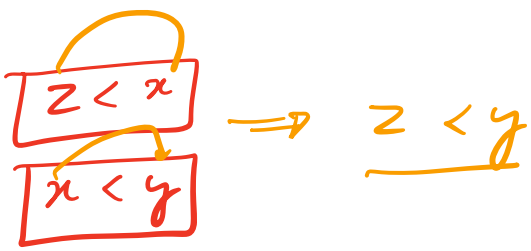
Given a MIN heap





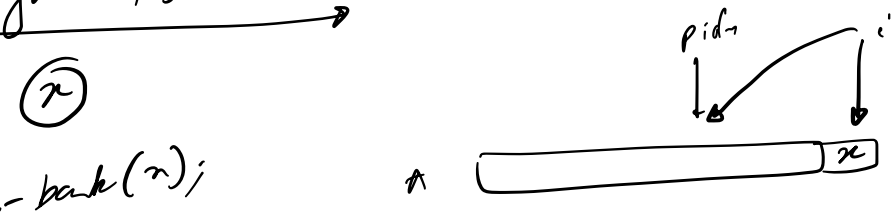
	0	1	2	3	4	5	6	7	8	9	10	11	12
A:	~~7~~	8	~~80~~	10	20	~~100~~	90	25	30	35	50	110	~~5~~
			5			5							100
	↖		7			80							





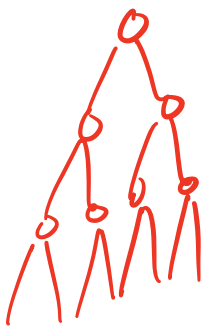
CODE for insertion in MIN HEAP

```
// A[], (x)
A.push_back(x);
int i = A.size() - 1;
while (i > 0) {
    pidx = (i-1)/2;
    if (A[i] < A[pidx]) {
        swap(A[i], A[pidx]);
        i = pidx;
    }
    else {
        break;
    }
}
```



TC = O(log N)

SL = O(1)



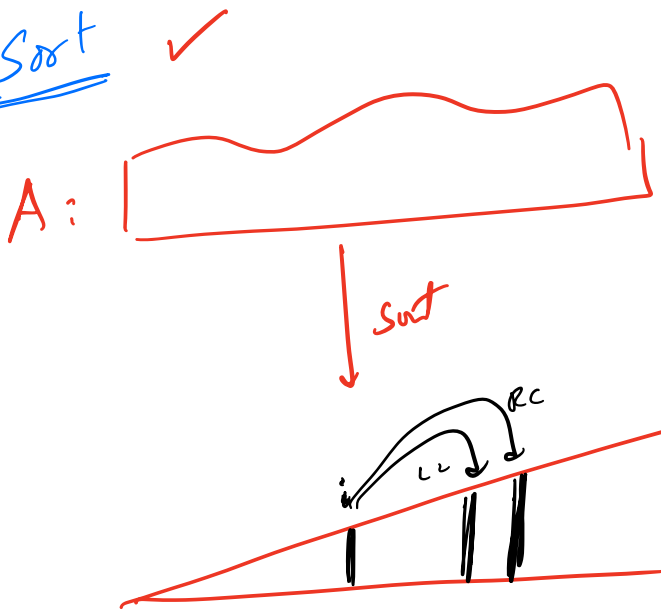
N = 1 → 0
N = 3 → 1
N = 7 → 2
N = 15 → 3

H → O(log N)

Q Given N elements. Create a MIN heap!

A: 3 5 1 2 6

1) Sort ✓



TC = O(N lg N)

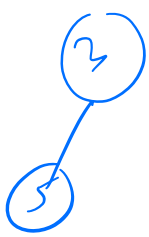
SC =

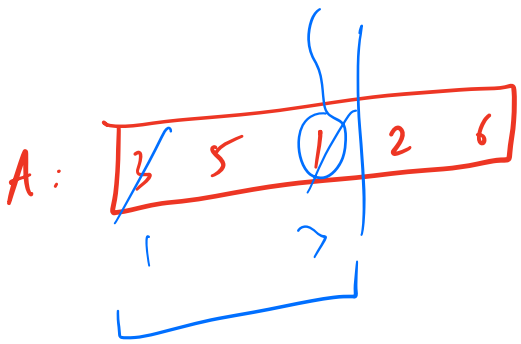
2)

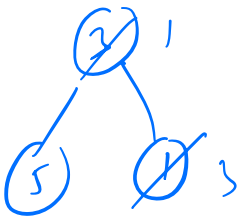
A: 3 5 1 2 6

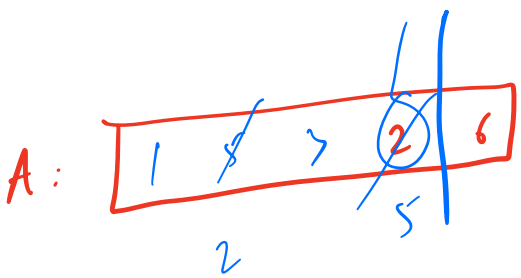
A: 3 5 1 2 6

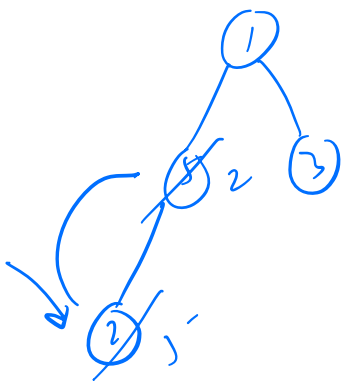
A: 3 5 1 2 6

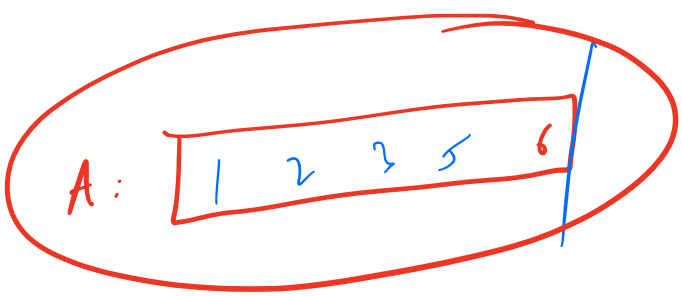


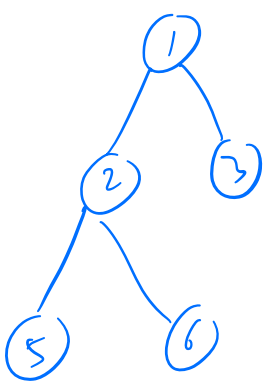












$TC = f(1) + f(2) + f(3) + \cdots + f(N)$

$f(N) \quad f(N) \quad \cdots \quad \cdots$

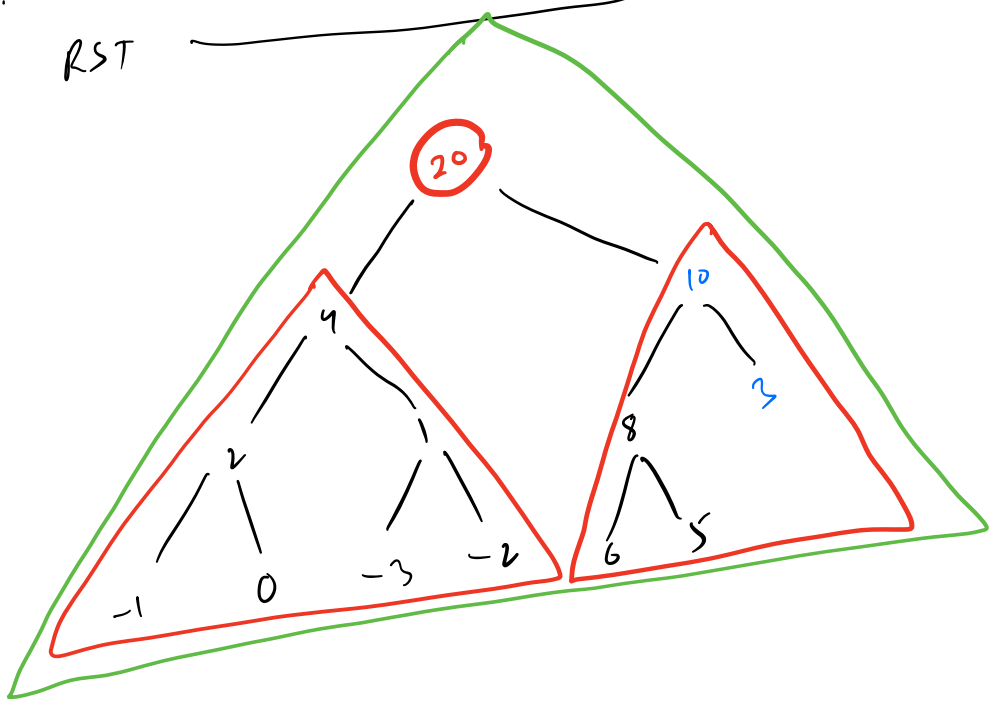
$TC = O(N f(N))$

$SC = O(1)$

④ Heapify → MAX heap

preconditions

1. Entire tree should be a CBT
2. LST should be a MAX heap.
3. RST



```
heapify(i) {
    LC
    RC
    figure out which is greater
    if( A[i] > (LC  RC) ) → break;
    else {
        swap( A[i], greater of LC, RC )
        heapify( index )
    }
}
```

SC = O(lg N)

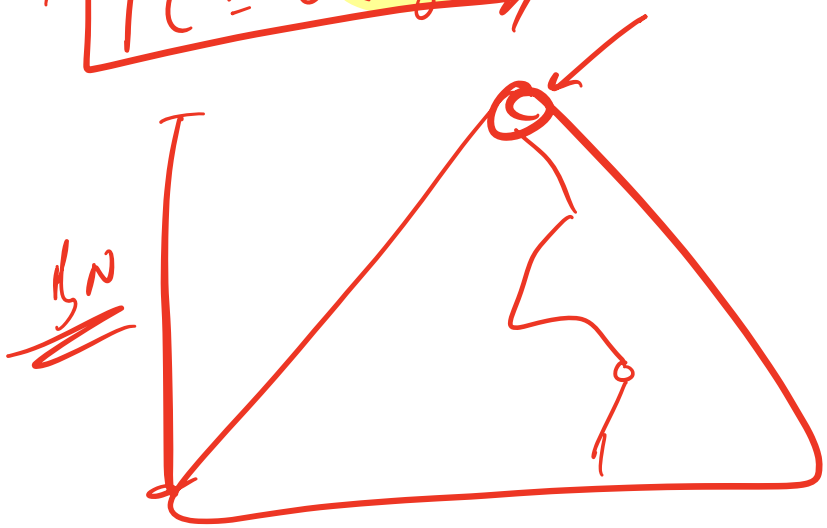
Rec.

O(1)

It

TC = O(lg N)

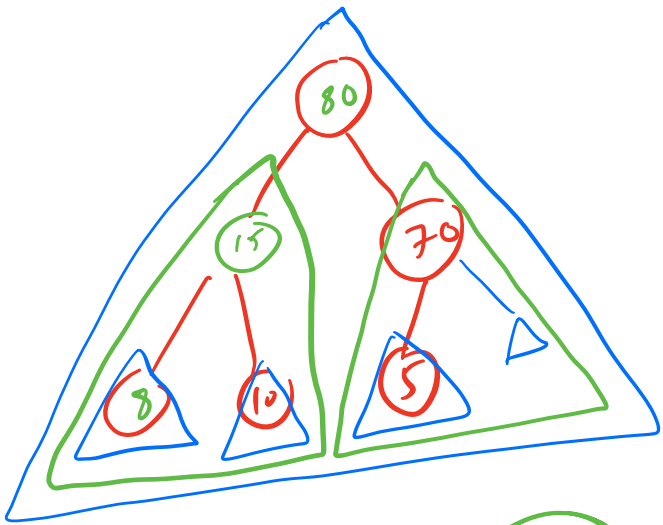
lg N

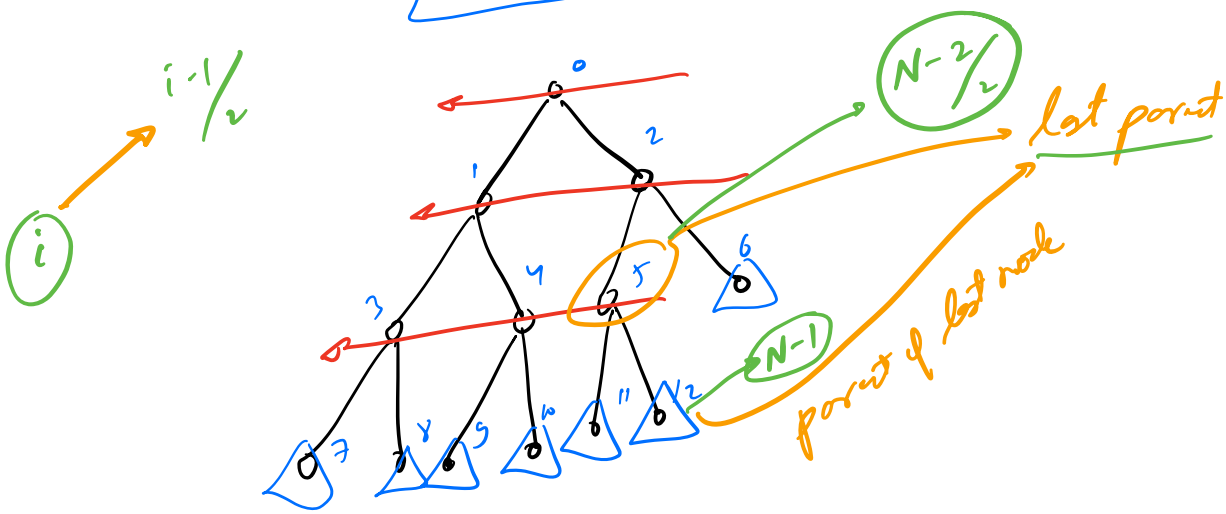


Q Given N elements. Create a MAX heap out of it!

A :

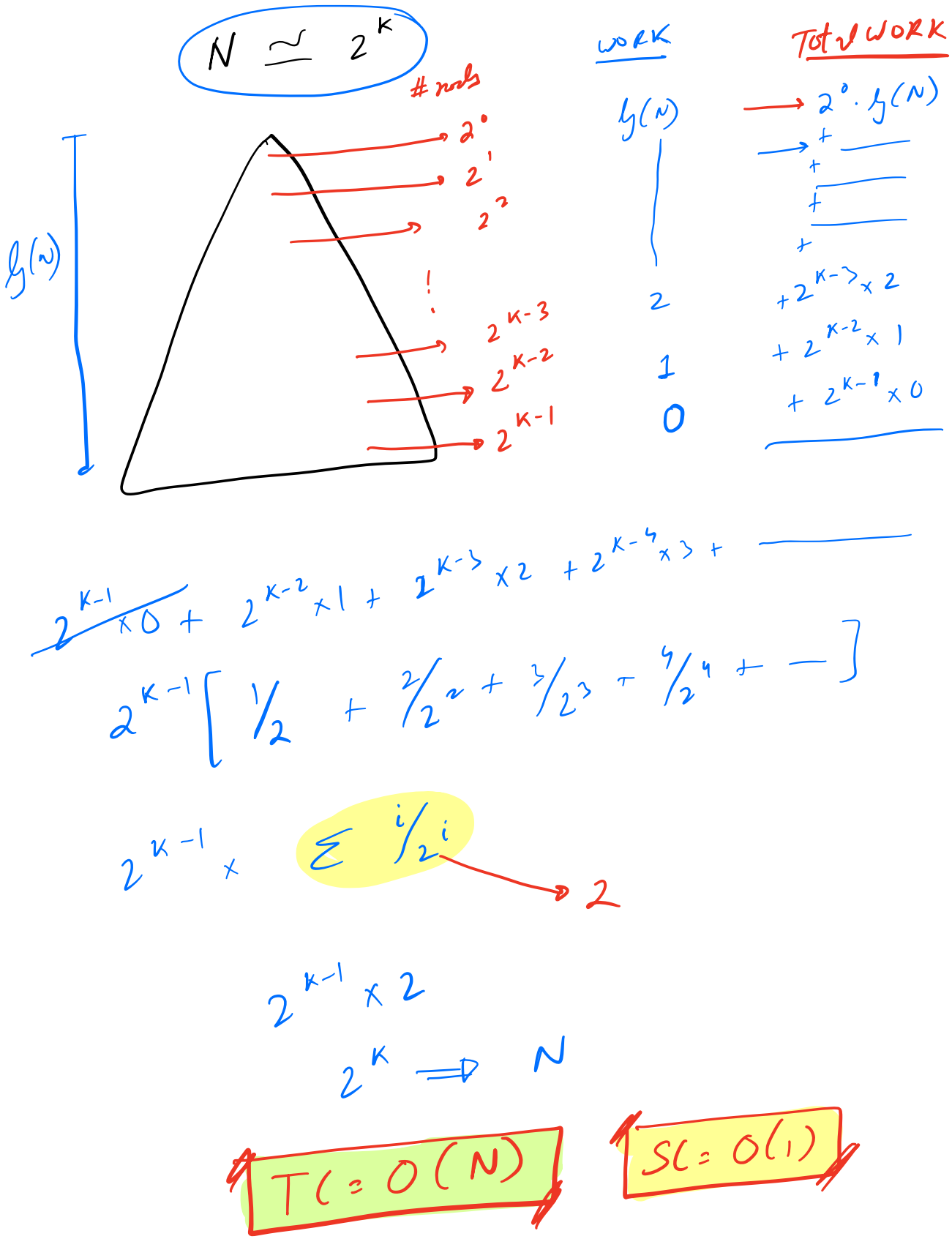
15	8	70	80	10	5

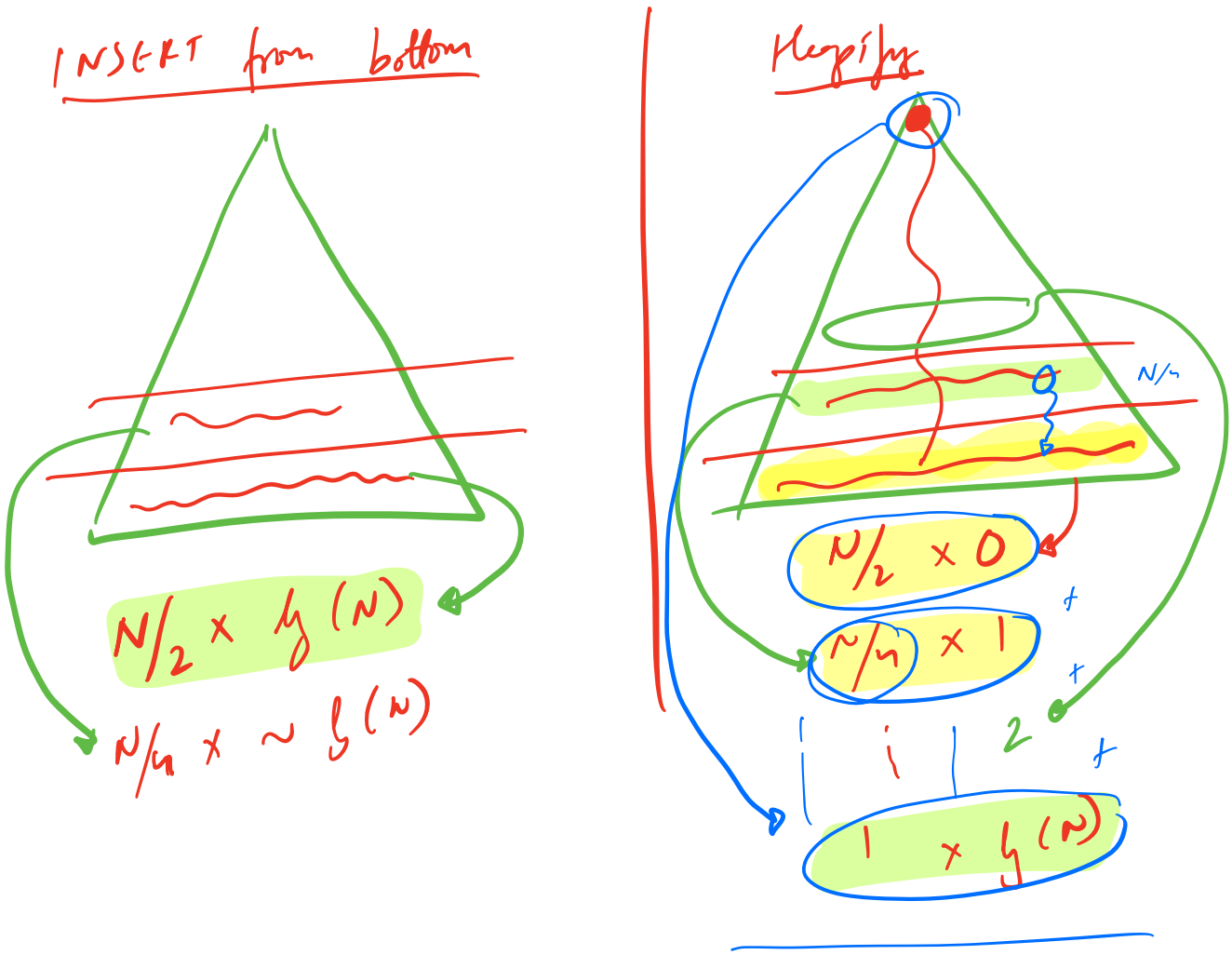





```
for (i = (N-2)/2 ⟶ 0) {
    heapify(i);
}
```

✓


N ≃ 2^K
nodes
2^0
2^1
2^2
2^{K-3}
2^{K-2}
2^{K-1}
f(N)
WORK
f(N)
2
1
0
Total WORK
2^0 · f(N)
+ 2^{K-3} x 2
+ 2^{K-2} x 1
+ 2^{K-1} x 0
2^{K-1} x 0 + 2^{K-2} x 1 + 2^{K-3} x 2 + 2^{K-4} x 3 + ——
2^{K-1} [1/2 + 2/2^2 + 3/2^3 + 4/2^4 + —]
2^{K-1} x Σ i/2^i → 2
2^{K-1} x 2
2^K ⇒ N
T(N) = O(N)
SC = O(1)


INSERT from bottom
N/2 x log(N)
N/4 x ~ log(N)
Heapify
N/4
N/2 x 0
N/4 x 1
2
i
1 x log(N)