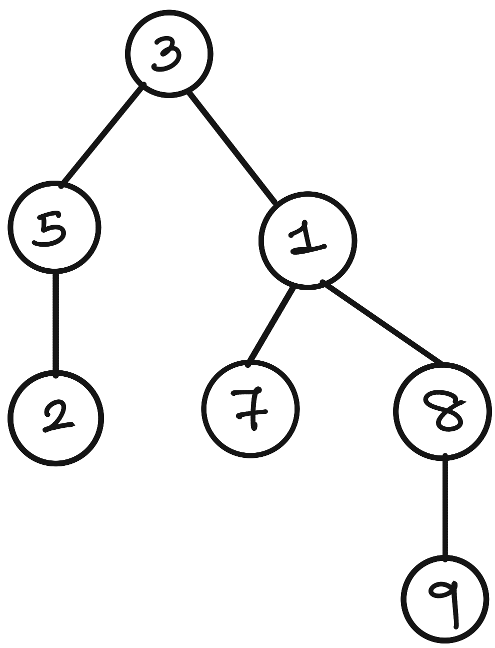
CONSIDER THE FOLLOWING DATA-STRUCTURE

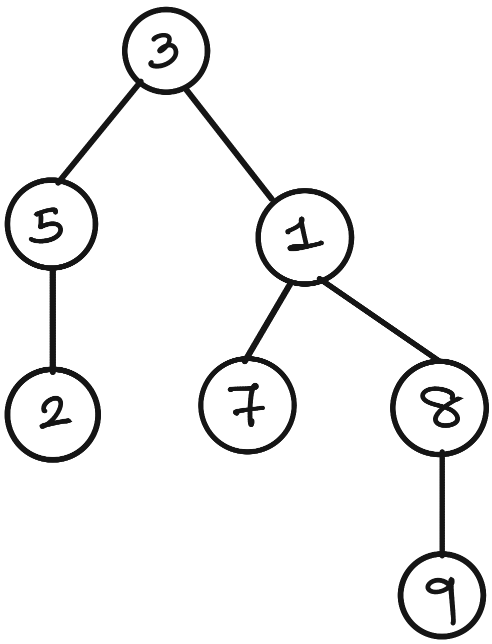
1) EACH NUMBER IS REPRESENTED BY A NODE
2) EACH NODE HAS UNIQUE CHILDRENS.



CONSIDER THE FOLLOWING DATA-STRUCTURE

1) EACH NUMBER IS REPRESENTED BY A NODE
2) EACH NODE HAS UNIQUE CHILDRENS.

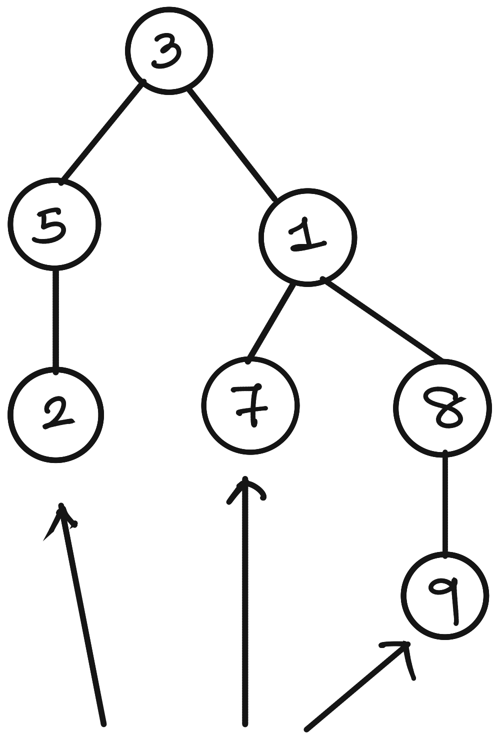
A TREE →



CONSIDER THE FOLLOWING DATA-STRUCTURE

1) EACH NUMBER IS REPRESENTED BY A NODE

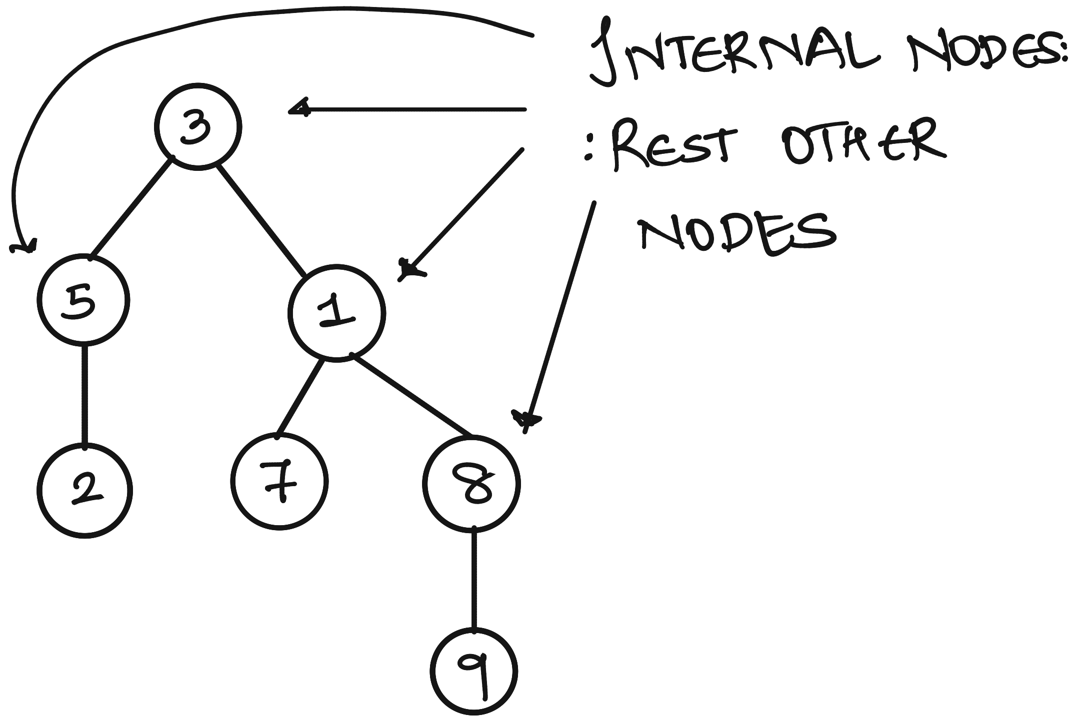
2) EACH NODE HAS UNIQUE CHILDRENS.



LEAF : NODES THAT DONOT HAVE ANY CHILDREN

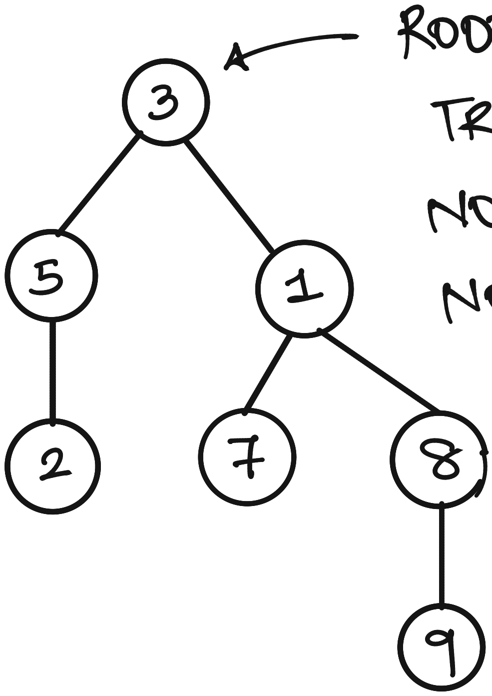
CONSIDER THE FOLLOWING DATA-STRUCTURE

1) EACH NUMBER IS REPRESENTED BY A NODE
2) EACH NODE HAS UNIQUE CHILDRENS.



CONSIDER THE FOLLOWING DATA-STRUCTURE

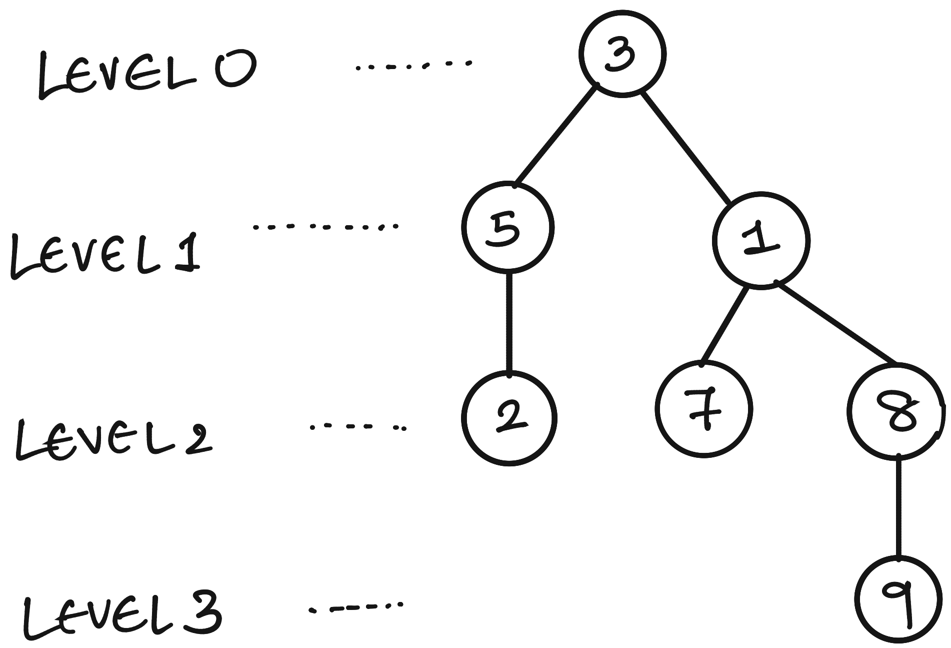
1) EACH NUMBER IS REPRESENTED BY A NODE
2) EACH NODE HAS UNIQUE CHILDRENS.

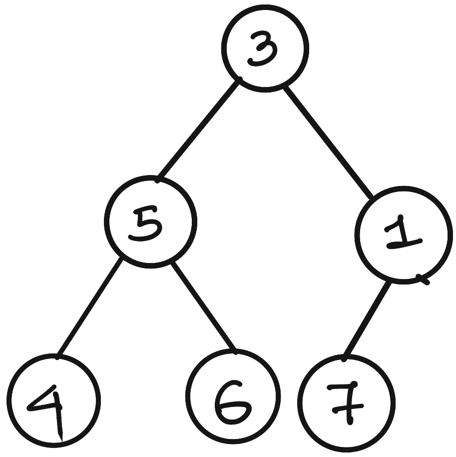


ROOT OF THE TREE: AN INTERNAL NODE THAT DOES NOT HAVE ANY PARENT

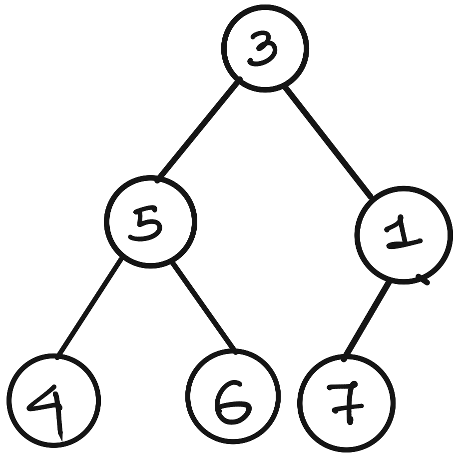
CONSIDER THE FOLLOWING DATA-STRUCTURE

1) EACH NUMBER IS REPRESENTED BY A NODE
2) EACH NODE HAS UNIQUE CHILDRENS.

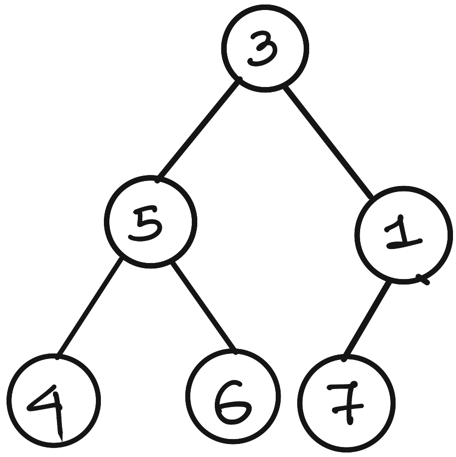




1) EACH INTERNAL NODE HAS ATMOST TWO CHILDREN

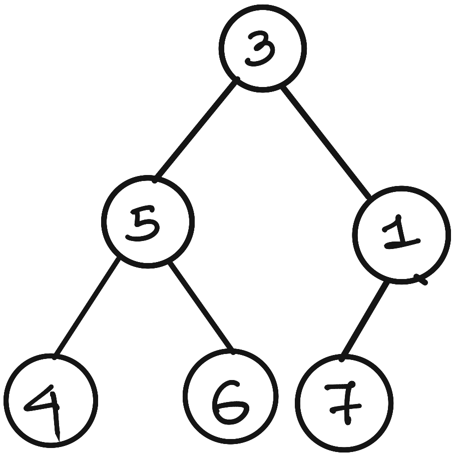


1) EACH INTERNAL NODE HAS ATMOST TWO CHILDREN
2) ALL THE LEVELS EXCEPT THE LAST ONE ARE FULL.
3) LEAVES APPEAR FROM LEFT TO RIGHT AT THE LAST LEVEL.



= COMPLETE BINARY TREE

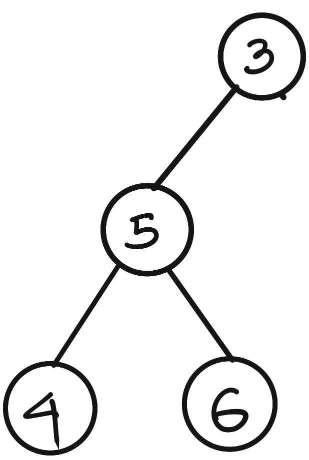
1) EACH INTERNAL NODE HAS ATMOST TWO CHILDREN
2) ALL THE LEVELS EXCEPT THE LAST ONE ARE FULL.
3) LEAVES APPEAR FROM LEFT TO RIGHT AT THE LAST LEVEL.

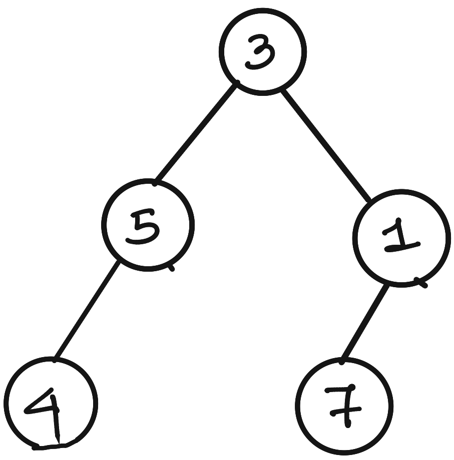


= COMPLETE BINARY TREE

1) EACH INTERNAL NODE HAS ATMOST TWO CHILDREN
2) ALL THE LEVELS EXCEPT THE LAST ONE ARE FULL.
3) LEAVES APPEAR FROM LEFT TO RIGHT AT THE LAST LEVEL.

NOT COMPLETE BINARY TREES





HEAP IS A COMPLETE BINARY TREE IN WHICH EACH INTERNAL NODE v SATISFIES:

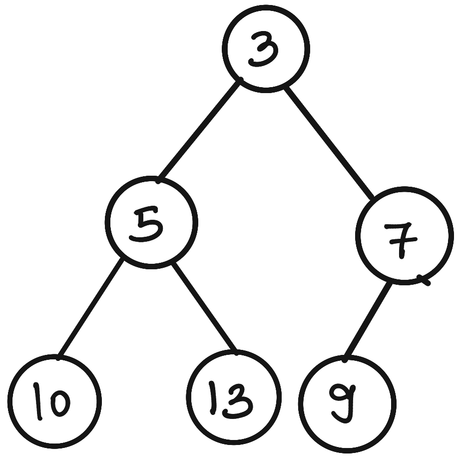
IF v HAS A LEFT CHILD

$$v.\text{value} < (v.\text{left}).\text{value}$$

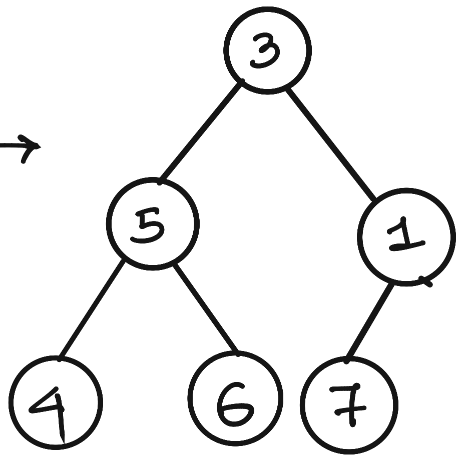
IF v HAS A RIGHT CHILD

$$v.\text{value} < (v.\text{right}).\text{value}$$

A HEAP →



NOT A HEAP →



HEAP IS A COMPLETE BINARY TREE IN WHICH EACH INTERNAL NODE v SATISFIES:

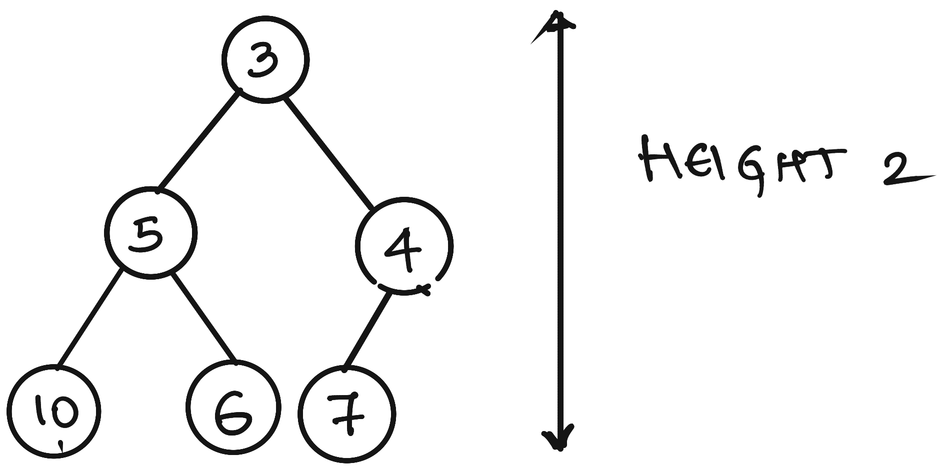
IF v HAS A LEFT CHILD

$$v.\text{value} < (v.\text{left}).\text{value}$$

IF v HAS A RIGHT CHILD

$$v.\text{value} < (v.\text{right}).\text{value}$$

THE HEIGHT OF THE HEAP IS THE DISTANCE FROM THE ROOT TO THE LEAF



HEAP IS A COMPLETE BINARY TREE IN WHICH EACH INTERNAL NODE v SATISFIES:

IF v HAS A LEFT CHILD

$$v.\text{value} < (v.\text{left}).\text{value}$$

IF v HAS A RIGHT CHILD

$$v.\text{value} < (v.\text{right}).\text{value}$$

Q: ASSUME THAT YOU ARE GIVEN A HEAP ON n NODES. WHAT IS THE HEIGHT OF THE HEAP (IN TERMS OF n)?

HEAP IS A COMPLETE BINARY TREE IN WHICH EACH INTERNAL NODE v SATISFIES:

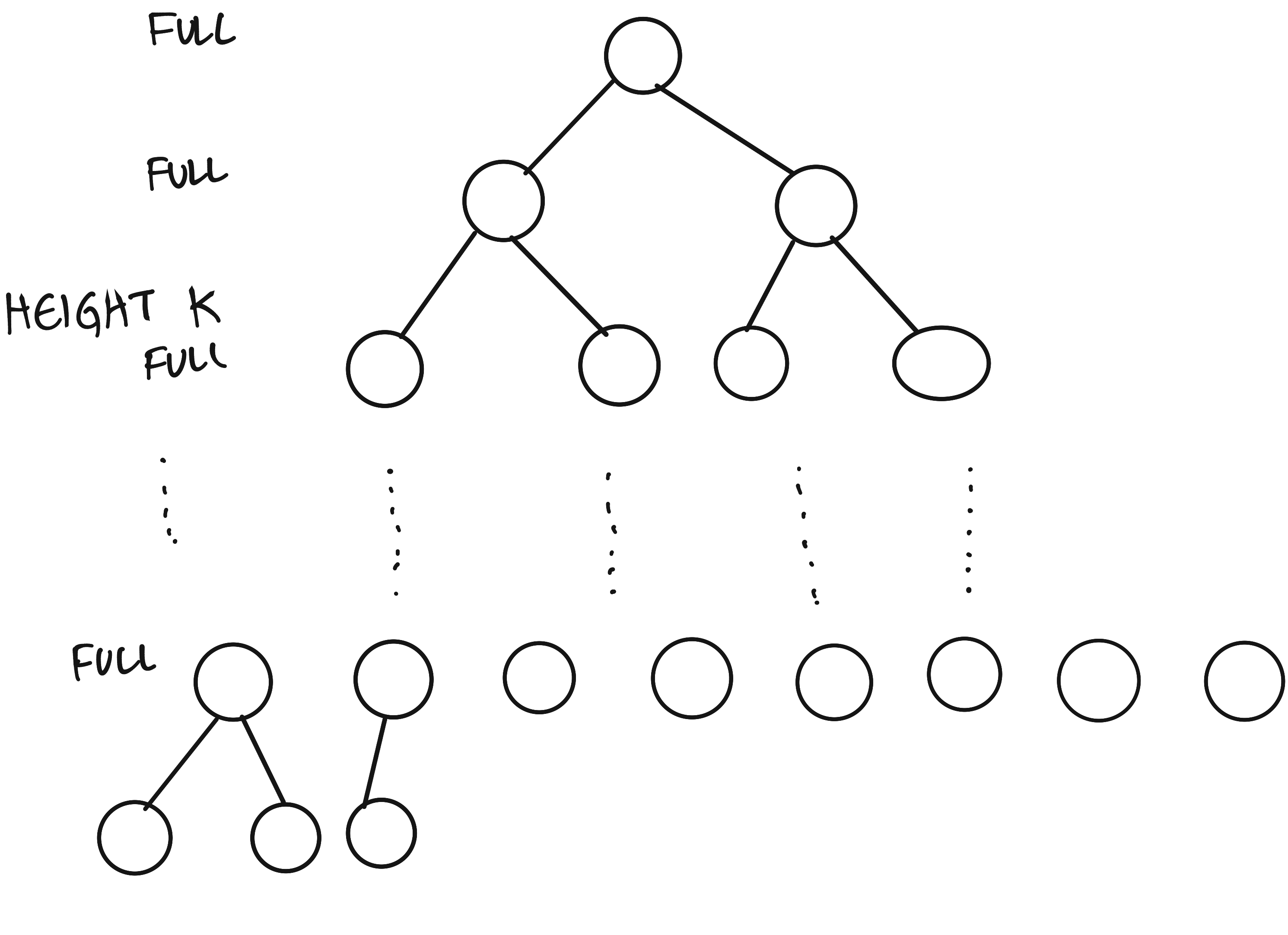
IF v HAS A LEFT CHILD

$v.\text{value} < (v.\text{left}).\text{value}$

IF v HAS A RIGHT CHILD

$v.\text{value} < (v.\text{right}).\text{value}$

Q: ASSUME THAT YOU ARE GIVEN A HEAP ON n NODES. WHAT IS THE HEIGHT OF THE HEAP (IN TERMS OF n)?



HEAP IS A COMPLETE BINARY TREE IN WHICH EACH INTERNAL NODE v SATISFIES:

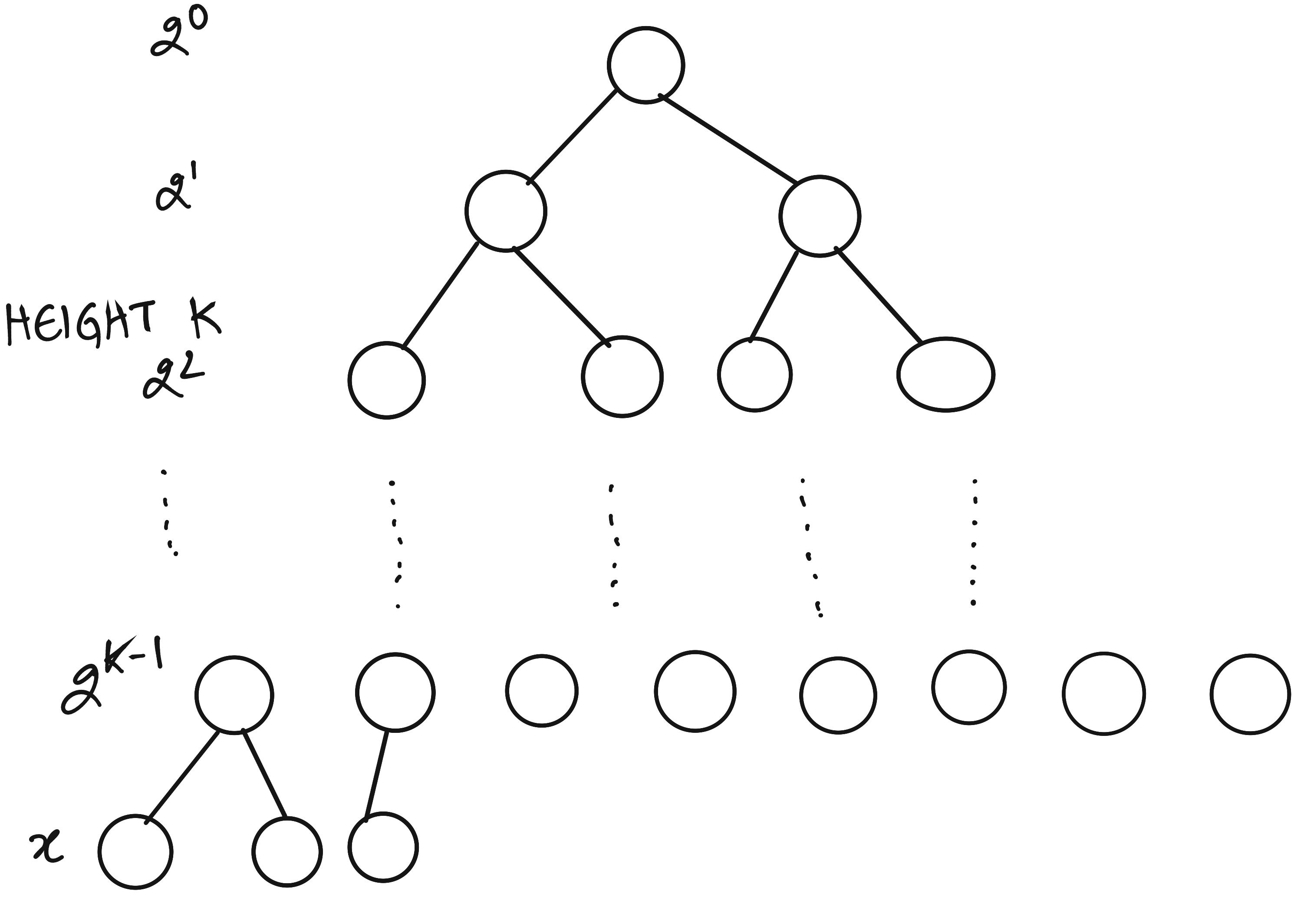
IF v HAS A LEFT CHILD

$$v.\text{value} < (v.\text{left}).\text{value}$$

IF v HAS A RIGHT CHILD

$$v.\text{value} < (v.\text{right}).\text{value}$$

Q: ASSUME THAT YOU ARE GIVEN A HEAP ON n NODES. WHAT IS THE HEIGHT OF THE HEAP (IN TERMS OF n)?



$$\Rightarrow \quad 2^0 + 2^1 + 2^2 + \cdots + 2^{k-1} + x = n$$

$$\Rightarrow \quad 2^0 + 2^1 + \cdots + 2^{k-1} \leq n$$

$$\Rightarrow \quad 2^0 + 2^1 + 2^2 + \dots + 2^{k-1} + x = n$$

$$\Rightarrow \quad 2^0 + 2^1 + \dots + 2^{k-1} \leq n$$

$$\Rightarrow \quad \frac{2^k - 1}{2 - 1} \leq n$$

$$\Rightarrow \quad 2^k \leq n + 1$$

$$\Rightarrow \quad 2^0 + 2^1 + 2^2 + \cdots + 2^{k-1} + x = n$$

$$\Rightarrow \quad 2^0 + 2^1 + \cdots + 2^{k-1} \leq n$$

$$\Rightarrow \quad \frac{2^k - 1}{2 - 1} \leq n$$

$$\Rightarrow \quad 2^k \leq n+1$$

$$\Rightarrow \quad K \leq \log(n+1)$$

$$\Rightarrow \quad 2^0 + 2^1 + 2^2 + \dots + 2^{k-1} + x = n$$

$$\Rightarrow \quad 2^0 + 2^1 + \dots + 2^{k-1} \leq n$$

$$\Rightarrow \quad \frac{2^k - 1}{2-1} \leq n$$

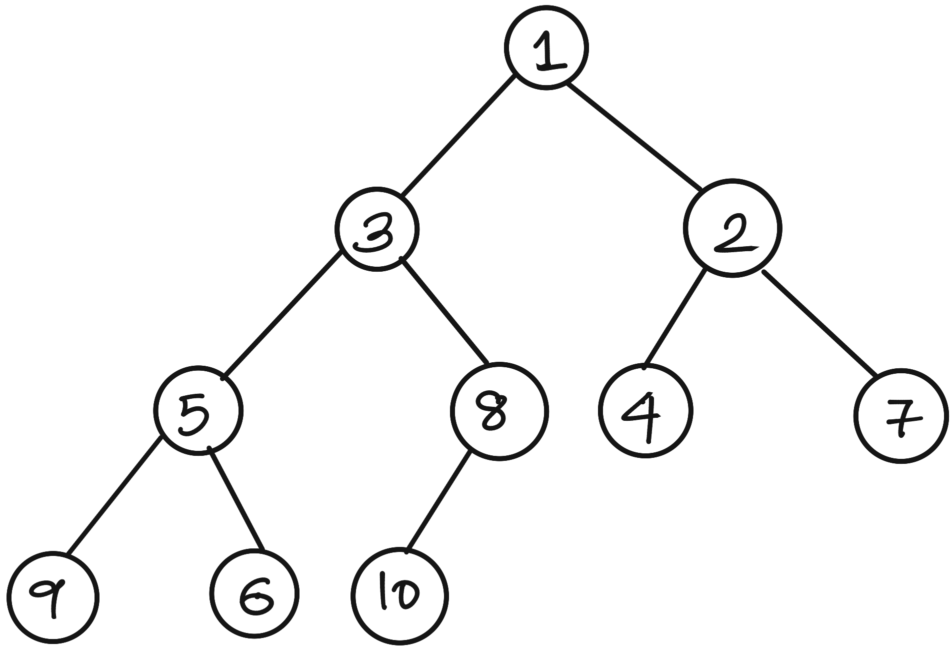
$$\Rightarrow \quad 2^k \leq n+1$$

$$\Rightarrow \quad K \leq \log(n+1)$$

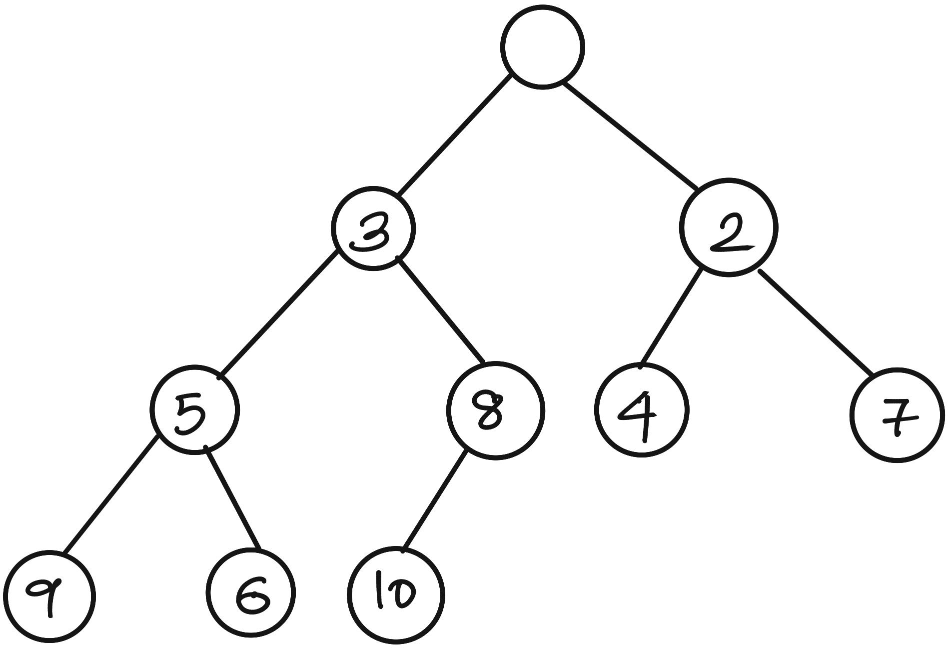
$$\Rightarrow \quad K = O(\log n)$$

<u>LEMMA</u>: THE HEIGHT OF A HEAP IS $O(\log n)$.

Q: GIVEN A MIN-HEAP, CAN IT BE USED TO SORT NUMBERS?



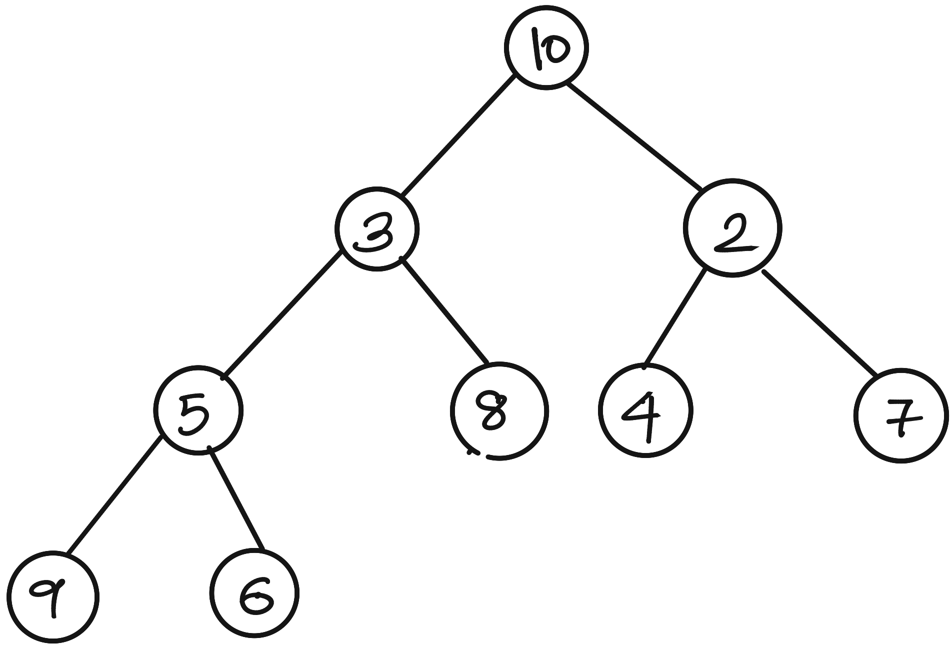
Q: GIVEN A MIN-HEAP, CAN IT BE USED TO SORT NUMBERS?



1

(1) REMOVE THE VALUE AT ROOT

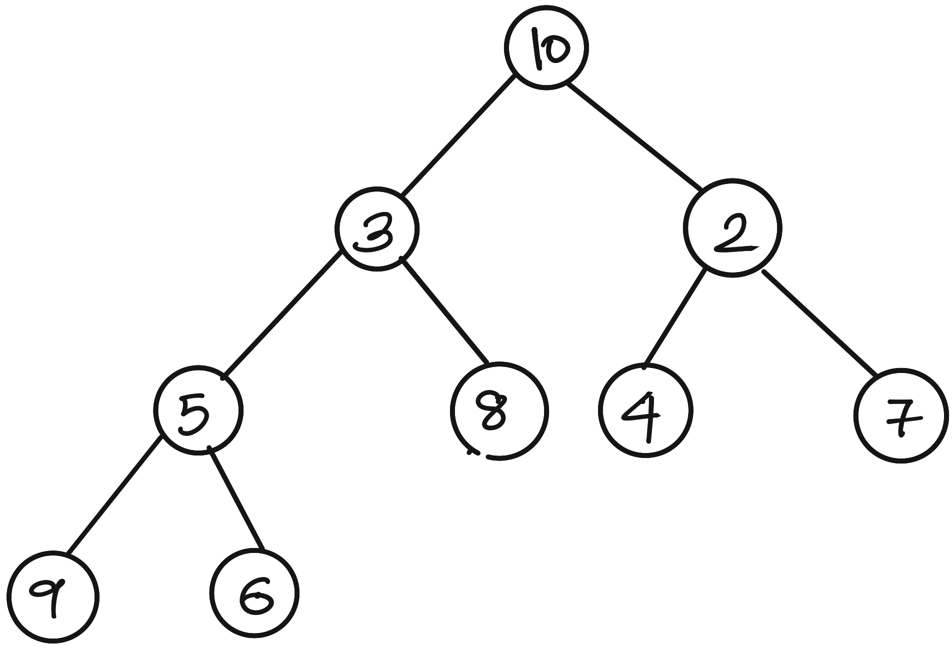
Q: GIVEN A MIN-HEAP, CAN IT BE USED TO SORT NUMBERS?



1

(1) REMOVE THE VALUE AT ROOT
(2) REPLACE THE LAST LEAF AS ROOT.

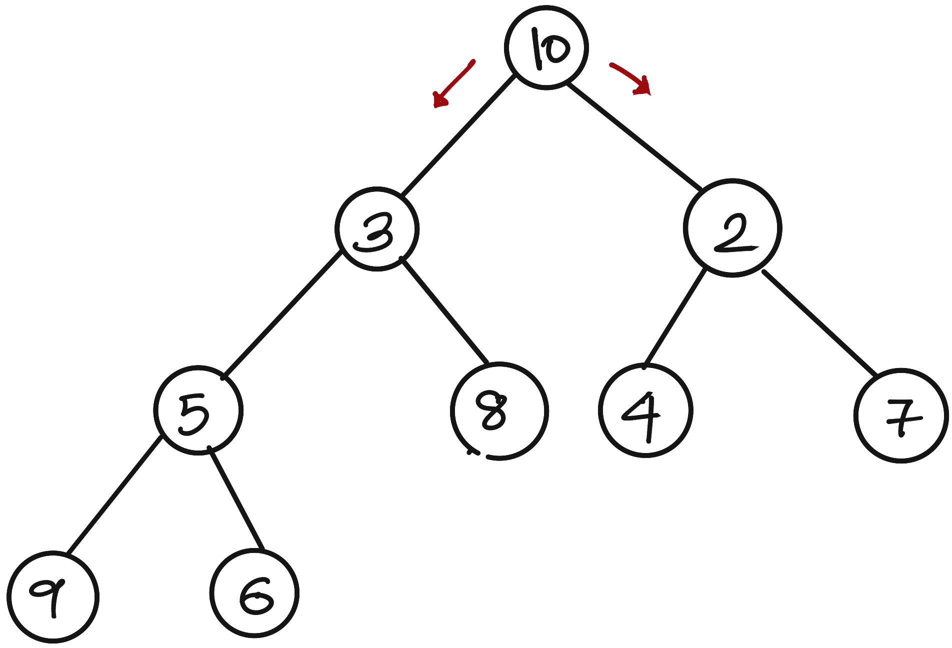
Q: GIVEN A MIN-HEAP, CAN IT BE USED TO SORT NUMBERS?



1

(1) REMOVE THE VALUE AT ROOT
(2) REPLACE THE LAST LEAF AS ROOT.
(3) SHIFT ROOT DOWN

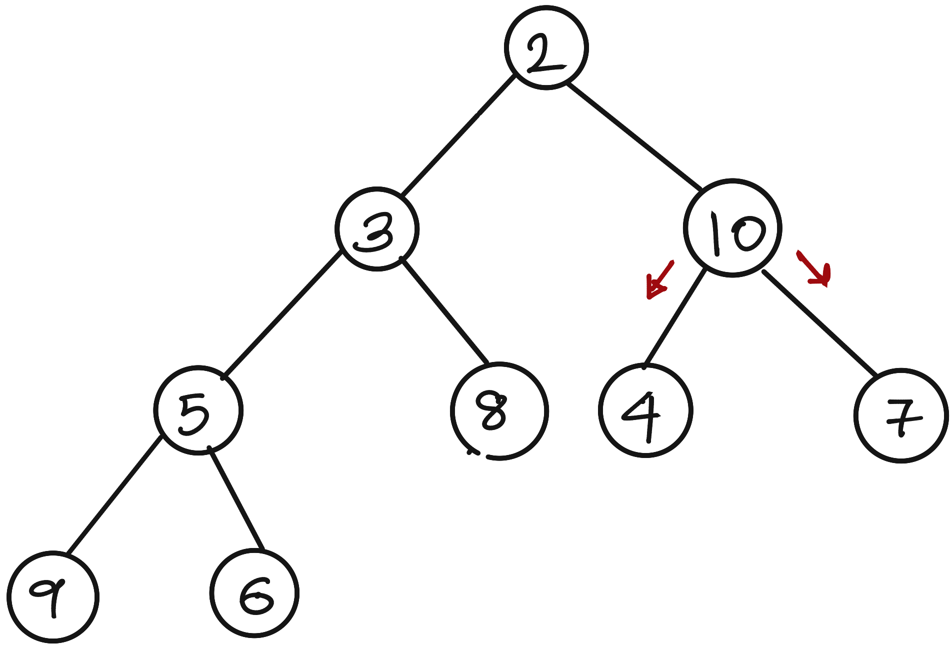
Q: GIVEN A MIN-HEAP, CAN IT BE USED TO SORT NUMBERS?



1

(1) REMOVE THE VALUE AT ROOT
(2) REPLACE THE LAST LEAF AS ROOT.
(3) SHIFT ROOT DOWN

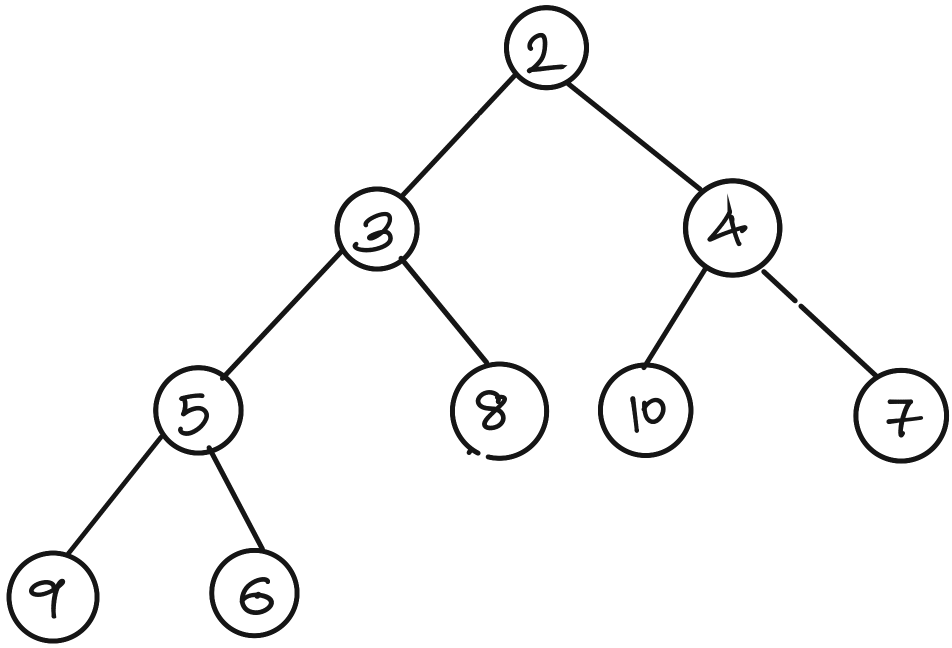
Q: GIVEN A MIN-HEAP, CAN IT BE USED TO SORT NUMBERS?



1

(1) REMOVE THE VALUE AT ROOT
(2) REPLACE THE LAST LEAF AS ROOT.
(3) SHIFT ROOT DOWN

Q: GIVEN A MIN-HEAP, CAN IT BE USED TO SORT NUMBERS?



1

(1) REMOVE THE VALUE AT ROOT
(2) REPLACE THE LAST LEAF AS ROOT.
(3) SHIFT ROOT DOWN

```
HEAPSORT ( HEAP H)
{   WHILE H IS NOT EMPTY
    {   OUTPUT THE VALUE AT THE ROOT OF H;
        v ← LAST LEAF OF H;
        root.value ← v.value ;
        delete node v;
        shift-down (root)
    }
}
```

```
HEAPSORT ( HEAP H)
{   WHILE H IS NOT EMPTY
    {   OUTPUT THE VALUE AT THE ROOT OF H;
        v ← LAST LEAF OF H;
        root.value ← v.value ;
        delete node v;
        shift-down(root)
    }
}


shift-down ( NODE v)
{   if( v.value < (v.left).value &
        v.value < (v.right).value)
          return;
    else
    {   u ← child of v with minimum
                               value ;
        swap( v.value , u.value)
        shift-down(u)
    }
}
```

```
HEAPSORT( HEAP H)
{  WHILE H IS NOT EMPTY
   {  OUTPUT THE VALUE AT THE ROOT OF H;
      v ← LAST LEAF OF H;
      root.value ← v.value ;
      delete node v;
      shift-down(root)
   }
}


shift-down( NODE v)
{  if( v.value < (v.left).value &
       v.value < (v.right).value)
          return;
   else
   {  u ← child of v with minimum
                          value ;
      swap( v.value , u.value)
      shift-down(u)
   }
}
```

RUNNING TIME = $O(\log n)$

RUNTIME OF HEAPSORT (GIVEN A HEAP)

$= O(n \log n)$

RUNTIME OF HEAPSORT (GIVEN A HEAP)

$= O(n \log n)$

A:

1	5	3	7	9	6	4	11	8	10	12

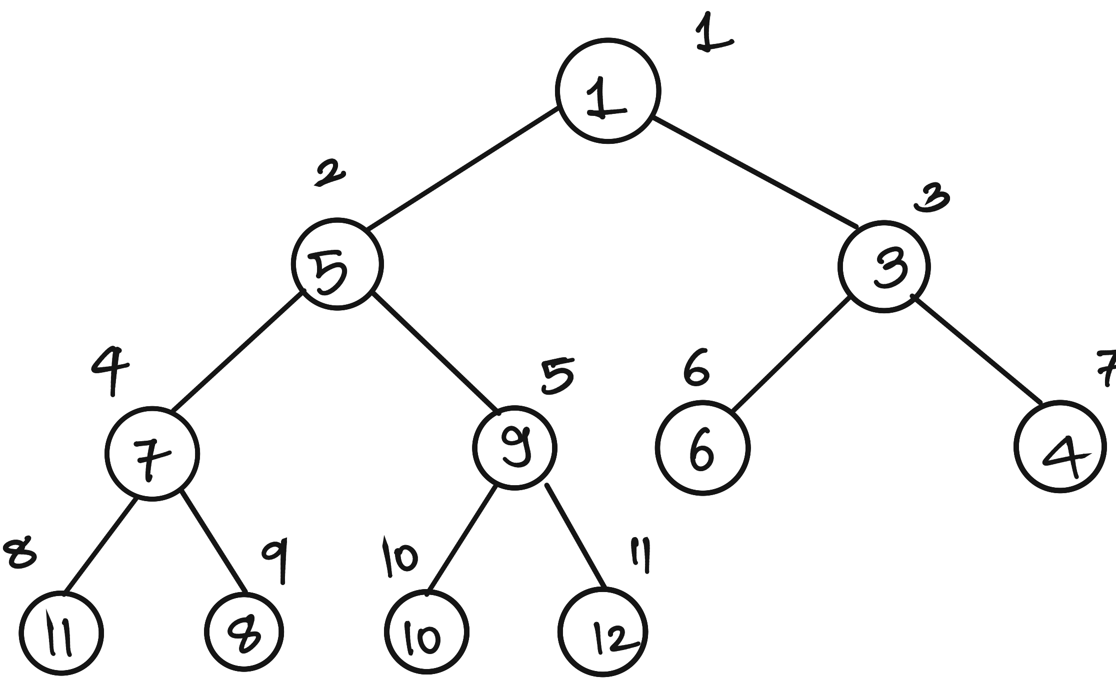
CAN YOU IDENTIFY THE HEAP IN THIS ARRAY ?

RUNTIME OF HEAPSORT (GIVEN A HEAP)
$= O(n \log n)$

A:

1	5	3	7	9	6	4	11	8	10	12

CAN YOU IDENTIFY THE HEAP IN THIS ARRAY ?



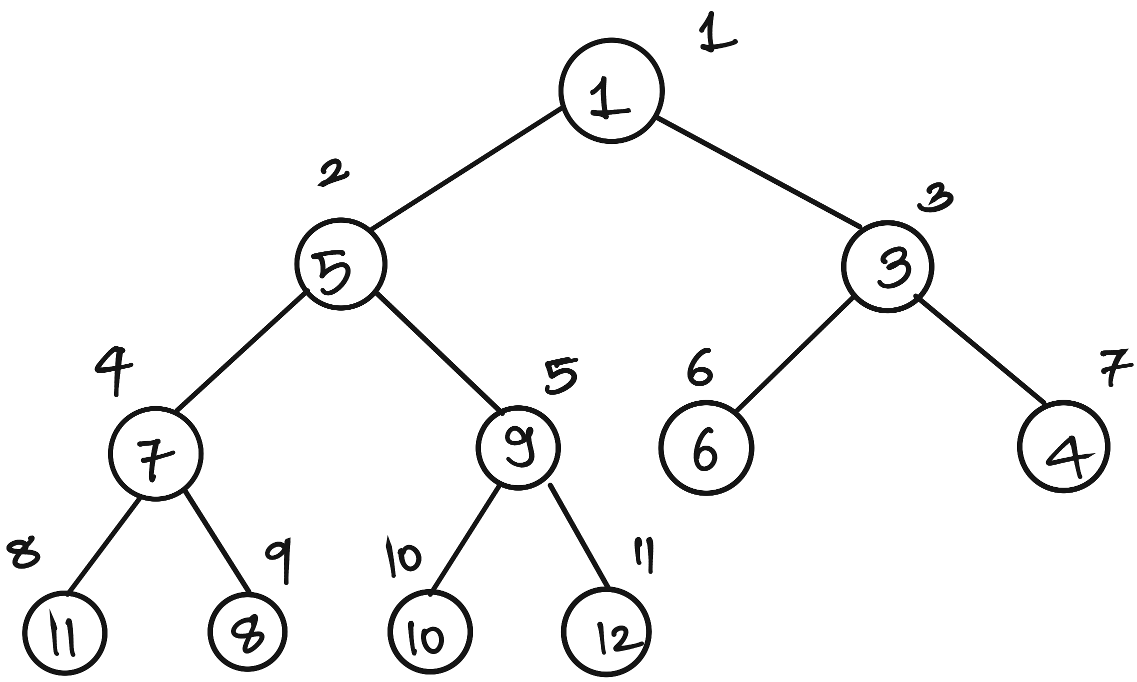
THIS ARRAY IMPLICITLY REPRESENTS A HEAP.

RUNTIME OF HEAPSORT (GIVEN A HEAP)
$= O(n \log n)$

A:

1	5	3	7	9	6	4	11	8	10	12

CAN YOU IDENTIFY THE HEAP IN THIS ARRAY?



THIS ARRAY IMPLICITLY REPRESENTS A HEAP.

THE CHILD OF INDEX i are

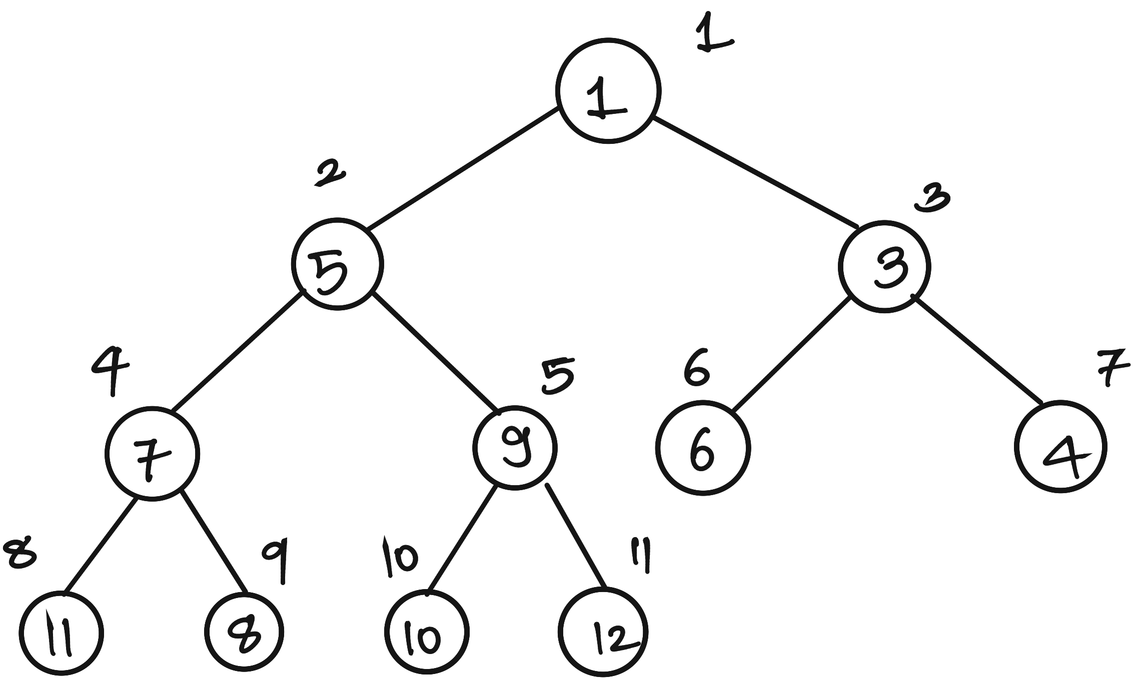
RUNTIME OF HEAPSORT (GIVEN A HEAP)

$= O(n \log n)$

A:

1	5	3	7	9	6	4	11	8	10	12

CAN YOU IDENTIFY THE HEAP IN THIS ARRAY ?



THIS ARRAY IMPLICITLY REPRESENTS A HEAP.

THE CHILD OF INDEX i are $2i$

$2i+1$

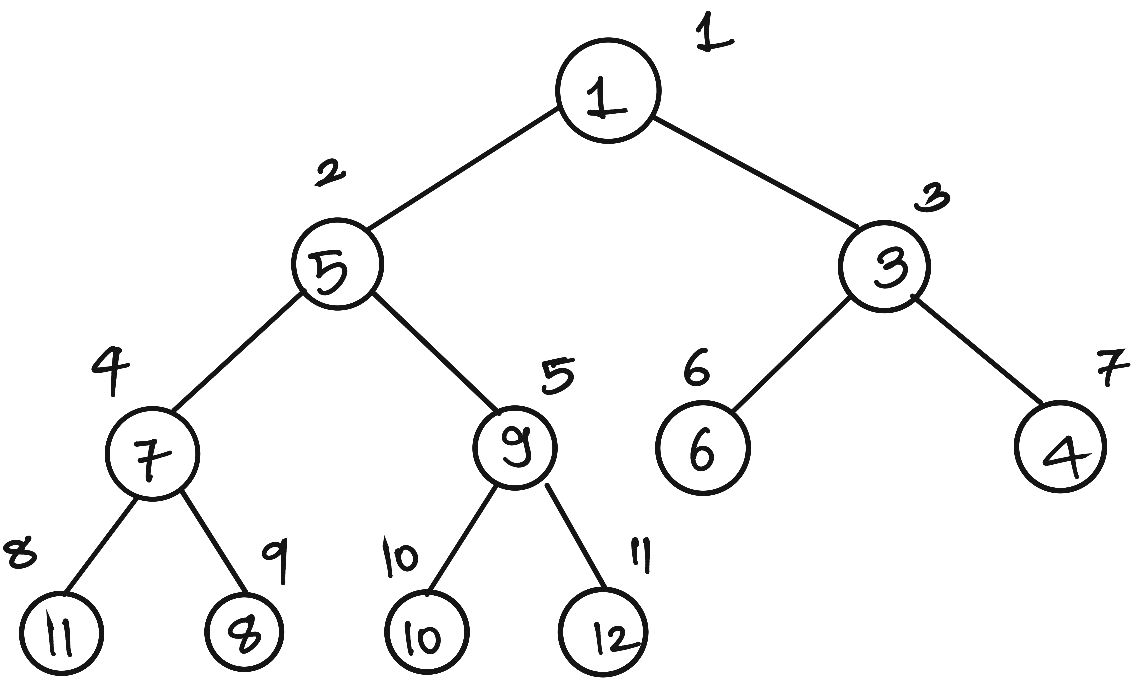
RUNTIME OF HEAPSORT (GIVEN A HEAP)

$= O(n \log n)$

A:

1	5	3	7	9	6	4	11	8	10	12

CAN YOU IDENTIFY THE HEAP IN THIS ARRAY ?



THIS ARRAY IMPLICITLY REPRESENTS A HEAP.

THE CHILD OF INDEX i ARE $2i$, $2i+1$

THE PARENT OF INDEX i IS $\lfloor i/2 \rfloor$

```
HEAPSORT ( HEAP H)
{   WHILE H IS NOT EMPTY
    {   OUTPUT THE VALUE AT THE ROOT OF H;
        v ← LAST LEAF OF H;
        root.value ← v.value ;
        delete node v;
        shift-down (root)
    }
}
```

```
HEAPSORT( HEAP H)
{   WHILE H IS NOT EMPTY
    {   OUTPUT THE VALUE AT THE ROOT OF H;
        v ← LAST LEAF OF H;
        root.value ← v.value ;
        delete node v;
        shift-down(root)
    }
}
```

⇓

(A IMPLICITLY REPRESENTS A HEAP)

```
HEAPSORT( ARRAY A)
{   i = n ;
    while i ≥ 1
    {   OUTPUT A[1];
        v ← i
        A[1] ← A[v] ;
        i ← i-1 ;
        shift-down(1) ;
    }
}
```

```
shift-down(NODE v)
{   if( v.value < (v.left).value &
        v.value < (v.right).value)
          return;
    else
    {   u ← child of v with minimum
                        value;
        swap(v.value , u.value)
        shift-down(u)
    }
}
```

```
shift-down( NODE v)
{   if( v.value < (v.left).value &
        v.value < (v.right).value)
          return;
    else
    {   u ← child of v with minimum
                          value;
        swap( v.value , u.value)
        shift-down(u)
    }
}
```

```
shift-down( INDEX i )
{   if( A[i]   <   A[2i]        & )
        A[i]   <   A[2i+1]
          return;
    else
    {   i' ← { 2i      ,   if A[2i] < A[2i+1]
             { 2i+1    ,   otherwise
        swap( A[i], A[i'])
        shift-down( i')
    }
}
```

RUNNING TIME OF HEAP-SORT (GIVEN AN IMPLICIT HEAP) IS $O(n \log n)$.

RUNNING TIME OF HEAP-SORT (GIVEN AN IMPLICIT HEAP) IS $O(n \log n)$.

Q: HOW DID WE GET HEAP IN THE FIRST PLACE ?

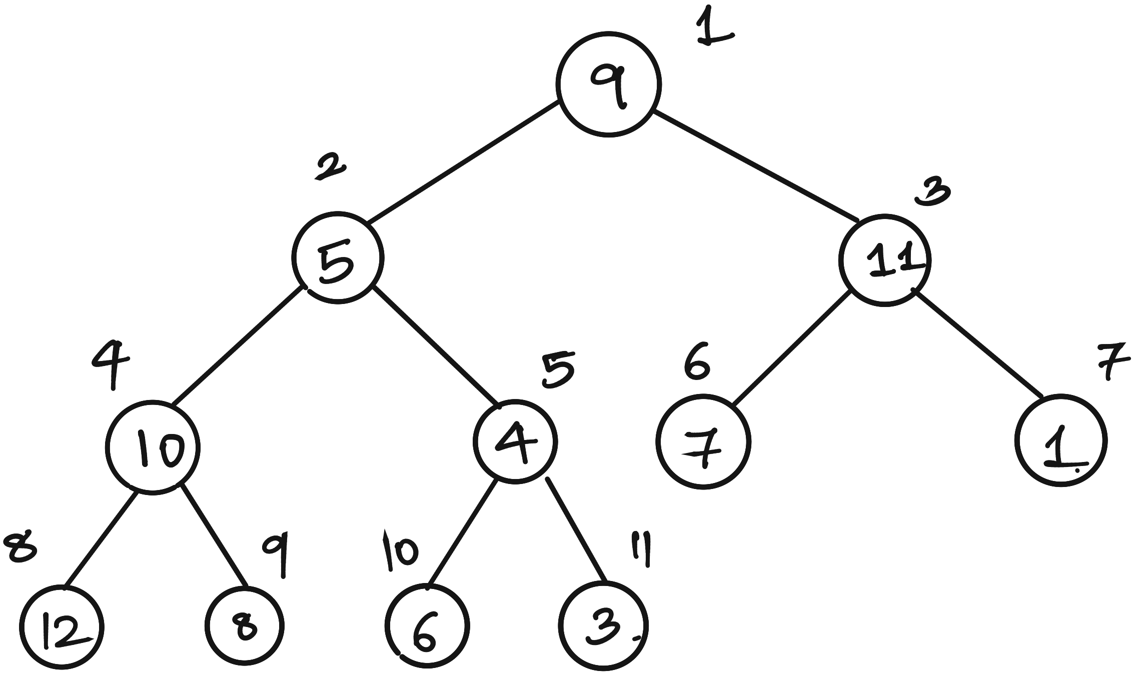
A	9	5	11	10	4	7	1	12	8	6	3.

$\Downarrow$ TRANSFORM INTO A HEAP

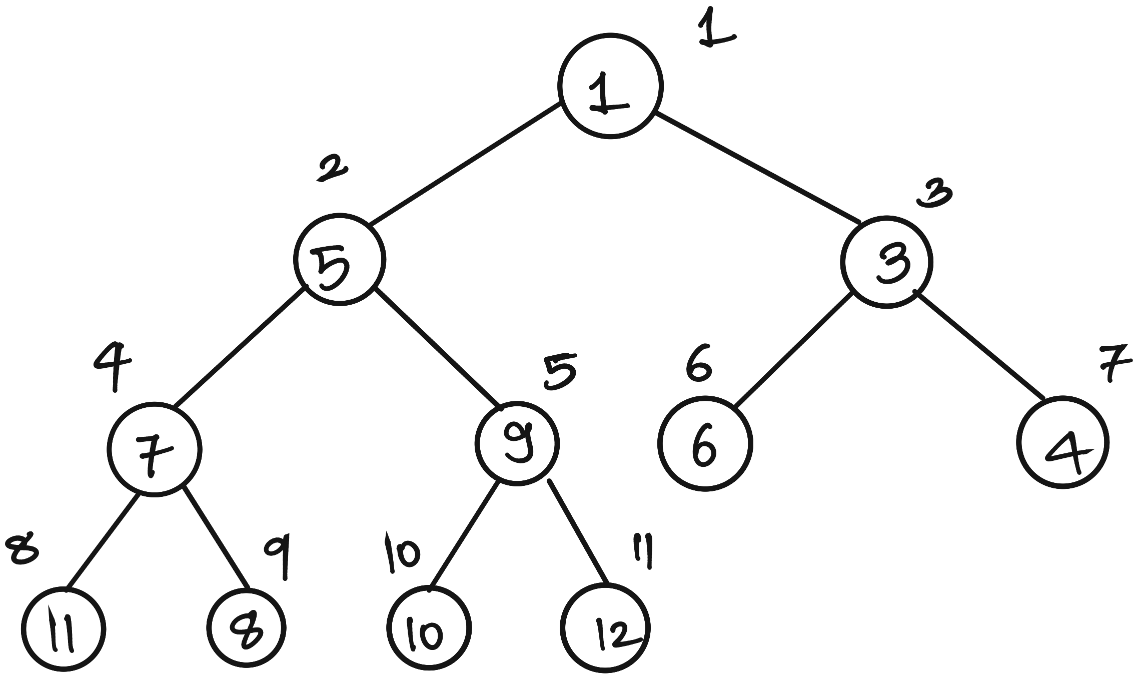
A	1	5	3	7	9	6	4	11	8	10	12

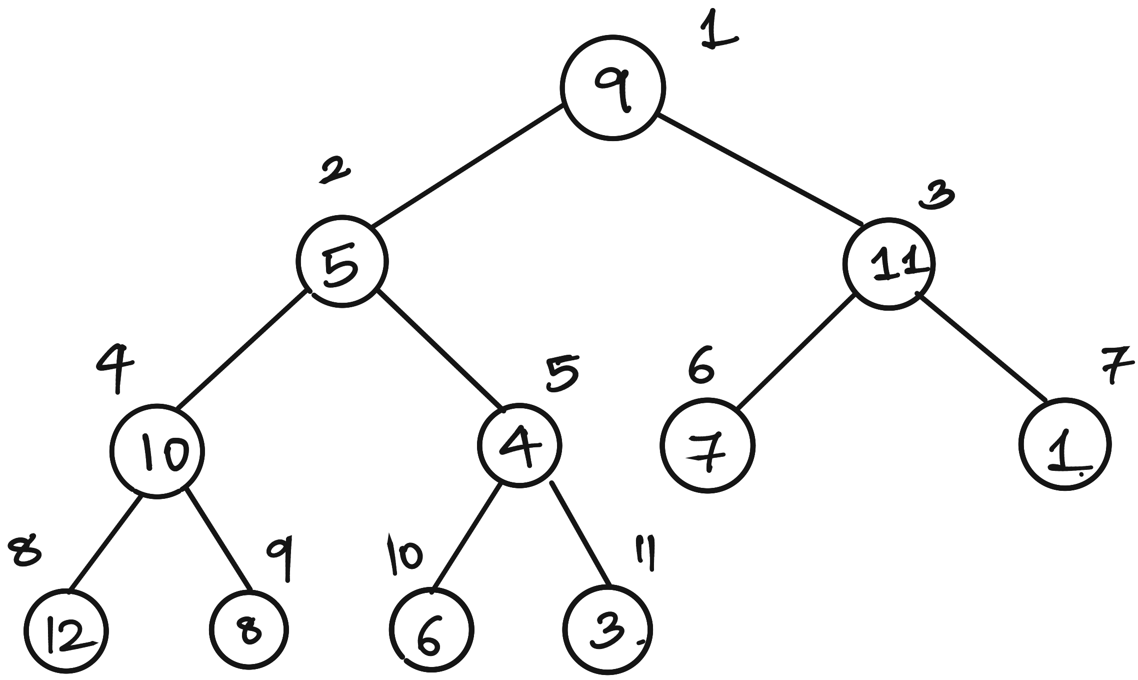
$\Downarrow$ HEAPSORT $O(n \log n)$

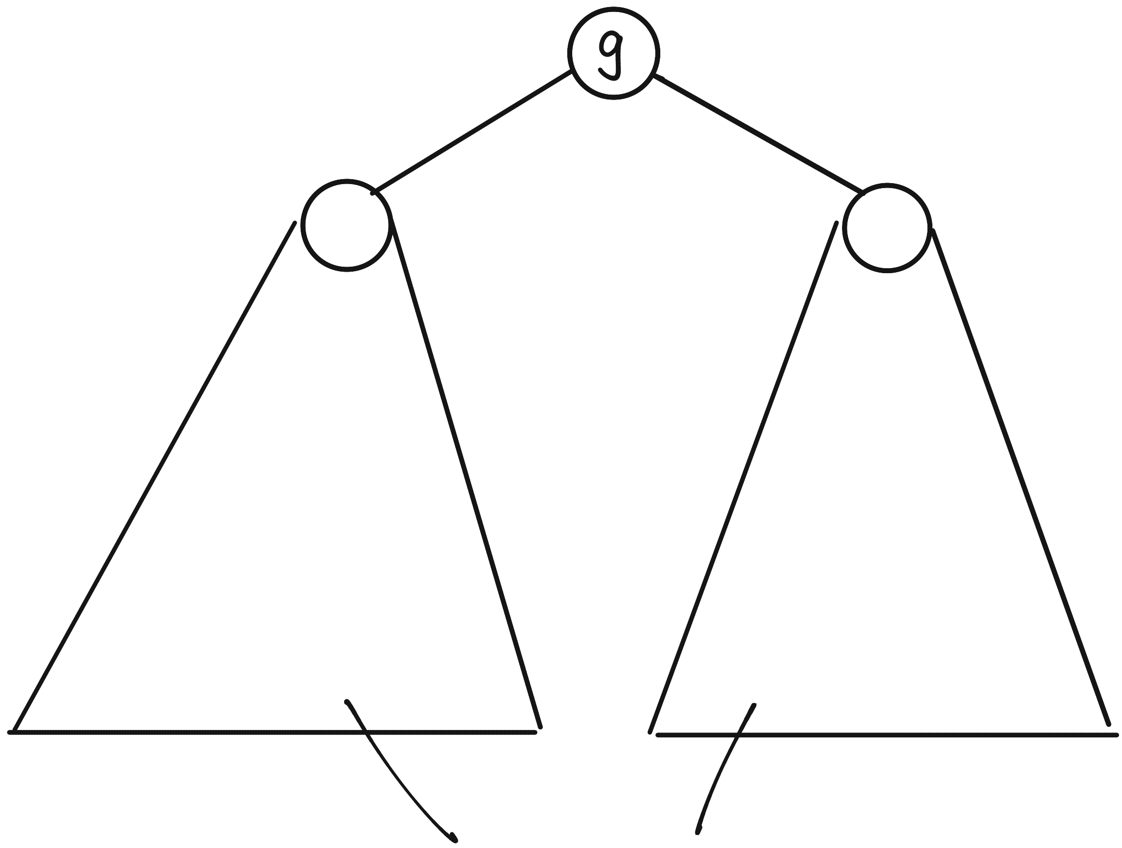
1 3 4 5 6 7 8 9 10 11 12



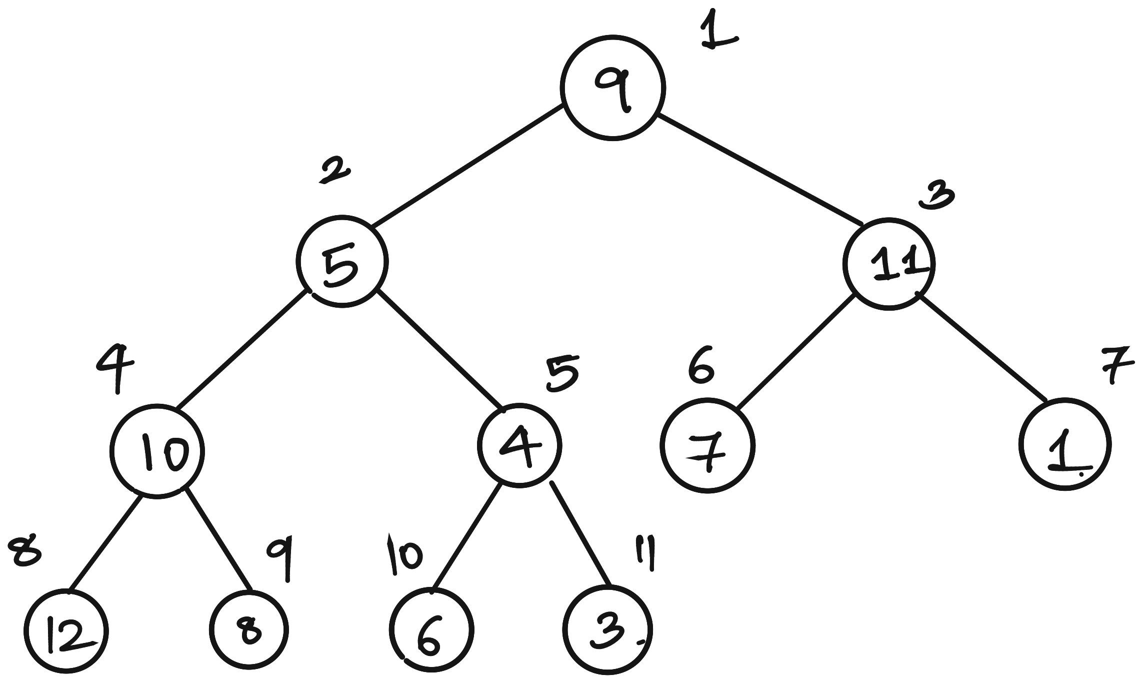
⇓ TRANSFORM TO A HEAP

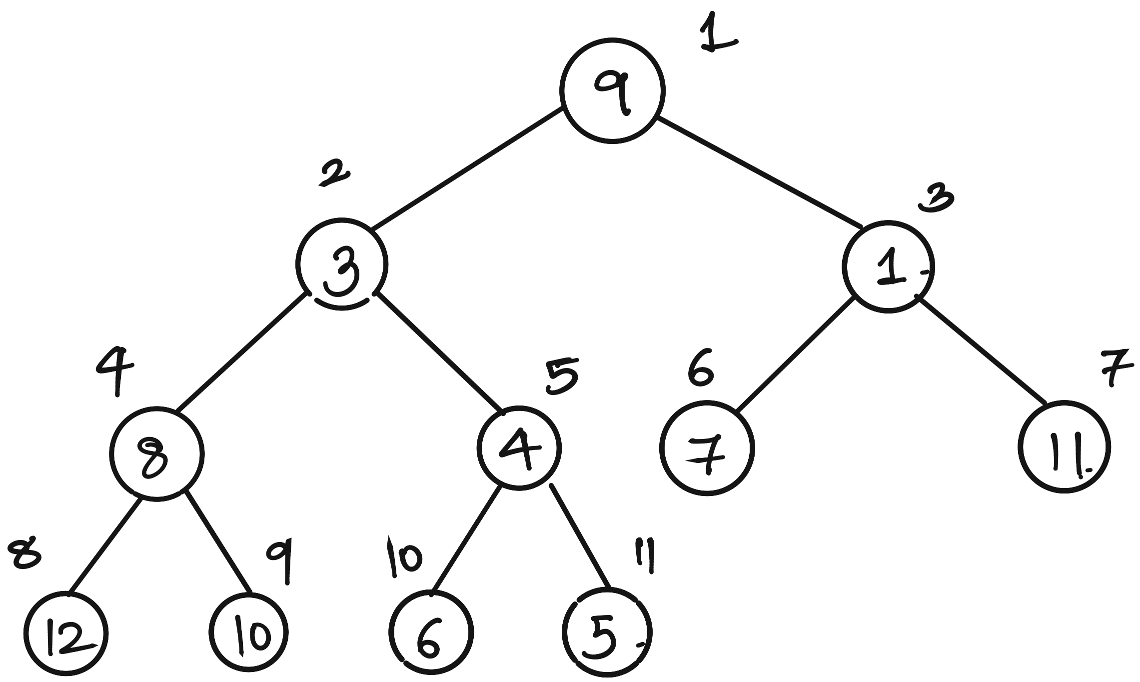




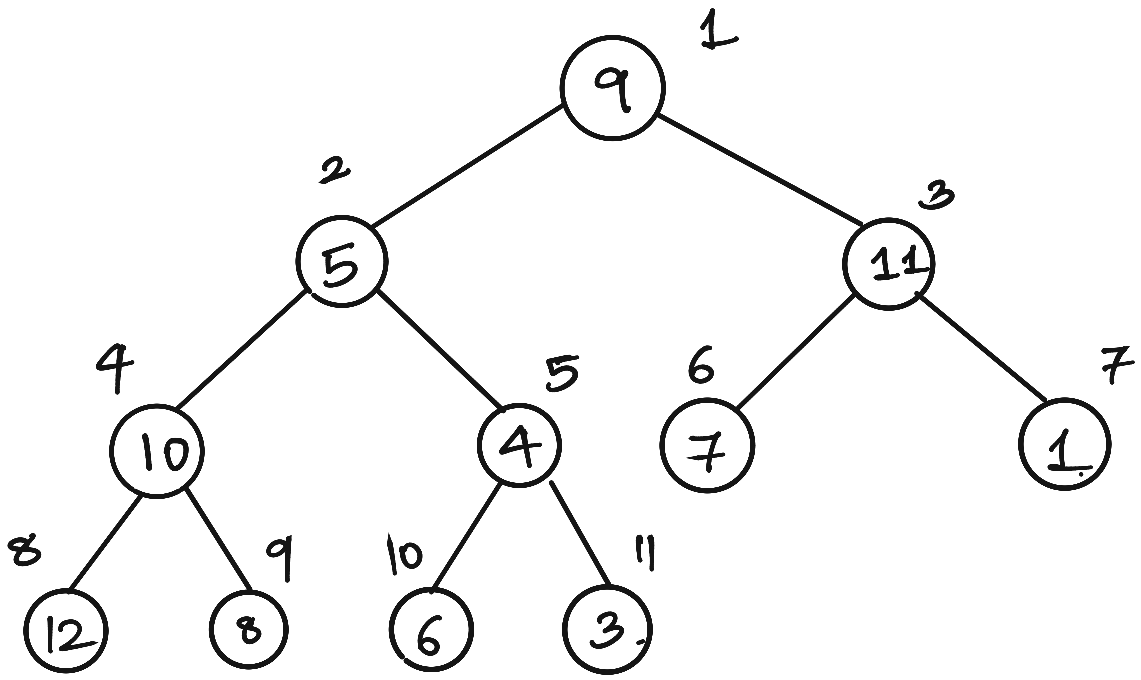


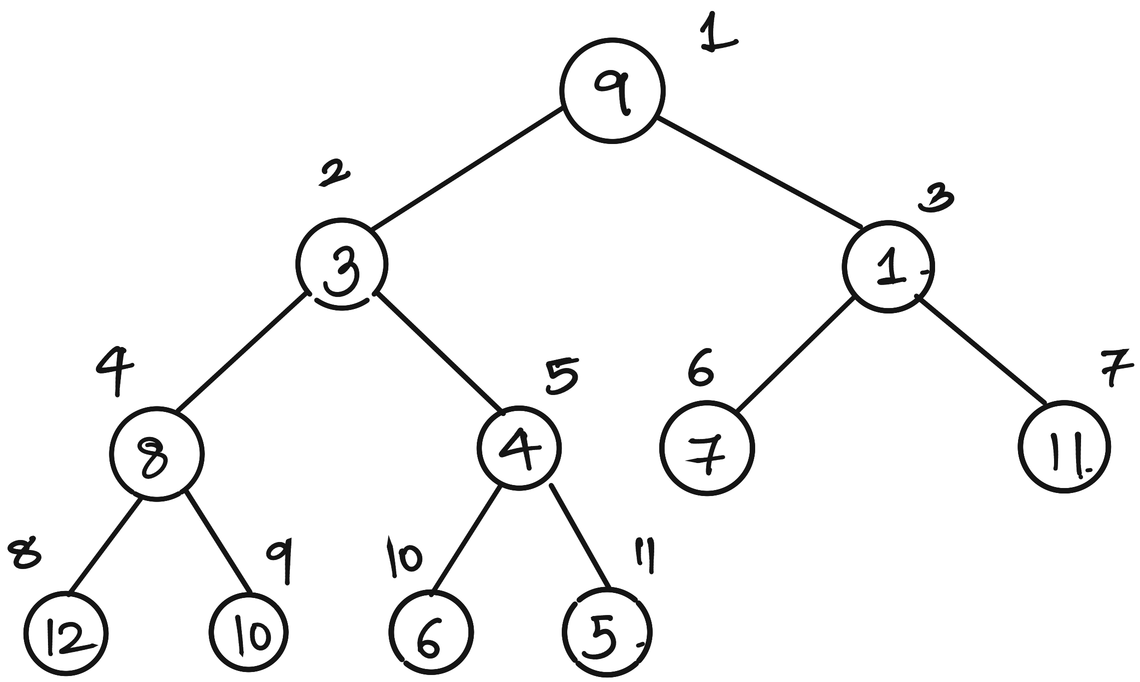
IF SOMEHOW, I CAN MAKE THESE A HEAP,





Q: WHAT WILL YOU DO NOW ?





Q: WHAT WILL YOU DO NOW?

A: Shift-down(1)

ALGORITHM TO BUILD HEAP

BUILD-HEAP (1)

```
BUILD-HEAP(i)
{   IF ( INDEX i HAS A LEFT CHILD)
        BUILD-HEAP(2i)
    IF (INDEX i HAS A RIGHT CHILD)
        BUILD-HEAP(2i+1)
    SHIFT-DOWN(i)
}
```

ALGORITHM TO BUILD HEAP

BUILD-HEAP(1)

```
BUILD-HEAP(i)
{   IF (INDEX i HAS A LEFT CHILD)
        BUILD-HEAP(2i)
    IF (INDEX i HAS A RIGHT CHILD)
        BUILD-HEAP(2i+1)
    SHIFT-DOWN(i)
}
```

<u>CORRECTNESS</u>

ALGORITHM TO BUILD HEAP
BUILD-HEAP (1)

```
BUILD-HEAP(i)
{   IF ( INDEX i HAS A LEFT CHILD)
        BUILD-HEAP(2i)
    IF (INDEX i HAS A RIGHT CHILD)
        BUILD-HEAP(2i+1)
    SHIFT-DOWN(i)
}
```

CORRECTNESS

→ THE PATTERN LIES IN THE ALGORITHM.

→ HEAPS OF SMALLER SIZE ARE BUILT FIRST

ALGORITHM TO BUILD HEAP

BUILD-HEAP (1)

```
BUILD-HEAP(i)
{   IF ( INDEX i HAS A LEFT CHILD)
          BUILD-HEAP(2i)
    IF (INDEX i HAS A RIGHT CHILD)
          BUILD-HEAP(2i+1)
    SHIFT-DOWN(i)
}
```

CORRECTNESS

- THE PATTERN LIES IN THE ALGORITHM.
- HEAPS OF SMALLER SIZE ARE BUILT FIRST

LEMMA: BUILD-HEAP CORRECTLY BUILDS A HEAP ON n NODES.

ALGORITHM TO BUILD HEAP

BUILD-HEAP(1)

```
BUILD-HEAP(i)
{   IF ( INDEX i HAS A LEFT CHILD)
        BUILD-HEAP(2i)
    IF (INDEX i HAS A RIGHT CHILD)
        BUILD-HEAP(2i+1)
    SHIFT-DOWN(i)
}
```

CORRECTNESS

→ THE PATTERN LIES IN THE ALGORITHM.

→ HEAPS OF SMALLER SIZE ARE BUILT FIRST

LEMMA: BUILD-HEAP CORRECTLY BUILDS A HEAP $\underbrace{\text{ON } n \text{ NODES}}_{\text{INDUCTION ON } n}$.

<u>LEMMA</u>: BUILD-HEAP CORRECTLY BUILDS A HEAP ON n NODES.

<u>PROOF</u>: 1) BASE CASE: $n = 1$

9 TRIVIALLY TRUE

LEMMA: BUILD-HEAP CORRECTLY BUILDS A HEAP ON n NODES.

PROOF: 1) BASE CASE: $n = 1$

9 TRIVIALLY TRUE

2) INDUCTION HYPOTHESIS

BUILD-HEAP CORRETLY BUILDS A HEAP ON $1, 2, 3, \ldots, n-1$ NODES

3) PROVE THAT BUILD-HEAP CORRECTLY BUILDS A HEAP ON n NODES

LEMMA: BUILD-HEAP CORRECTLY BUILDS A HEAP ON n NODES.

PROOF: 1) BASE CASE: $n = 1$

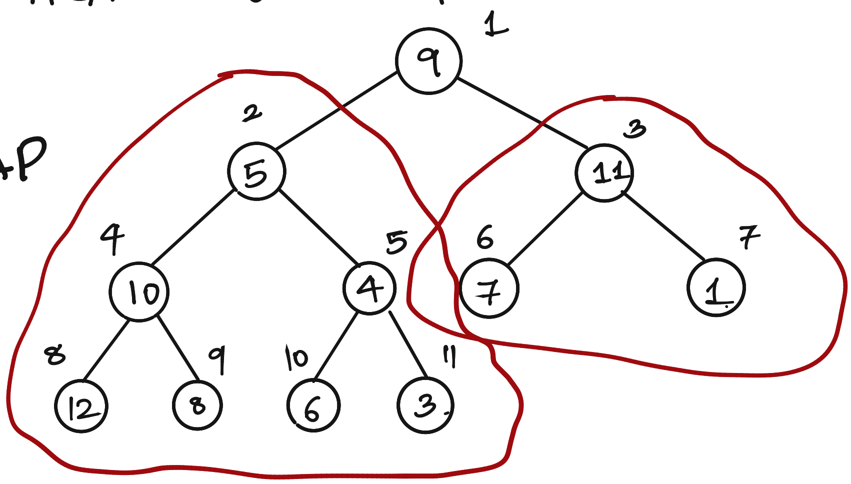
TRIVIALLY TRUE

2) INDUCTION HYPOTHESIS

BUILD-HEAP CORRETLY BUILDS A HEAP ON $1, 2, 3, \ldots, n-1$ NODES

3) PROVE THAT BUILD-HEAP CORRECTLY BUILDS A HEAP ON n NODES

BUILD HEAP LEFT



BUILD HEAP RIGHT

LEMMA: BUILD-HEAP CORRECTLY BUILDS A HEAP ON n NODES.

PROOF: 1) BASE CASE: $n = 1$

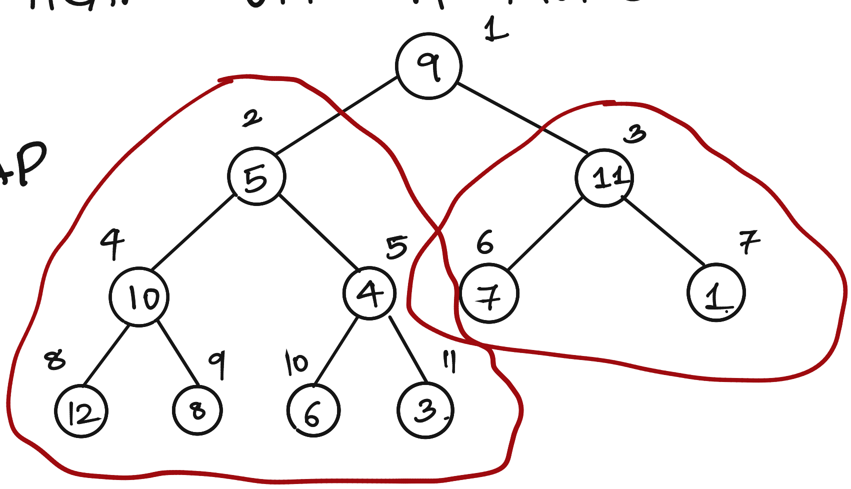
TRIVIALLY TRUE

2) INDUCTION HYPOTHESIS

BUILD-HEAP CORRETLY BUILDS A HEAP ON $1, 2, 3, \ldots, n-1$ NODES

3) PROVE THAT BUILD-HEAP CORRECTLY BUILDS A HEAP ON n NODES

BUILD HEAP LEFT



BUILD HEAP RIGHT

THE NUMBER OF NODES IN THE LEFT AND RIGHT SUBTRE $< n$

$\Rightarrow$ WE CAN USE INDUCTION HYPOTHESIS

LEMMA: BUILD-HEAP CORRECTLY BUILDS A HEAP ON n NODES.

PROOF: 1) BASE CASE: $n = 1$

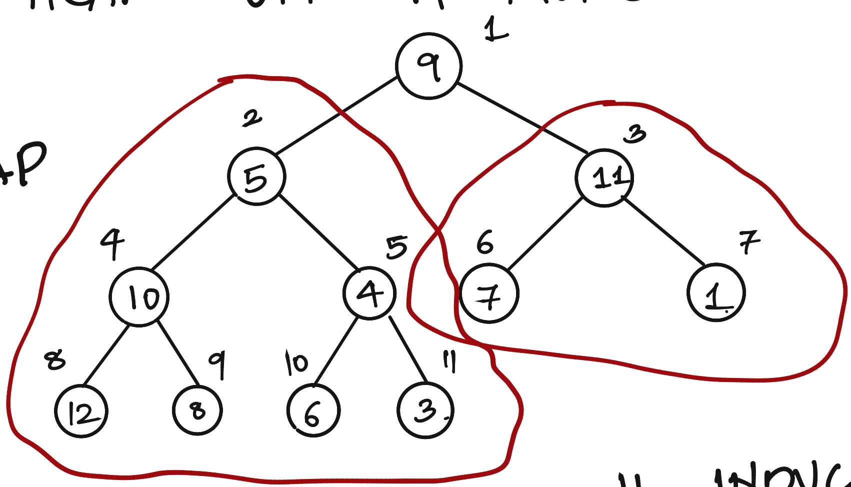
TRIVIALLY TRUE

2) INDUCTION HYPOTHESIS

BUILD-HEAP CORRETLY BUILDS A HEAP ON $1, 2, 3, \ldots, n-1$ NODES

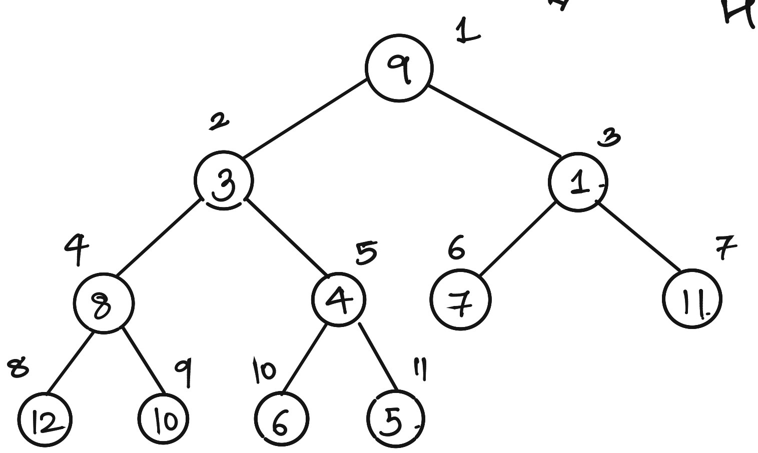
3) PROVE THAT BUILD-HEAP CORRECTLY BUILDS A HEAP ON n NODES

BUILD HEAP LEFT



BUILD HEAP RIGHT

$\Downarrow$ INDUCTION HYPOTHESIS



LAST STEP : SHOW THAT SHIFT-DOWN WORKS CORRECTLY

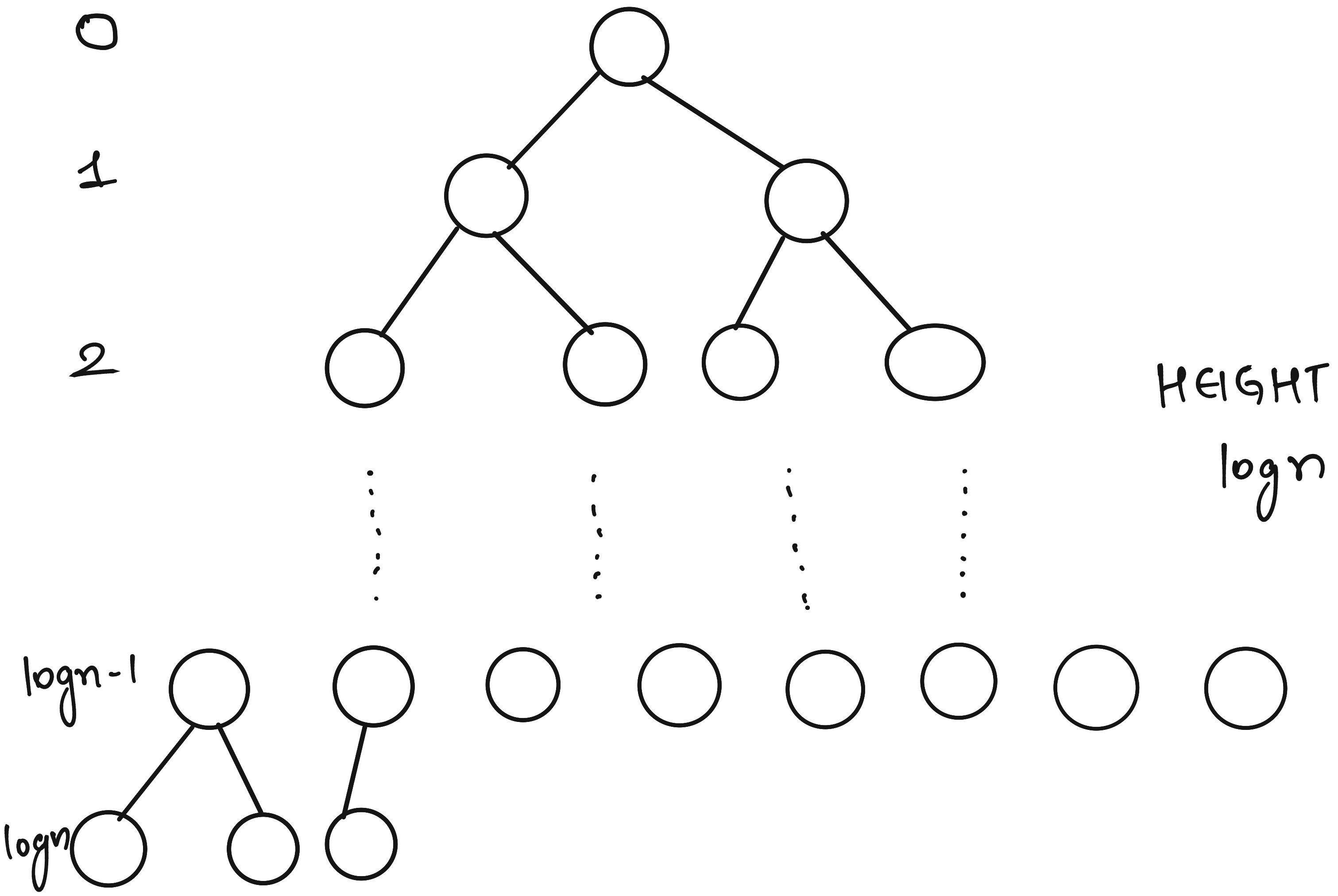
1) HOME-WORK
2) SEE NOTES
   ↳ IN NOTES, NON-RECURSIVE VERSION OF ALGORITHM.

ALGORITHM TO BUILD HEAP

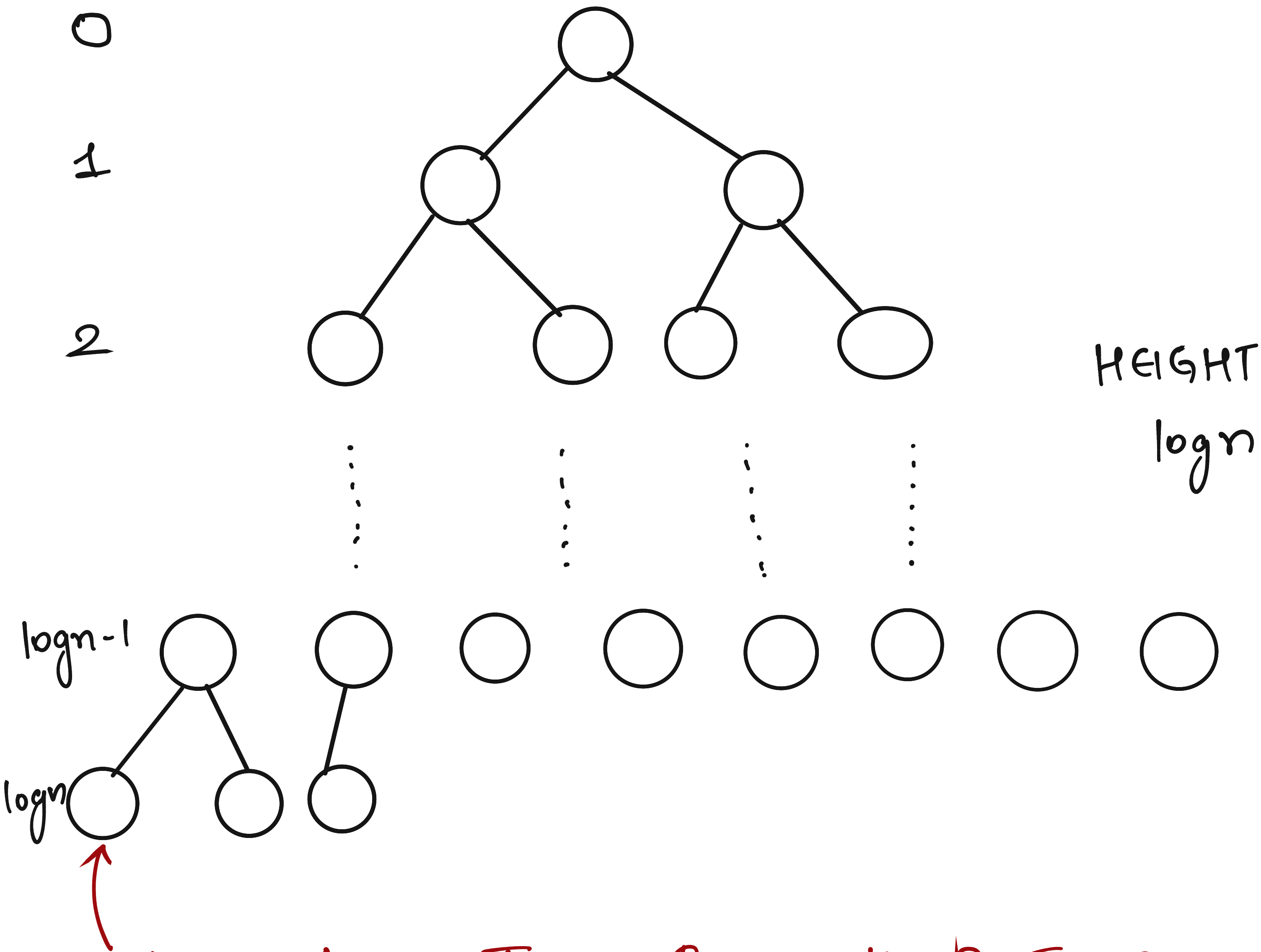
BUILD-HEAP(1)

```
BUILD-HEAP(i)
{   IF (INDEX i HAS A LEFT CHILD)
        BUILD-HEAP(2i)
    IF (INDEX i HAS A RIGHT CHILD)
        BUILD-HEAP(2i+1)
    SHIFT-DOWN(i)
}
```

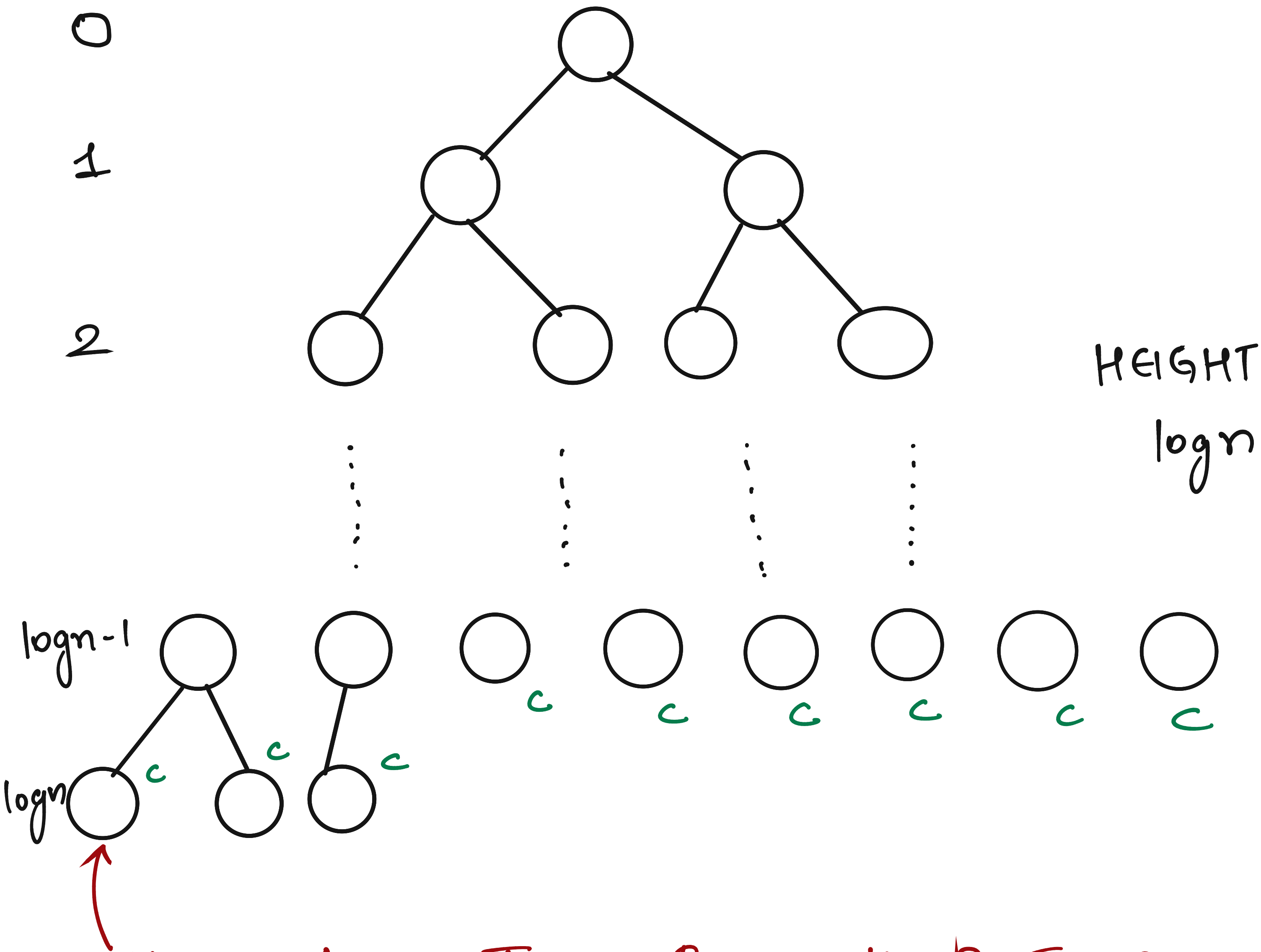
RUNNING TIME OF BUILD-HEAP(1)



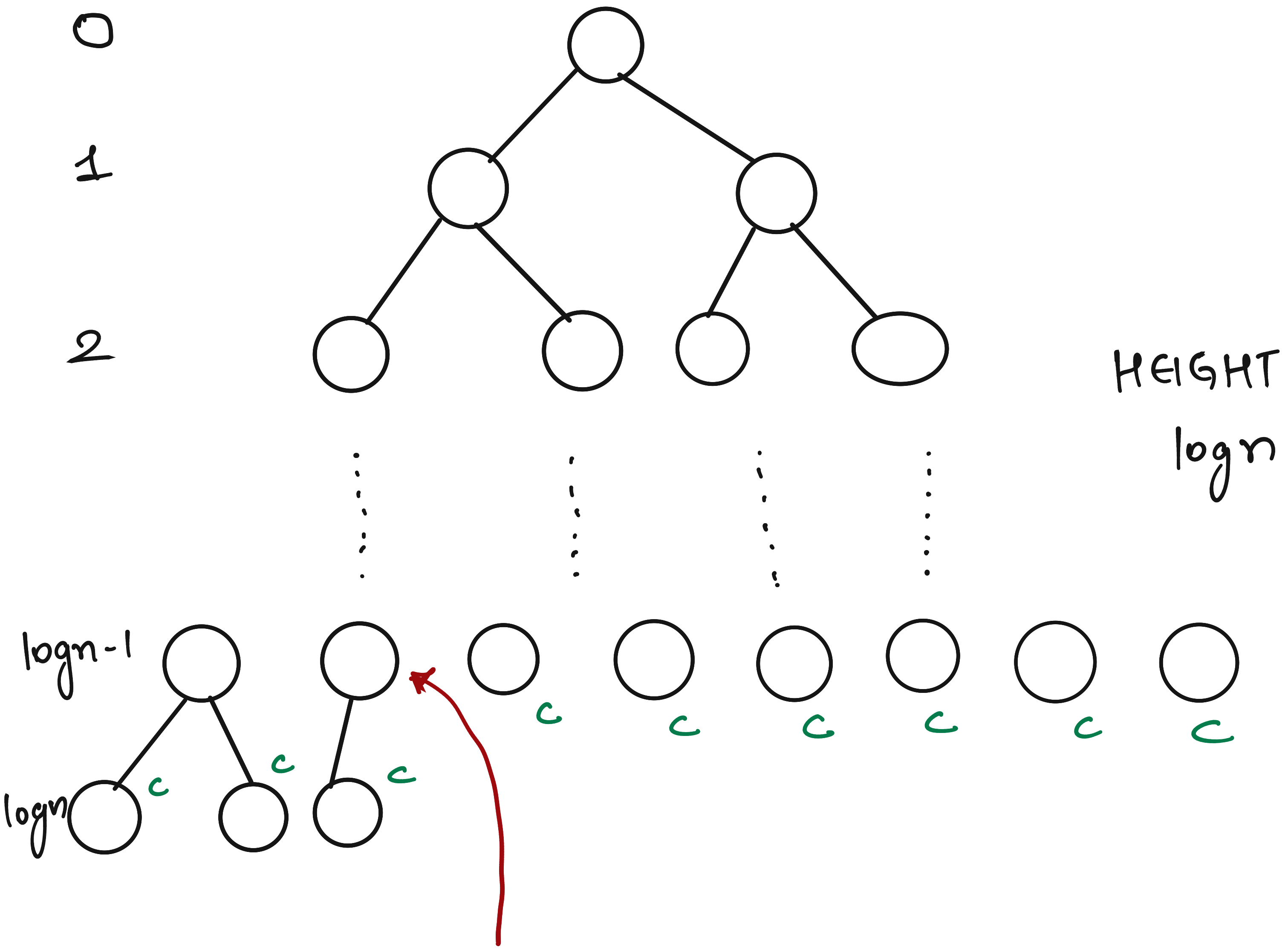
0
1
2
HEIGHT
logn
logn-1
logn



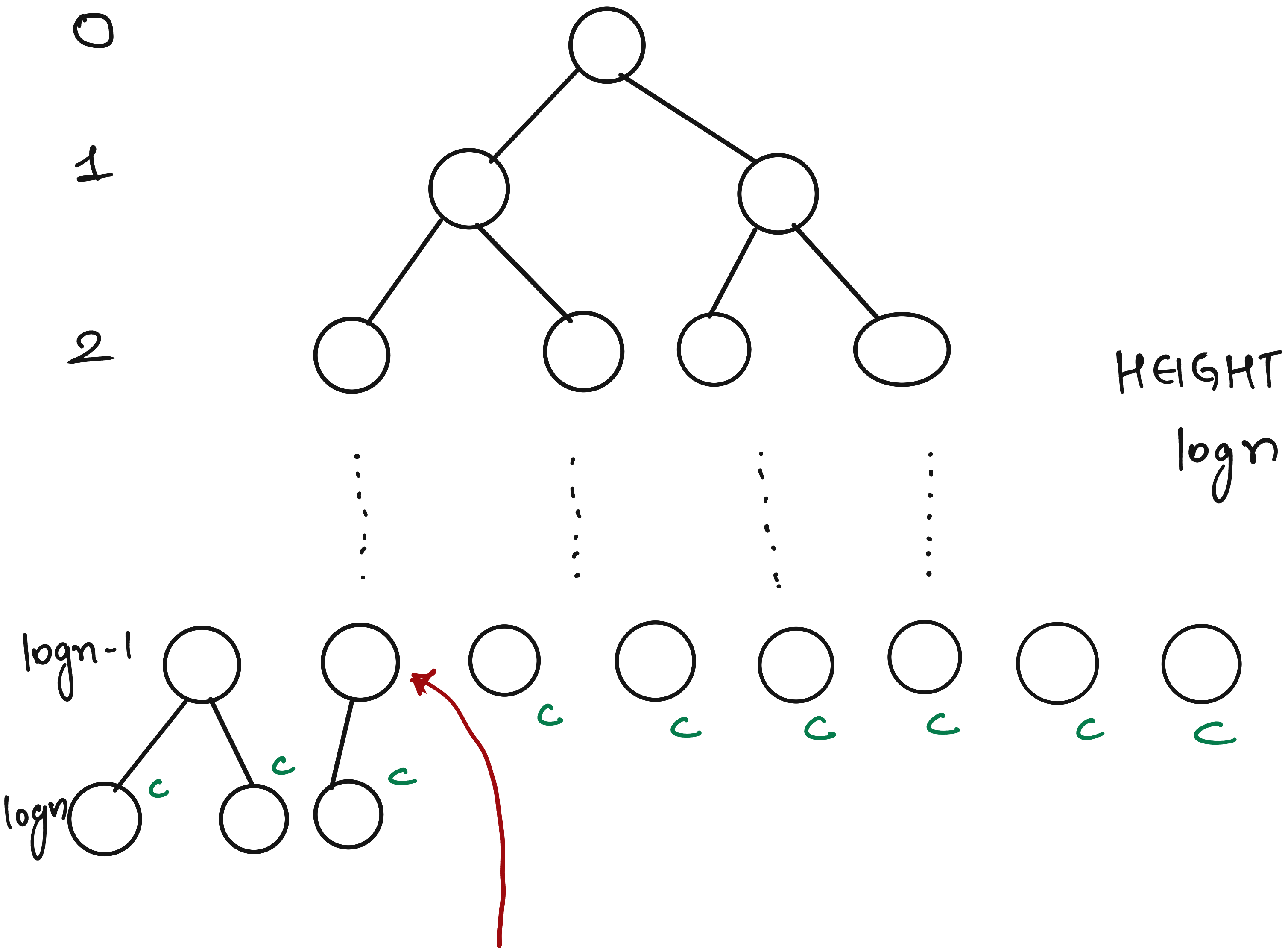
HOW MUCH TIME BUILD HEAP TAKES FOR THE BASE-CASE ?



HOW MUCH TIME BUILD HEAP TAKES FOR THE BASE-CASE ?

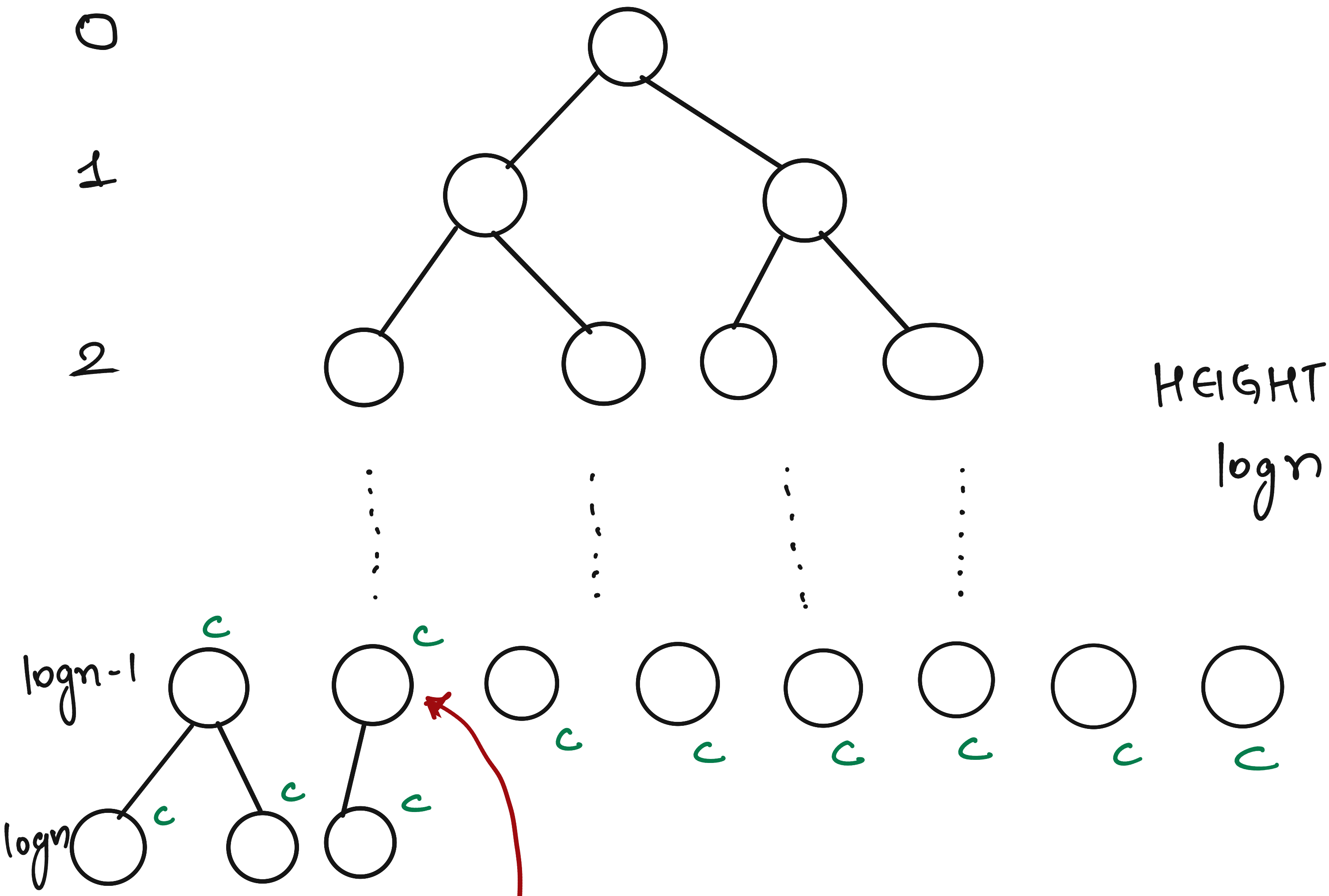


HOW MUCH TIME BUILD HEAP TAKES FOR INTERNAL NODES AT SECOND-LAST LAYER ?



HOW MUCH TIME BUILD HEAP TAKES FOR INTERNAL NODES AT SECOND-LAST LAYER ?

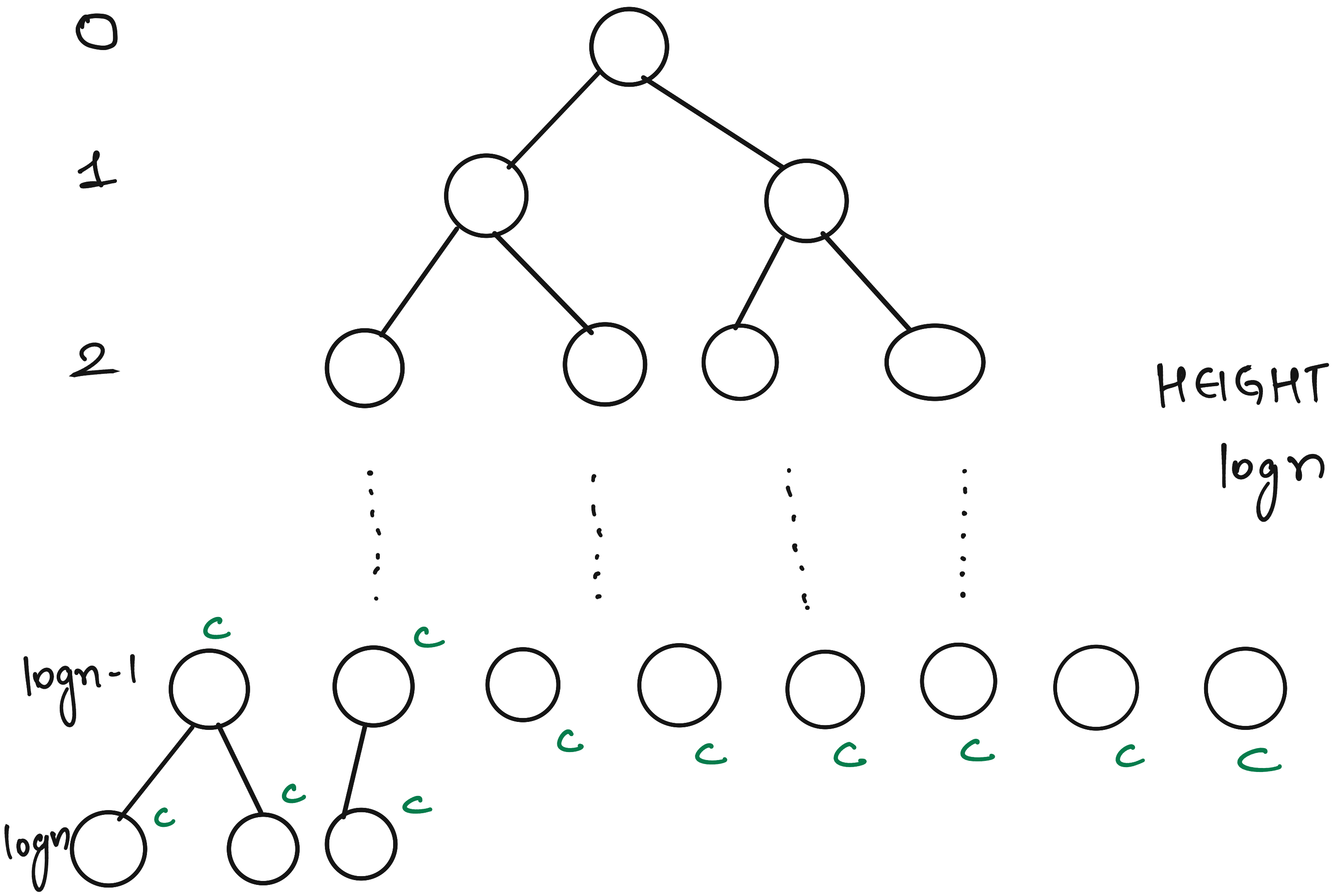
DOMINATED BY THE RUNNING TIME OF SHIFT-DOWN



HOW MUCH TIME BUILD HEAP TAKES FOR INTERNAL NODES AT SECOND-LAST LAYER ?

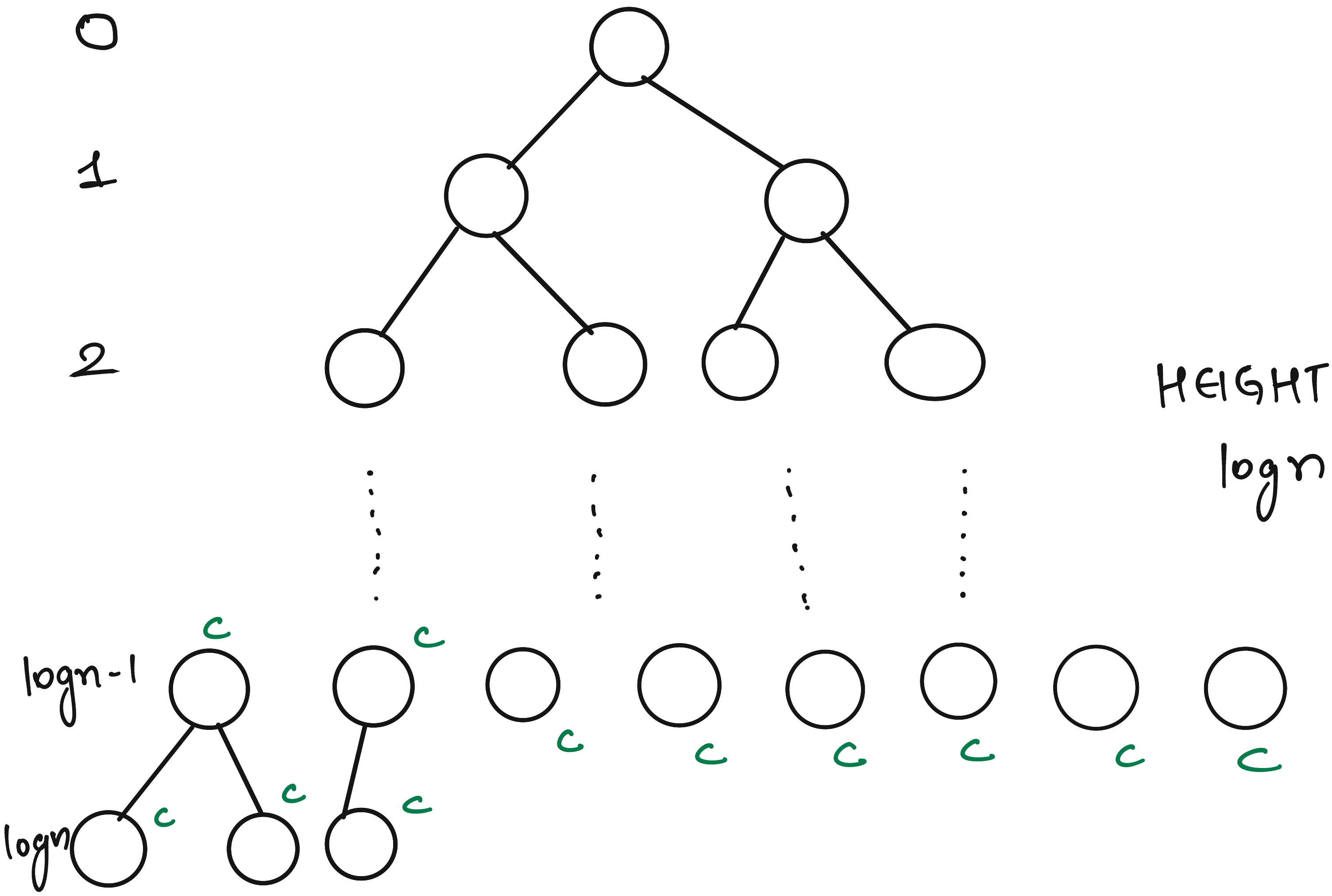
DOMINATED BY THE RUNNING TIME OF SHIFT-DOWN

C



HOW MUCH TIME BUILD HEAP TAKES FOR INTERNAL NODES AT THIRD - LAST LAYER ?

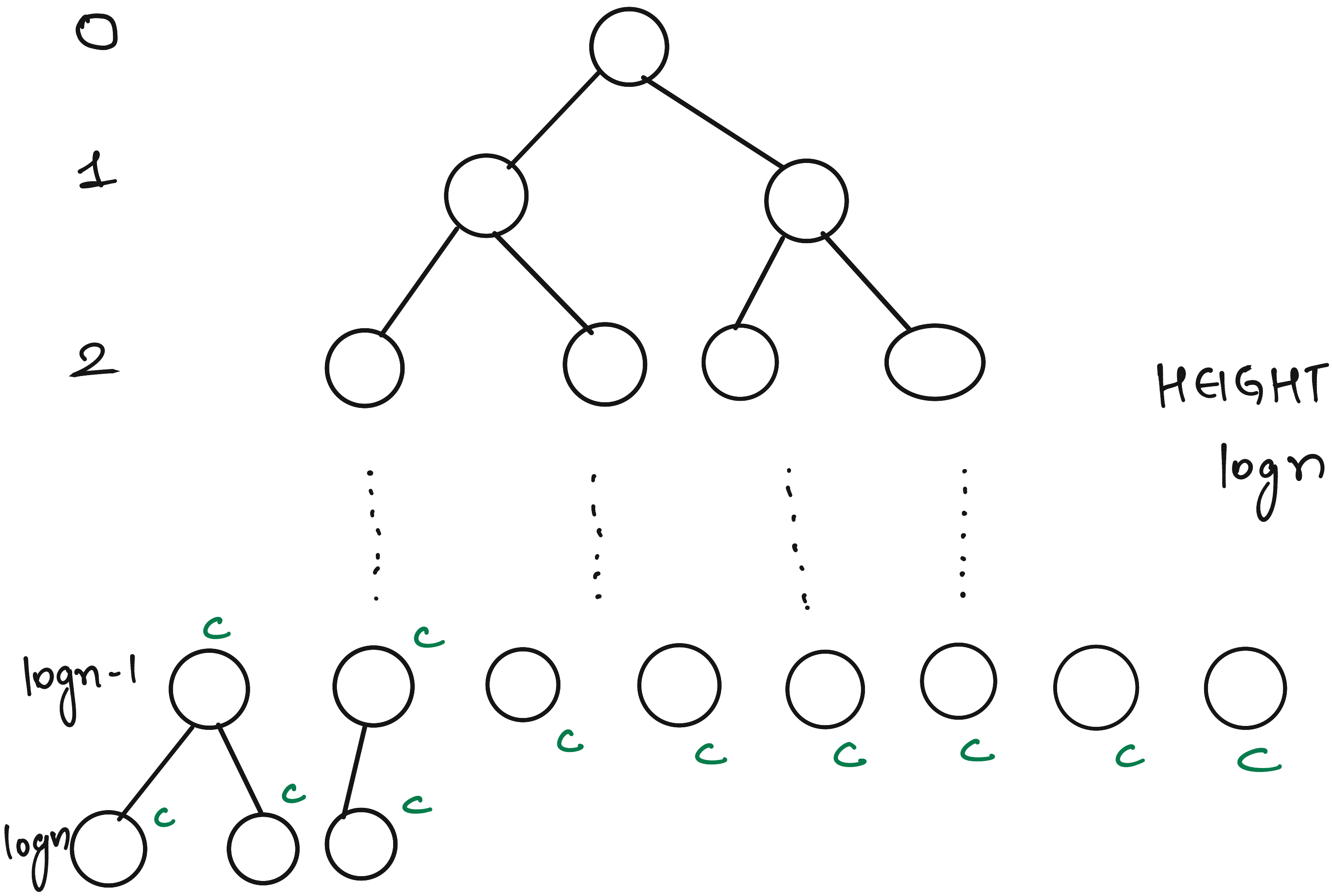
DOMINATED BY THE RUNNING TIME OF SHIFT- DOWN



HOW MUCH TIME BUILD HEAP TAKES FOR INTERNAL NODES AT THIRD - LAST LAYER ?

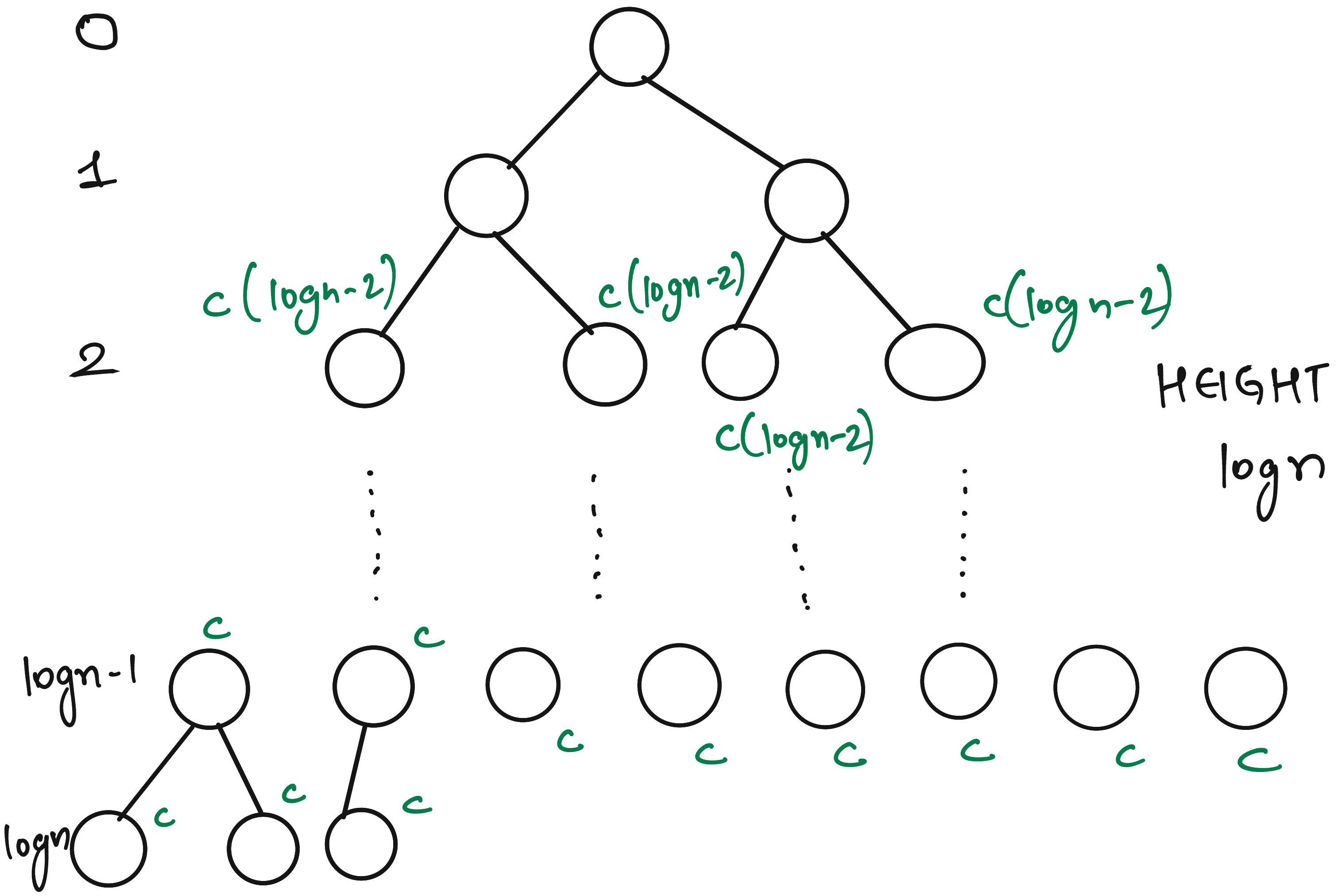
DOMINATED BY THE RUNNING TIME OF SHIFT- DOWN

2c



HOW MUCH TIME BUILD HEAP TAKES FOR INTERNAL NODES AT LAYER 2?

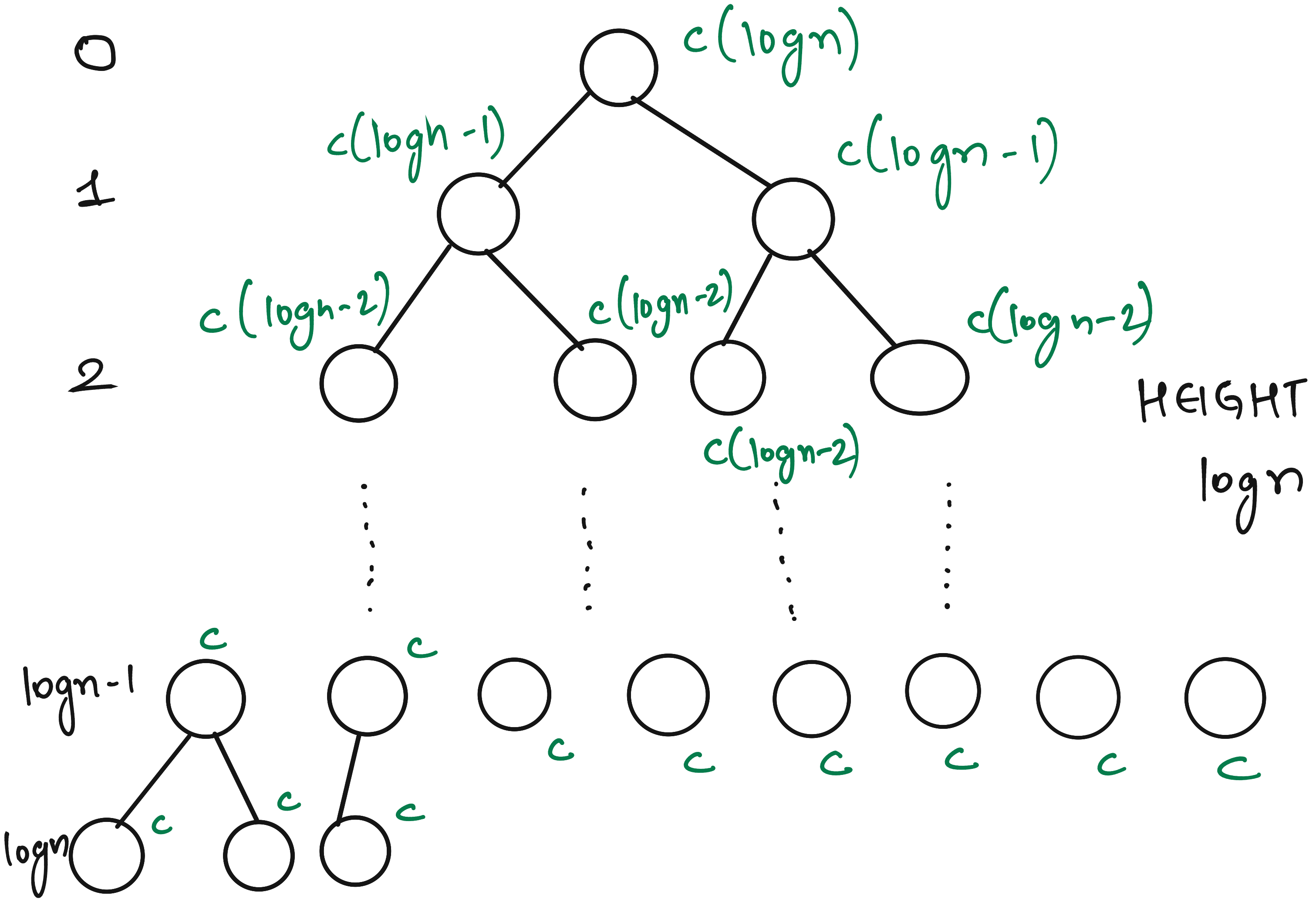
DOMINATED BY THE RUNNING TIME OF SHIFT-DOWN



HOW MUCH TIME BUILD HEAP TAKES FOR INTERNAL NODES AT LAYER 2?

DOMINATED BY THE RUNNING TIME OF SHIFT-DOWN

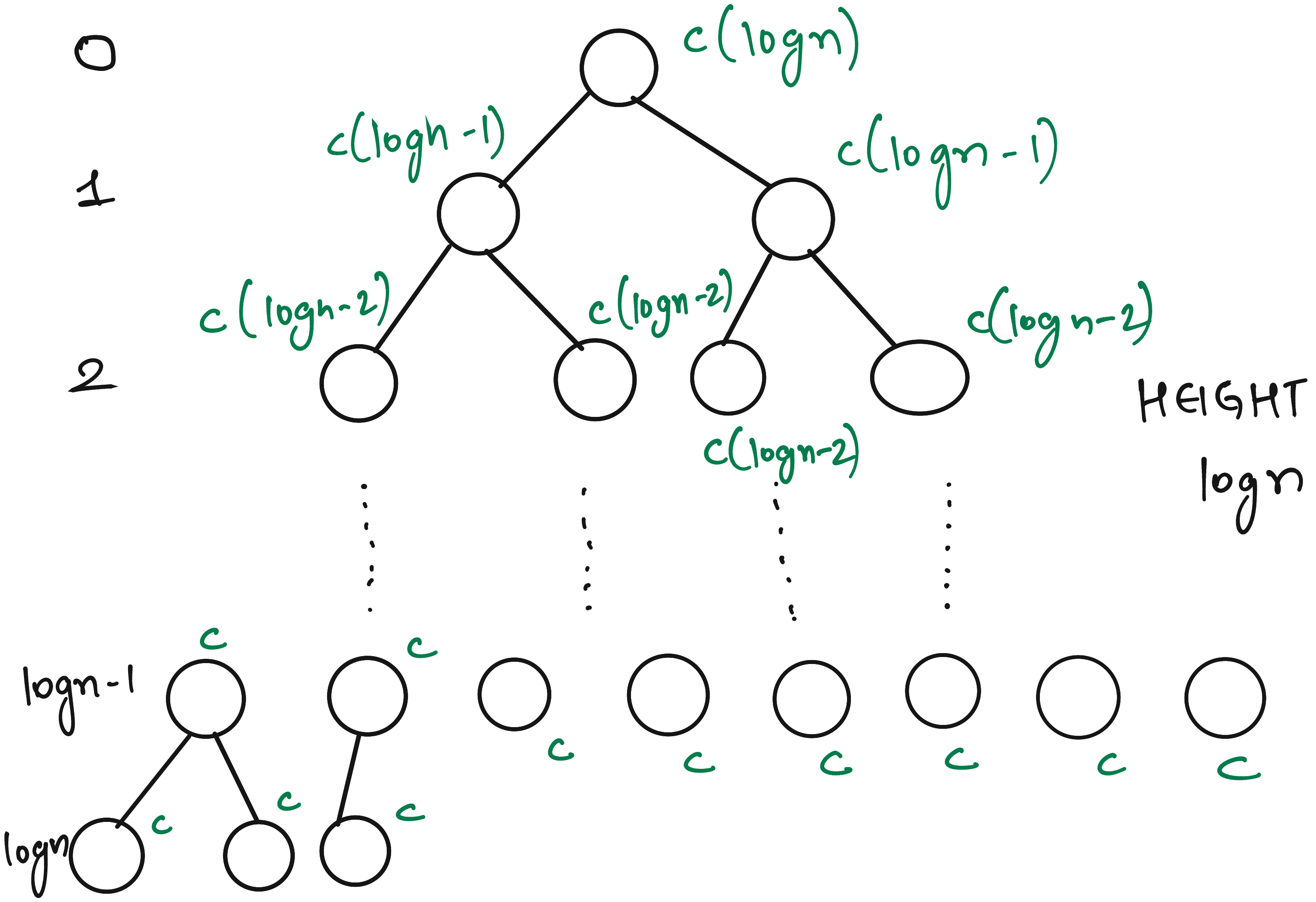
$$c(\log n - 2)$$



HOW MUCH TIME BUILD HEAP TAKES FOR INTERNAL NODES AT LAYER 2?

DOMINATED BY THE RUNNING TIME OF SHIFT- DOWN

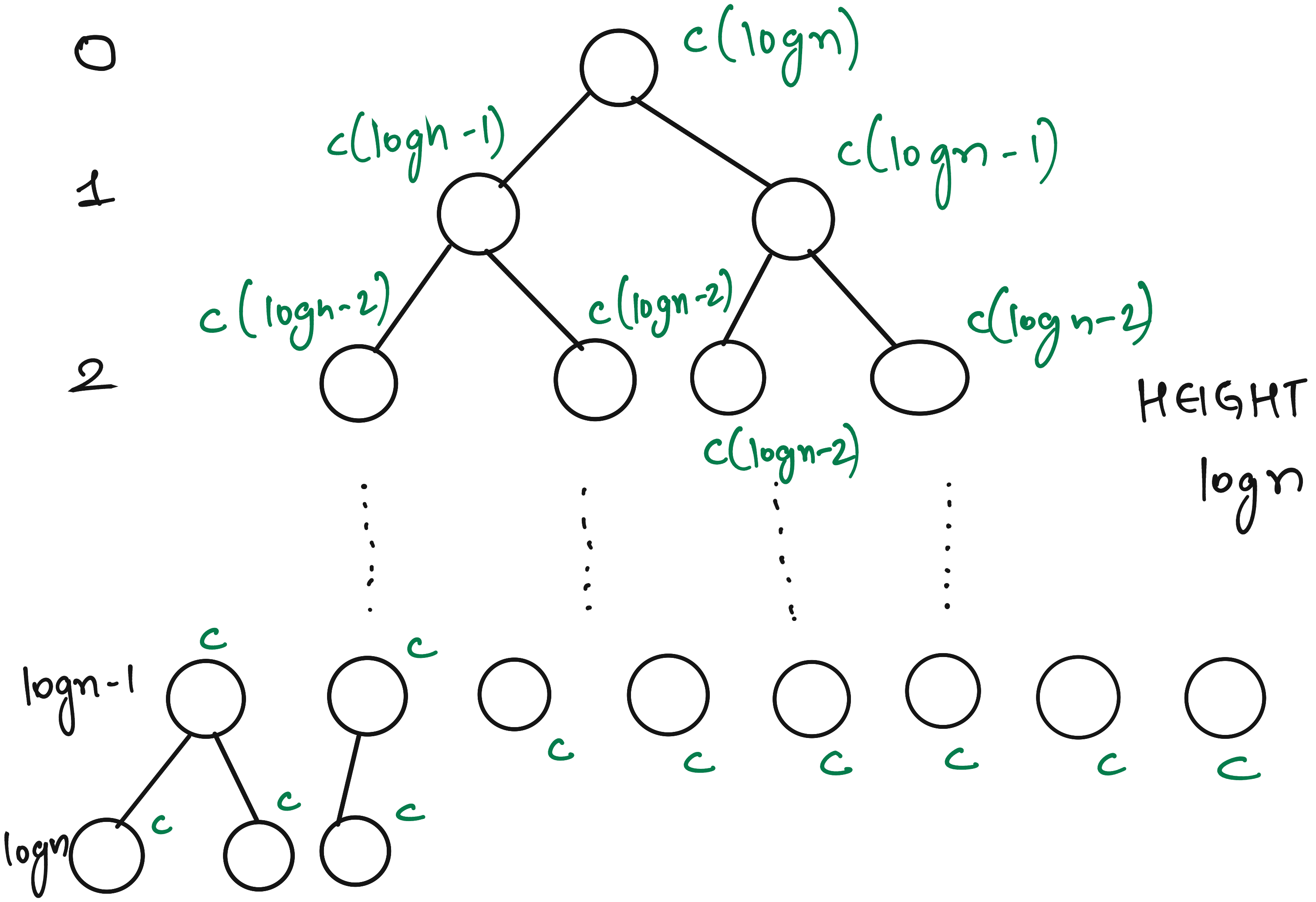
$c(\log n - 2)$



RUNTIME OF BUILD-HEAP

$$= \sum_{i=0}^{\log n - 1} 2^{i} \left(\qquad \right) + x \cdot c$$

LEAF NODES



RUNTIME OF BUILD-HEAP

$$= \sum_{i=0}^{\log n - 1} 2^i \left(\qquad \right) + \underset{\substack{\uparrow \\ \text{LEAF NODES}}}{x} \cdot c$$

$$\leq \sum_{i=0}^{\log n - 1} 2^i c(\log n - i) + 2^{\log n} \cdot c$$

$$= S + nc$$

$$S = \sum_{i=0}^{\log n - 1} 2^i c(\log n - i)$$

$$S = \sum_{i=0}^{\log n - 1} 2^i c(\log n - i)$$

$$S = 2^0 c(\log n) + 2^1 c(\log n - 1) + \cdots + 2^{\log n - 1} c(1)$$

$$2S = \qquad 2^1 c \log n + \cdots + 2^{\log n - 1} c(2) \pm 2^{\log n} c(1)$$

$$-S = 2^0 c \log n - 2^1 c - 2^2 \cdot c - \cdots - 2^{\log n - 1} \cdot c - 2^{\log n} \cdot c$$

$$S = \sum_{i=0}^{\log n - 1} 2^i c(\log n - i)$$

$$S = 2^0 c(\log n) + 2^1 c(\log n - 1) + \dots + 2^{\log n - 1} c(1)$$

$$2S = \qquad 2^1 c \log n + \dots + 2^{\log n - 1} c(2) + 2^{\log n} c(1)$$

$$- \qquad - \qquad - \qquad - \qquad \pm$$

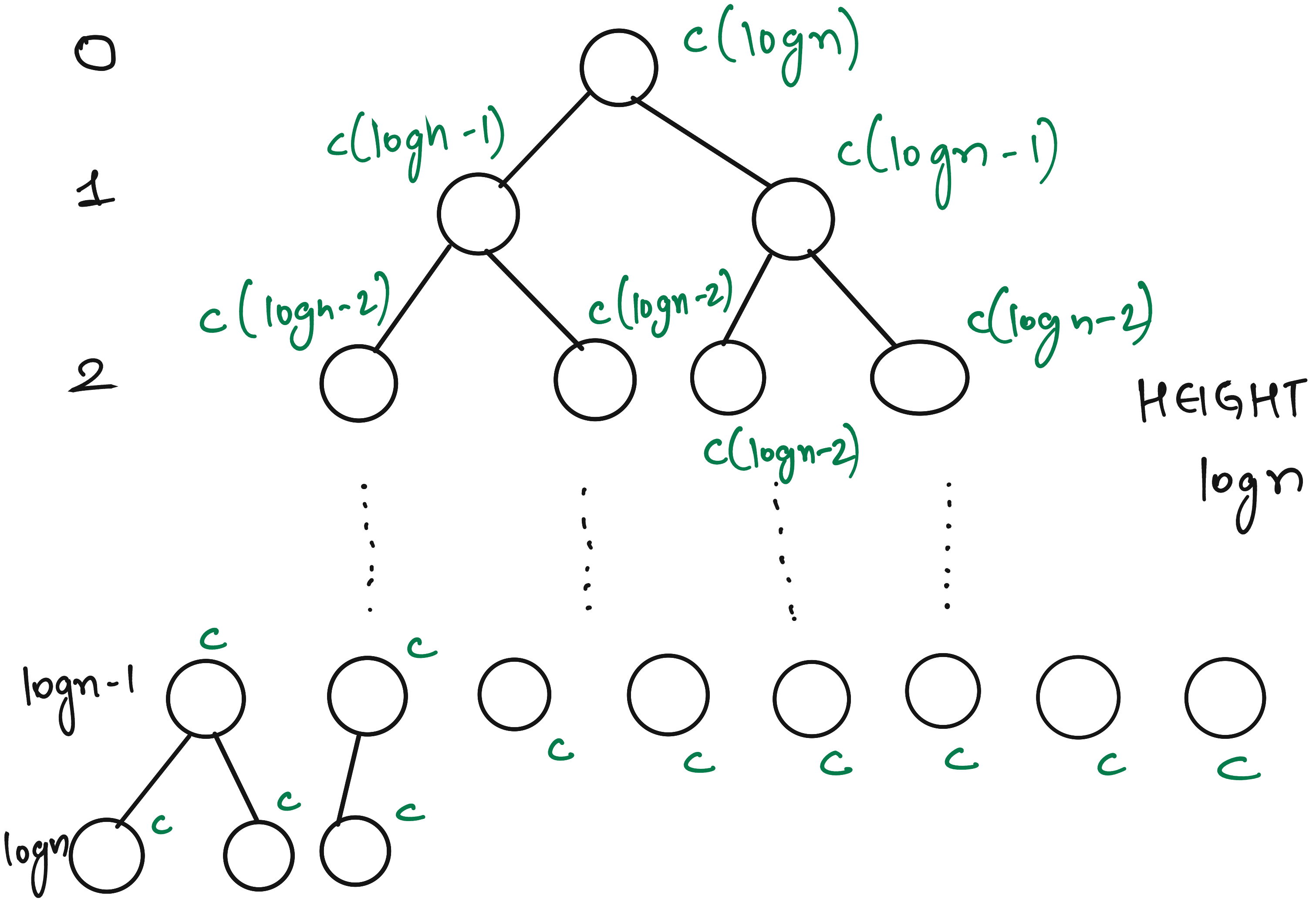
$$-S = 2^0 c \log n - 2^1 c - 2^2 \cdot c - \dots - 2^{\log n - 1} \cdot c - 2^{\log n} \cdot c$$

$$\Rightarrow S = \left(\sum_{i=1}^{\log n} c \cdot 2^i \right) - 2^0 c \log n$$

$$= \frac{c \cdot (2^{\log n + 1} - 1)}{2 - 1} - 2^0 c \log n$$

$$= c \cdot 2n - c - 2^0 \cdot c \log n$$

$$\Rightarrow S \leq 2nc$$



RUNTIME OF BUILD-HEAP

$$= \sum_{i=0}^{\log n-1} 2^{i} \left(\qquad \right) + x \cdot c$$

$\uparrow$

LEAF NODES

$$\leq \sum_{i=0}^{\log n-1} 2^{i} c(\log n - i) + 2^{\log n} \cdot c$$

$$= S + nc$$

$$\leq 2nc + nc$$

$$= 3nc$$

$$= O(n)$$

TOTAL RUNNING TIME OF HEAPSORT

= BUILD-HEAP +

RUNTIME OF HEAPSORT (ASSUMING AN IMPLICIT HEAP)

TOTAL RUNNING TIME OF HEAPSORT

= BUILD-HEAP +

RUNTIME OF HEAPSORT (ASSUMING AN IMPLICIT HEAP)

$= O(n) + O(n \log n)$

$= O(n \log n).$