CSE 548 / AMS 542: Analysis of Algorithms

Prerequisites Review 5
(Heaps and Heapsort)

Rezaul Chowdhury
Department of Computer Science
SUNY Stony Brook
Fall 2023

Selection Sort

Input: An array $A[\,1 : n\,]$ of n numbers.

Output: Elements of $A[\,1 : n\,]$ rearranged in non-decreasing order of value.

SELECTION-SORT (*A*)

for $j = A.length$ **downto** 2
    // find the index of an entry with the largest value in $A[1..j]$
    $max = 1$
    for $i = 2$ **to** j
        if $A[i] > A[max]$
            $max = i$
    // swap $A[j]$ and $A[max]$
    $A[j] \leftrightarrow A[max]$

This way of finding the index of an entry with the largest value in a subarray of length m takes $\Theta(m)$ time, which is bad!

Selection Sort

```
SELECTION-SORT ( A )

for j = A.length downto 2
    // find the index of an entry with the largest value in A[1..j]
    max = 1
    for i = 2 to j
        if A[i] > A[max]
            max = i
    // swap A[j] and A[max]
    A[j] ↔ A[max]
```

This way of finding the index of an entry with the largest value in a subarray of length m takes $\Theta(m)$ time, which is bad!

Let $L(m)$ be the time needed to find the index of an entry with the largest value in a subarray of length m.

Then running time of SELECTION-SORT, $T(n) = \sum_{2 \le j \le n} L(j)$

$$= \sum_{2 \le j \le n} \Theta(j) = \Theta\left(\sum_{2 \le j \le n} j\right) = \Theta(n^2)$$

Selection Sort

```
SELECTION-SORT ( A )

for j = A.length downto 2
    // find the index of an entry with the largest value in A[1..j]
    max = 1
    for i = 2 to j
        if A[i] > A[max]
            max = i
    // swap A[j] and A[max]
    A[j] ↔ A[max]
```

This way of finding the index of an entry with the largest value in a subarray of length m takes $\Theta(m)$ time, which is bad!

If we can decrease $L(m)$, then the running time of SELECTION-SORT will also decrease. For example, if we have $L(m) = O(\log m)$,

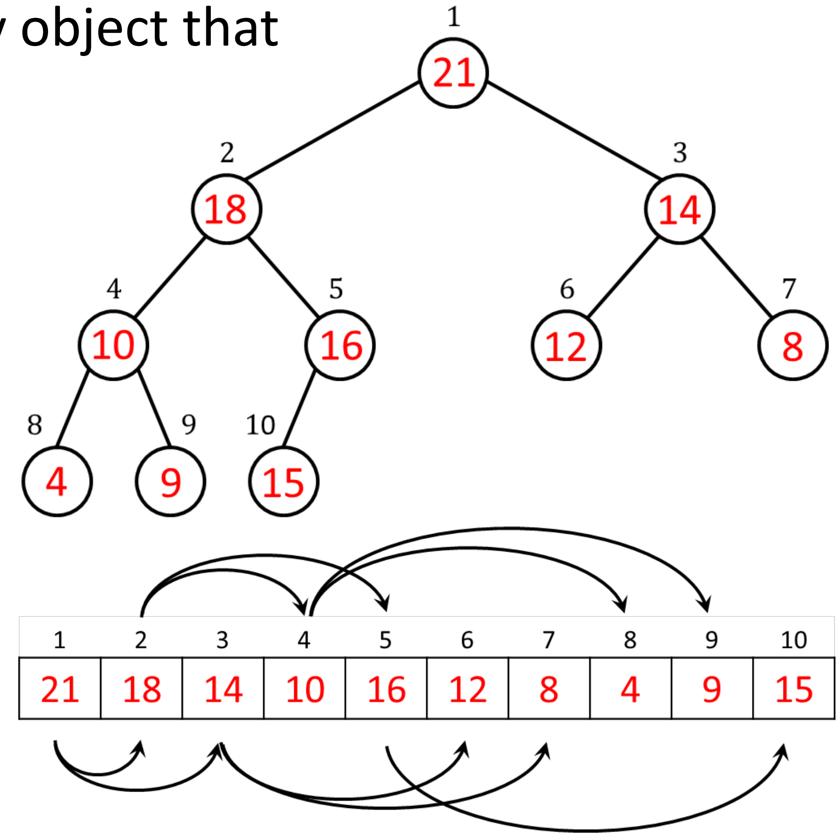
running time of SELECTION-SORT will be, $T(n) = \sum_{2 \le j \le n} L(j)$

$$= \sum_{2 \le j \le n} O(\log(j)) = O\left(\sum_{2 \le j \le n} \log(j)\right) = O(n \log n)$$

How can you decrease $L(m)$ to $O(\log m)$?

Heap (Binary Heap)

A (*binary*) *heap* data structure is an array object that can be viewed as a complete binary tree.



Each node of the tree corresponds to an element of the array.

The tree is completely filled on all levels except possibly the last, which is filled from the left up to a point.

An array A that represents a heap is an object with two attributes:

$A.length$, which gives the number of elements in the array.

$A.heapsize$, which represents how many elements in the heap are stored within array A.

Though $A[1..A.length]$ may contain numbers, only $A[1..A.heapsize]$ contain valid elements of the heap, where $1 \leq A.heapsize \leq A.length$.

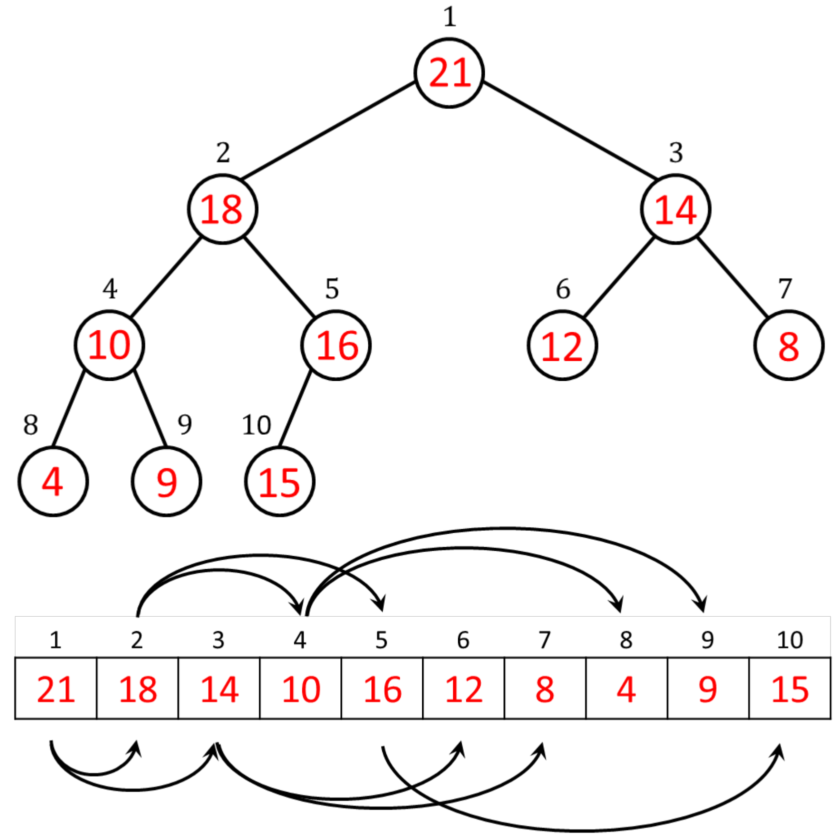
Parent and Children

The root of the tree is $A[1]$.

A node has 0, 1 or 2 children.

A node with no child is called a *leaf*.

A node with at least one child is called an *internal node*.



Given the index i of a node, we can easily compute the indices of its *parent*, *left child* and *right child*.

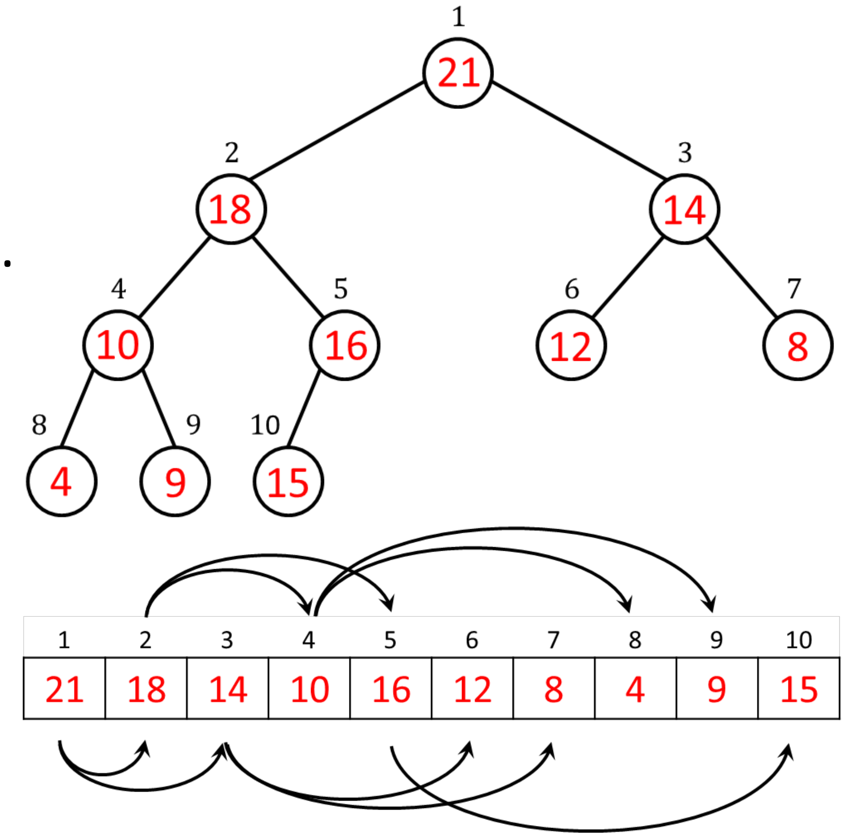
PARENT (*i*)
1. ***return*** $\left\lfloor \frac{i}{2} \right\rfloor$

LEFT (*i*)
1. ***return*** $2i$

RIGHT (*i*)
1. ***return*** $2i + 1$

Max-Heap and Min-Heap

The root of the tree is $A[1]$.



Max-heap. Each node $i > 1$ satisfies the *max-heap property*: $A[\textsc{Parent}(i)] \geq A[i]$.

Hence, the largest element in a max-heap is stored at the root.

We will use max-heaps in the *heapsort* algorithm which can be viewed as improved selection sort.

Min-heap. Each node $i > 1$ satisfies the *min-heap property*: $A[\textsc{Parent}(i)] \leq A[i]$.

Hence, the smallest element in a min-heap is stored at the root.

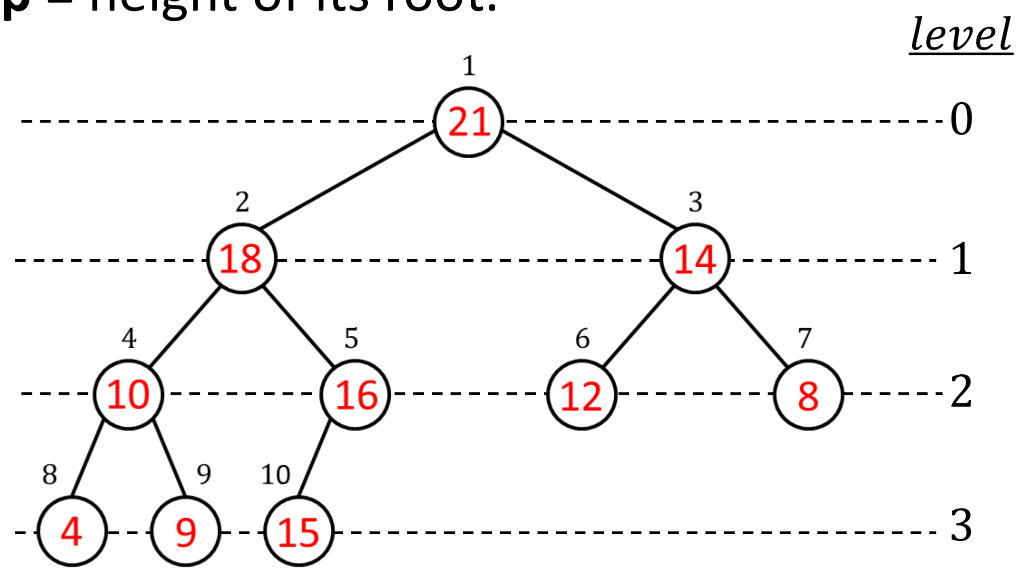
Min-heaps commonly implement priority queues which have many applications, e.g., in shortest paths computation.

Height and Levels of a Heap

The root of the tree is $A[1]$.

Height of a node = Number of edges on the longest simple downward path from that node to a leaf.

Height of a heap = height of its root.



Levels:

Level of the root, $\text{LEVEL}(1) = 0$

Level of node $i > 1$, $\text{LEVEL}(i) = \text{LEVEL}\big(\text{PARENT}(i)\big) + 1$

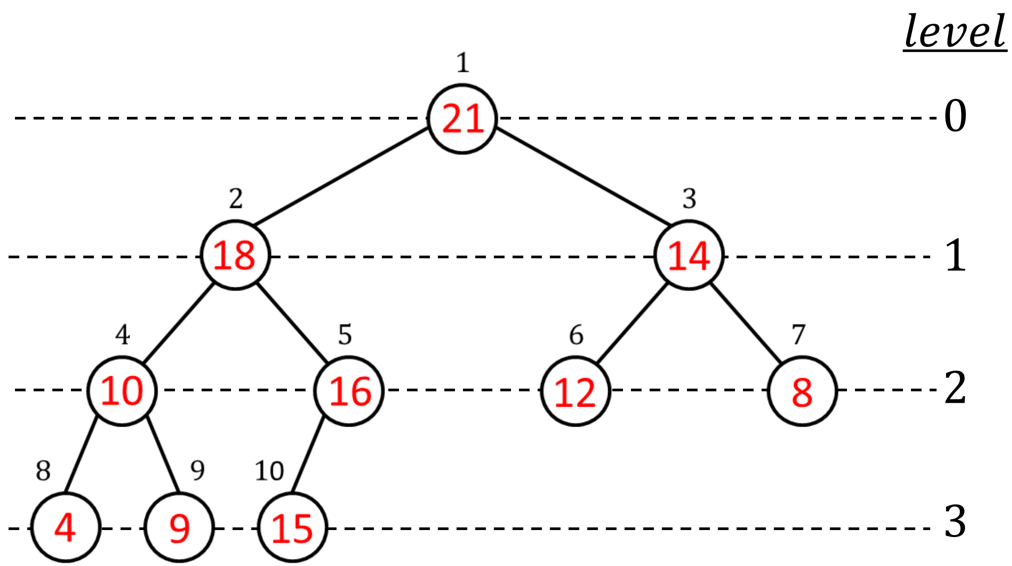
A heap of height h has exactly $h + 1$ levels, numbered from 0 to h.

Height of an n-node Binary Heap

Let h be the height of a heap containing $n > 0$ elements.

So, the heap will have exactly $h + 1$ levels.

Let n_l be the number of nodes at level l, where $0 \leq l \leq h$.



Clearly, $n_l = 2^l$ for $0 \leq l \leq h - 1$,
and $1 \leq n_l \leq 2^l$ for $l = h$.

Also $n = n_0 + n_1 + \cdots + n_h = \sum_{l=0}^{h} n_l$.

Height of an n-node Binary Heap

We have, $n = \sum_{l=0}^{h} n_l = n_h + \sum_{l=0}^{h-1} n_l = n_h + \sum_{l=0}^{h-1} 2^l = n_h + (2^h - 1)$.

But $1 \le n_h \le 2^h$

$\Rightarrow 1 + (2^h - 1) \le n_h + (2^h - 1) \le 2^h + (2^h - 1)$

$\Rightarrow 2^h \le n \le 2^{h+1} - 1$

$\Rightarrow 2^h \le n < 2^{h+1}$

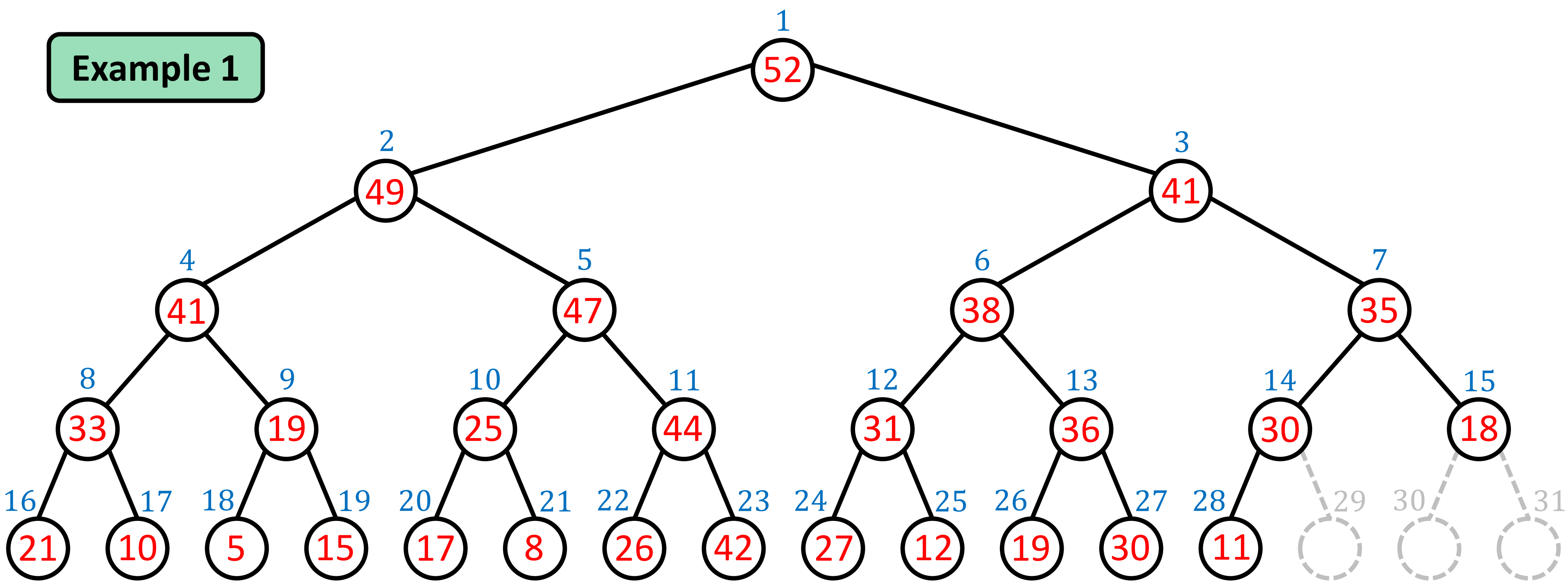
$\Rightarrow \log_2 n - 1 < h \le \log_2 n$

Since h is an integer, and the only integer $> \log_2 n - 1$ and $\le \log_2 n$ is $\lfloor \log_2 n \rfloor$, we have $h = \lfloor \log_2 n \rfloor$.

Maintaining Heap Property

This is a valid max-heap:

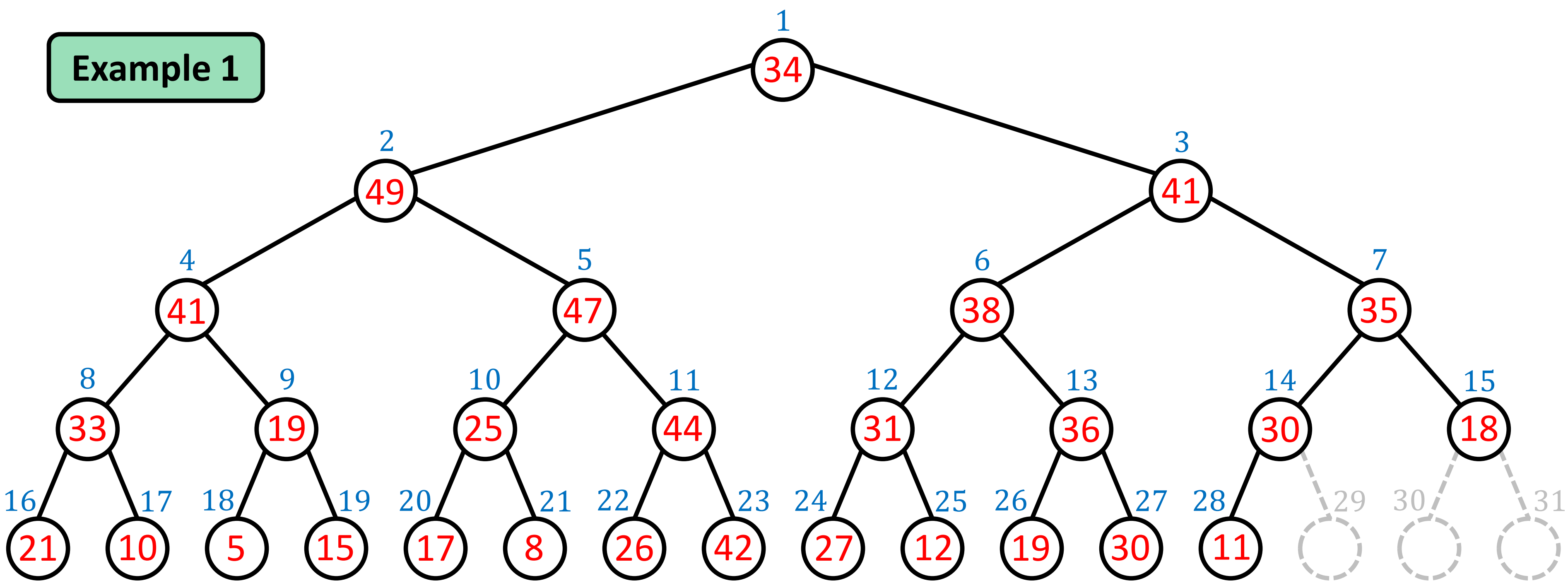
Example 1



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	52	49	41	41	47	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11			

Maintaining Heap Property

But this is not:

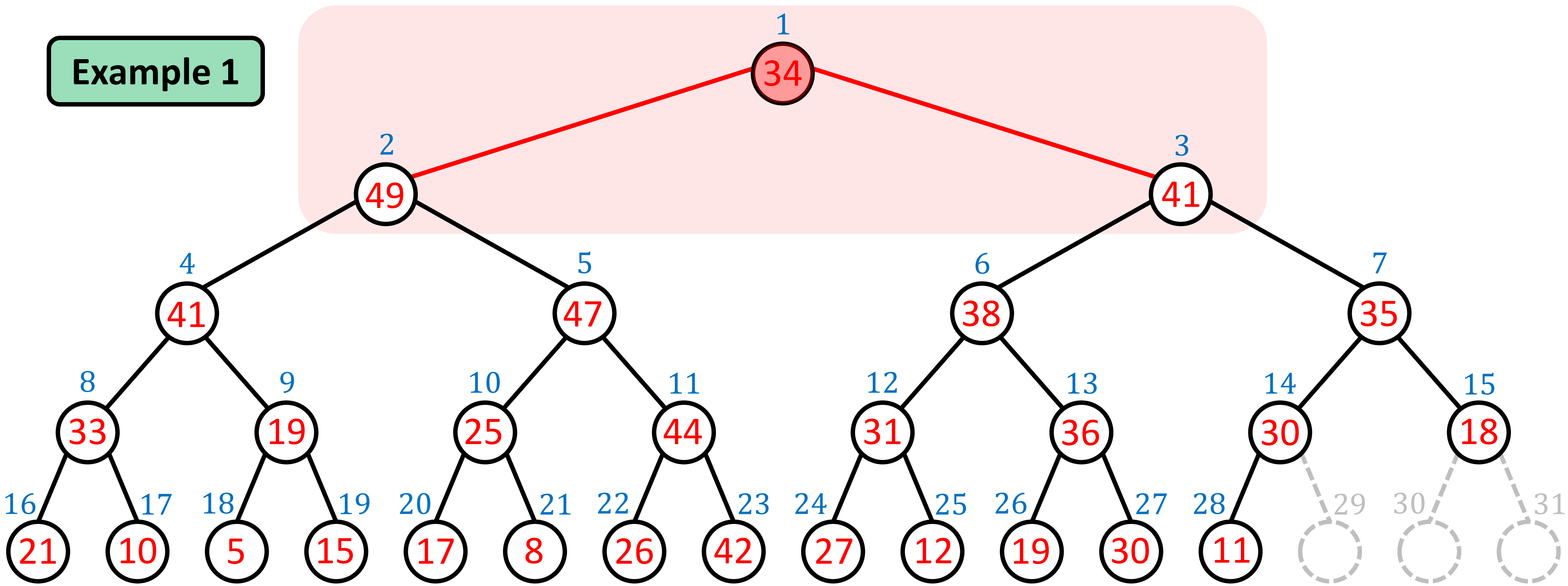


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	34	49	41	41	47	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11			

Maintaining Heap Property

The max-heap property is violated at $A[1]$ though the subtrees rooted at $A[2]$ and $A[3]$ are both valid max-heaps.

Example 1



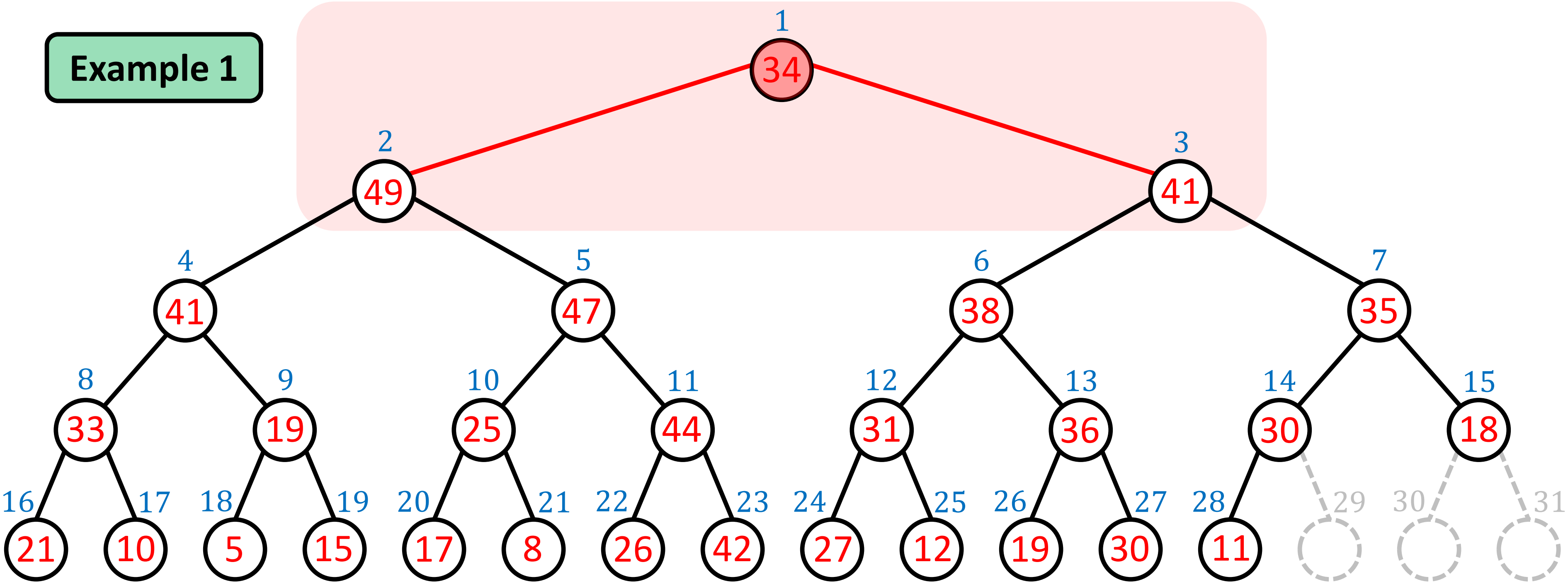
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	34	49	41	41	47	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11			

Maintaining Heap Property

To fix the max-heap property at $A[1]$:

- Find the largest among $A[1]$, $A[2]$ and $A[3]$, which is $A[2]$.
- Swap $A[1]$ and $A[2]$.

Example 1



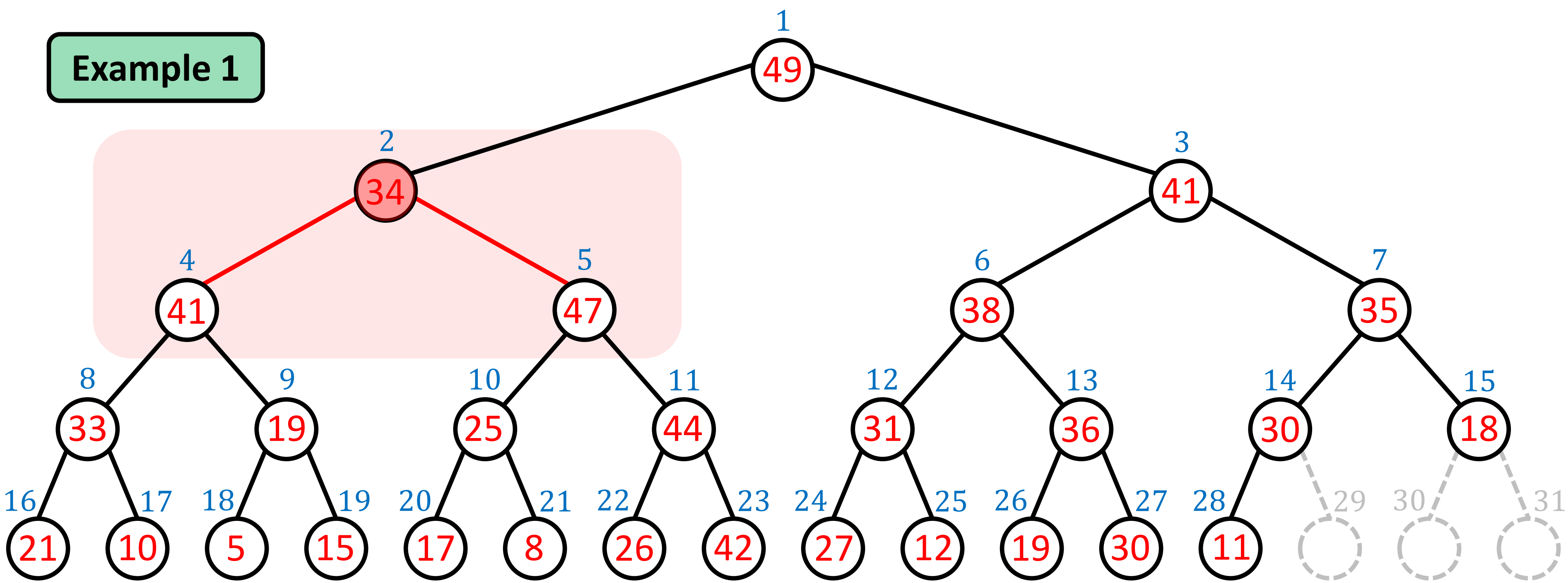
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	34	49	41	41	47	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11			

Maintaining Heap Property

To fix the max-heap property at $A[1]$:

- Find the largest among $A[1]$, $A[2]$ and $A[3]$, which is $A[2]$.
- Swap $A[1]$ and $A[2]$.

Example 1

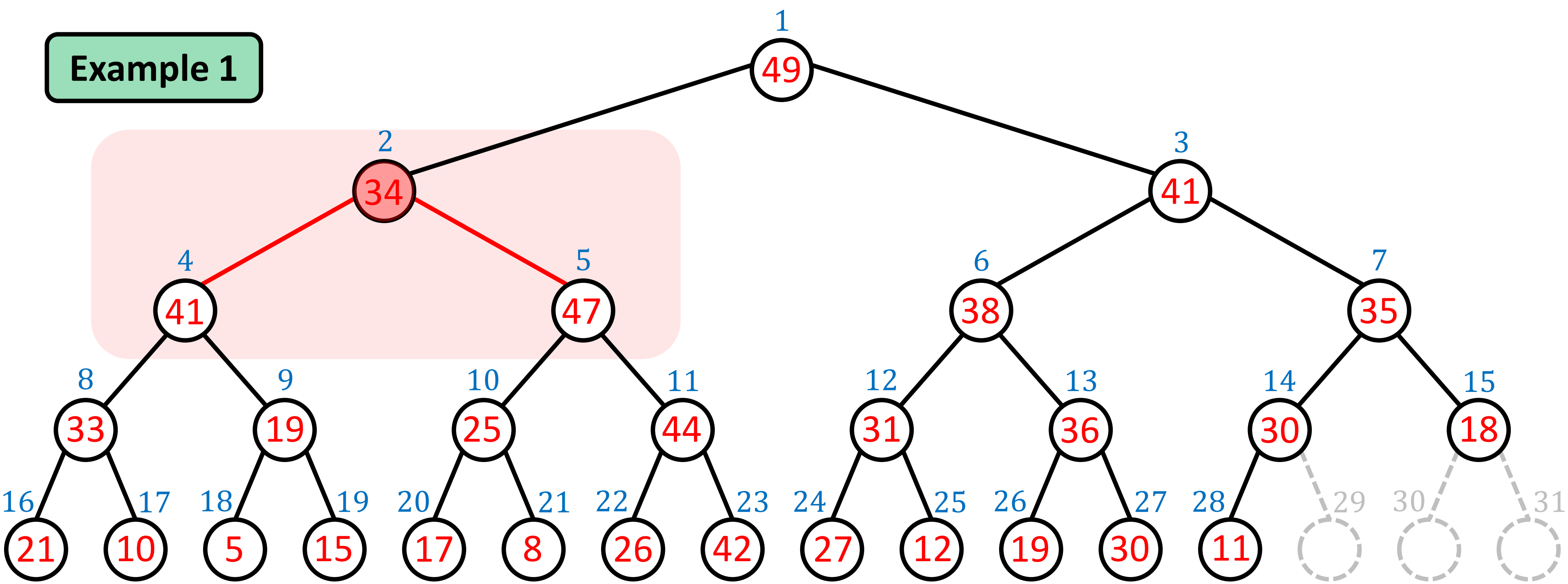


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	49	34	41	41	47	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11			

Maintaining Heap Property

The max-heap property is violated at $A[2]$ though the subtrees rooted at $A[4]$ and $A[5]$ are both valid max-heaps.

Example 1



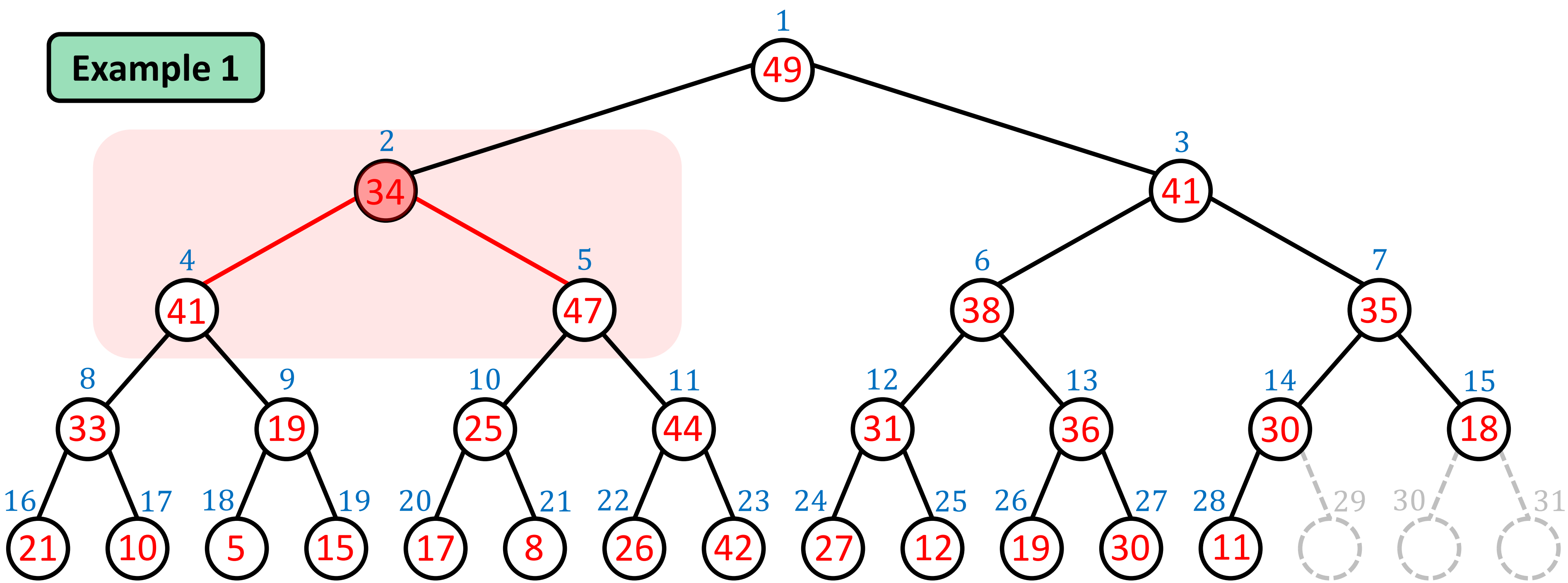
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	49	34	41	41	47	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11			

Maintaining Heap Property

To fix the max-heap property at $A[2]$:

- Find the largest among $A[2]$, $A[4]$ and $A[5]$, which is $A[5]$.
- Swap $A[2]$ and $A[5]$.

Example 1



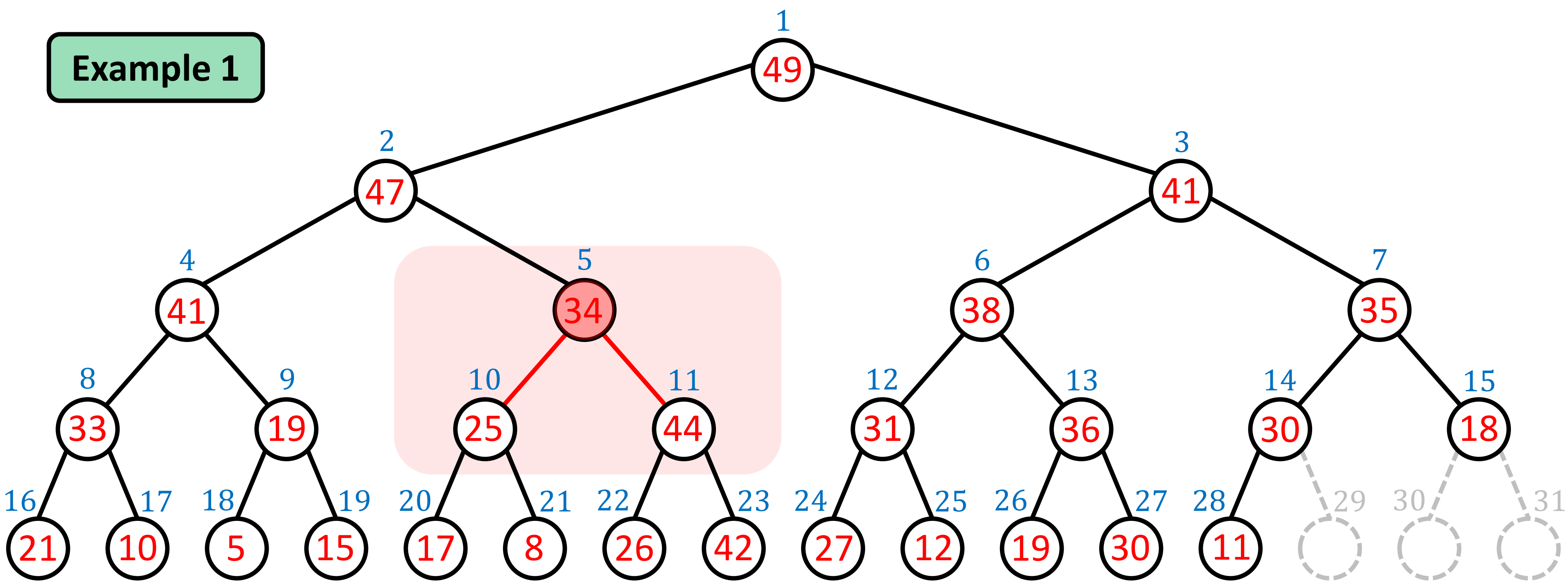
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	49	34	41	41	47	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11			

Maintaining Heap Property

To fix the max-heap property at $A[2]$:

- Find the largest among $A[2]$, $A[4]$ and $A[5]$, which is $A[5]$.
- Swap $A[2]$ and $A[5]$.

Example 1

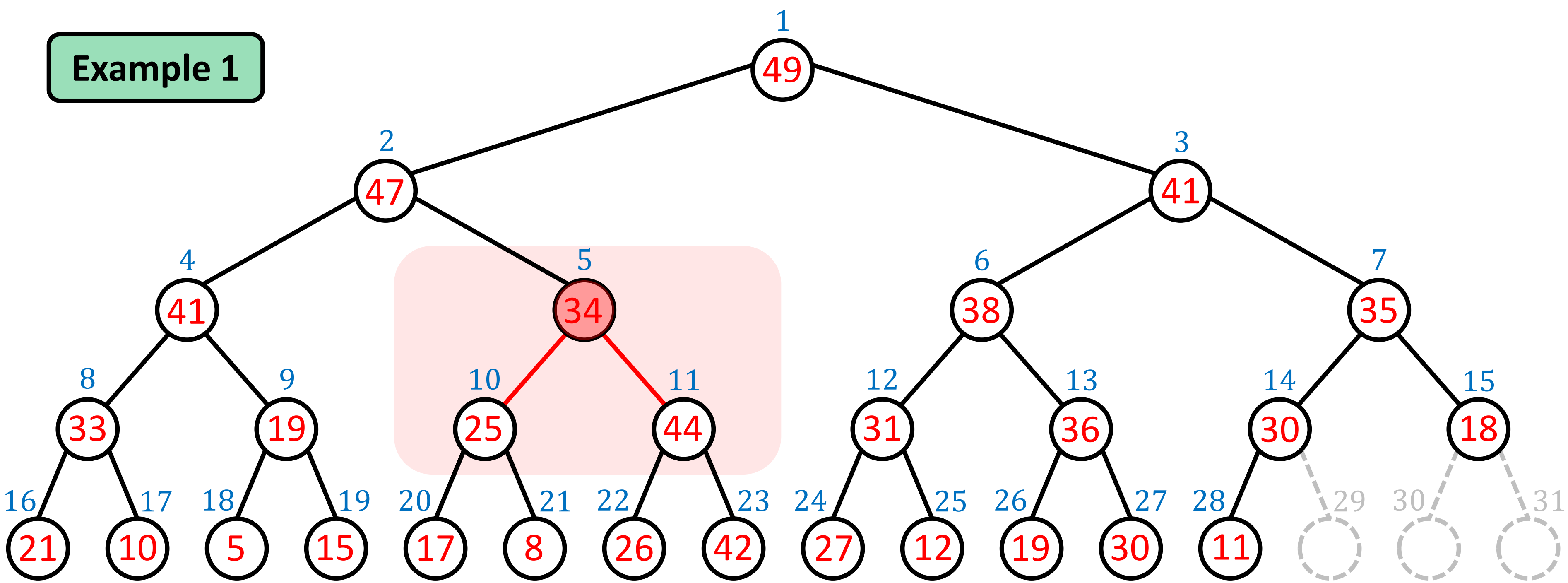


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	49	47	41	41	34	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11			

Maintaining Heap Property

The max-heap property is violated at $A[5]$ though the subtrees rooted at $A[10]$ and $A[11]$ are both valid max-heaps.

Example 1



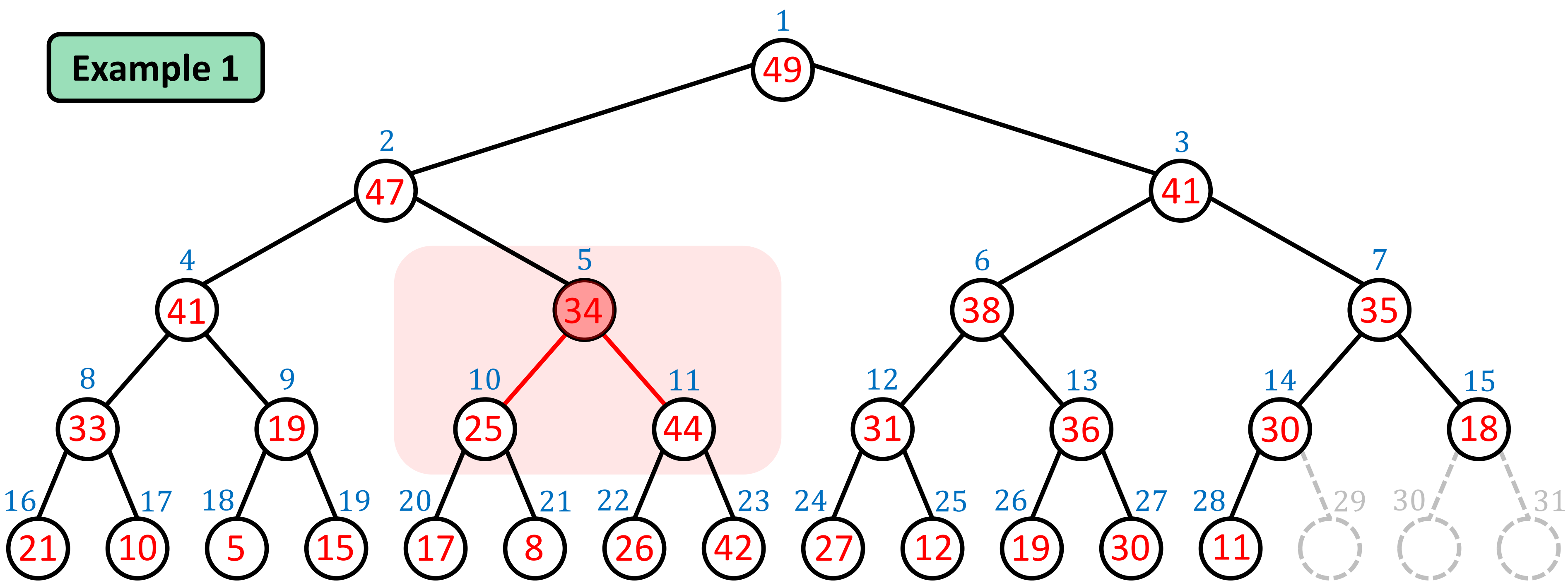
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	49	47	41	41	34	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11			

Maintaining Heap Property

To fix the max-heap property at $A[5]$:

- Find the largest among $A[5]$, $A[10]$ and $A[11]$, which is $A[11]$.
- Swap $A[5]$ and $A[11]$.

Example 1



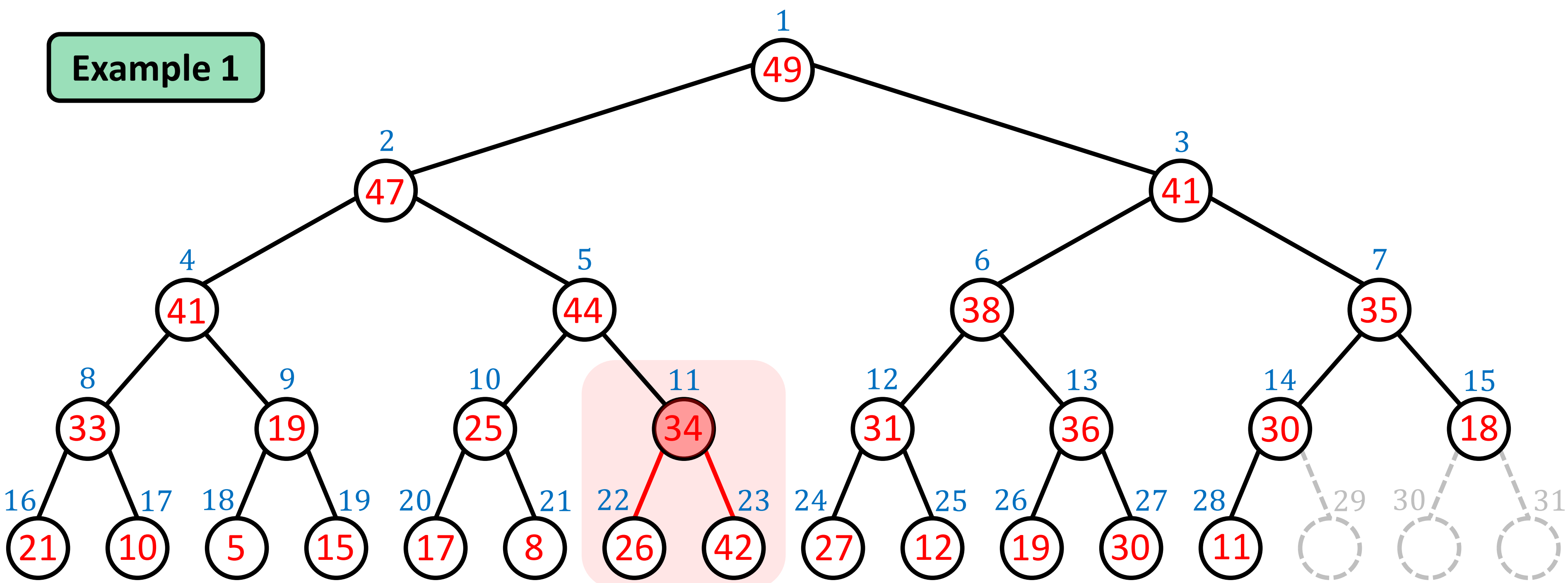
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	49	47	41	41	34	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11			

Maintaining Heap Property

To fix the max-heap property at $A[5]$:

- Find the largest among $A[5]$, $A[10]$ and $A[11]$, which is $A[11]$.
- Swap $A[5]$ and $A[11]$.

Example 1

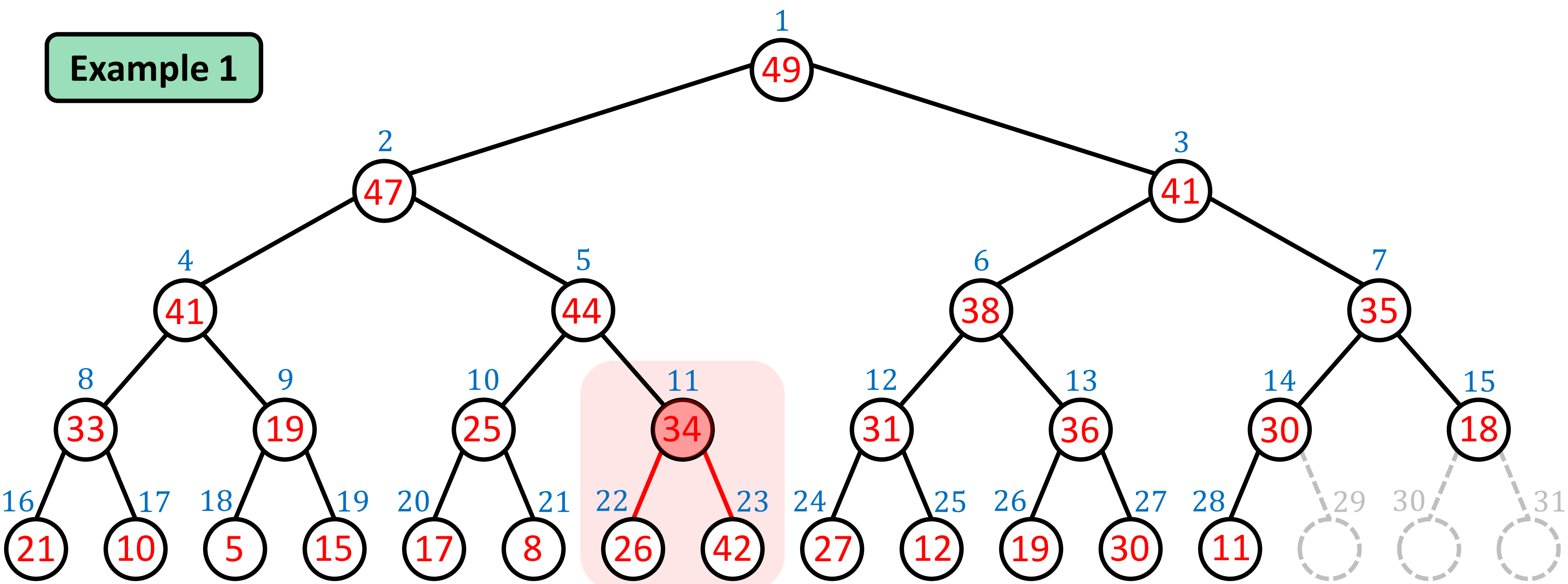


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	49	47	41	41	44	38	35	33	19	25	34	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11			

Maintaining Heap Property

The max-heap property is violated at $A[11]$ though the subtrees rooted at $A[22]$ and $A[23]$ are both valid max-heaps.

Example 1



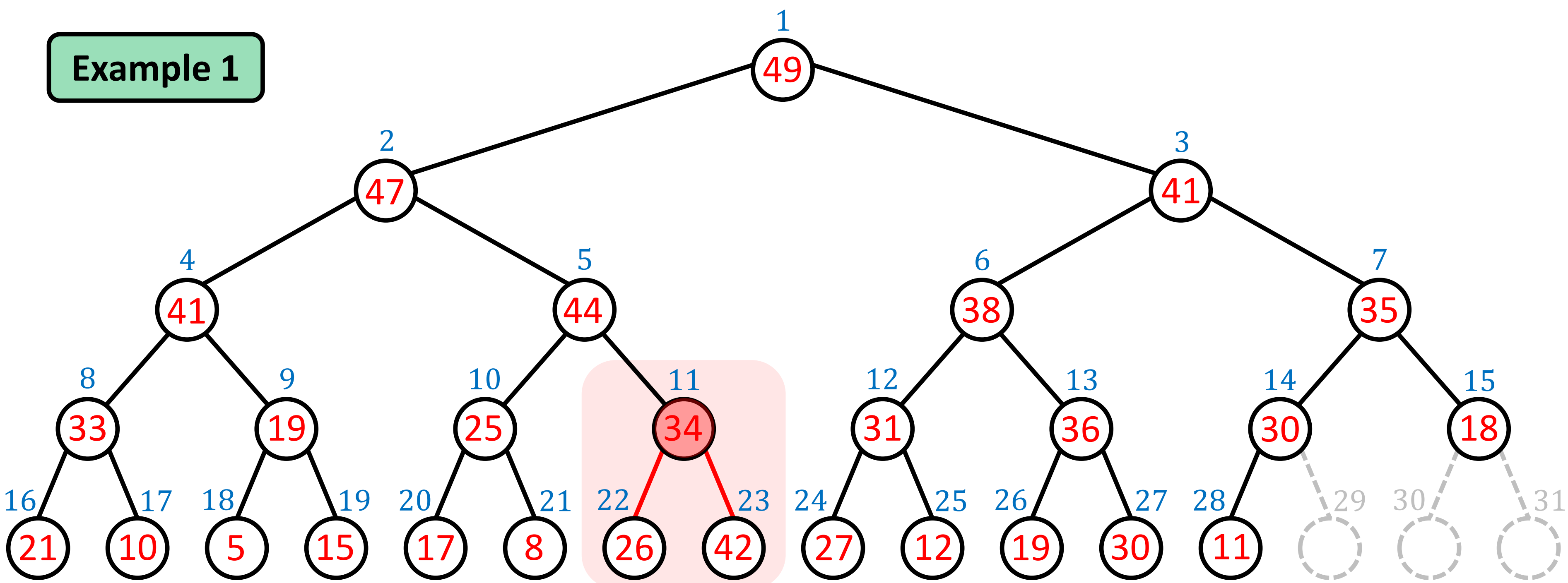
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	49	47	41	41	44	38	35	33	19	25	34	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11			

Maintaining Heap Property

To fix the max-heap property at $A[11]$:

- Find the largest among $A[11]$, $A[22]$ and $A[23]$, which is $A[23]$.
- Swap $A[11]$ and $A[23]$.

Example 1



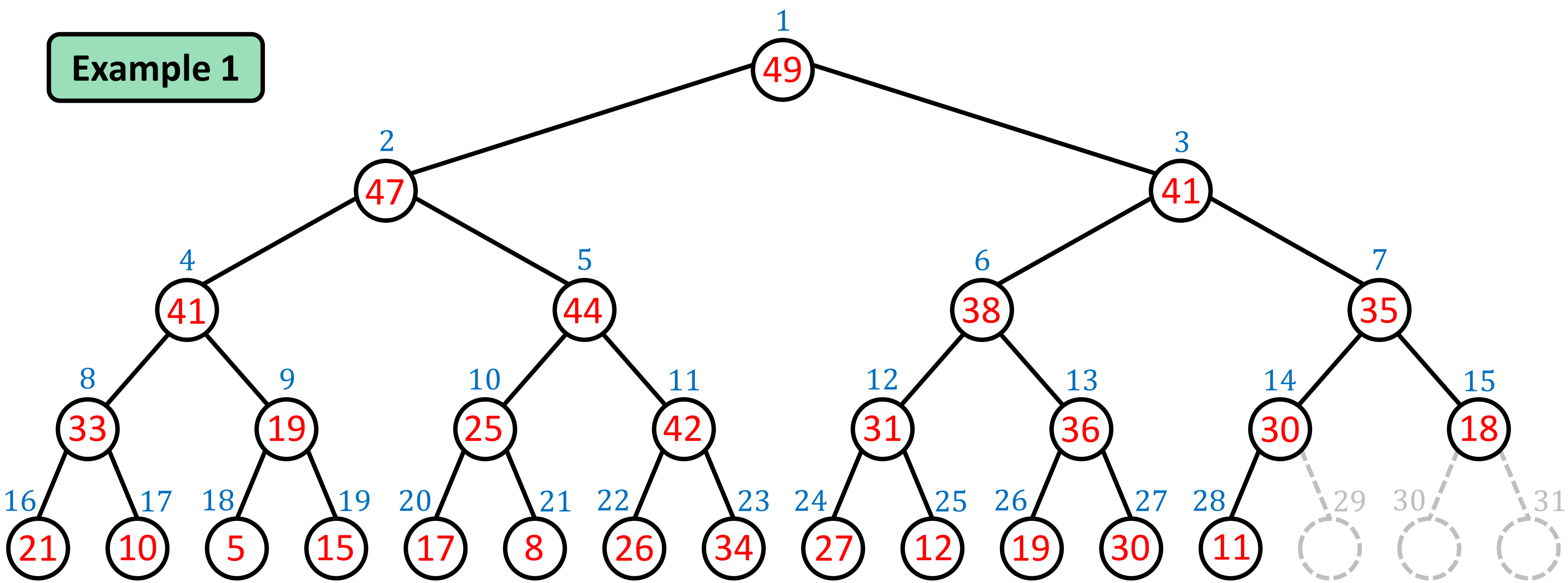
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	49	47	41	41	44	38	35	33	19	25	34	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11			

Maintaining Heap Property

To fix the max-heap property at $A[11]$:

- Find the largest among $A[11]$, $A[22]$ and $A[23]$, which is $A[23]$.
- Swap $A[11]$ and $A[23]$.

Example 1

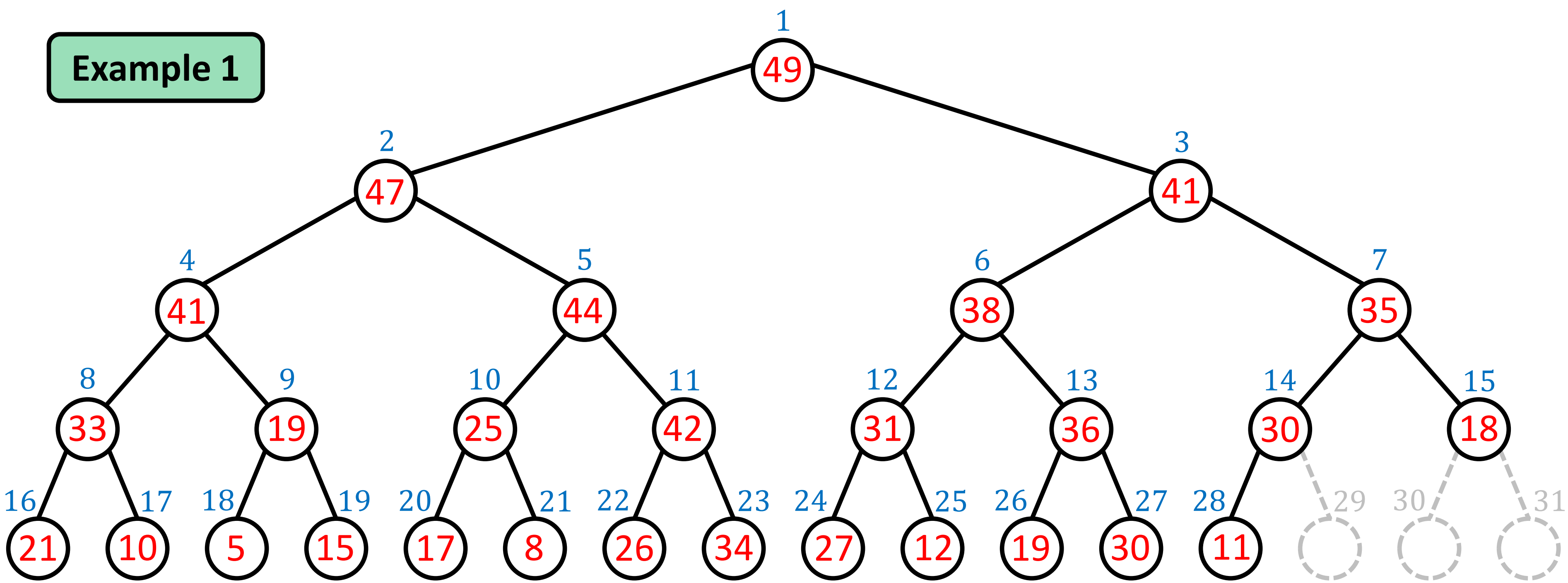


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	49	47	41	41	44	38	35	33	19	25	42	31	36	30	18	21	10	5	15	17	8	26	34	27	12	19	30	11			

Maintaining Heap Property

This is now a valid max-heap:

Example 1

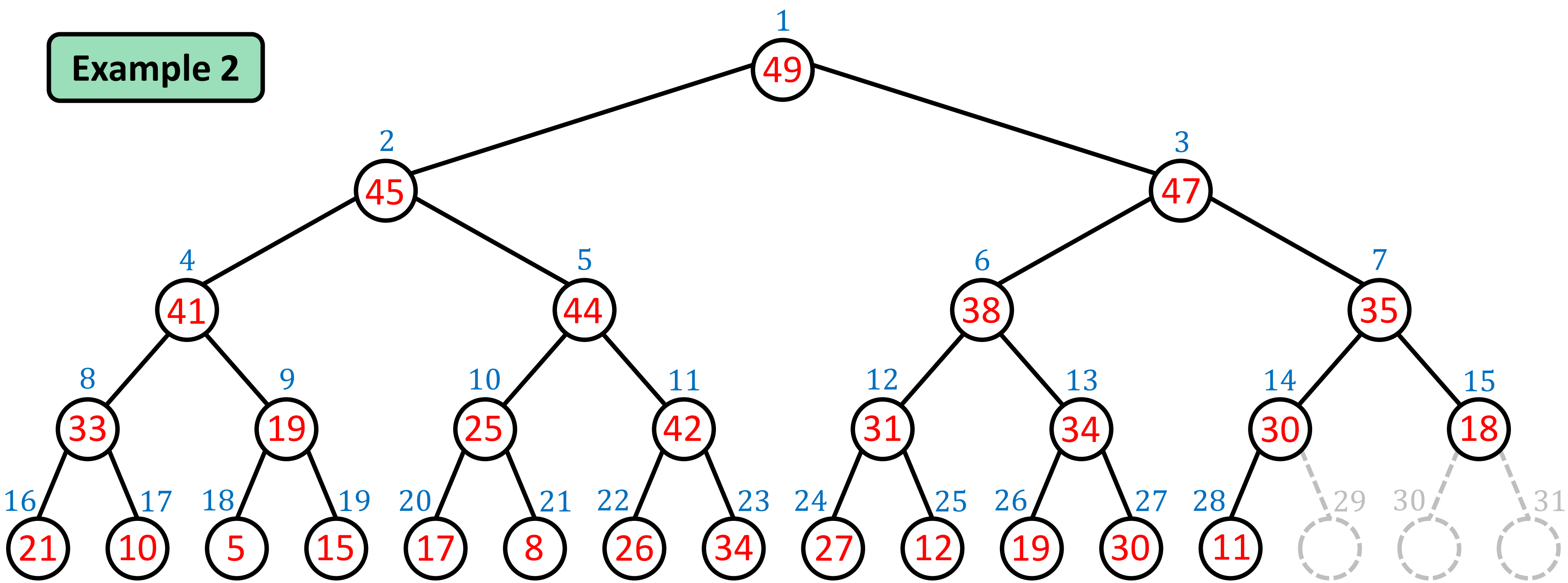


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	49	47	41	41	44	38	35	33	19	25	42	31	36	30	18	21	10	5	15	17	8	26	34	27	12	19	30	11			

Maintaining Heap Property

This is another valid max-heap:

Example 2

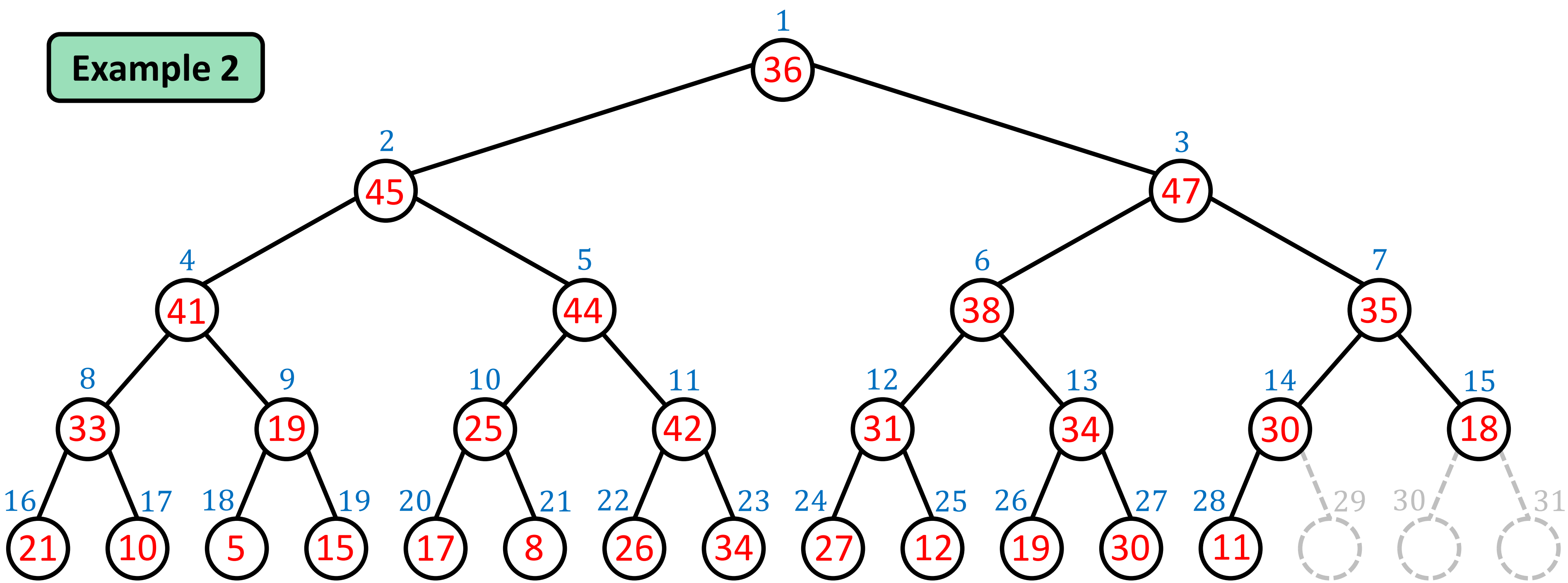


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	49	45	47	41	44	38	35	33	19	25	42	31	34	30	18	21	10	5	15	17	8	26	34	27	12	19	30	11			

Maintaining Heap Property

But this is not:

Example 2

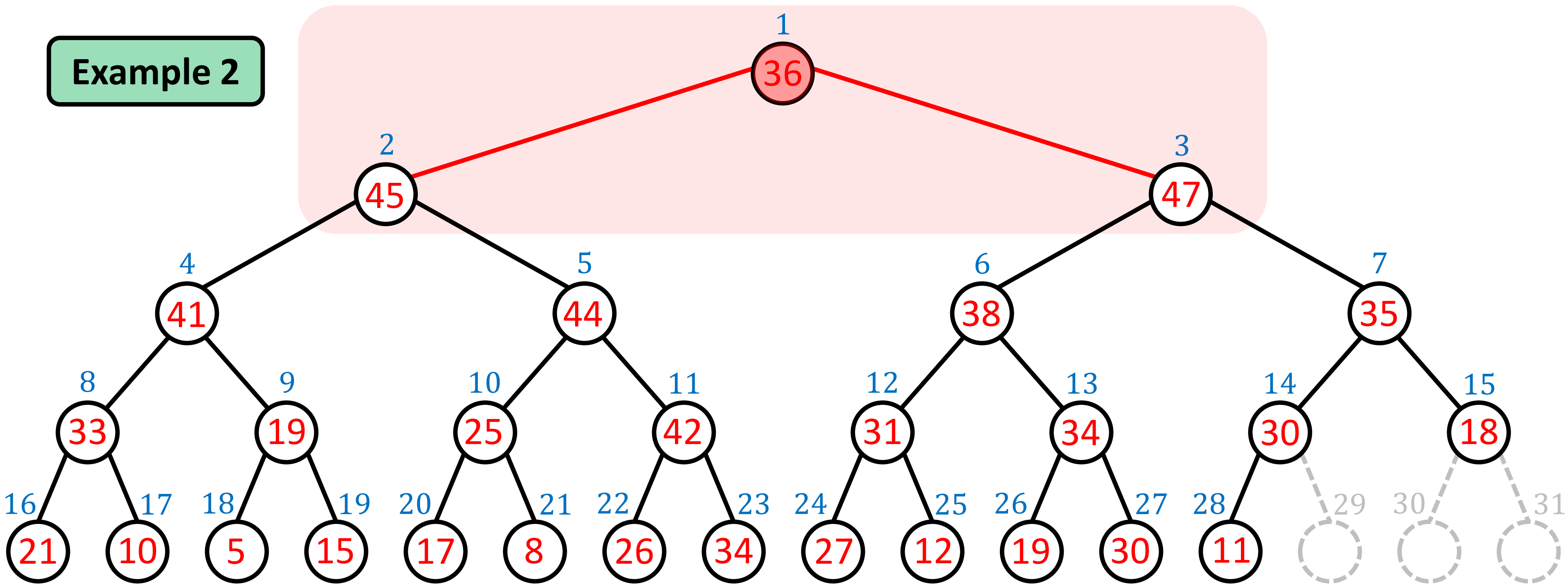


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	36	45	47	41	44	38	35	33	19	25	42	31	34	30	18	21	10	5	15	17	8	26	34	27	12	19	30	11			

Maintaining Heap Property

The max-heap property is violated at $A[1]$ though the subtrees rooted at $A[2]$ and $A[3]$ are both valid max-heaps.

Example 2



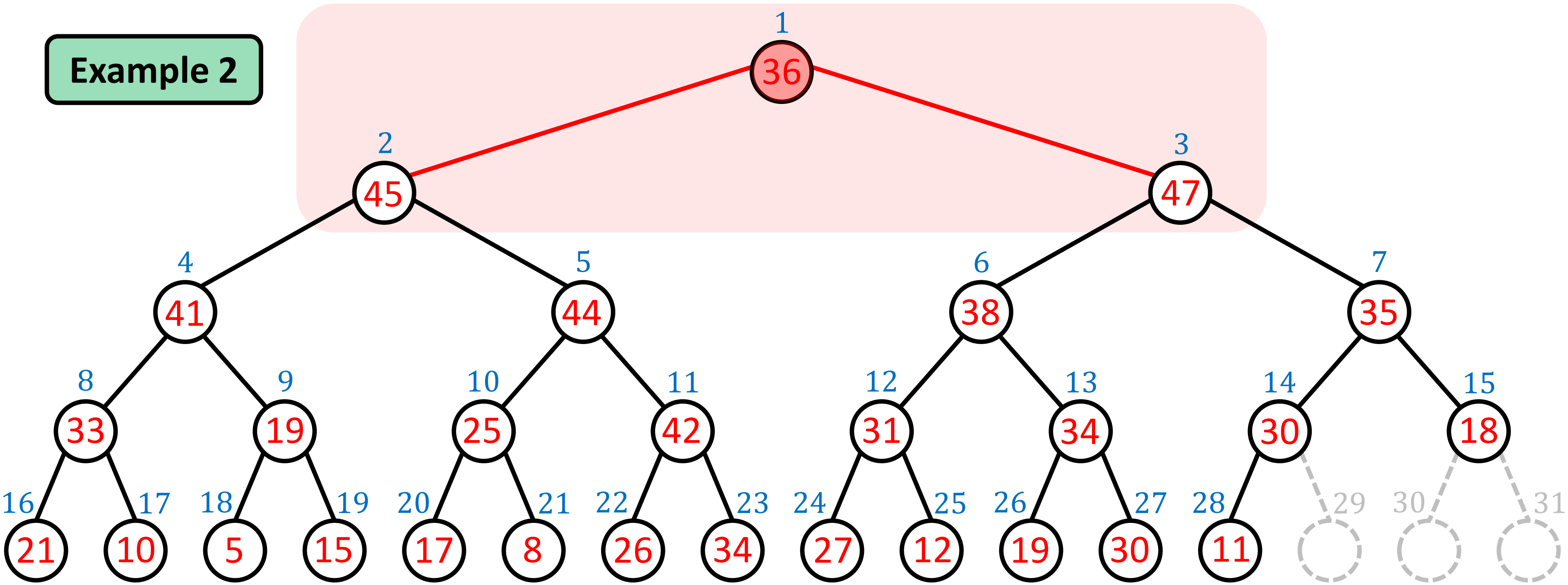
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	36	45	47	41	44	38	35	33	19	25	42	31	34	30	18	21	10	5	15	17	8	26	34	27	12	19	30	11			

Maintaining Heap Property

To fix the max-heap property at $A[1]$:

- Find the largest among $A[1]$, $A[2]$ and $A[3]$, which is $A[3]$.
- Swap $A[1]$ and $A[3]$.

Example 2



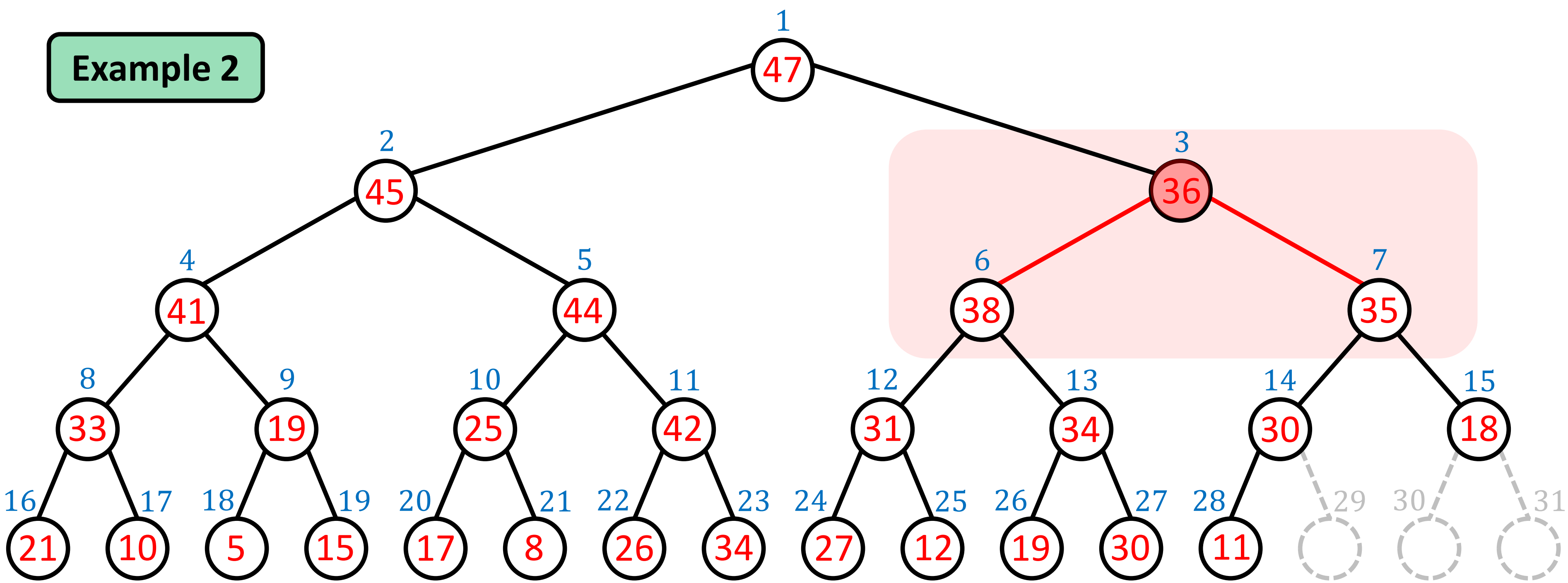
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	36	45	47	41	44	38	35	33	19	25	42	31	34	30	18	21	10	5	15	17	8	26	34	27	12	19	30	11			

Maintaining Heap Property

To fix the max-heap property at $A[1]$:

- Find the largest among $A[1]$, $A[2]$ and $A[3]$, which is $A[3]$.
- Swap $A[1]$ and $A[3]$.

Example 2

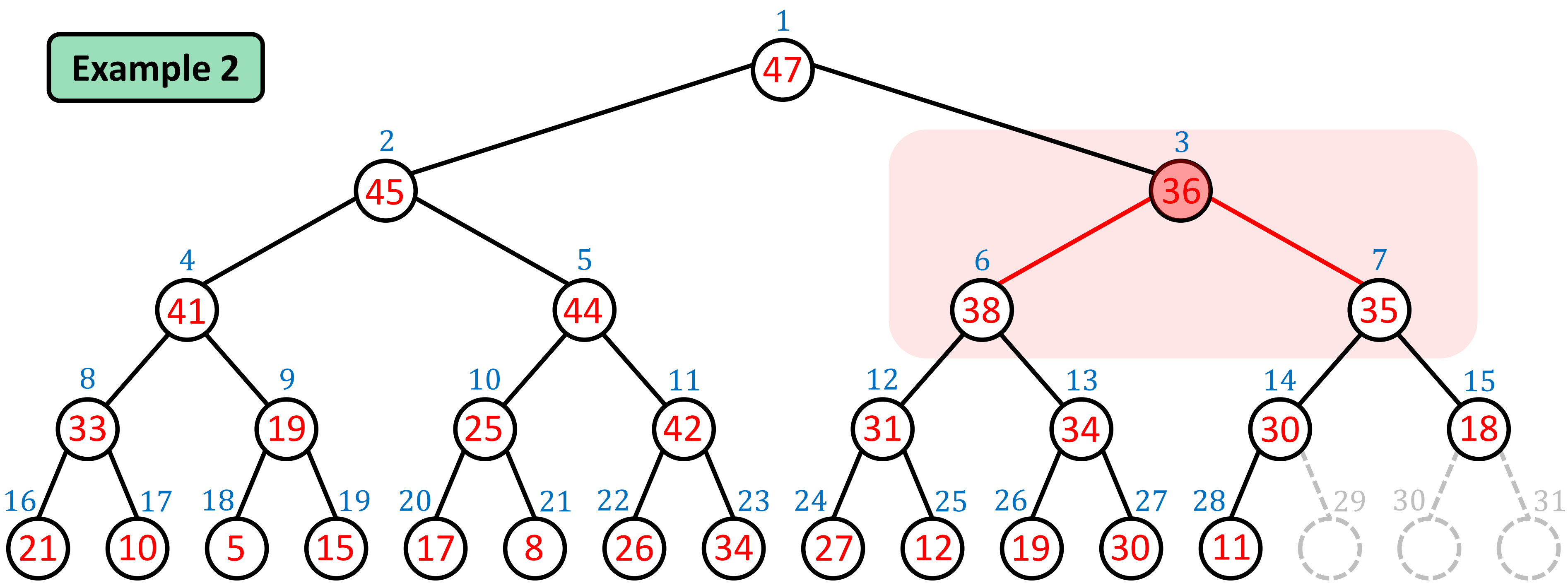


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	47	45	36	41	44	38	35	33	19	25	42	31	34	30	18	21	10	5	15	17	8	26	34	27	12	19	30	11			

Maintaining Heap Property

The max-heap property is violated at $A[3]$ though the subtrees rooted at $A[6]$ and $A[7]$ are both valid max-heaps.

Example 2



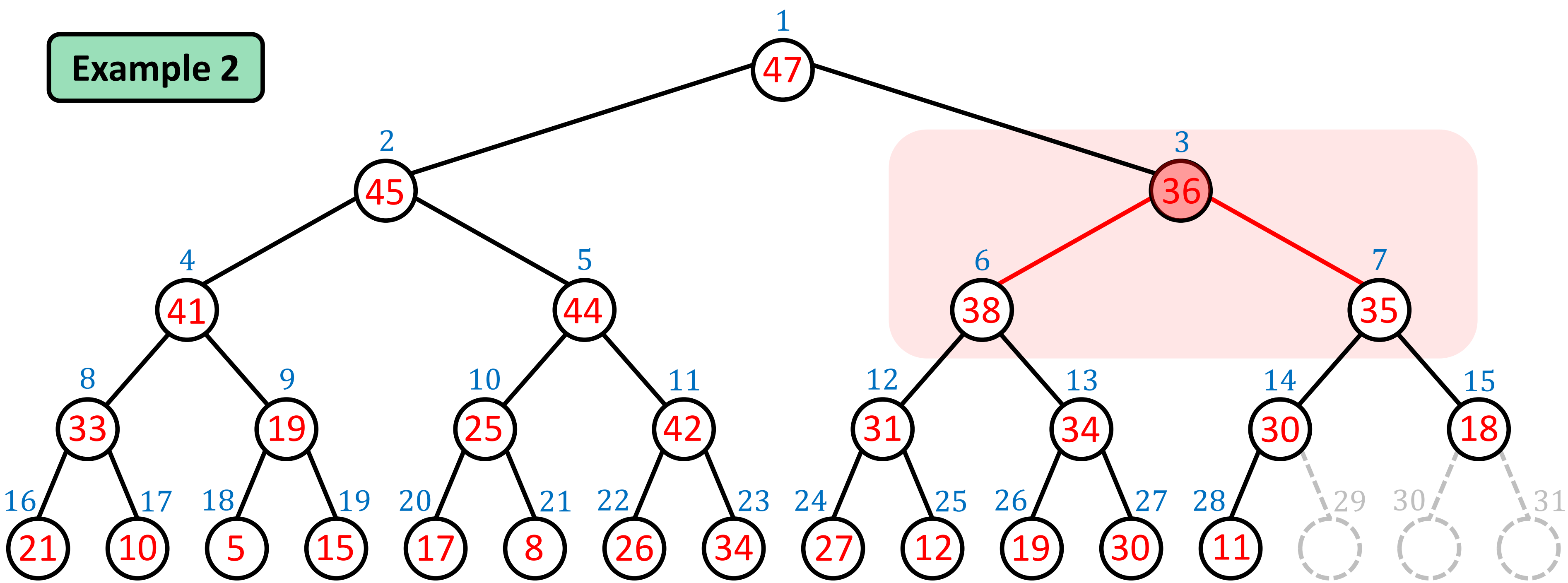
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	47	45	36	41	44	38	35	33	19	25	42	31	34	30	18	21	10	5	15	17	8	26	34	27	12	19	30	11			

Maintaining Heap Property

To fix the max-heap property at $A[3]$:

- Find the largest among $A[3]$, $A[6]$ and $A[7]$, which is $A[6]$.
- Swap $A[3]$ and $A[6]$.

Example 2



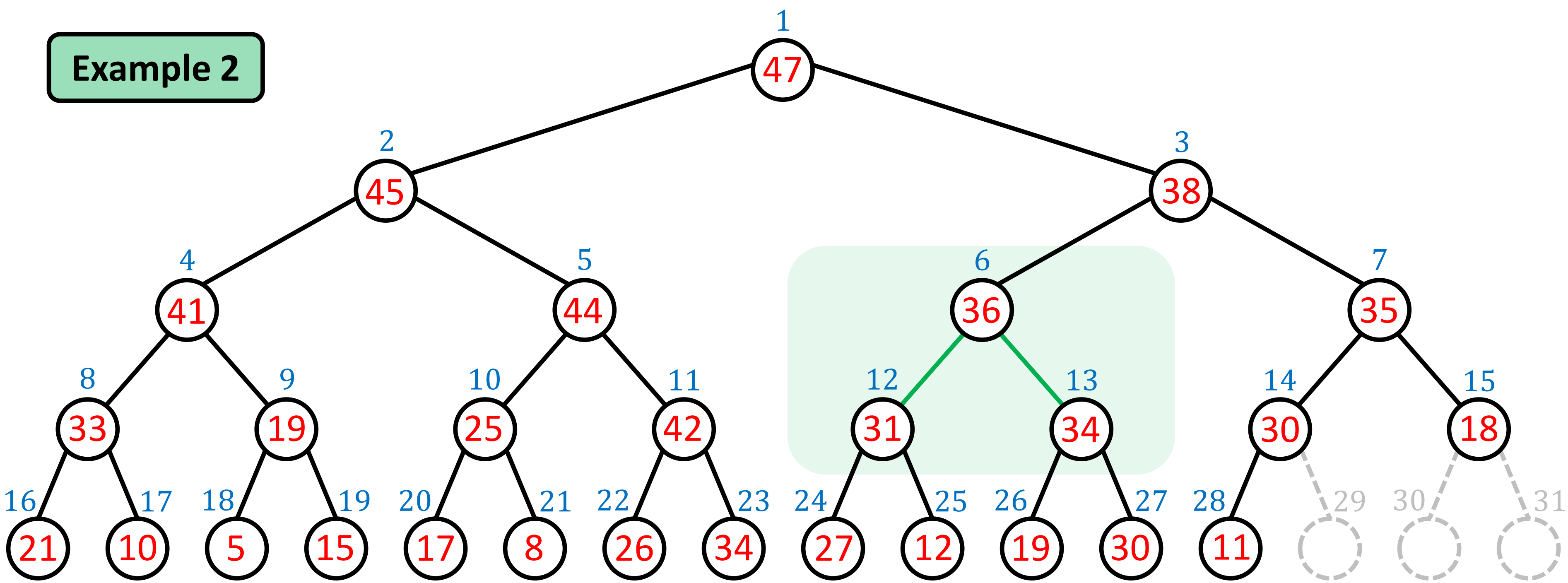
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	47	45	36	41	44	38	35	33	19	25	42	31	34	30	18	21	10	5	15	17	8	26	34	27	12	19	30	11			

Maintaining Heap Property

To fix the max-heap property at $A[3]$:

- Find the largest among $A[3]$, $A[6]$ and $A[7]$, which is $A[6]$.
- Swap $A[3]$ and $A[6]$.

Example 2

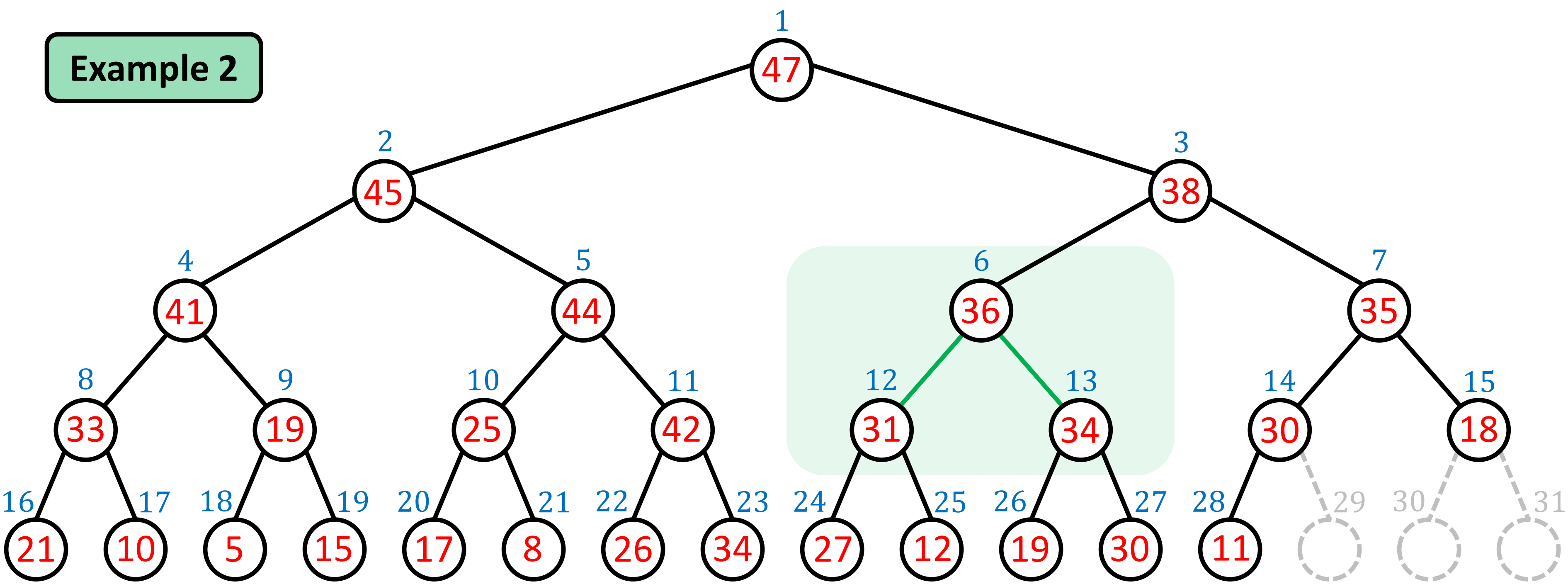


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	47	45	38	41	44	36	35	33	19	25	42	31	34	30	18	21	10	5	15	17	8	26	34	27	12	19	30	11			

Maintaining Heap Property

The max-heap property is not violated at $A[6]$.

Example 2

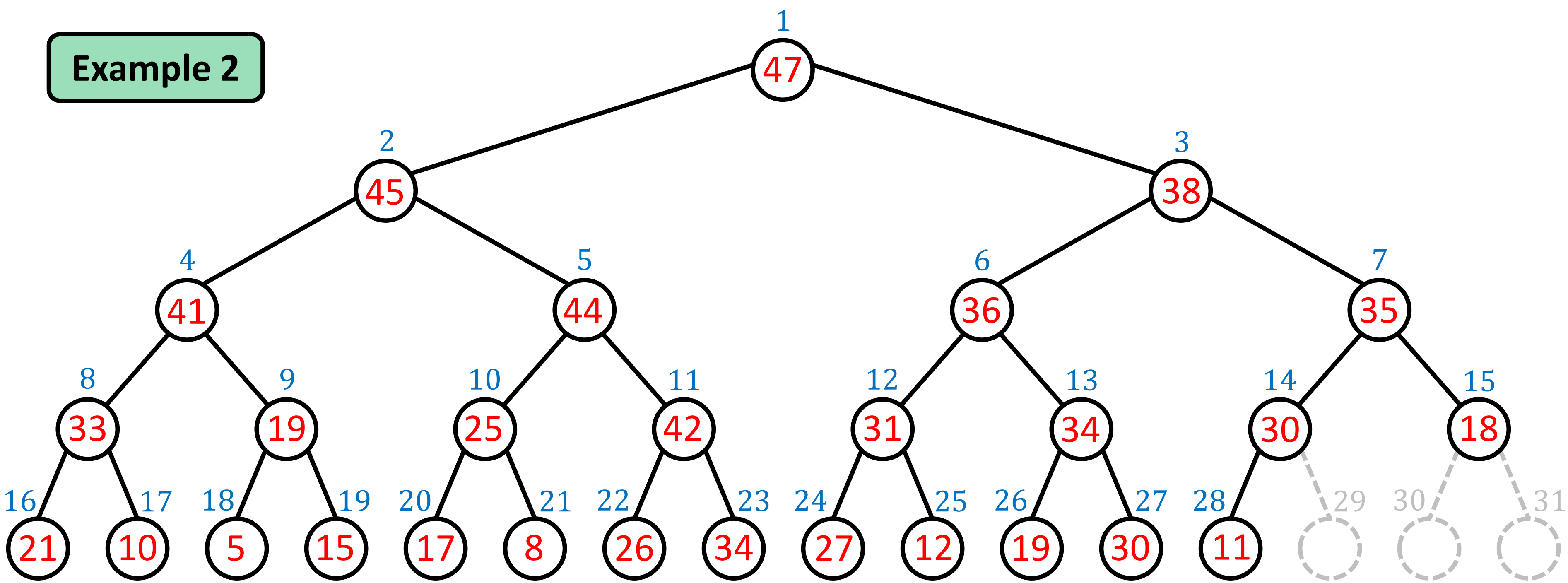


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	47	45	38	41	44	36	35	33	19	25	42	31	34	30	18	21	10	5	15	17	8	26	34	27	12	19	30	11			

Maintaining Heap Property

This is now a valid max-heap:

Example 2



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	47	45	38	41	44	36	35	33	19	25	42	31	34	30	18	21	10	5	15	17	8	26	34	27	12	19	30	11			

Maintaining Heap Property

Input: An array A and an index i into the array with the subtrees rooted at $\text{LEFT}(i)$ and $\text{RIGHT}(i)$ are max-heaps, but $A[i]$ might be smaller than its children and thus violating the max-heap property.

Output: Array A with its elements rearranged so that the subtree rooted at index i is a max-heap.

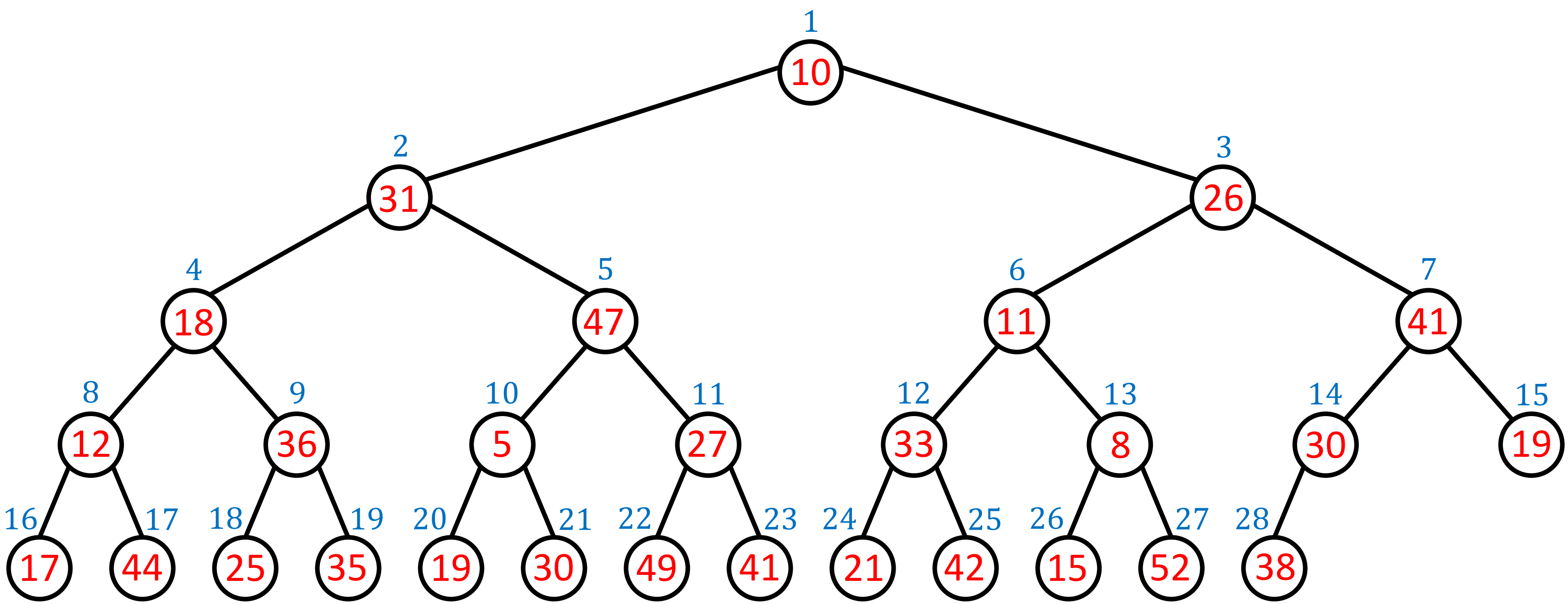
```
MAX-HEAPIFY ( A, i )

    l = LEFT(i)
    r = RIGHT(i)
    if l ≤ A.heapsize and A[l] > A[i]
        largest = l
    else largest = i
    if r ≤ A.heapsize and A[r] > A[largest]
        largest = r
    if largest ≠ i
        exchange A[i] with A[largest]
        MAX-HEAPIFY ( A, largest )
```

Building a Max-Heap

The items in $A[1..28]$ are not correctly ordered to form a valid max-heap.

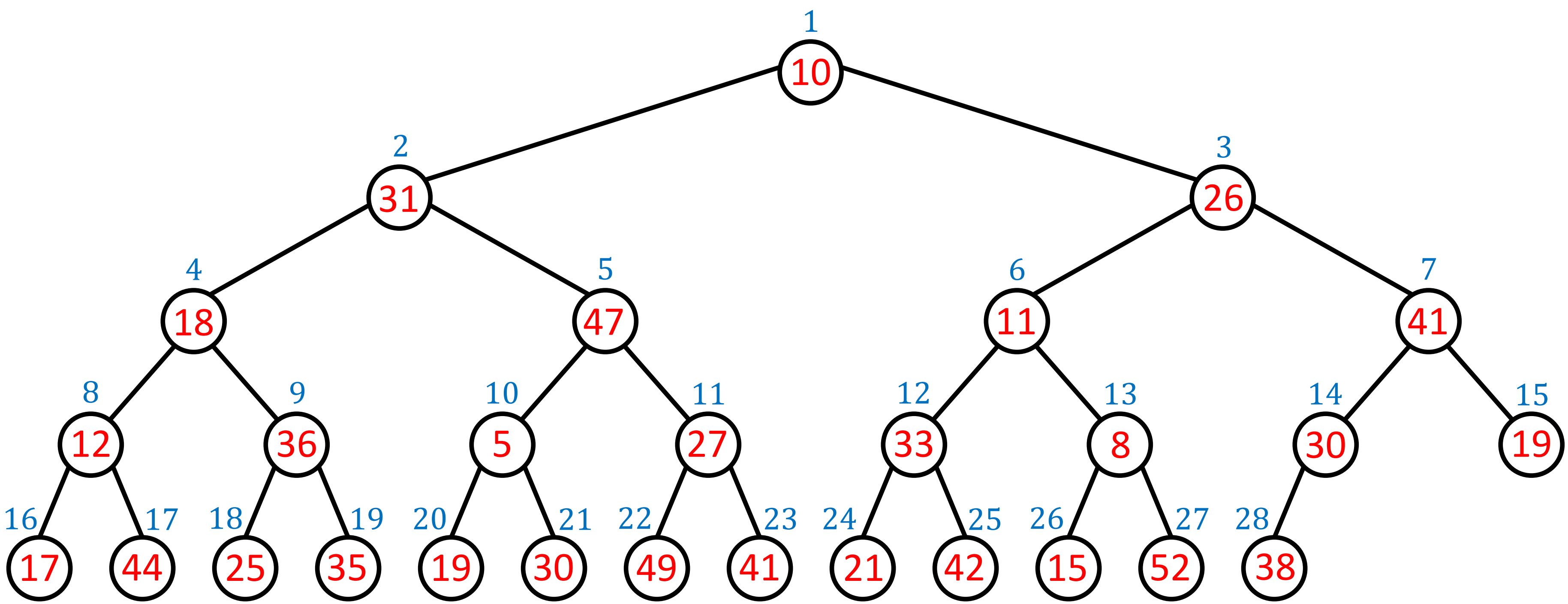
We want to reorder them so that they form a valid max-heap.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	5	27	33	8	30	19	17	44	25	35	19	30	49	41	21	42	15	52	38

Building a Max-Heap

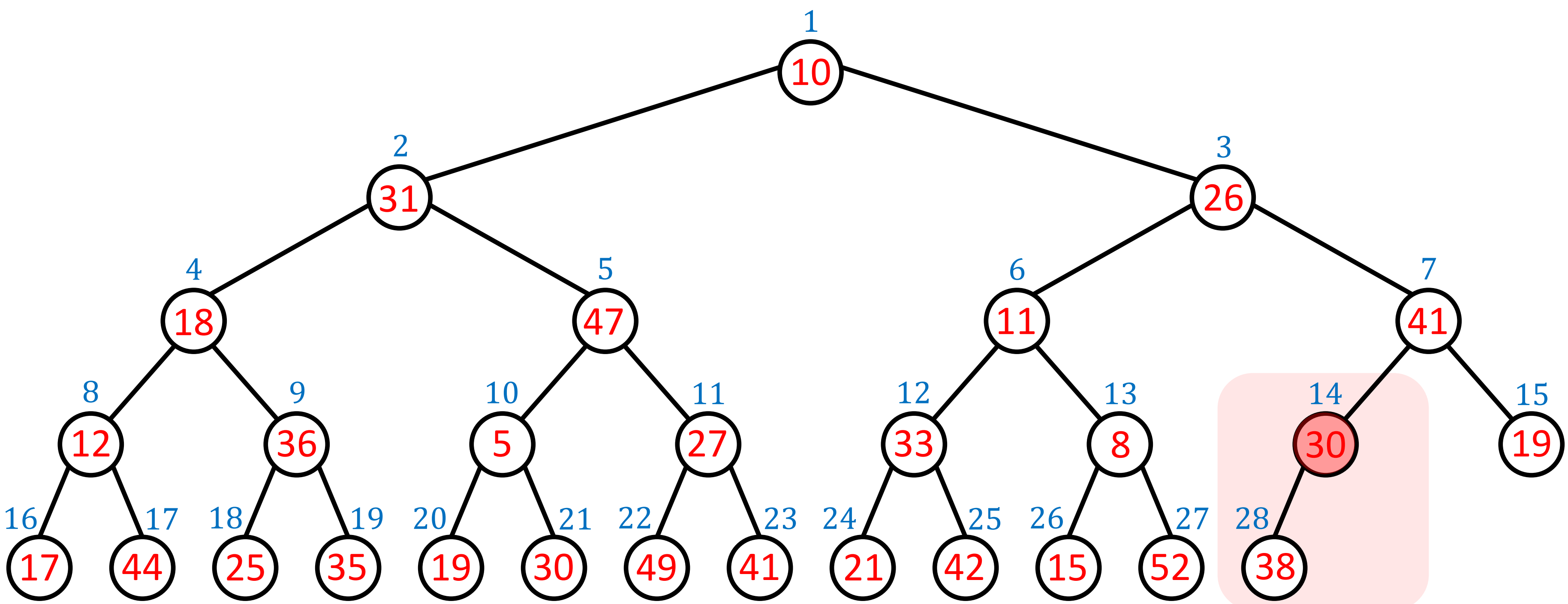
We reorder the items by calling MAX-HEAPIFY(A, i) on each $A[i]$ starting from $i = \left\lfloor \frac{A.length}{2} \right\rfloor = \left\lfloor \frac{28}{2} \right\rfloor = 14$ down to $i = 1$.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	5	27	33	8	30	19	17	44	25	35	19	30	49	41	21	42	15	52	38

Building a Max-Heap

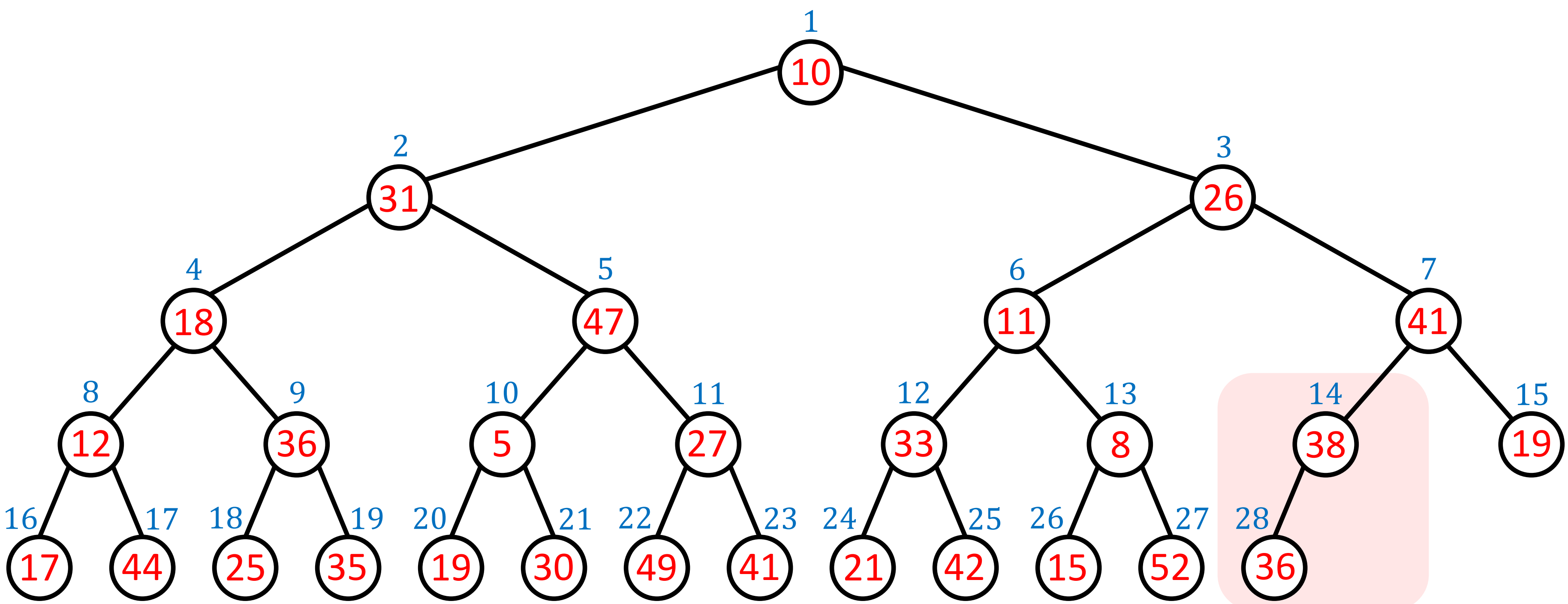
MAX-HEAPIFY(A, 14):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	5	27	33	8	30	19	17	44	25	35	19	30	49	41	21	42	15	52	38

Building a Max-Heap

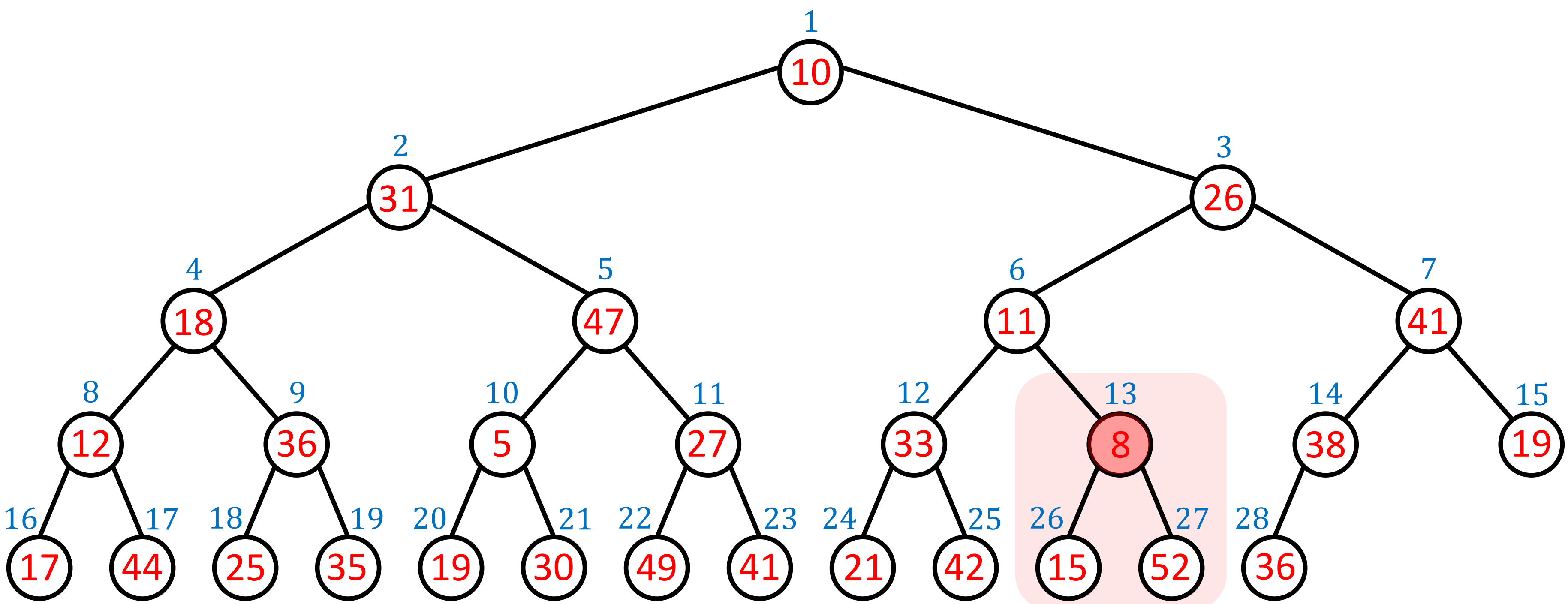
MAX-HEAPIFY(A, 14):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	5	27	33	8	38	19	17	44	25	35	19	30	49	41	21	42	15	52	36

Building a Max-Heap

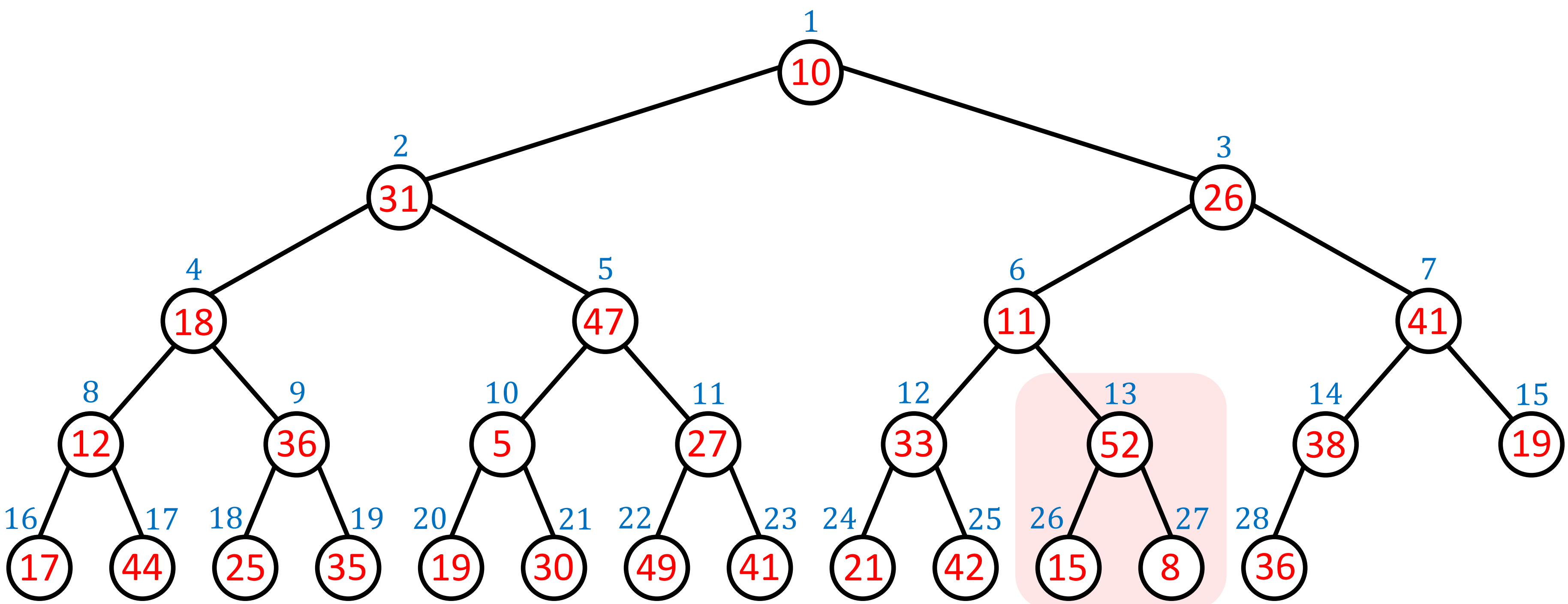
MAX-HEAPIFY(A, 13):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	5	27	33	8	38	19	17	44	25	35	19	30	49	41	21	42	15	52	36

Building a Max-Heap

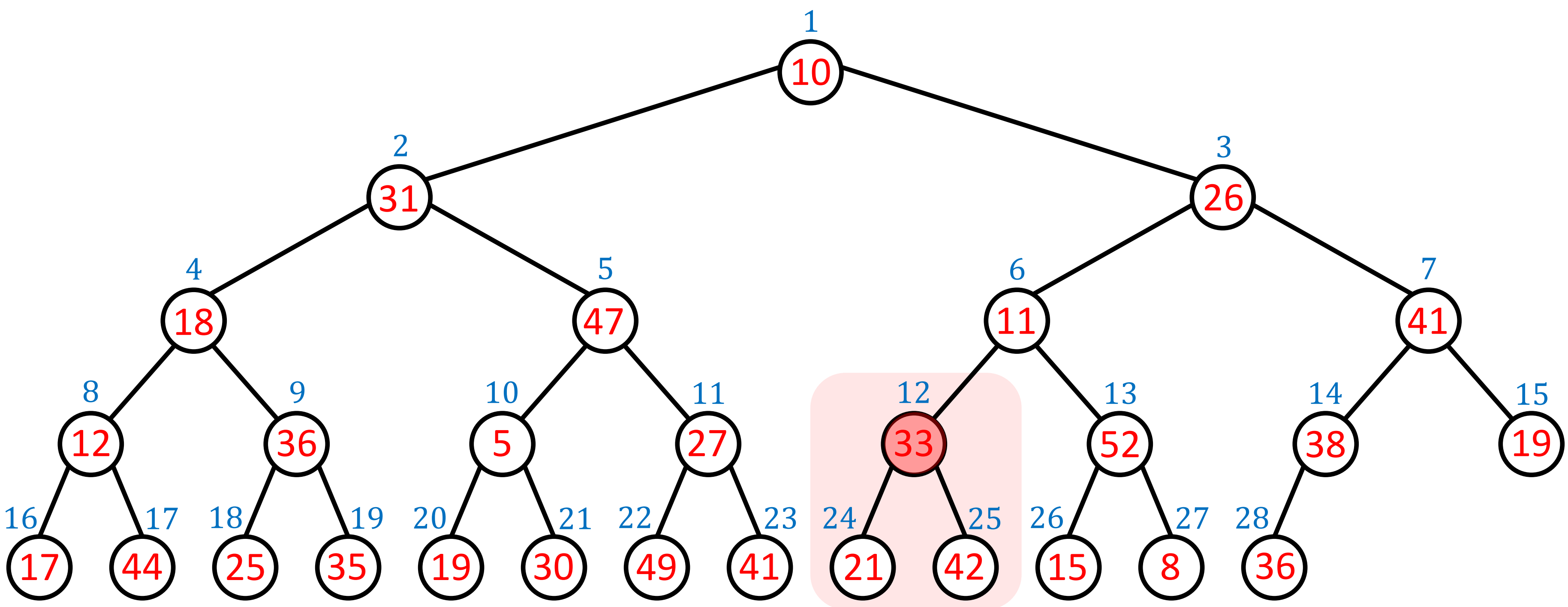
MAX-HEAPIFY(A, 13):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	5	27	33	52	38	19	17	44	25	35	19	30	49	41	21	42	15	8	36

Building a Max-Heap

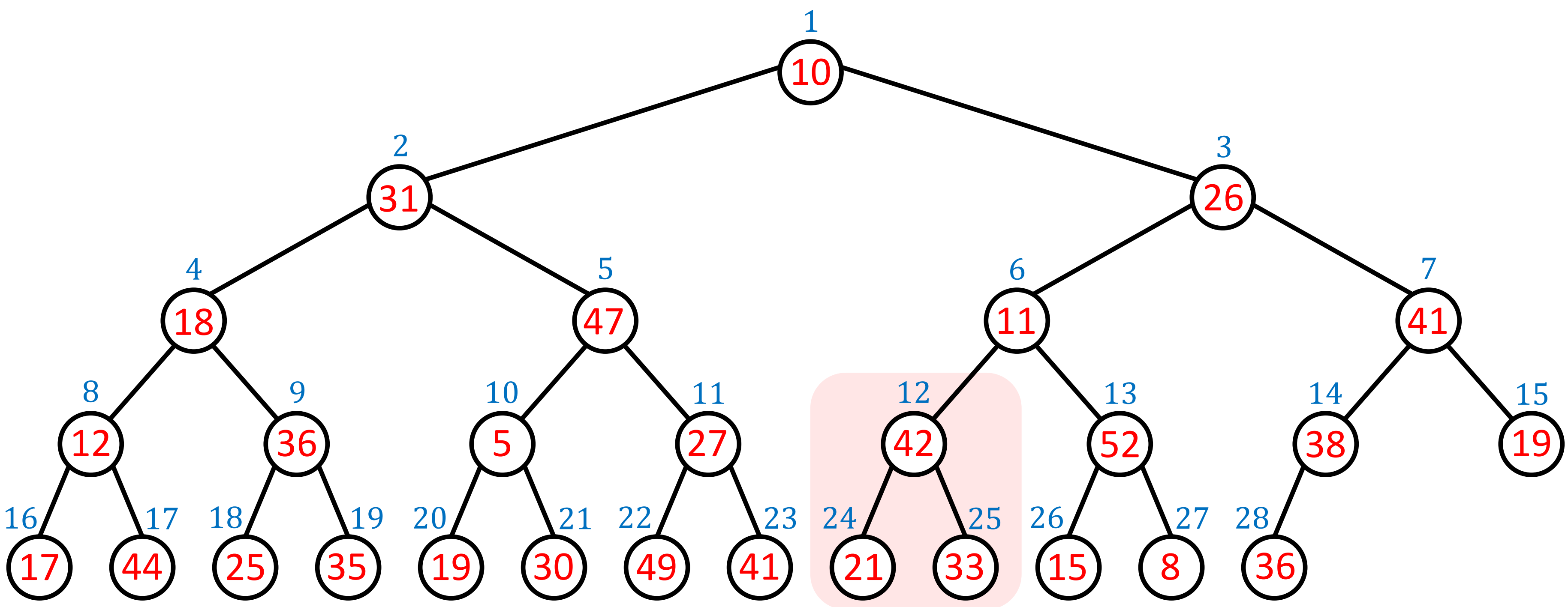
Max-Heapify(A, 12):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	5	27	33	52	38	19	17	44	25	35	19	30	49	41	21	42	15	8	36

Building a Max-Heap

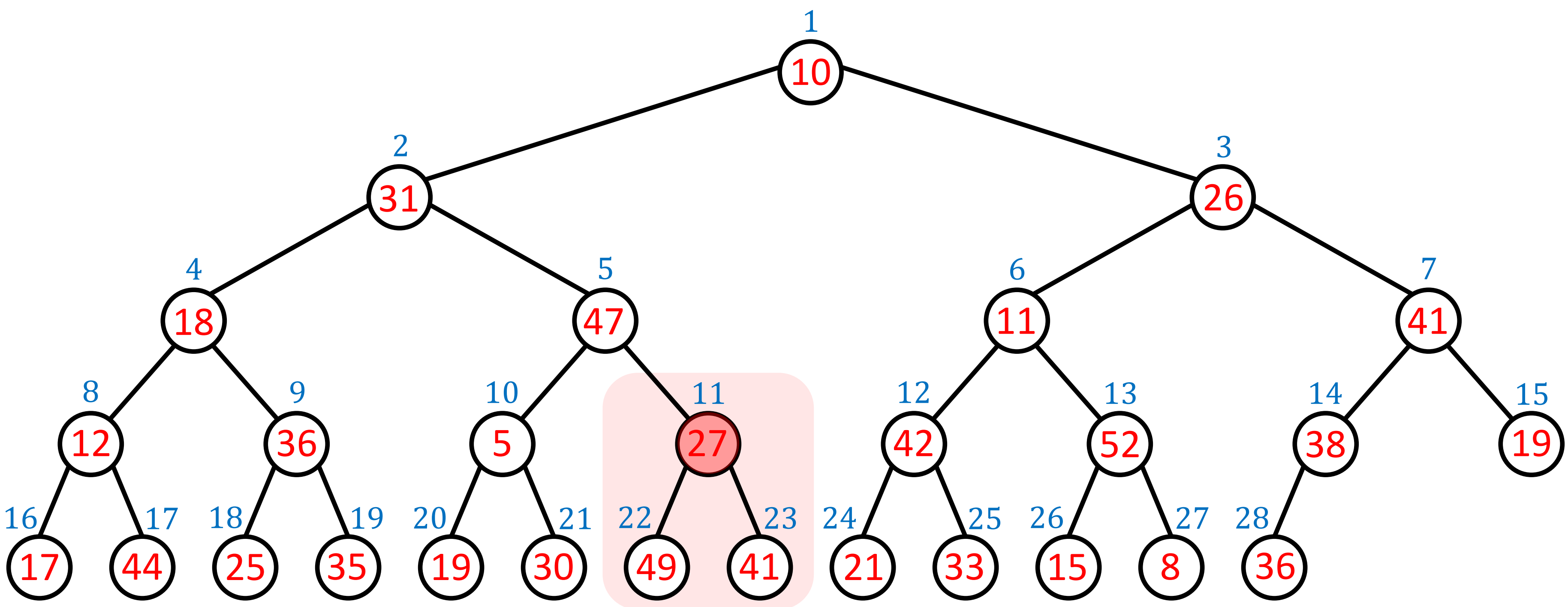
MAX-HEAPIFY(A, 12):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	5	27	42	52	38	19	17	44	25	35	19	30	49	41	21	33	15	8	36

Building a Max-Heap

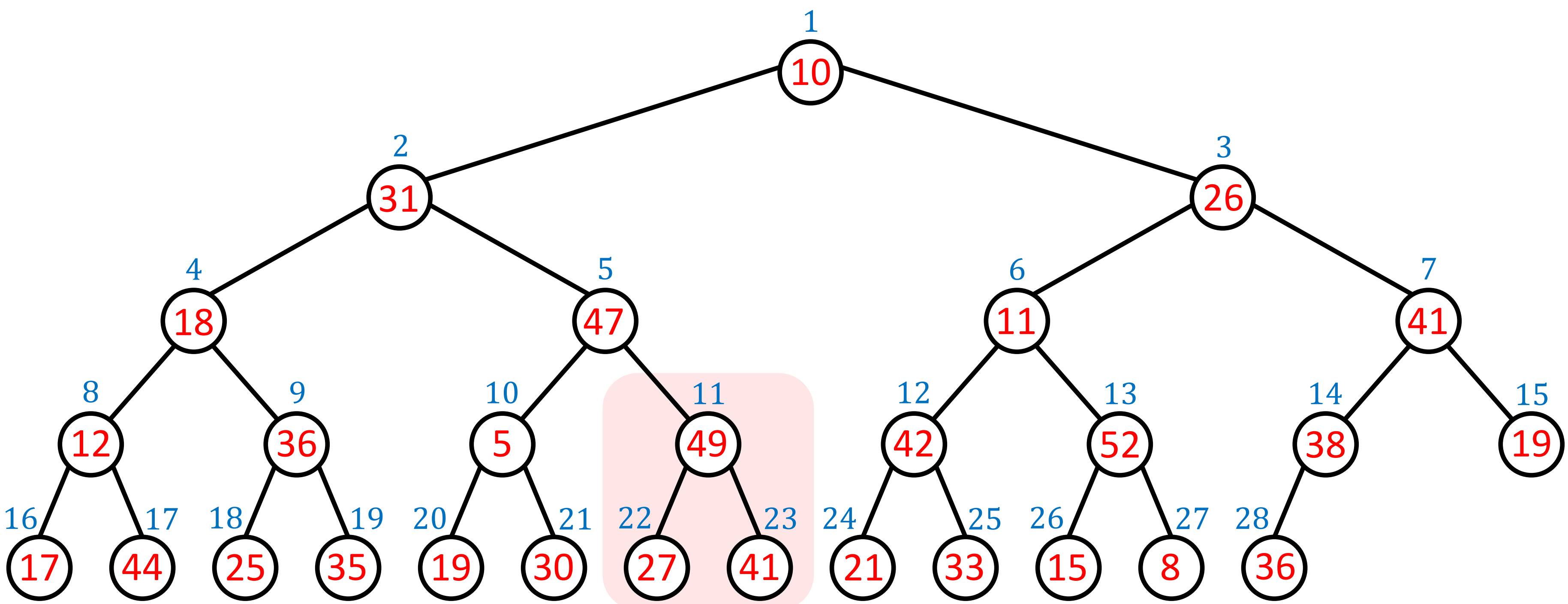
MAX-HEAPIFY(A, 11):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	5	27	42	52	38	19	17	44	25	35	19	30	49	41	21	33	15	8	36

Building a Max-Heap

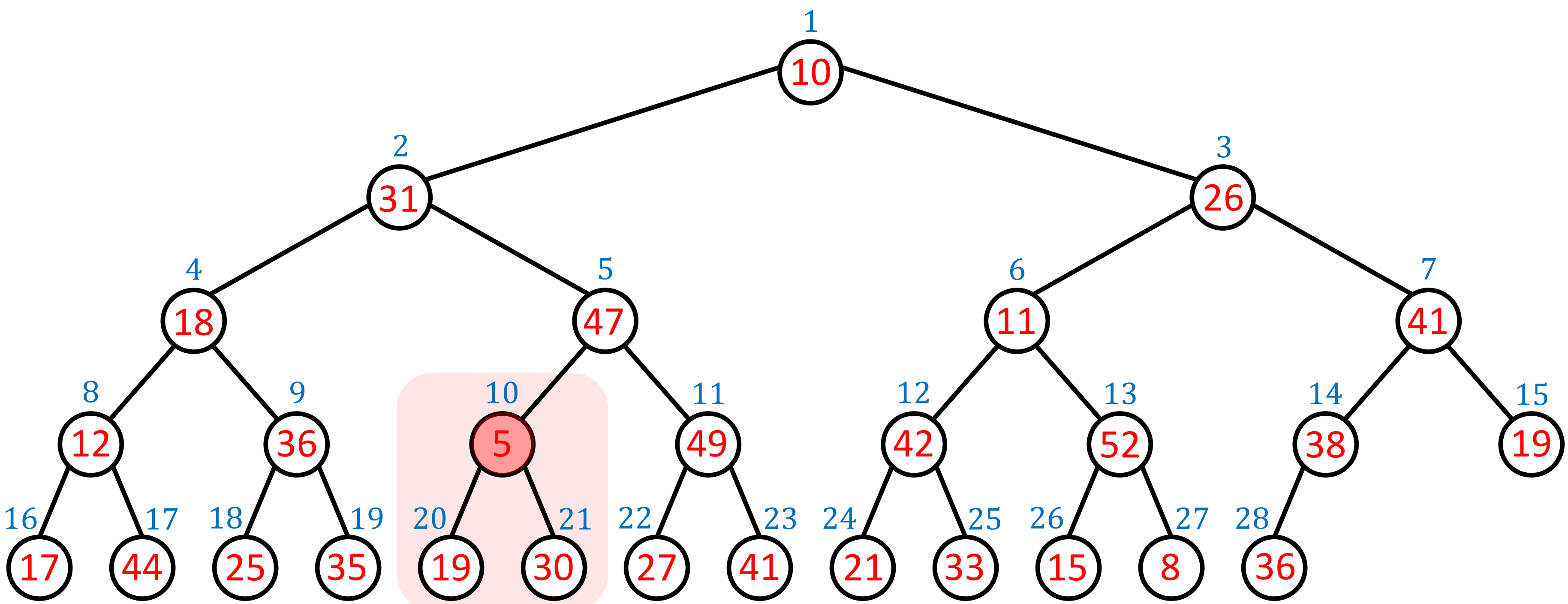
MAX-HEAPIFY(A, 11):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	5	49	42	52	38	19	17	44	25	35	19	30	27	41	21	33	15	8	36

Building a Max-Heap

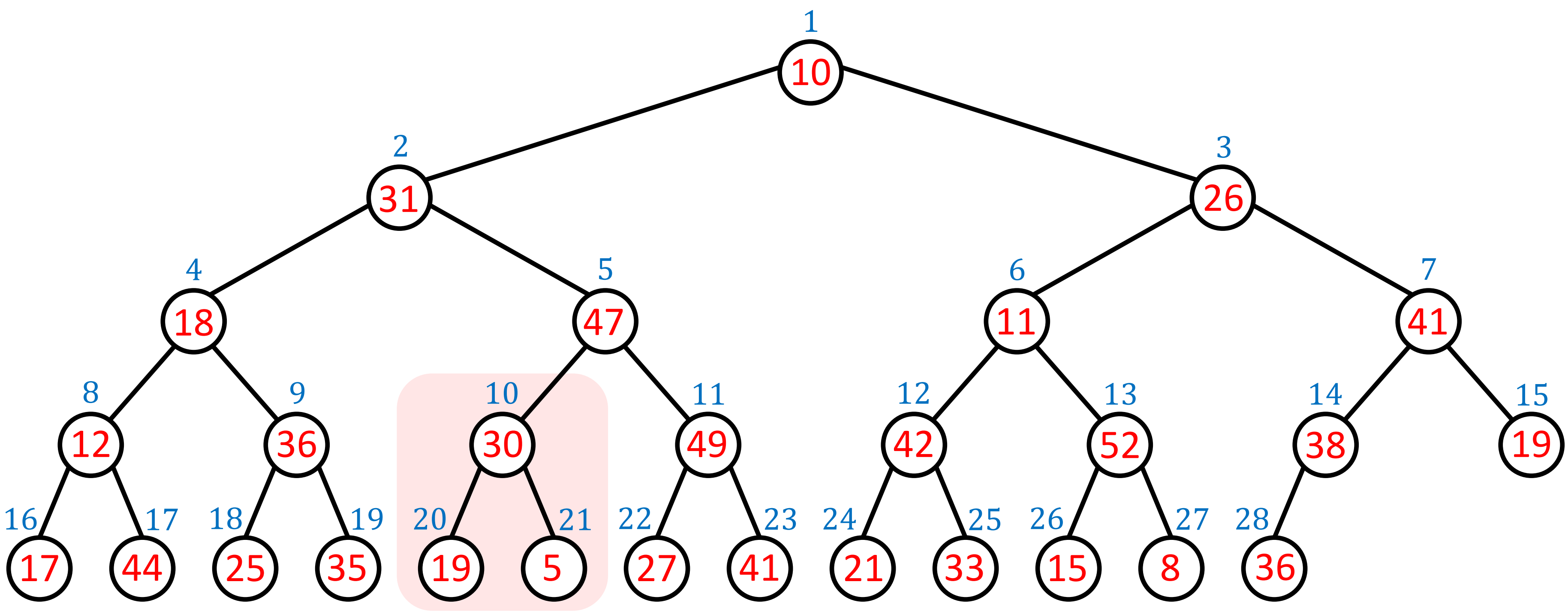
MAX-HEAPIFY(A, 10):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	5	49	42	52	38	19	17	44	25	35	19	30	27	41	21	33	15	8	36

Building a Max-Heap

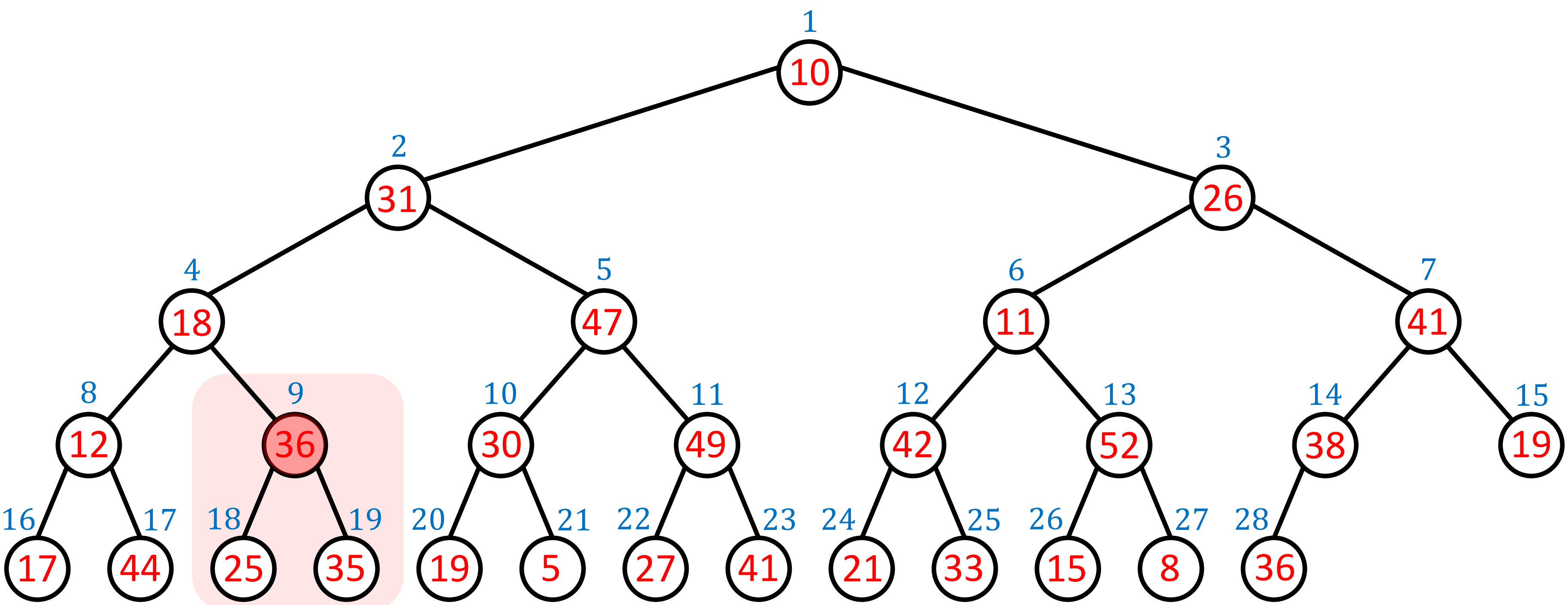
MAX-HEAPIFY(A, 10):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	30	49	42	52	38	19	17	44	25	35	19	5	27	41	21	33	15	8	36

Building a Max-Heap

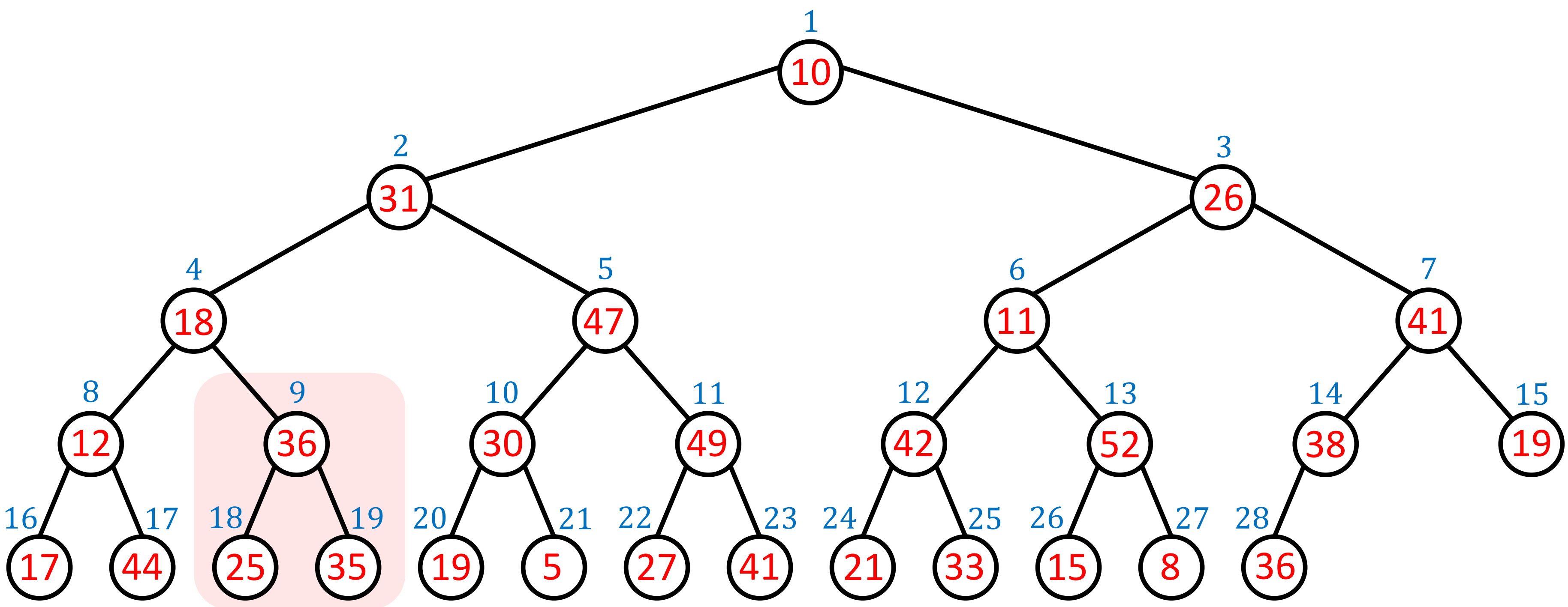
MAX-HEAPIFY(A, 9):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	30	49	42	52	38	19	17	44	25	35	19	5	27	41	21	33	15	8	36

Building a Max-Heap

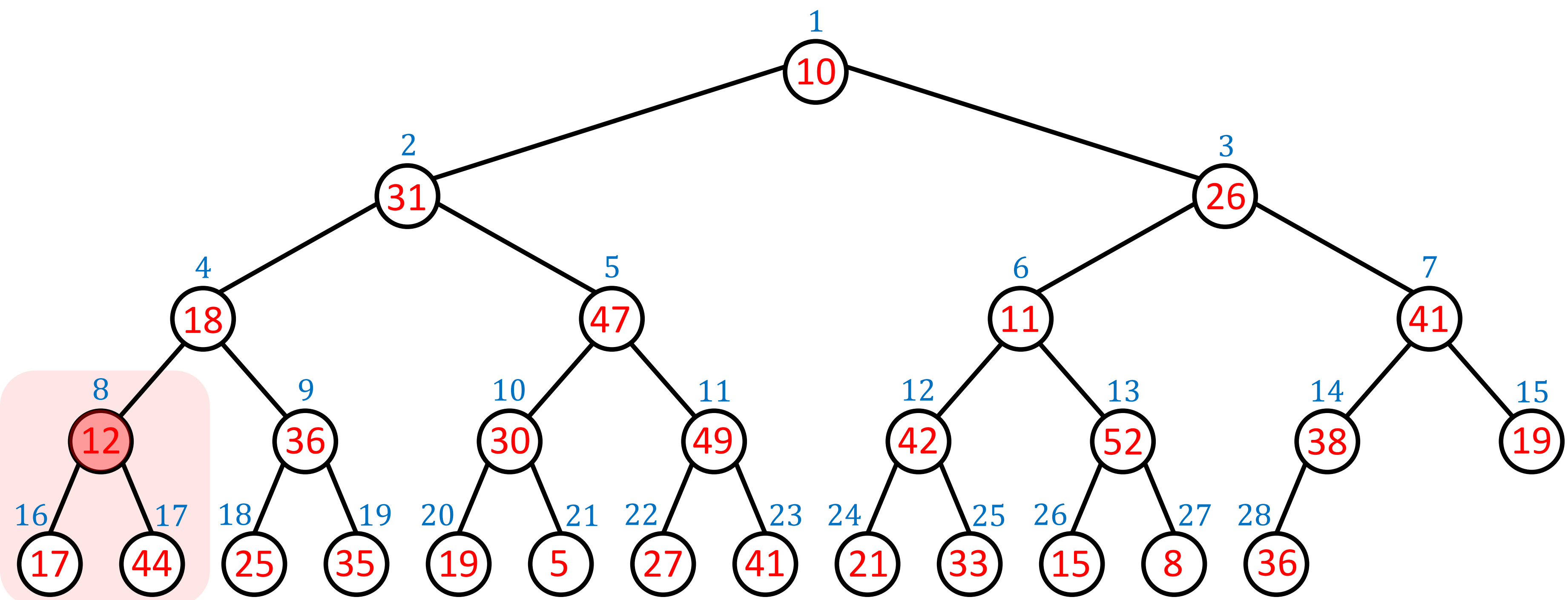
MAX-HEAPIFY(A, 9):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	30	49	42	52	38	19	17	44	25	35	19	5	27	41	21	33	15	8	36

Building a Max-Heap

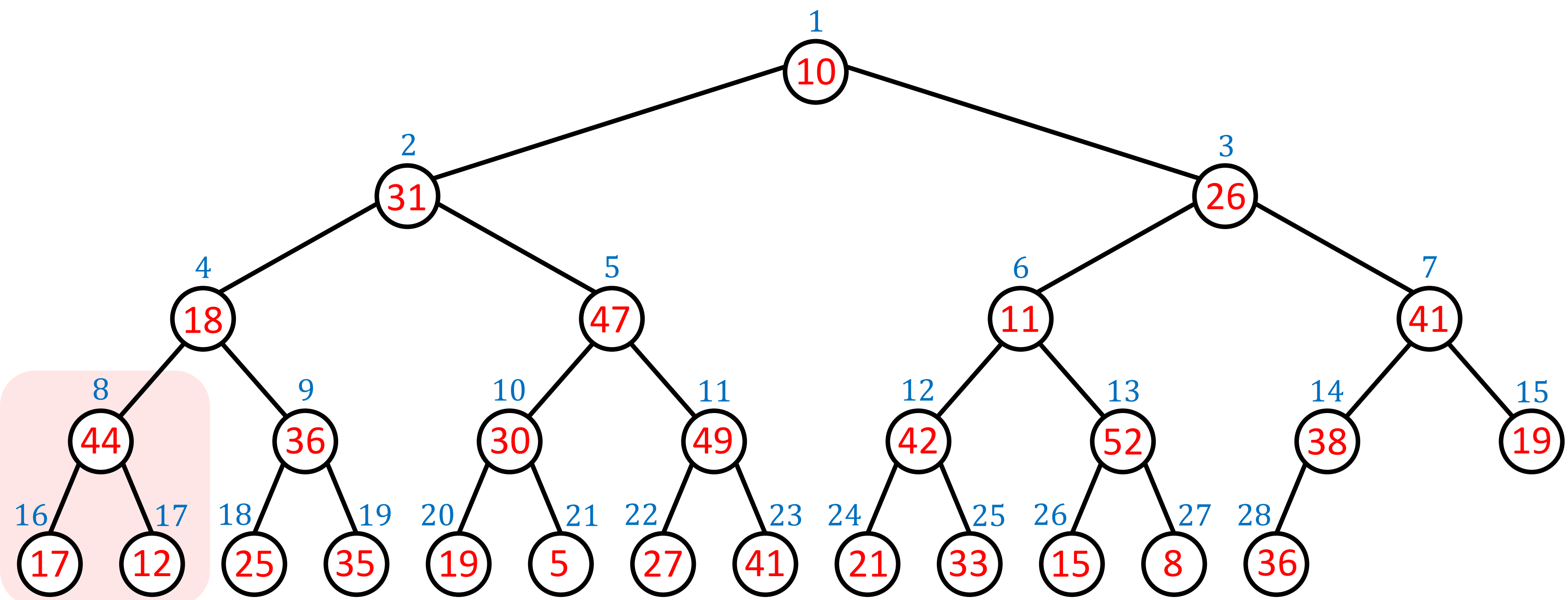
MAX-HEAPIFY(A, 8):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	30	49	42	52	38	19	17	44	25	35	19	5	27	41	21	33	15	8	36

Building a Max-Heap

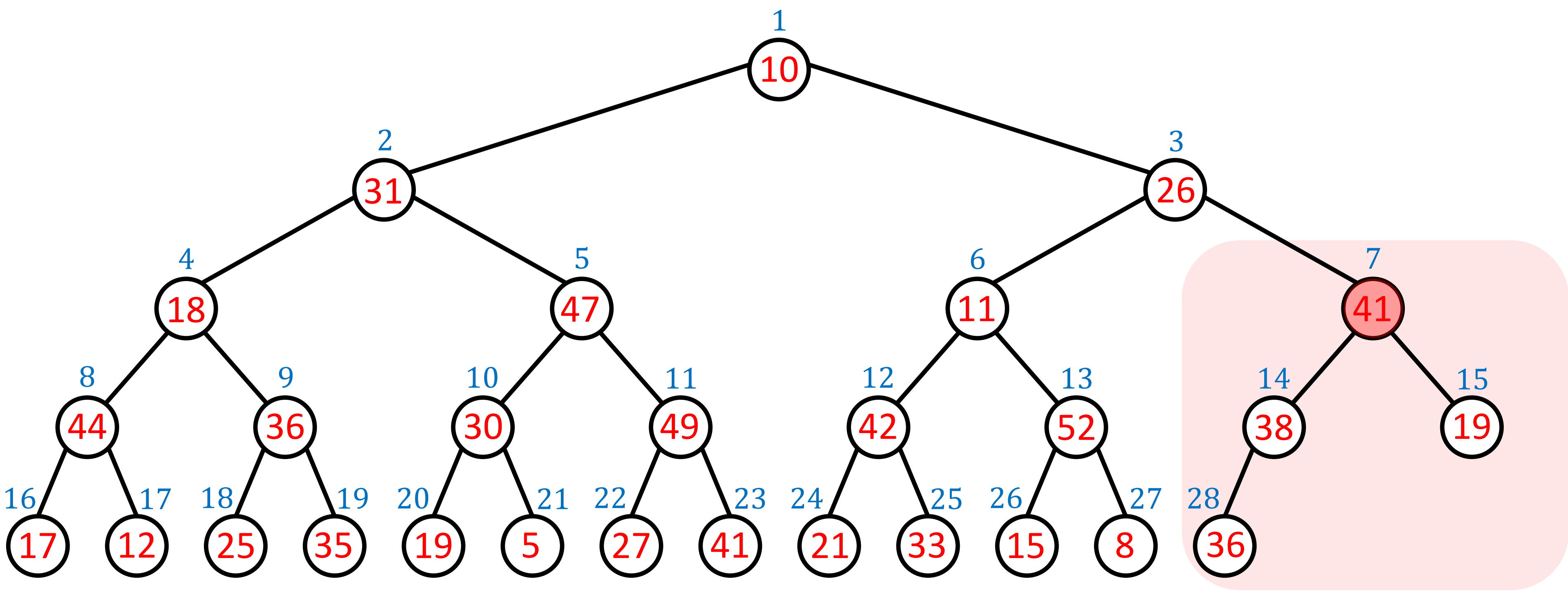
MAX-HEAPIFY(A, 8):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	44	36	30	49	42	52	38	19	17	12	25	35	19	5	27	41	21	33	15	8	36

Building a Max-Heap

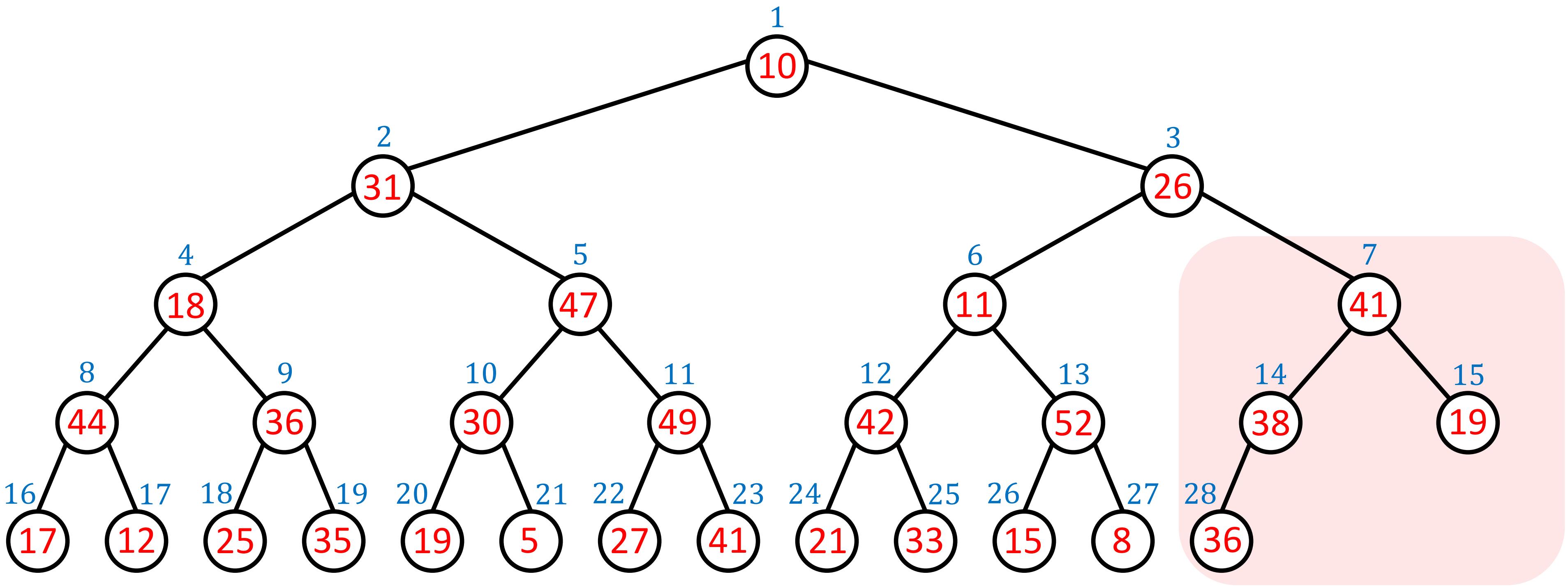
MAX-HEAPIFY(A, 7):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	44	36	30	49	42	52	38	19	17	12	25	35	19	5	27	41	21	33	15	8	36

Building a Max-Heap

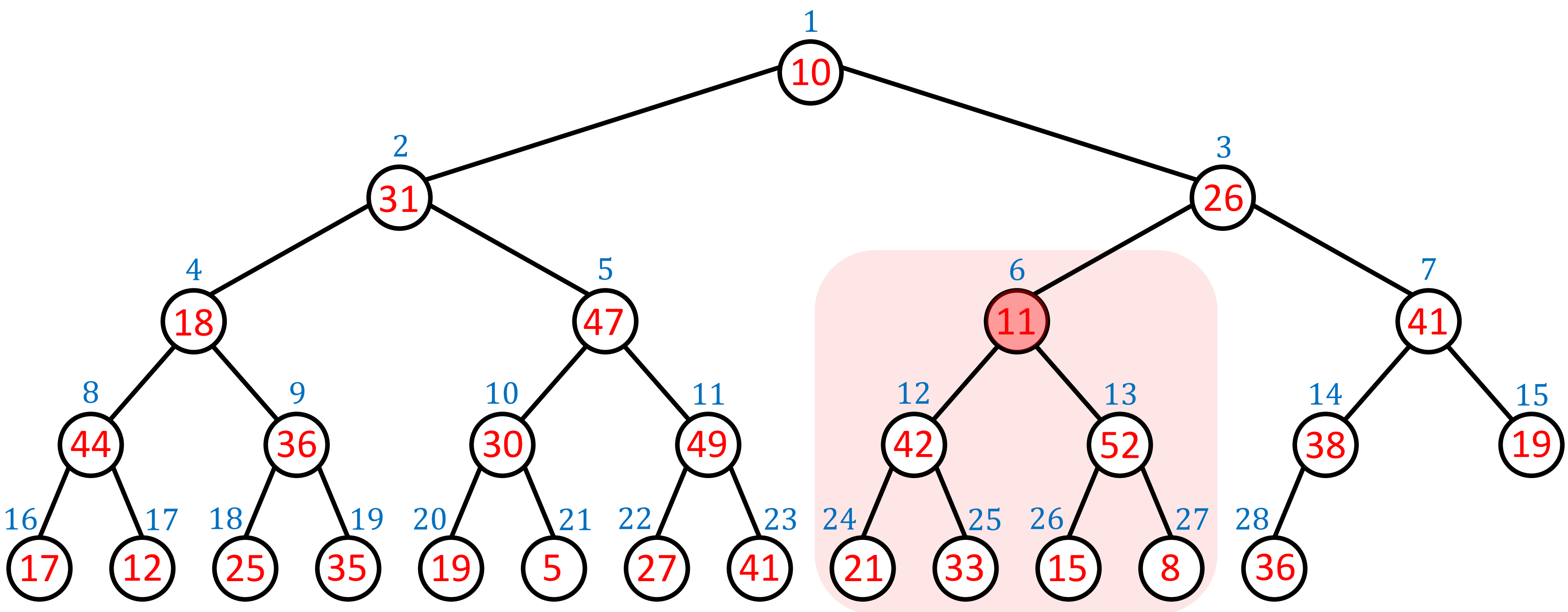
MAX-HEAPIFY(A, 7):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	44	36	30	49	42	52	38	19	17	12	25	35	19	5	27	41	21	33	15	8	36

Building a Max-Heap

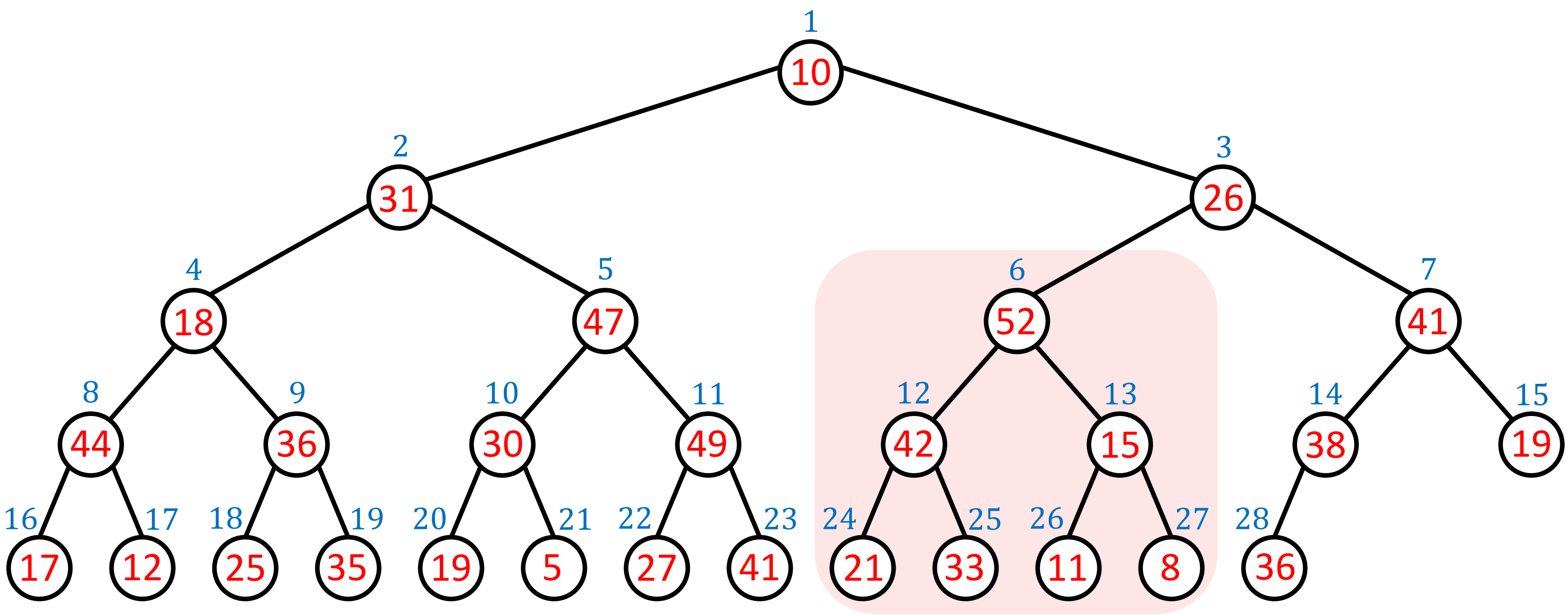
MAX-HEAPIFY(A, 6):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	44	36	30	49	42	52	38	19	17	12	25	35	19	5	27	41	21	33	15	8	36

Building a Max-Heap

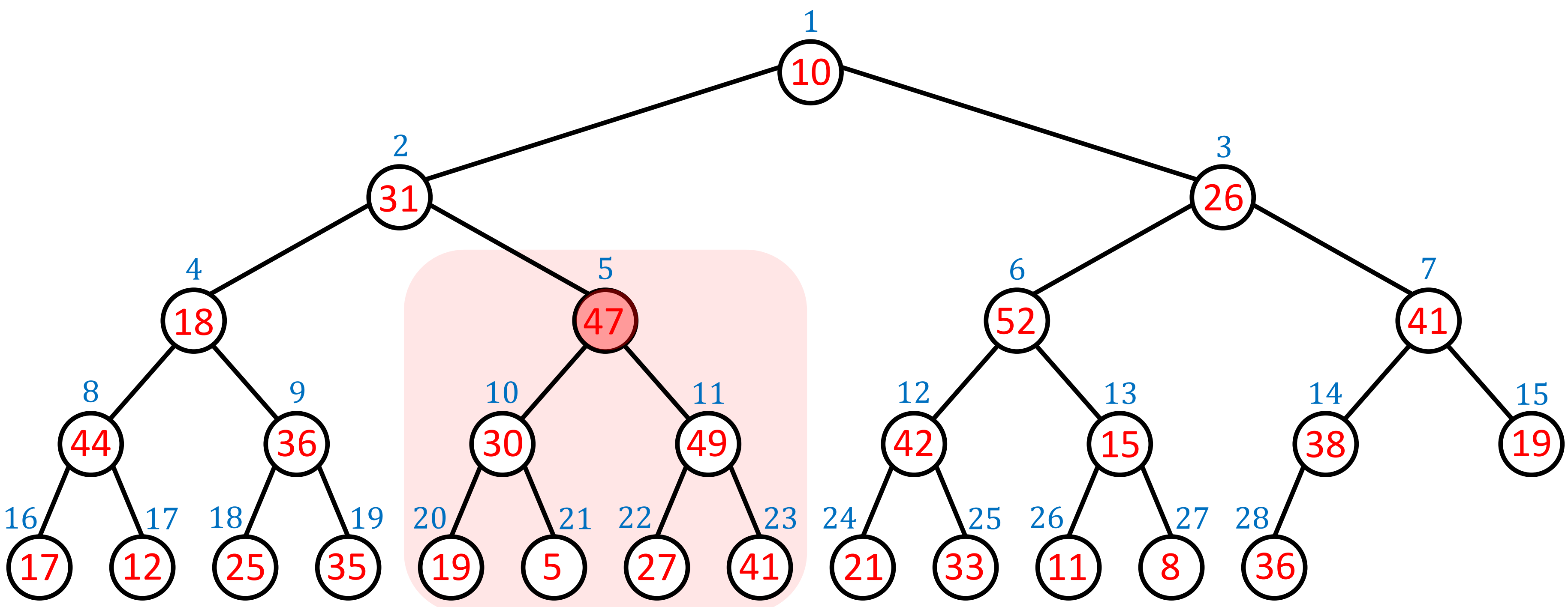
MAX-HEAPIFY(A, 6):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	52	41	44	36	30	49	42	15	38	19	17	12	25	35	19	5	27	41	21	33	11	8	36

Building a Max-Heap

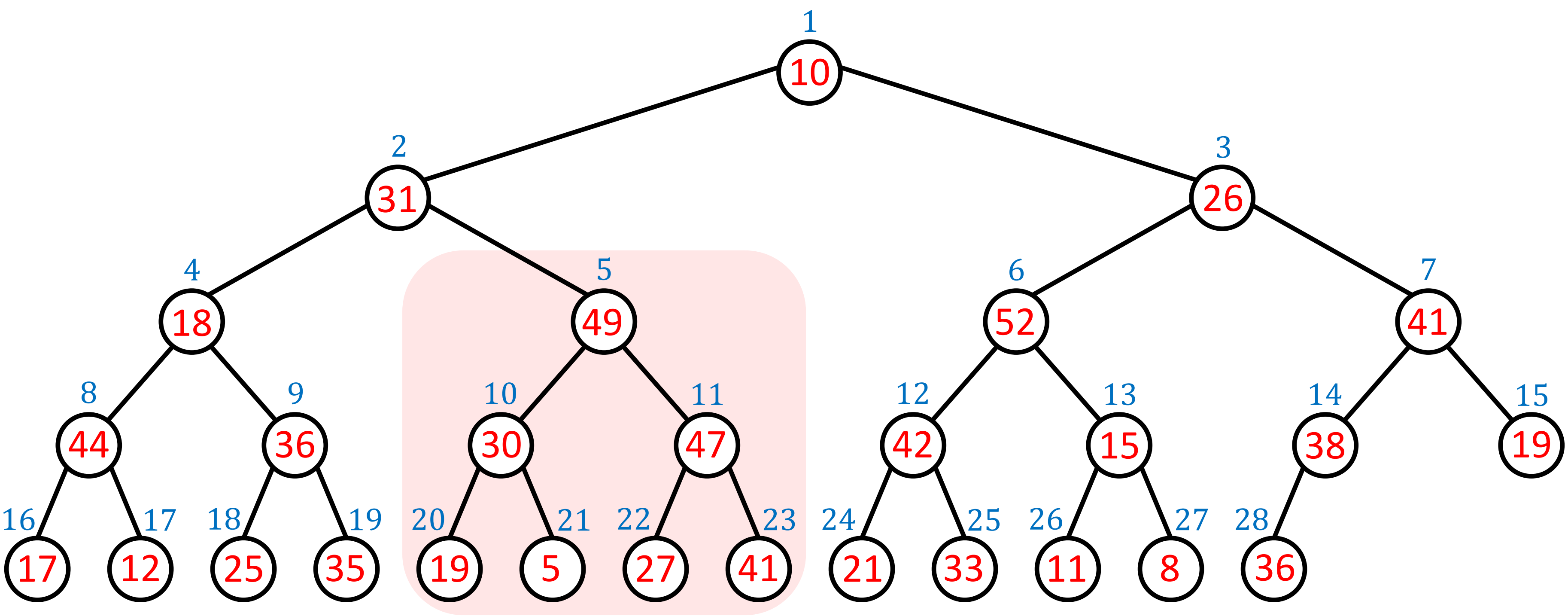
MAX-HEAPIFY(A, 5):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	52	41	44	36	30	49	42	15	38	19	17	12	25	35	19	5	27	41	21	33	11	8	36

Building a Max-Heap

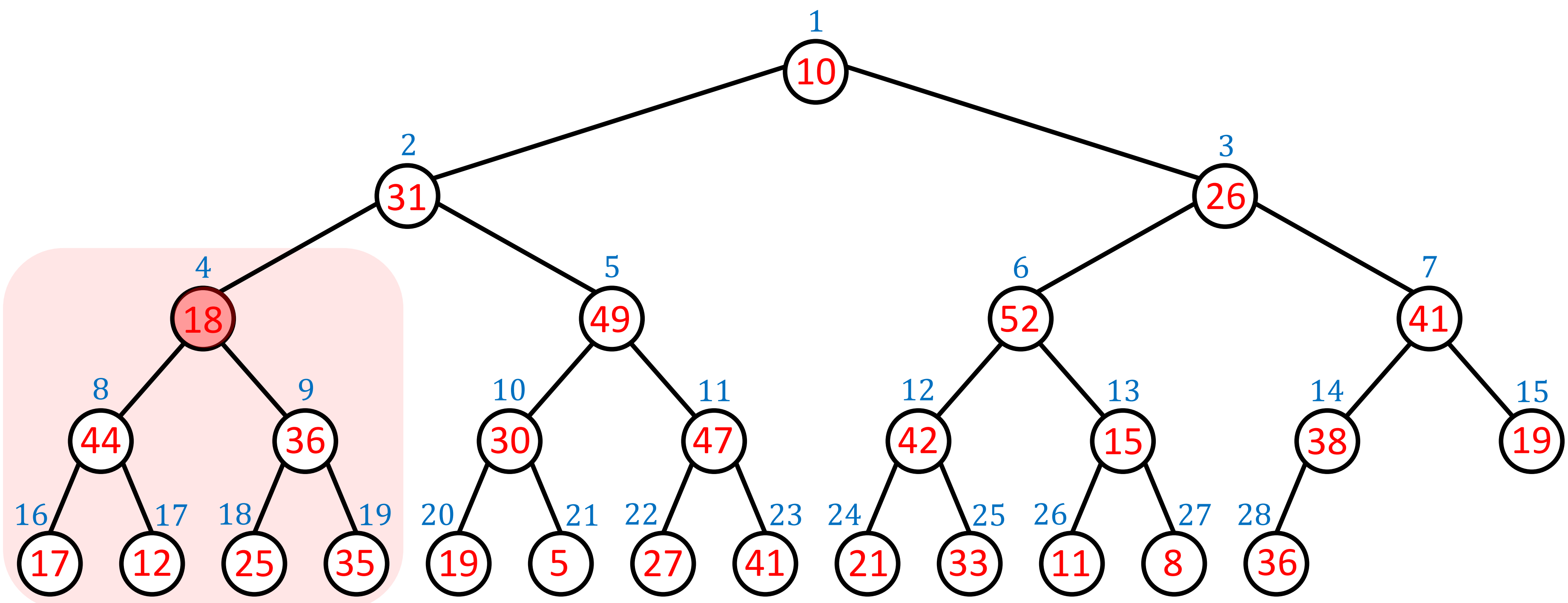
MAX-HEAPIFY(A, 5):



A	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
	10	31	26	18	49	52	41	44	36	30	47	42	15	38	19	17	12	25	35	19	5	27	41	21	33	11	8	36

Building a Max-Heap

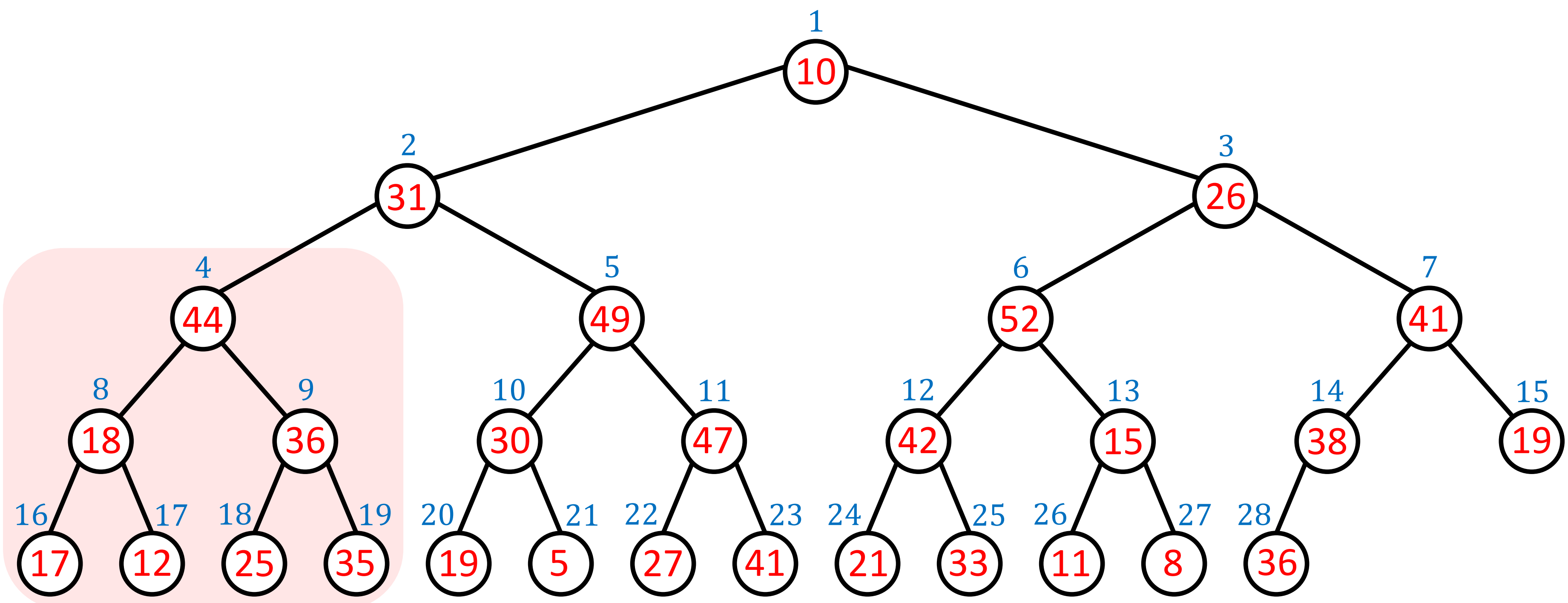
MAX-HEAPIFY(A, 4):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	49	52	41	44	36	30	47	42	15	38	19	17	12	25	35	19	5	27	41	21	33	11	8	36

Building a Max-Heap

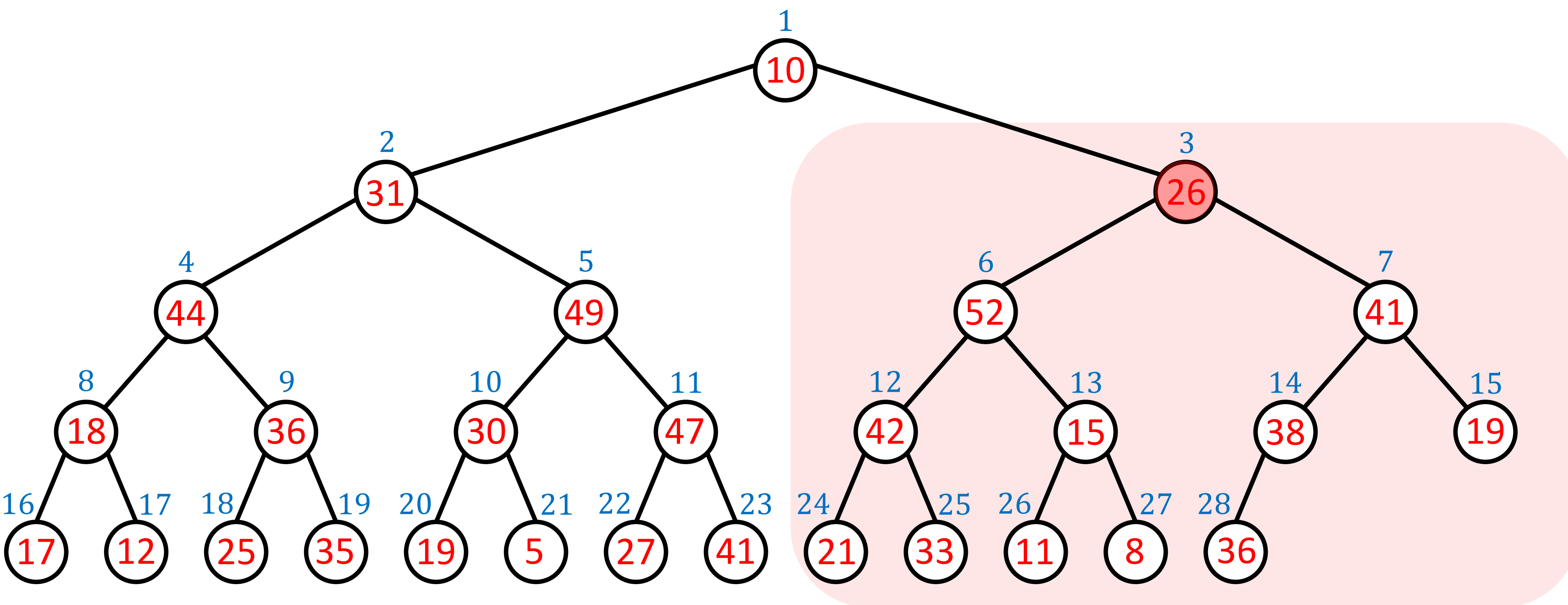
MAX-HEAPIFY(A, 4):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	44	49	52	41	18	36	30	47	42	15	38	19	17	12	25	35	19	5	27	41	21	33	11	8	36

Building a Max-Heap

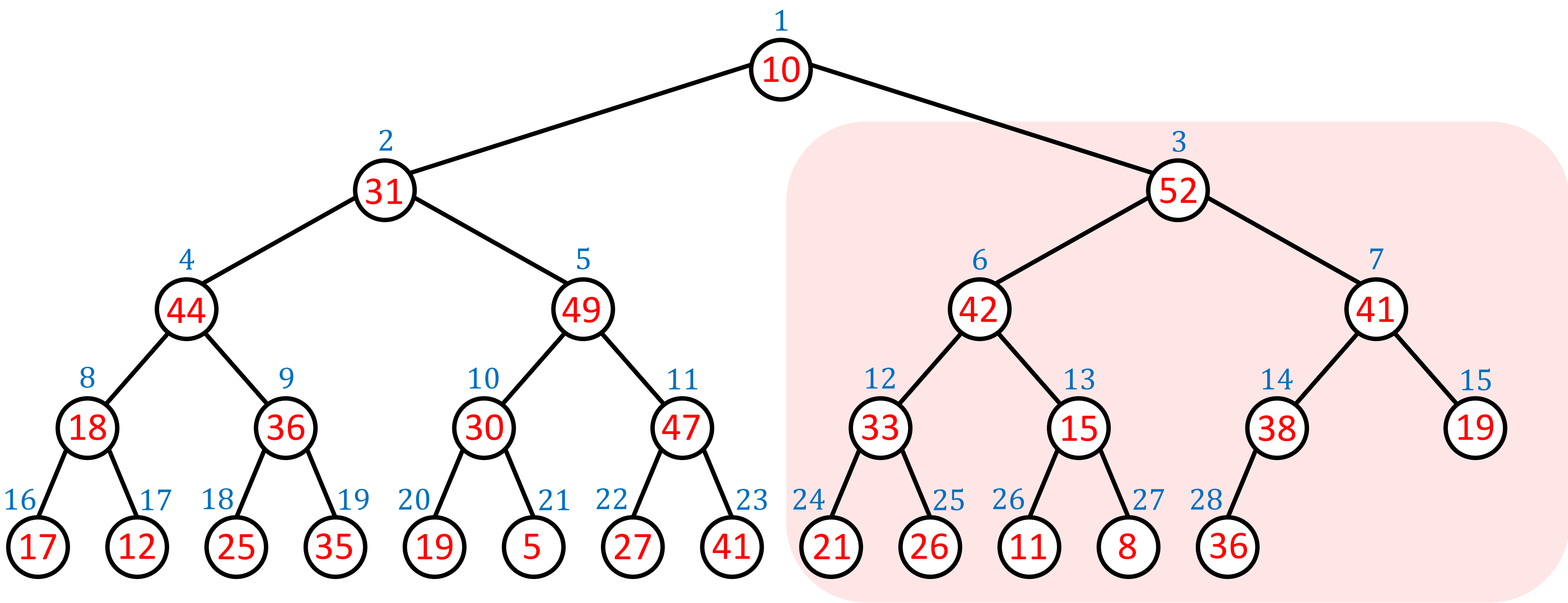
MAX-HEAPIFY(A, 3):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	44	49	52	41	18	36	30	47	42	15	38	19	17	12	25	35	19	5	27	41	21	33	11	8	36

Building a Max-Heap

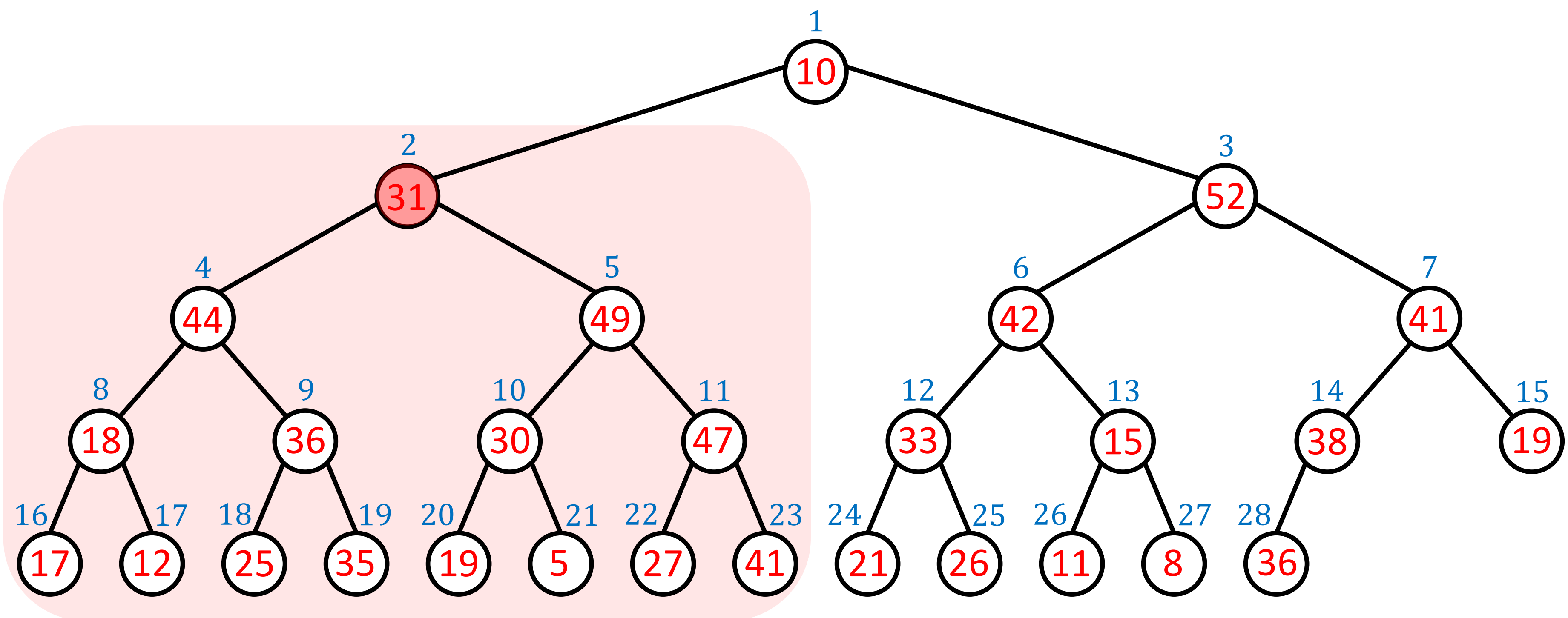
MAX-HEAPIFY(A, 3):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	52	44	49	42	41	18	36	30	47	33	15	38	19	17	12	25	35	19	5	27	41	21	26	11	8	36

Building a Max-Heap

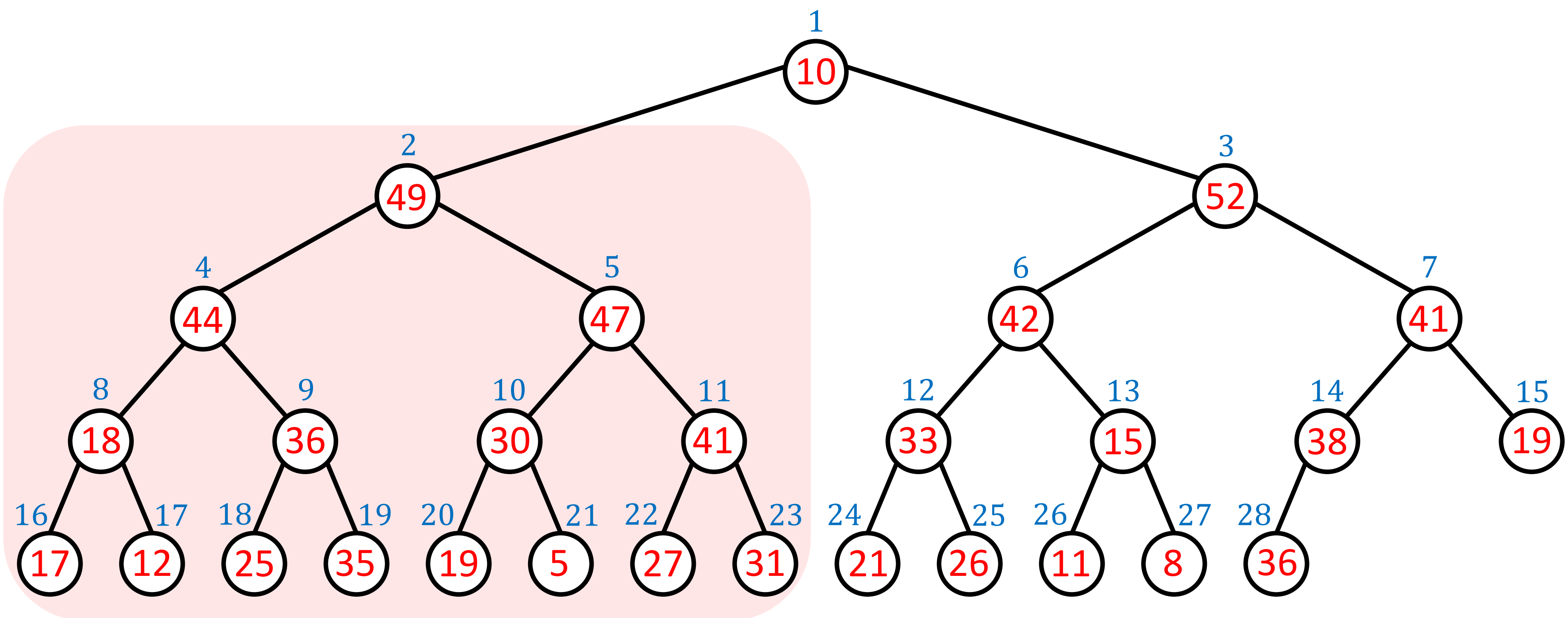
MAX-HEAPIFY(A, 2):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	52	44	49	42	41	18	36	30	47	33	15	38	19	17	12	25	35	19	5	27	41	21	26	11	8	36

Building a Max-Heap

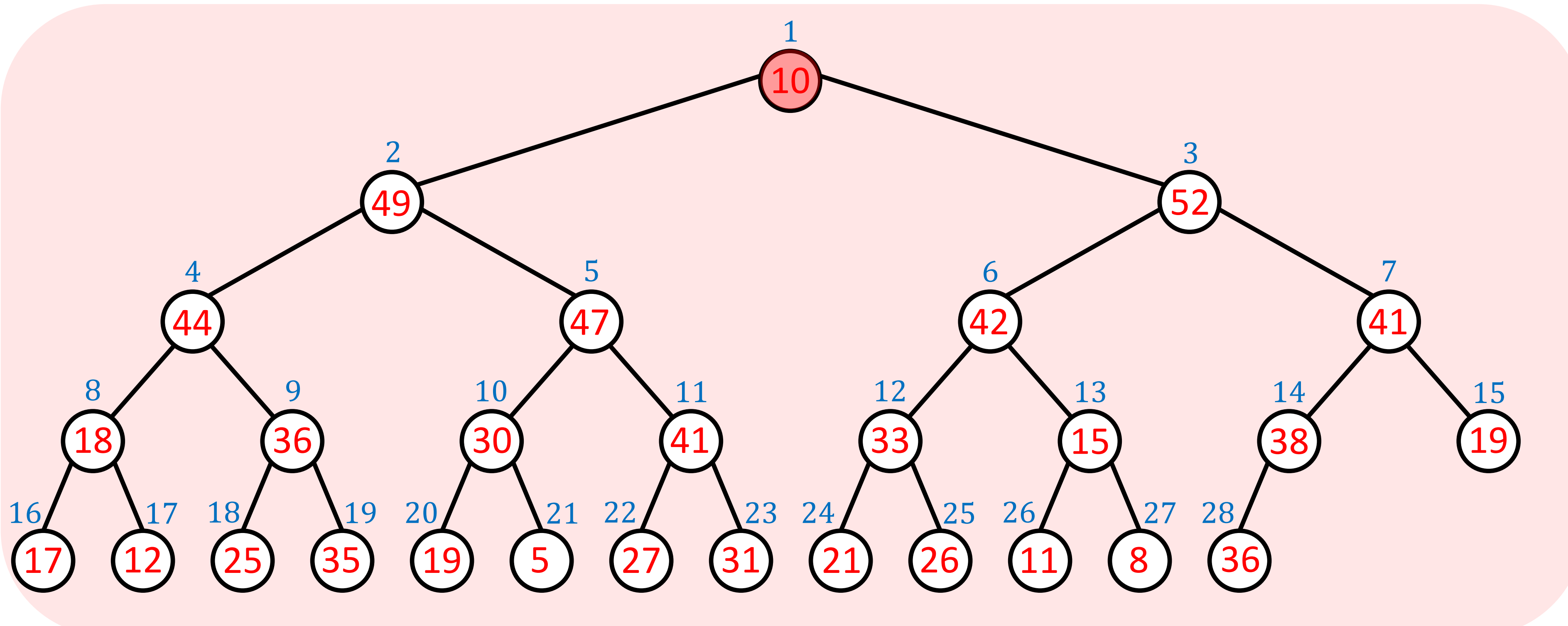
MAX-HEAPIFY(A, 2):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	49	52	44	47	42	41	18	36	30	41	33	15	38	19	17	12	25	35	19	5	27	31	21	26	11	8	36

Building a Max-Heap

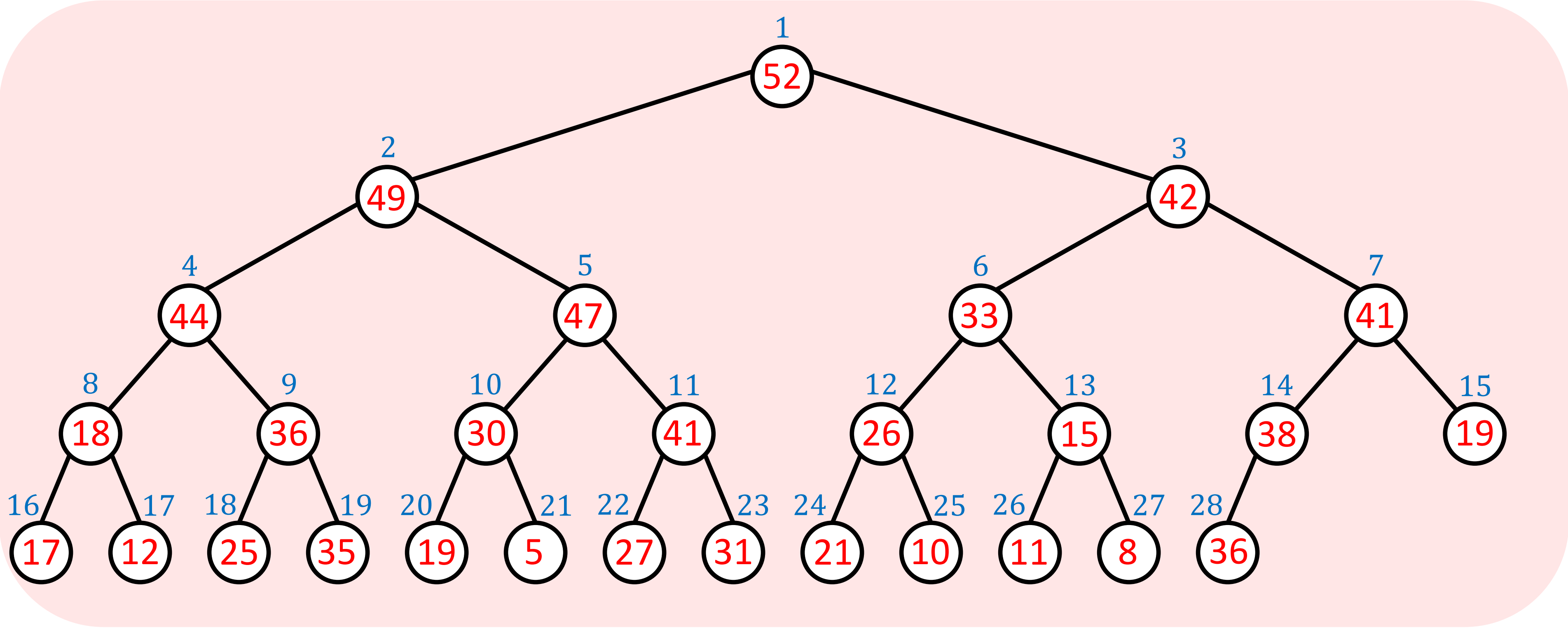
MAX-HEAPIFY(A, 1):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	49	52	44	47	42	41	18	36	30	41	33	15	38	19	17	12	25	35	19	5	27	31	21	26	11	8	36

Building a Max-Heap

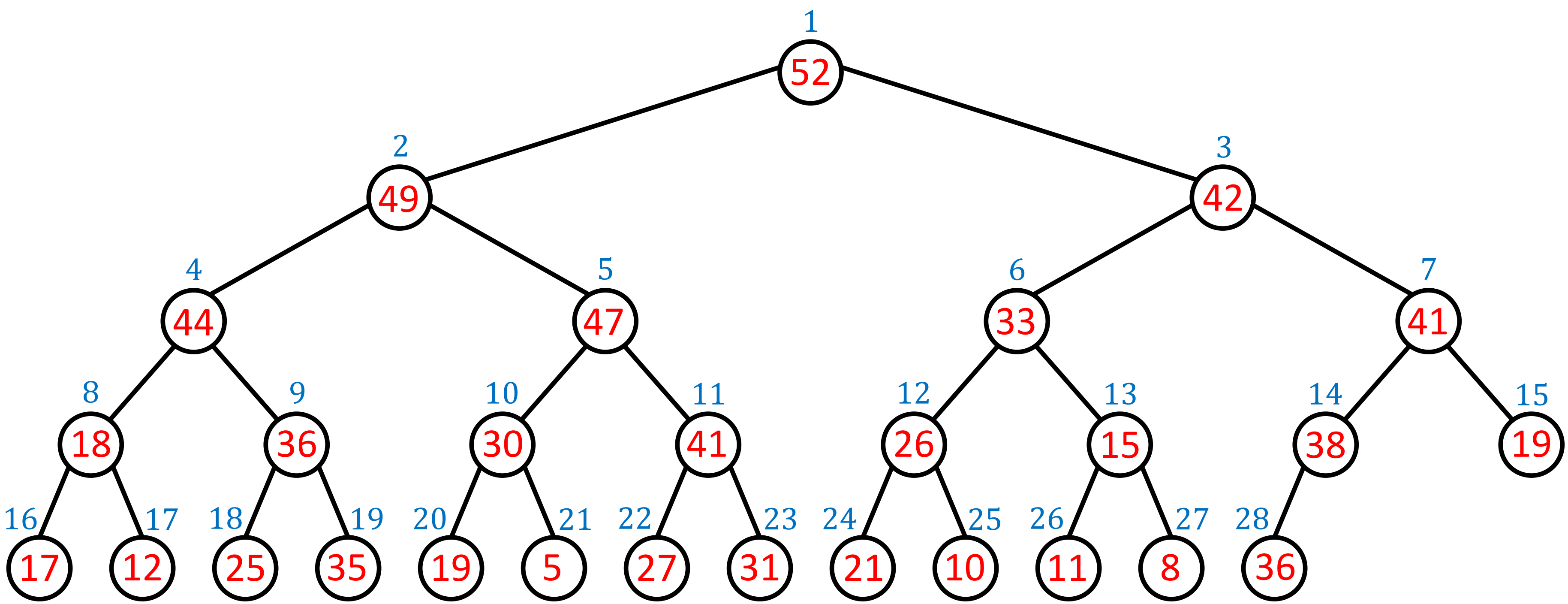
MAX-HEAPIFY(A, 1):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	36

Building a Max-Heap

The items in $A[1..28]$ are now correctly ordered to form a valid max-heap.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	36

Building a Max-Heap

Input: An array $A[1{:}n]$, where $n = A.length$.

Output: Array A with its elements rearranged so that the entire array is now a max-heap.

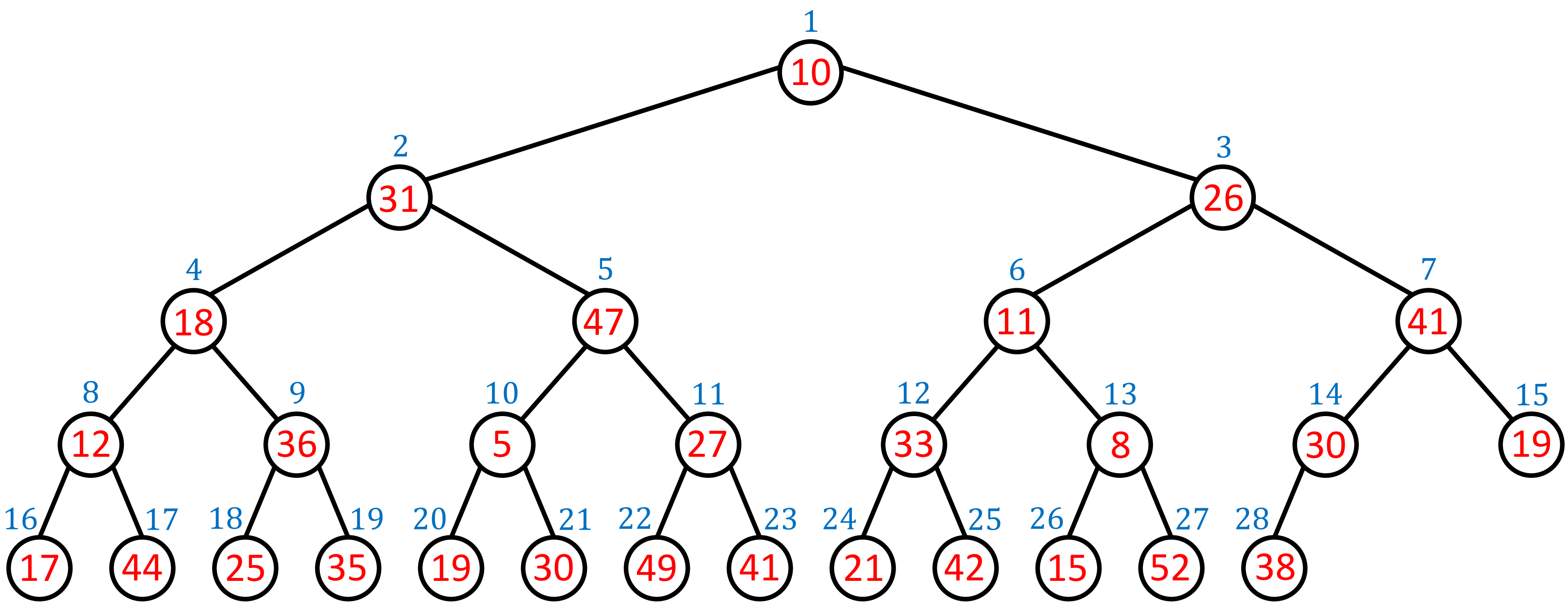
BUILD-MAX-HEAP (A)

1. $A.heapsize = A.length$
2. ***for*** $i = \lfloor A.length/2 \rfloor$ ***downto*** 1
3. MAX-HEAPIFY (A, i)

The Heapsort Algorithm

Suppose we are given an array $A[1..28]$ of $n = 28$ numbers.

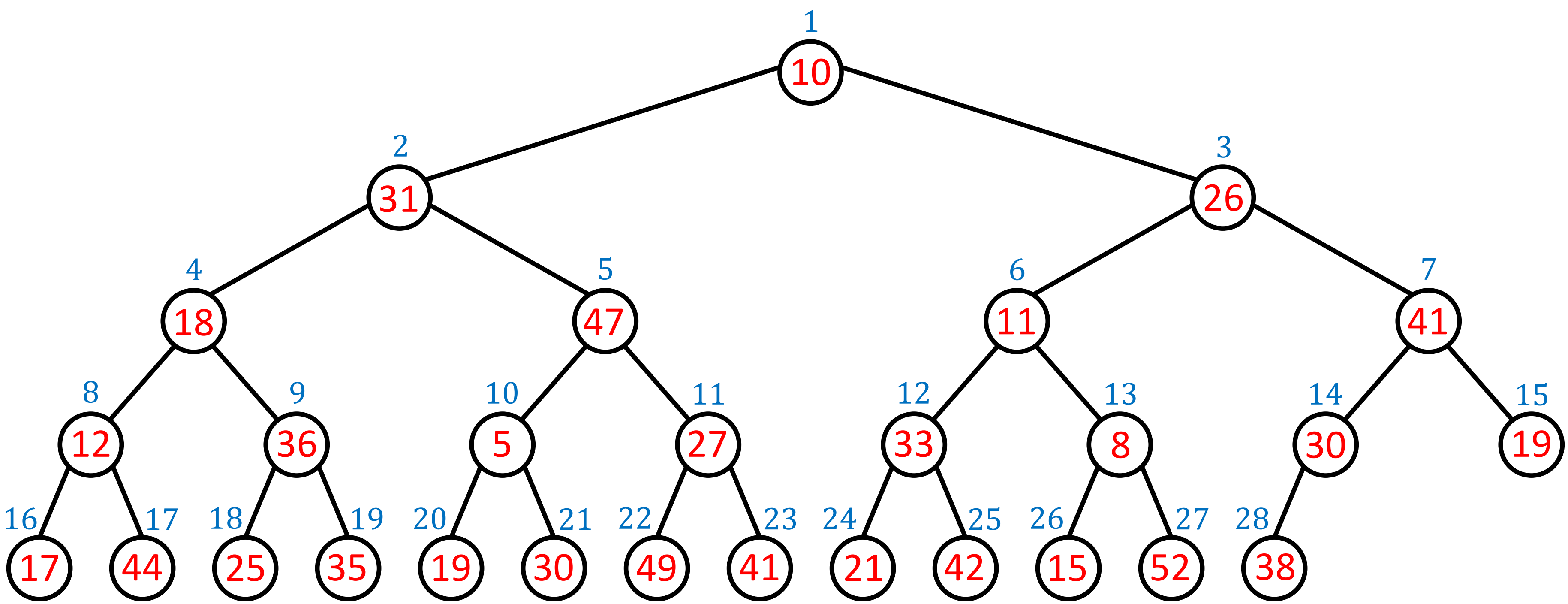
We want to sort $A[1..28]$ in nondecreasing order of value.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	5	27	33	8	30	19	17	44	25	35	19	30	49	41	21	42	15	52	38

The Heapsort Algorithm

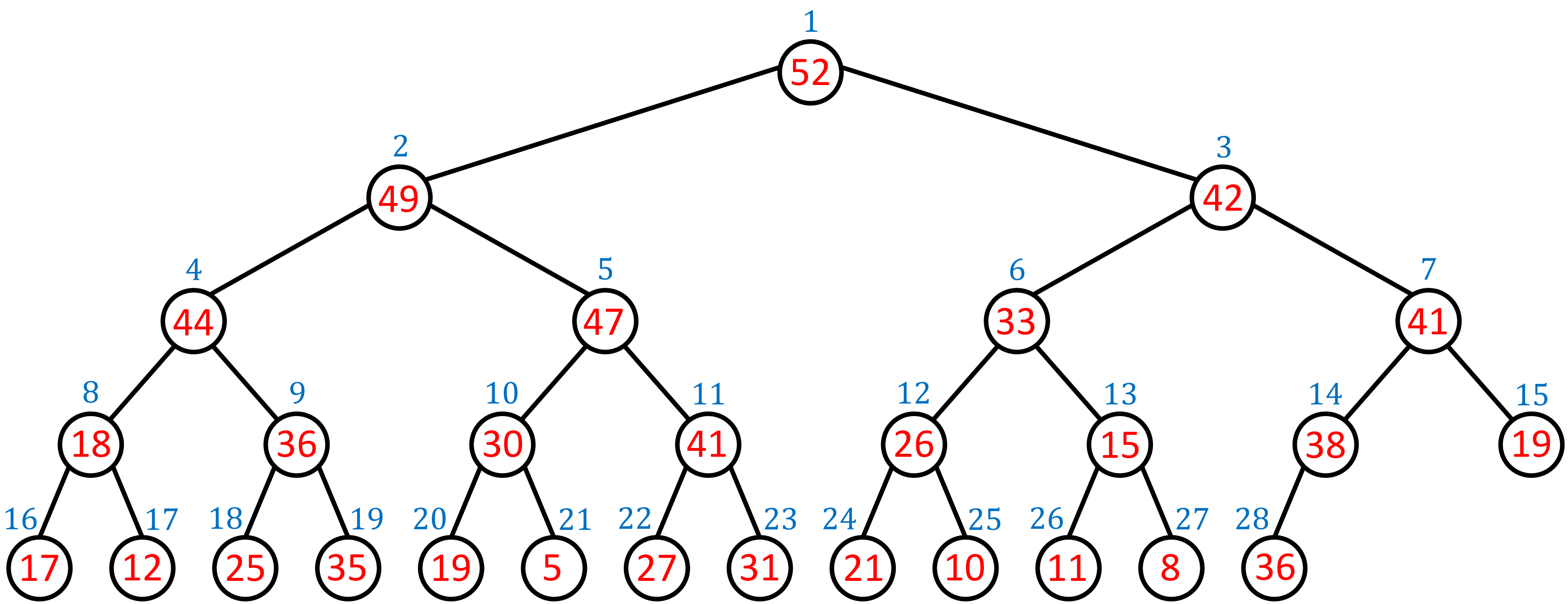
First, we make $A[1..28]$ a valid max-heap by calling BUILD-MAX-HEAP (A).



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	31	26	18	47	11	41	12	36	5	27	33	8	30	19	17	44	25	35	19	30	49	41	21	42	15	52	38

The Heapsort Algorithm

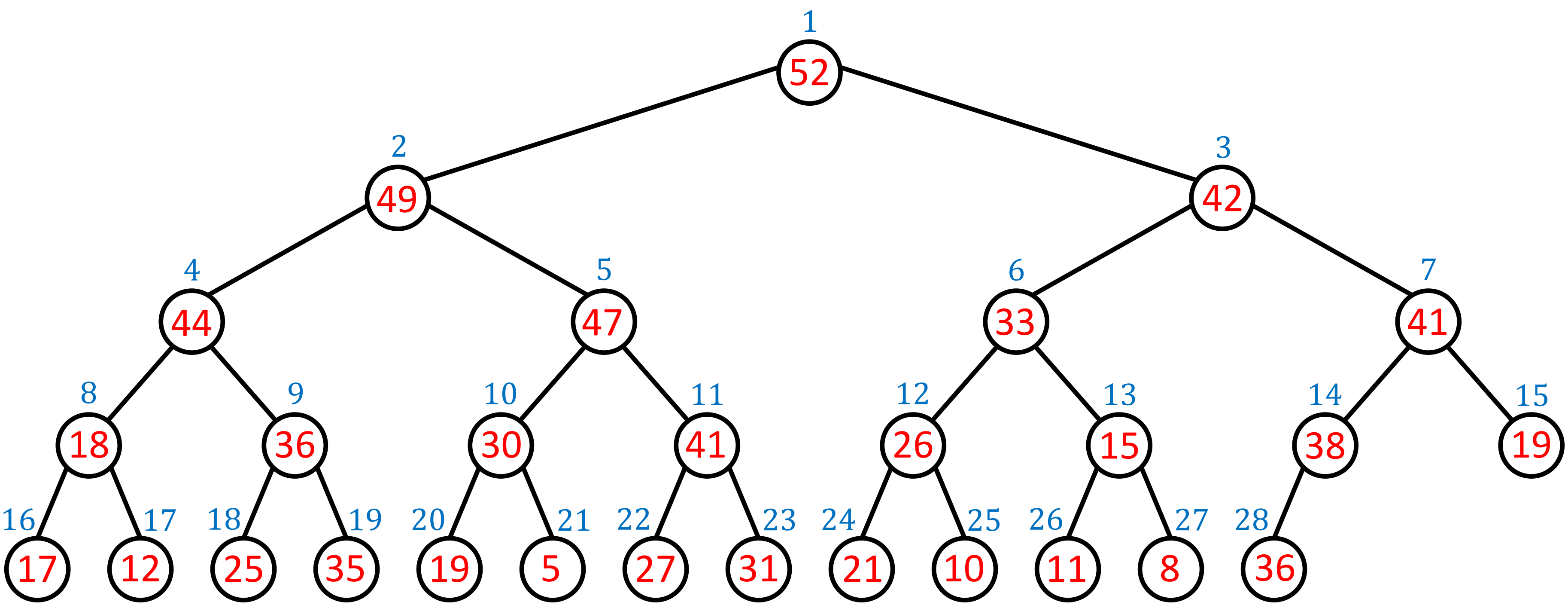
Here is what we get after BUILD-MAX-HEAP (A) finishes execution (A is now a valid max-heap):



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	36

The Heapsort Algorithm

$A[1] = 52$ is the largest number in the max-heap $A[1..28]$.

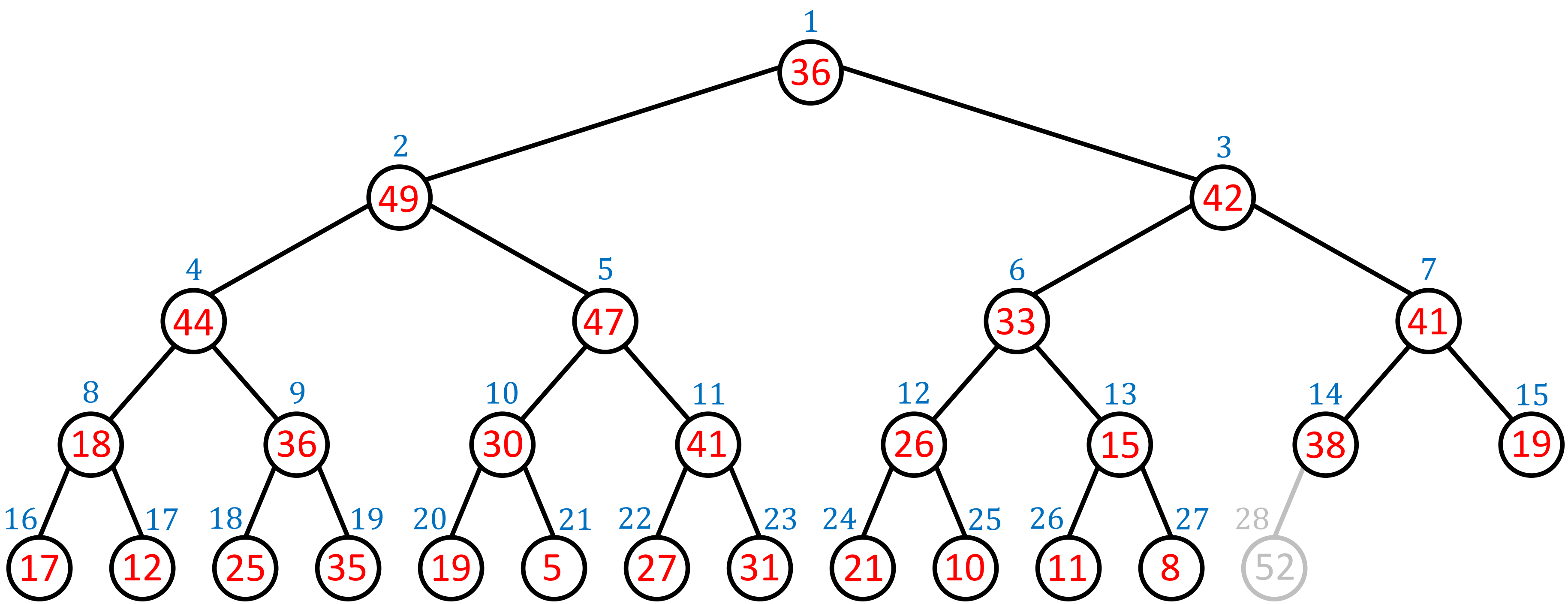


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	36

The Heapsort Algorithm

$A[1] = 52$ is the largest number in the max-heap $A[1..28]$.

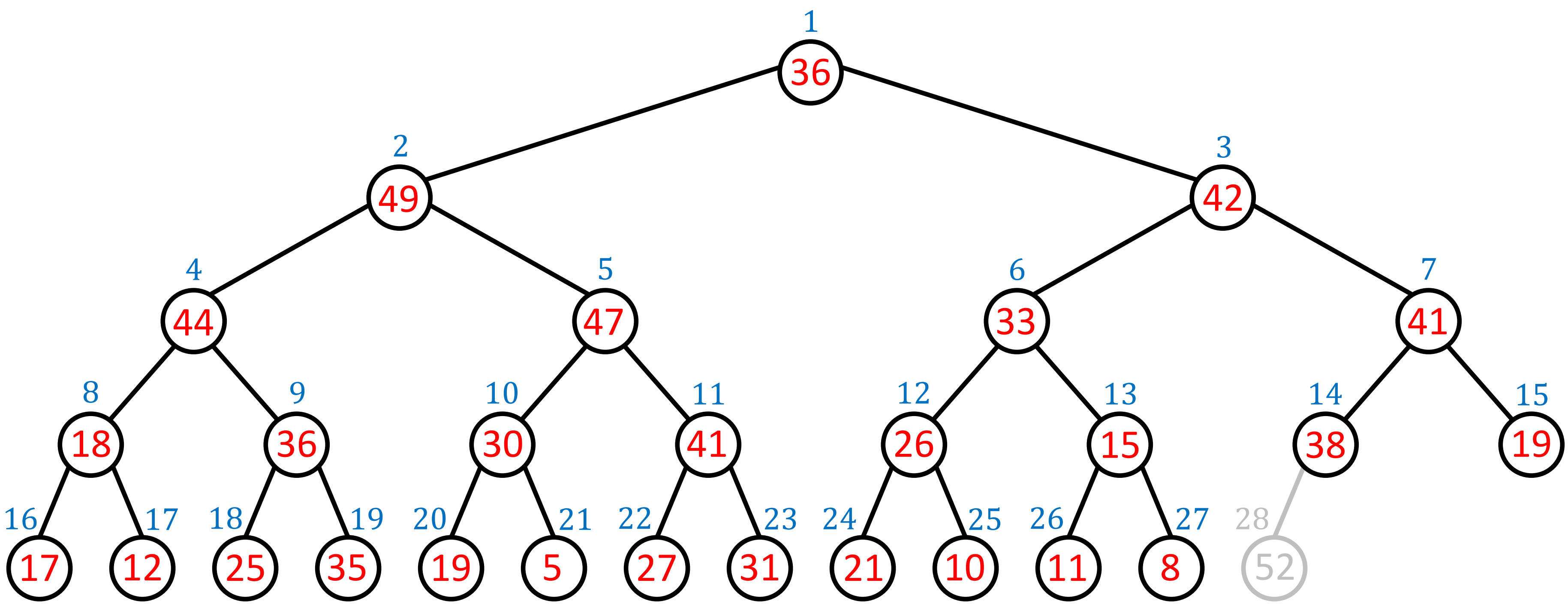
We swap $A[1]$ with $A[28]$ and treat $A[1..27]$ as the max-heap from now on.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	36	49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	52

The Heapsort Algorithm

But $A[1..27]$ is not a valid max-heap!

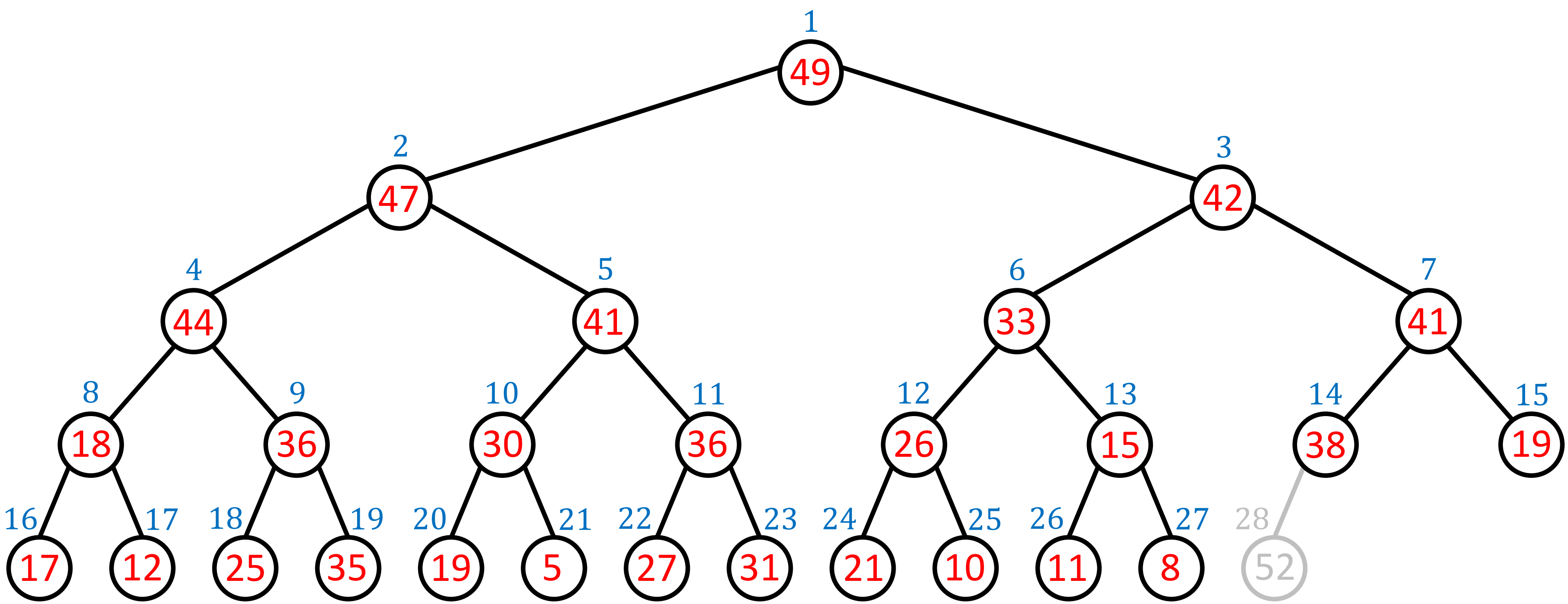


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	36	49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	52

The Heapsort Algorithm

But $A[1..27]$ is not a valid max-heap!

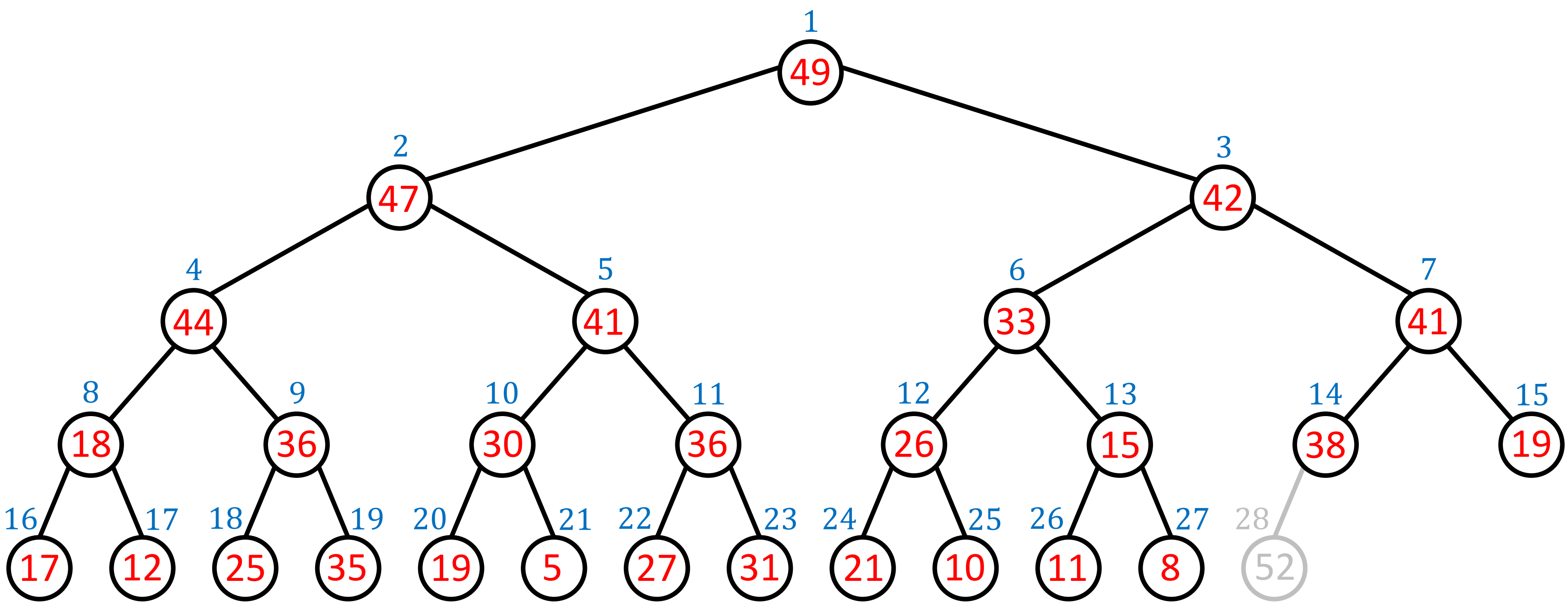
We make $A[1..27]$ a valid max-heap by calling MAX-HEAPIFY(A, 1).



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	49	47	42	44	41	33	41	18	36	30	36	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	52

The Heapsort Algorithm

$A[1] = 49$ is the largest number in the max-heap $A[1..27]$.

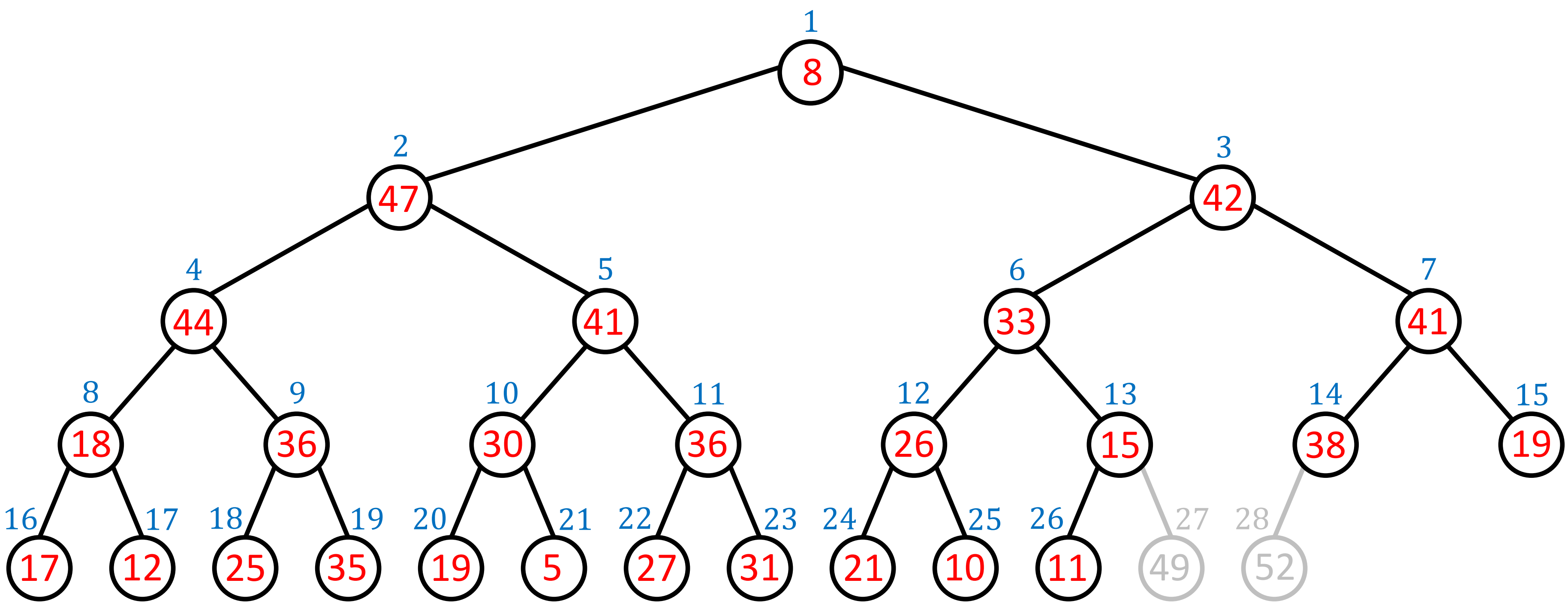


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	49	47	42	44	41	33	41	18	36	30	36	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	52

The Heapsort Algorithm

$A[1] = 49$ is the largest number in the max-heap $A[1..27]$.

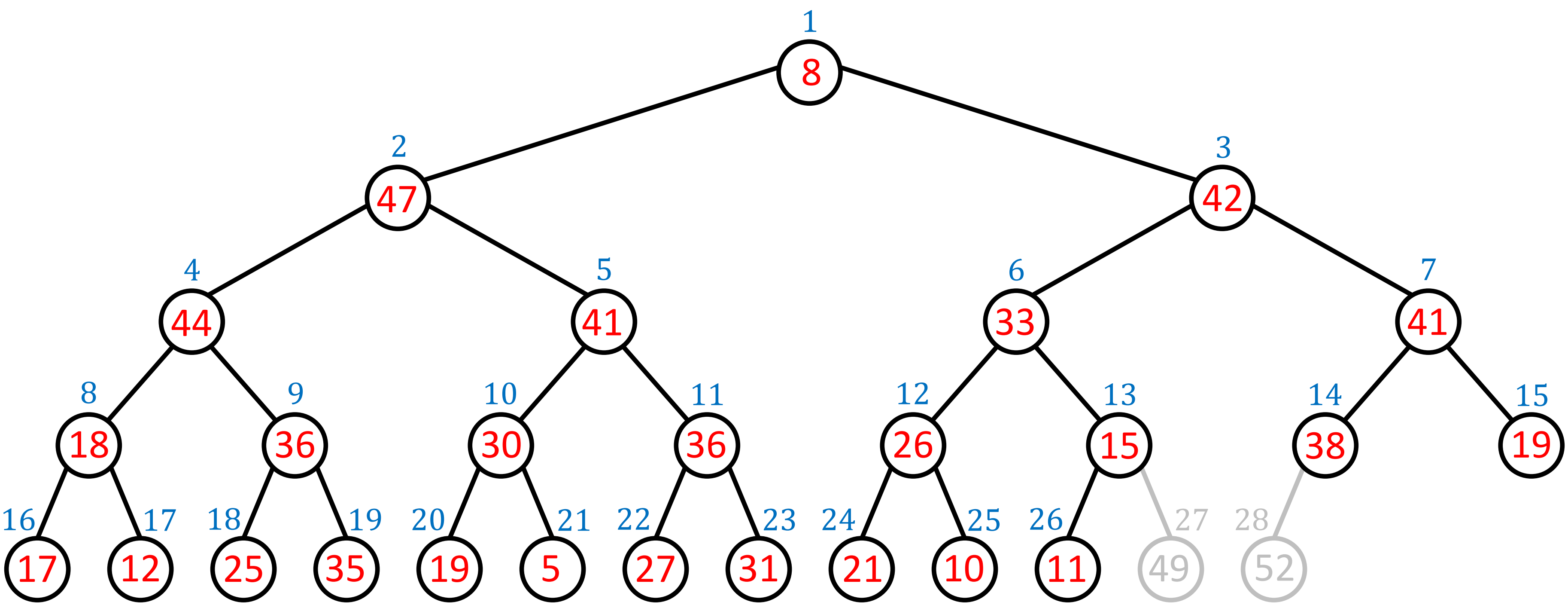
We swap $A[1]$ with $A[27]$ and treat $A[1..26]$ as the max-heap from now on.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	8	47	42	44	41	33	41	18	36	30	36	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	49	52

The Heapsort Algorithm

But $A[1..26]$ is not a valid max-heap!

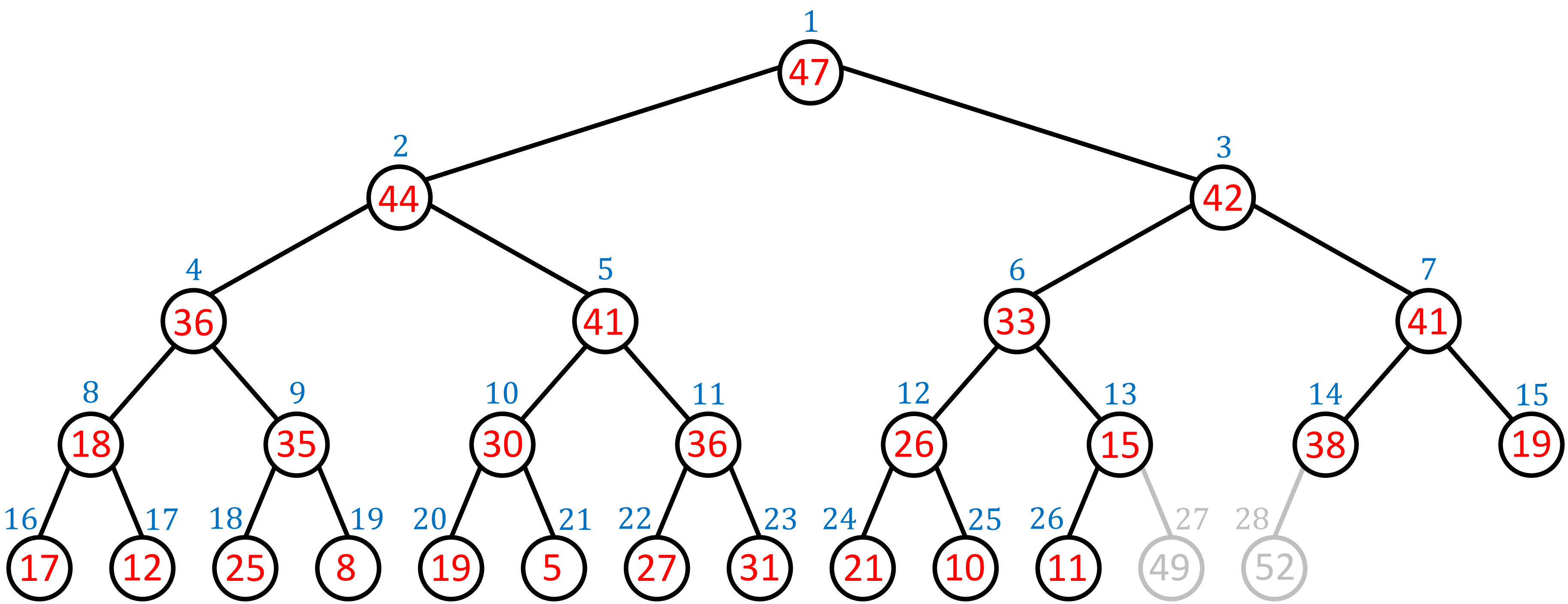


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	8	47	42	44	41	33	41	18	36	30	36	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	49	52

The Heapsort Algorithm

But $A[1..26]$ is not a valid max-heap!

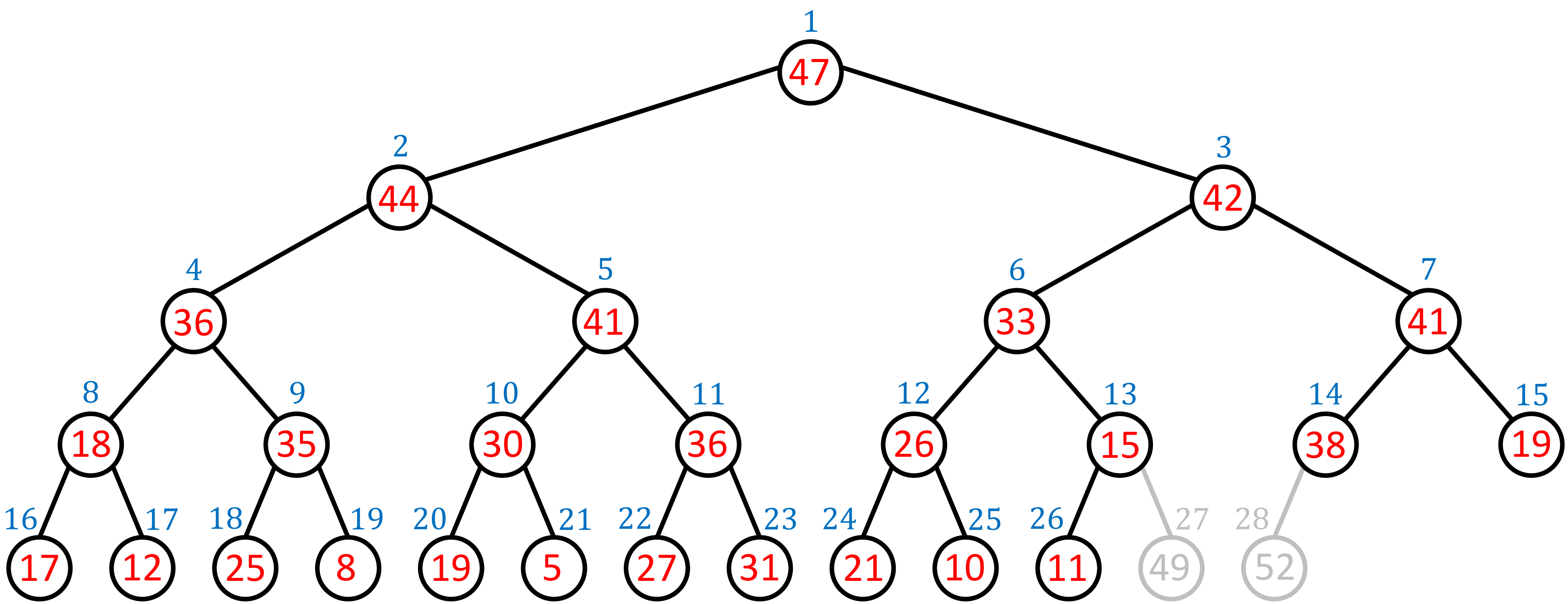
We make $A[1..26]$ a valid max-heap by calling MAX-HEAPIFY(A, 1).



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	47	44	42	36	41	33	41	18	35	30	36	26	15	38	19	17	12	25	8	19	5	27	31	21	10	11	49	52

The Heapsort Algorithm

$A[1] = 47$ is the largest number in the max-heap $A[1..26]$.

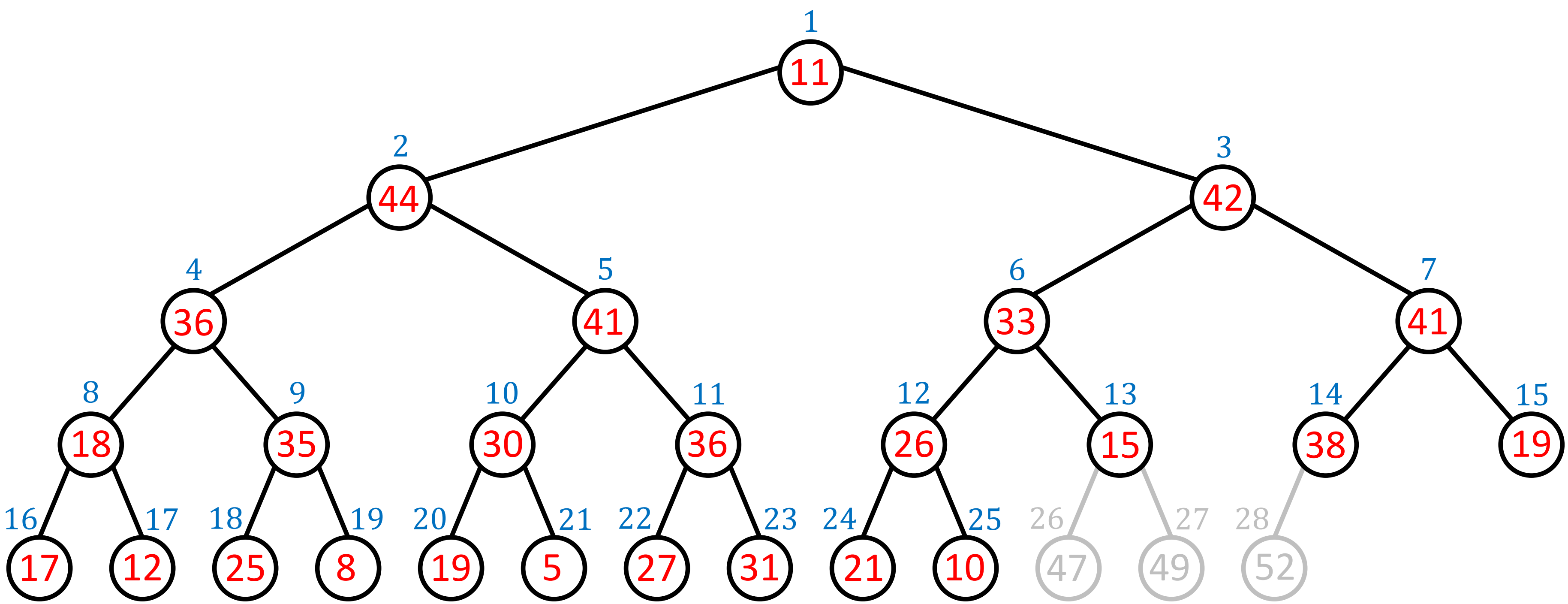


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	47	44	42	36	41	33	41	18	35	30	36	26	15	38	19	17	12	25	8	19	5	27	31	21	10	11	49	52

The Heapsort Algorithm

$A[1] = 47$ is the largest number in the max-heap $A[1..26]$.

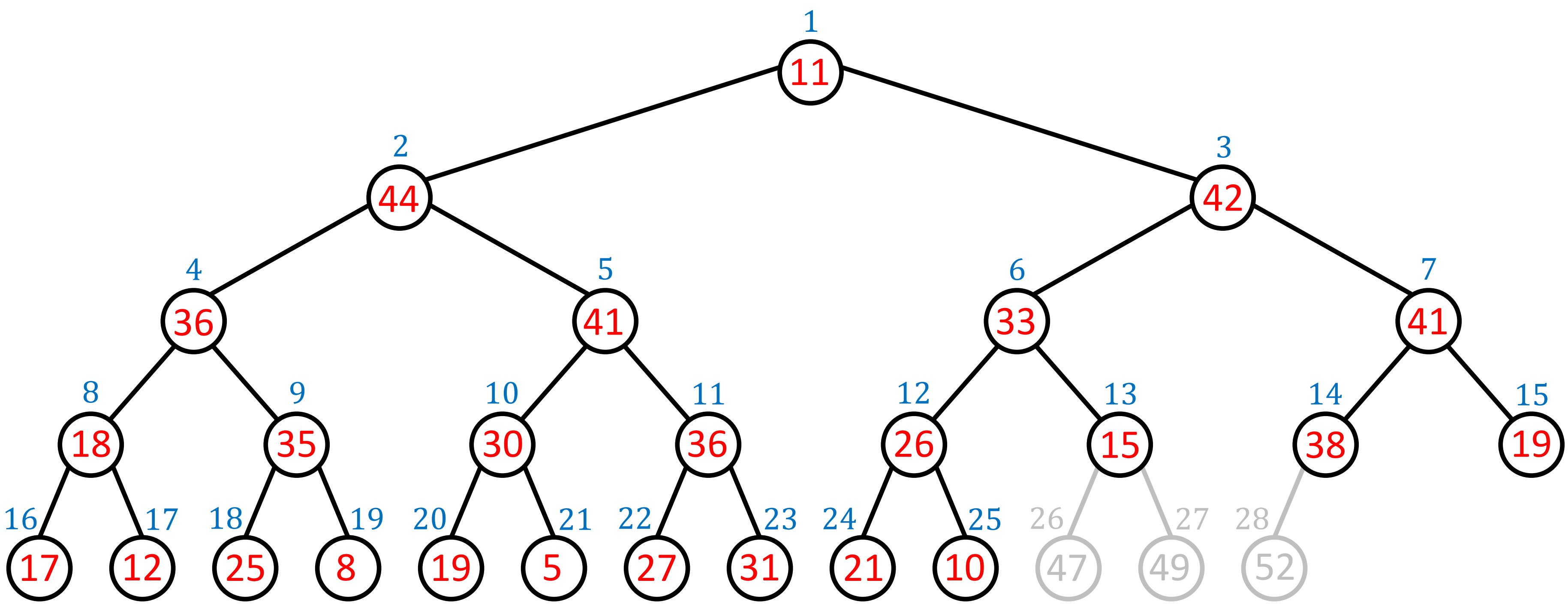
We swap $A[1]$ with $A[26]$ and treat $A[1..25]$ as the max-heap from now on.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	11	44	42	36	41	33	41	18	35	30	36	26	15	38	19	17	12	25	8	19	5	27	31	21	10	47	49	52

The Heapsort Algorithm

But $A[1..25]$ is not a valid max-heap!

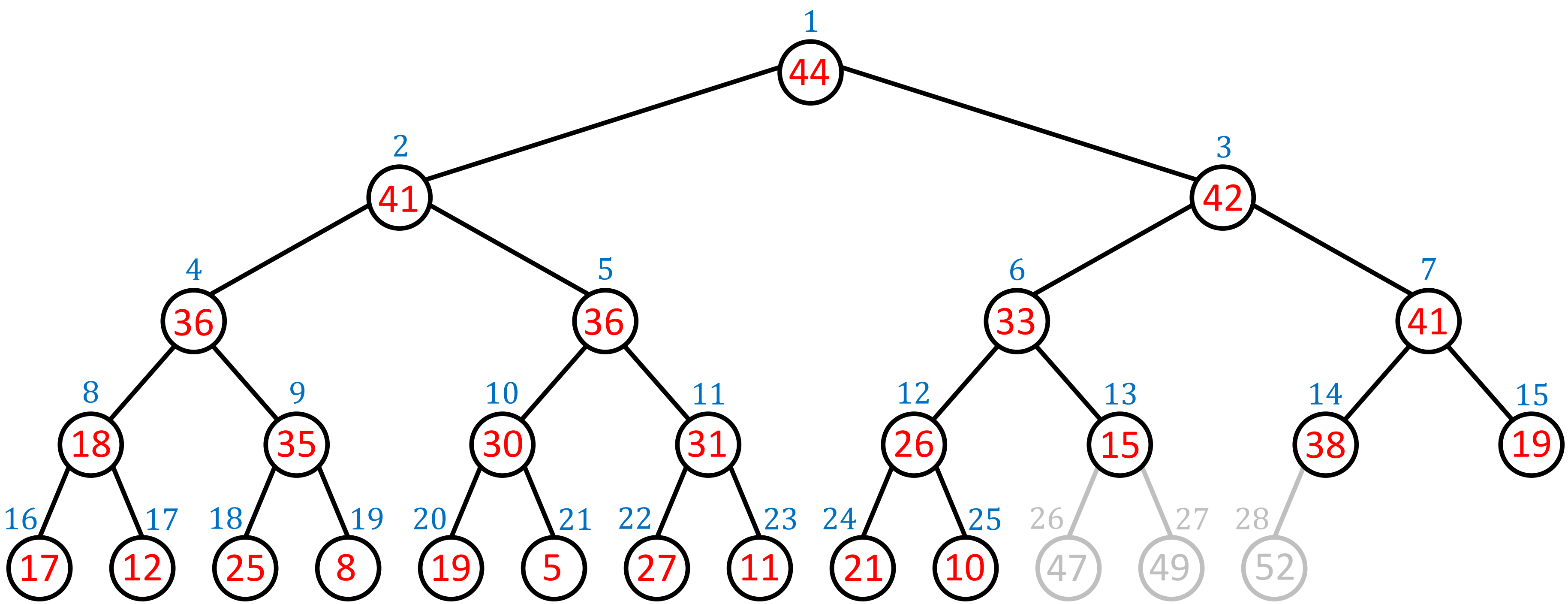


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	11	44	42	36	41	33	41	18	35	30	36	26	15	38	19	17	12	25	8	19	5	27	31	21	10	47	49	52

The Heapsort Algorithm

But $A[1..25]$ is not a valid max-heap!

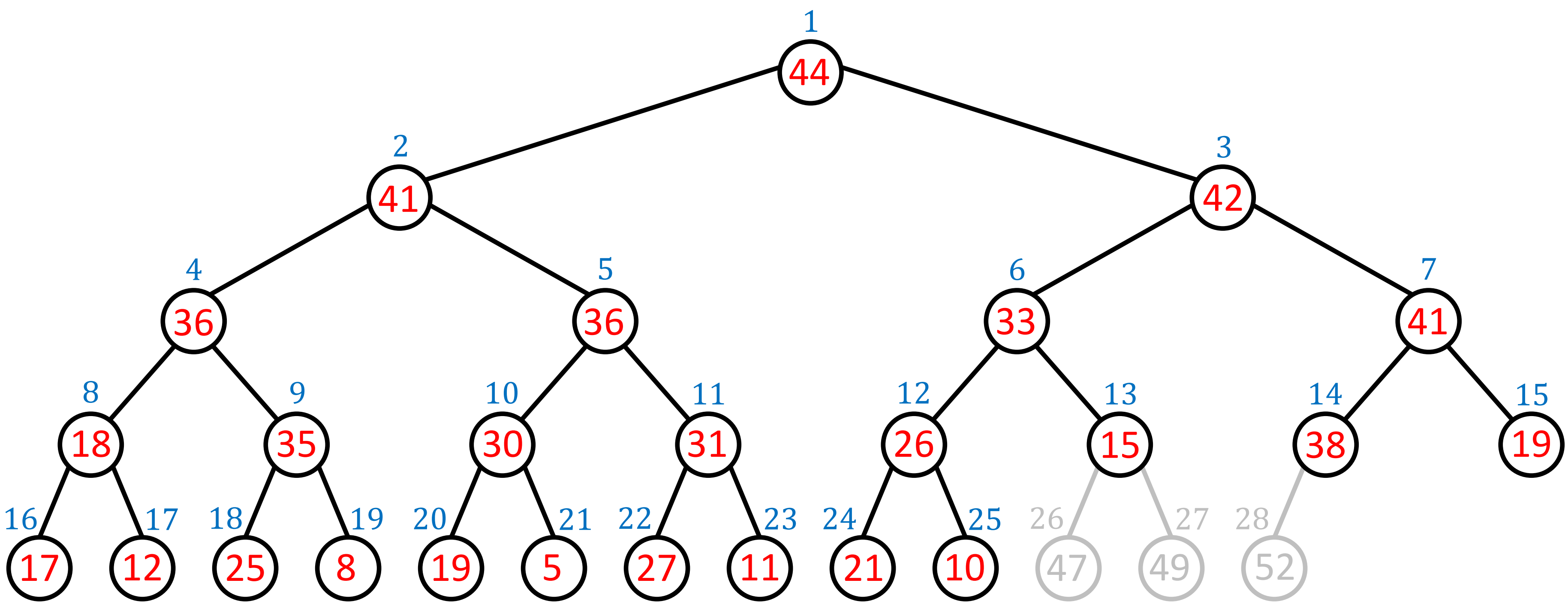
We make $A[1..25]$ a valid max-heap by calling MAX-HEAPIFY(A, 1).



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	44	41	42	36	36	33	41	18	35	30	31	26	15	38	19	17	12	25	8	19	5	27	11	21	10	47	49	52

The Heapsort Algorithm

$A[1] = 44$ is the largest number in the max-heap $A[1..25]$.

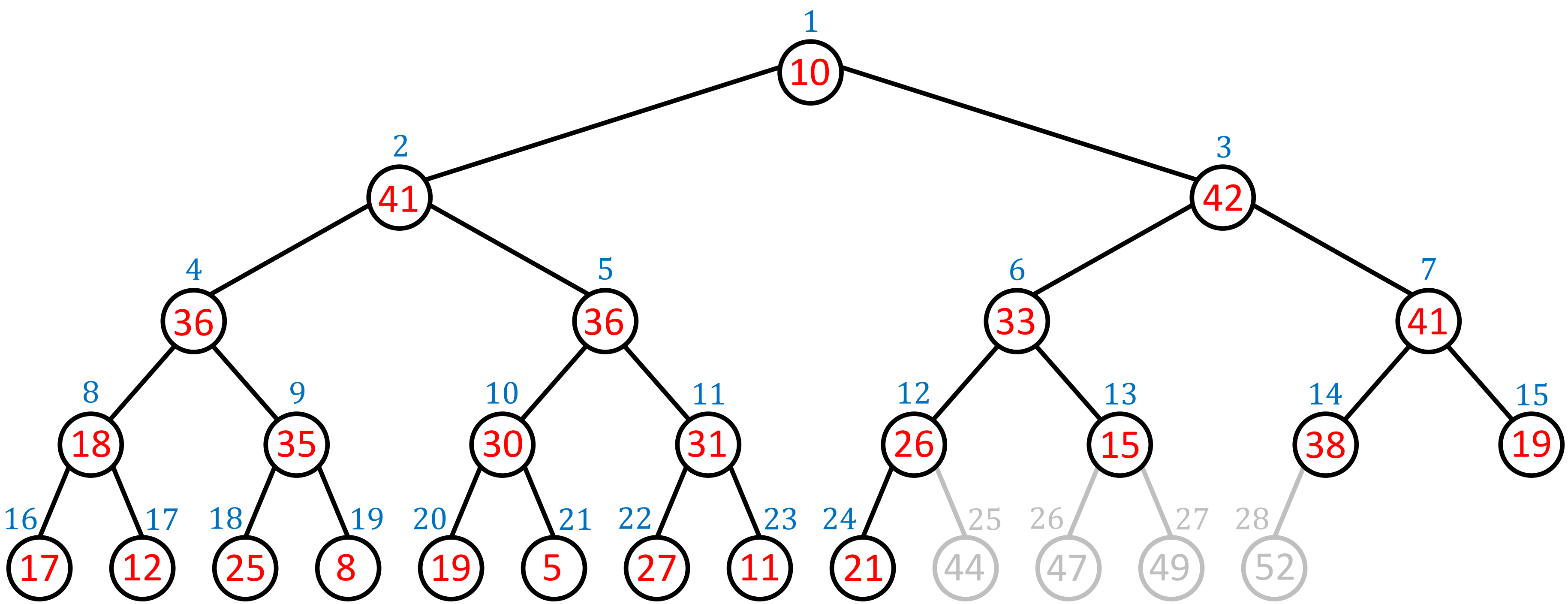


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	44	41	42	36	36	33	41	18	35	30	31	26	15	38	19	17	12	25	8	19	5	27	11	21	10	47	49	52

The Heapsort Algorithm

$A[1] = 44$ is the largest number in the max-heap $A[1..25]$.

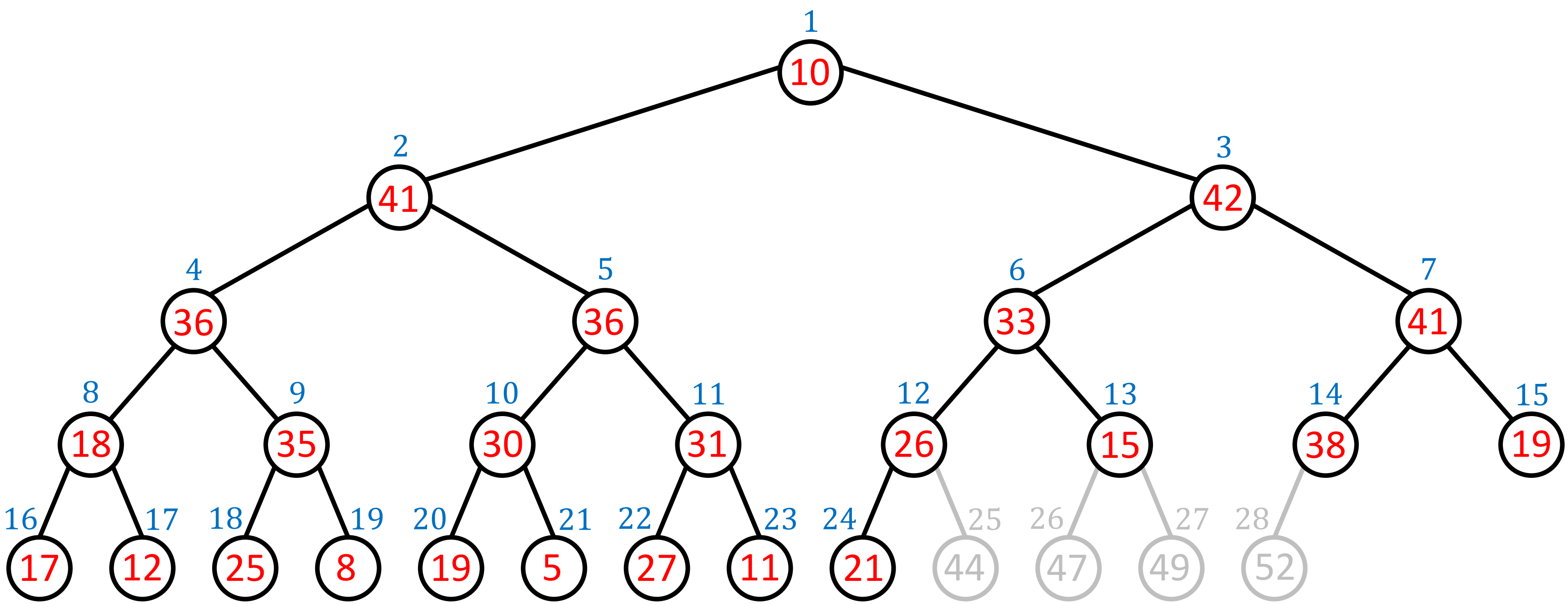
We swap $A[1]$ with $A[25]$ and treat $A[1..24]$ as the max-heap from now on.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	41	42	36	36	33	41	18	35	30	31	26	15	38	19	17	12	25	8	19	5	27	11	21	44	47	49	52

The Heapsort Algorithm

But $A[1..24]$ is not a valid max-heap!

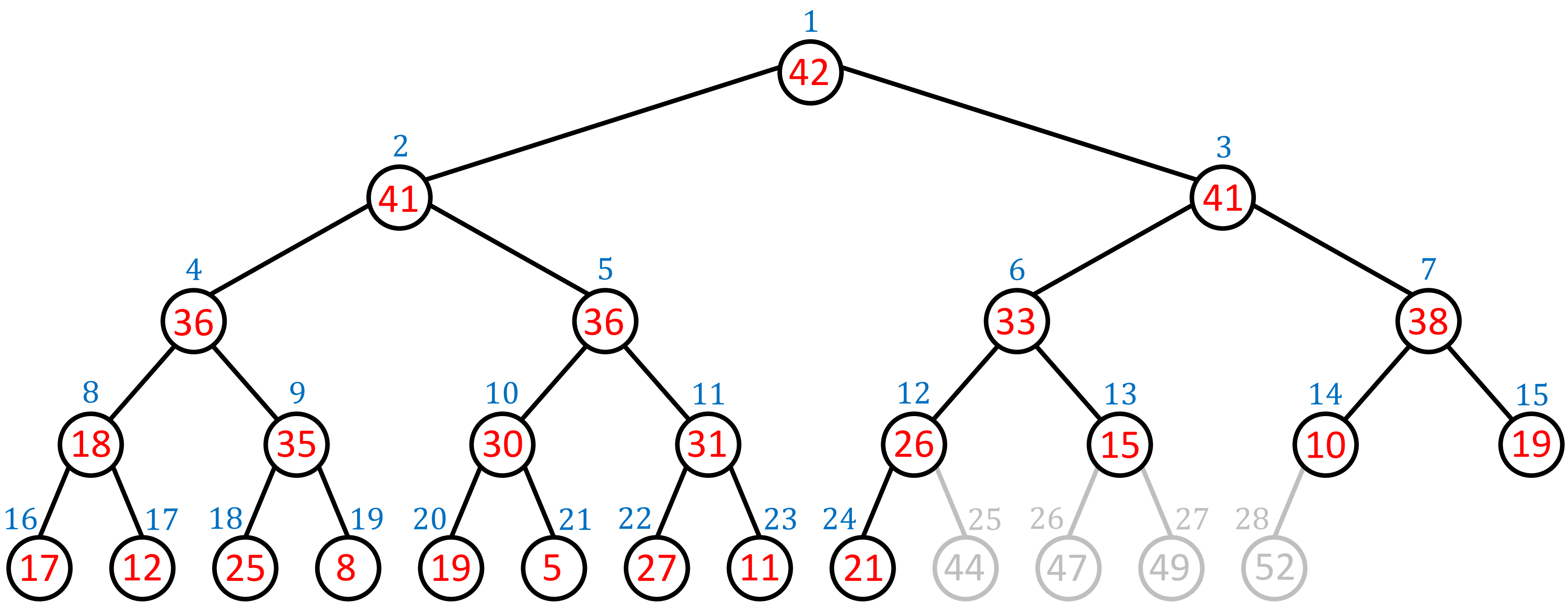


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	10	41	42	36	36	33	41	18	35	30	31	26	15	38	19	17	12	25	8	19	5	27	11	21	44	47	49	52

The Heapsort Algorithm

But $A[1..24]$ is not a valid max-heap!

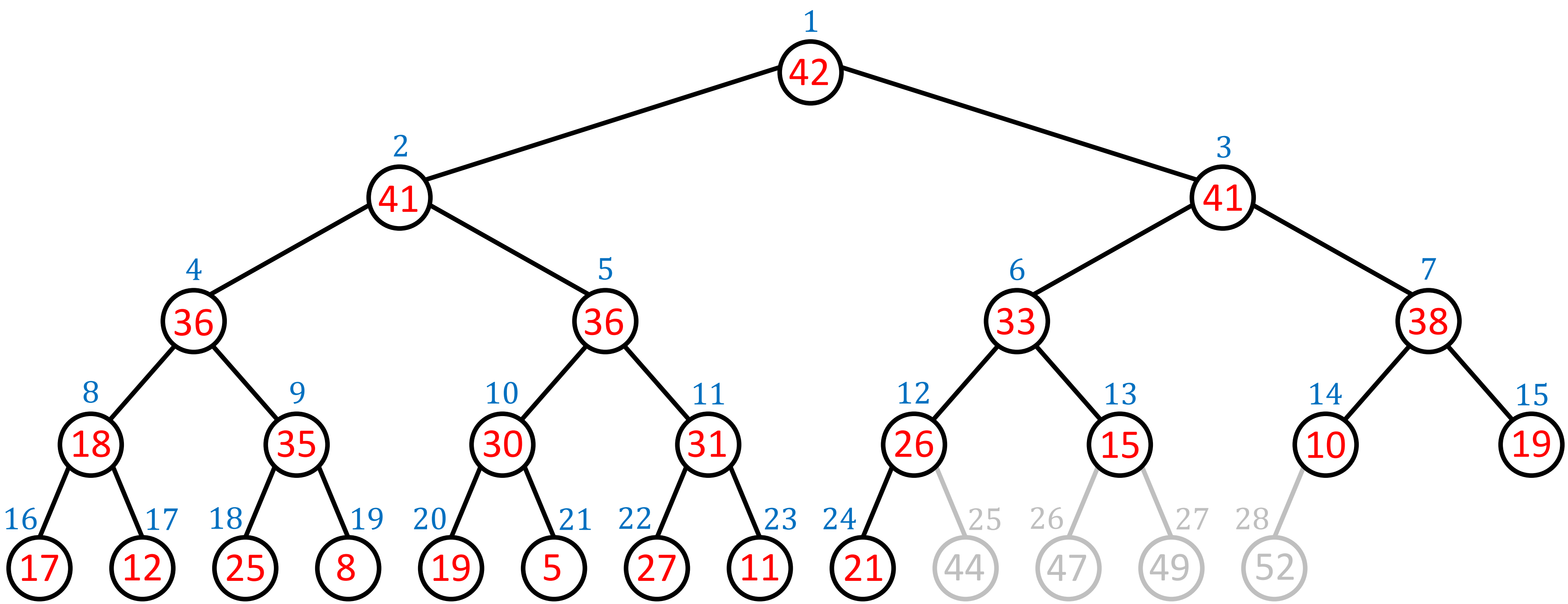
We make $A[1..24]$ a valid max-heap by calling MAX-HEAPIFY(A, 1).



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	42	41	41	36	36	33	38	18	35	30	31	26	15	10	19	17	12	25	8	19	5	27	11	21	44	47	49	52

The Heapsort Algorithm

$A[1] = 42$ is the largest number in the max-heap $A[1..24]$.

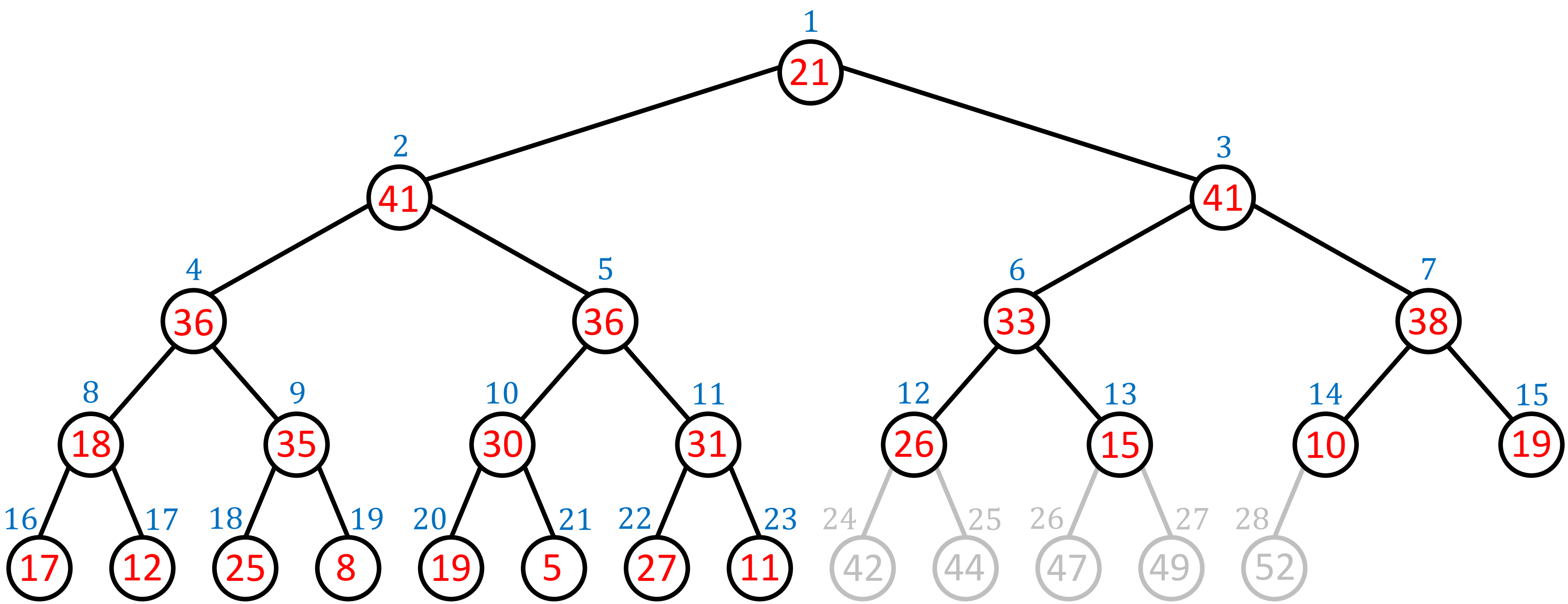


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	42	41	41	36	36	33	38	18	35	30	31	26	15	10	19	17	12	25	8	19	5	27	11	21	44	47	49	52

The Heapsort Algorithm

$A[1] = 42$ is the largest number in the max-heap $A[1..24]$.

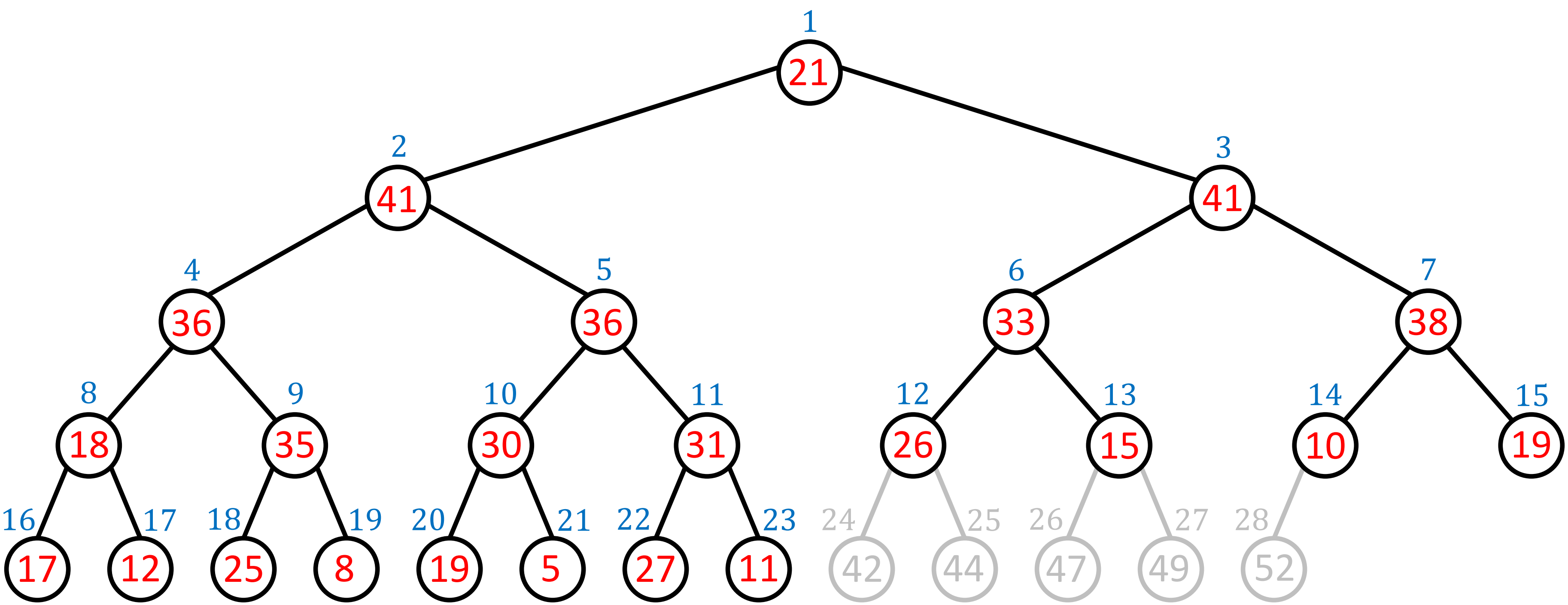
We swap $A[1]$ with $A[24]$ and treat $A[1..23]$ as the max-heap from now on.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	21	41	41	36	36	33	38	18	35	30	31	26	15	10	19	17	12	25	8	19	5	27	11	42	44	47	49	52

The Heapsort Algorithm

But $A[1..23]$ is not a valid max-heap!

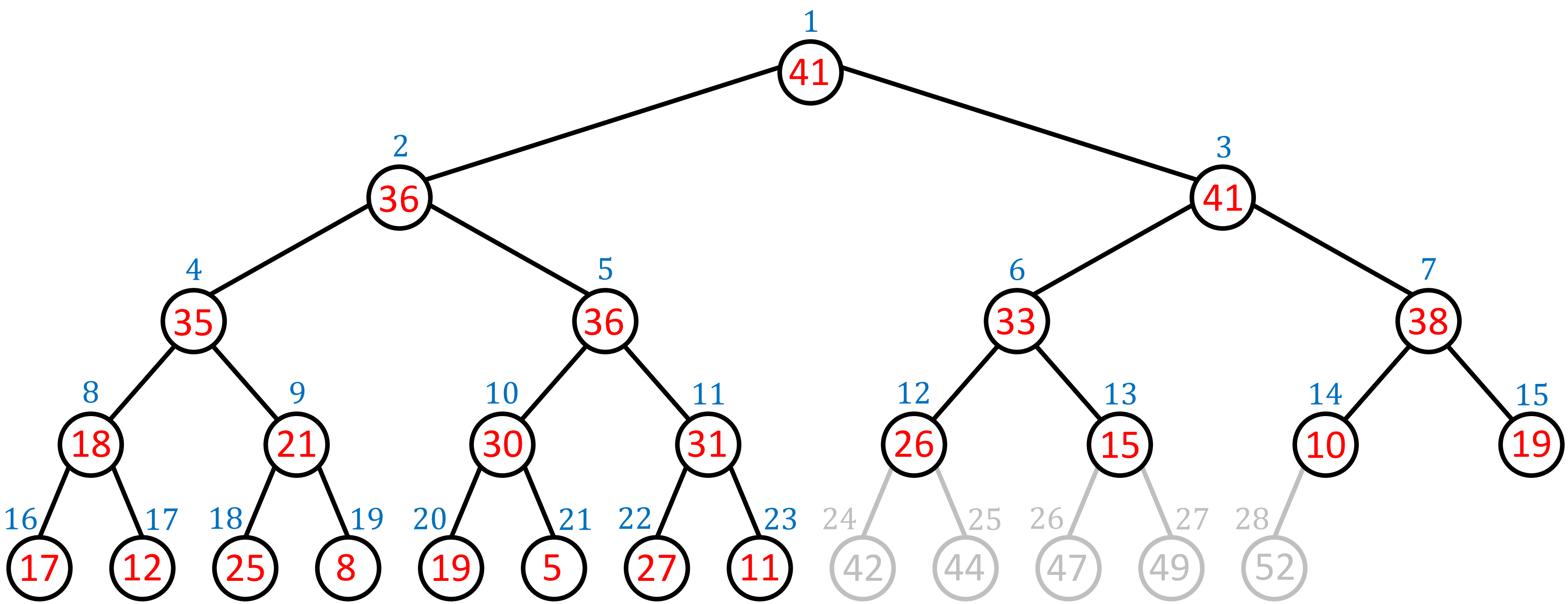


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	21	41	41	36	36	33	38	18	35	30	31	26	15	10	19	17	12	25	8	19	5	27	11	42	44	47	49	52

The Heapsort Algorithm

But $A[1..23]$ is not a valid max-heap!

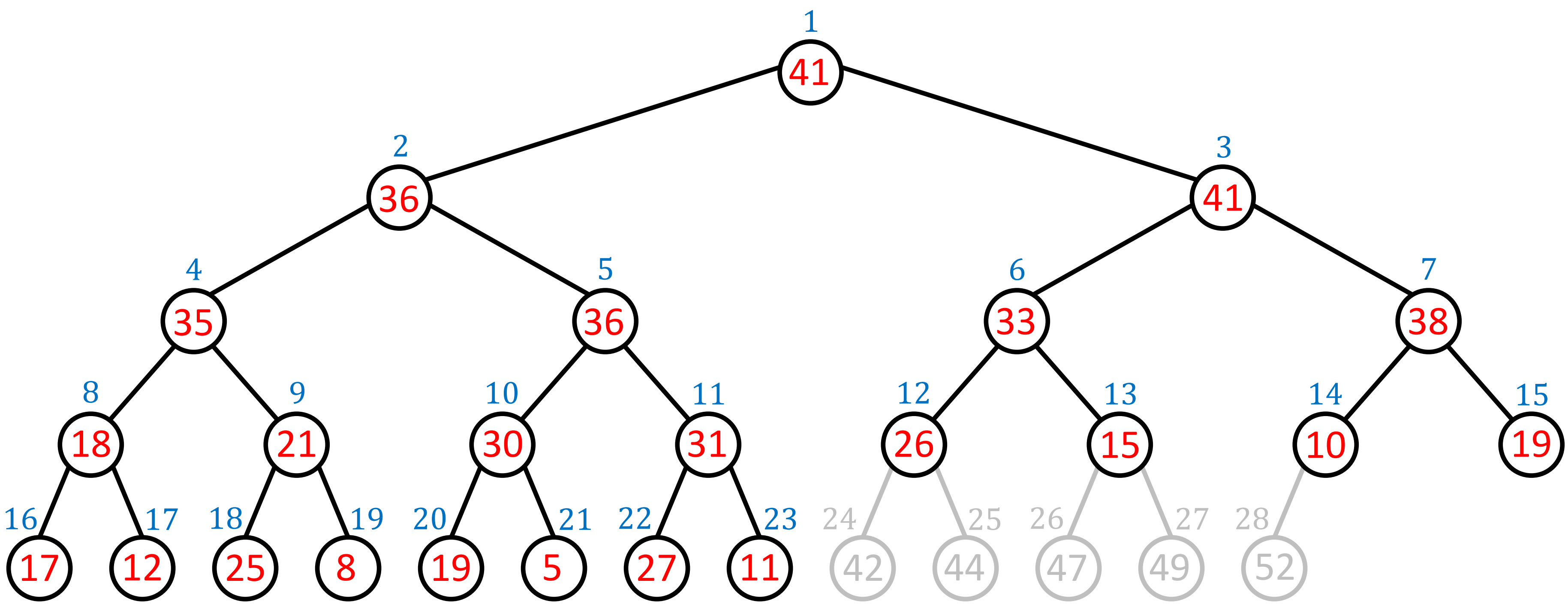
We make $A[1..23]$ a valid max-heap by calling MAX-HEAPIFY(A, 1).



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	41	36	41	35	36	33	38	18	21	30	31	26	15	10	19	17	12	25	8	19	5	27	11	42	44	47	49	52

The Heapsort Algorithm

$A[1] = 41$ is a largest number in the max-heap $A[1..23]$.

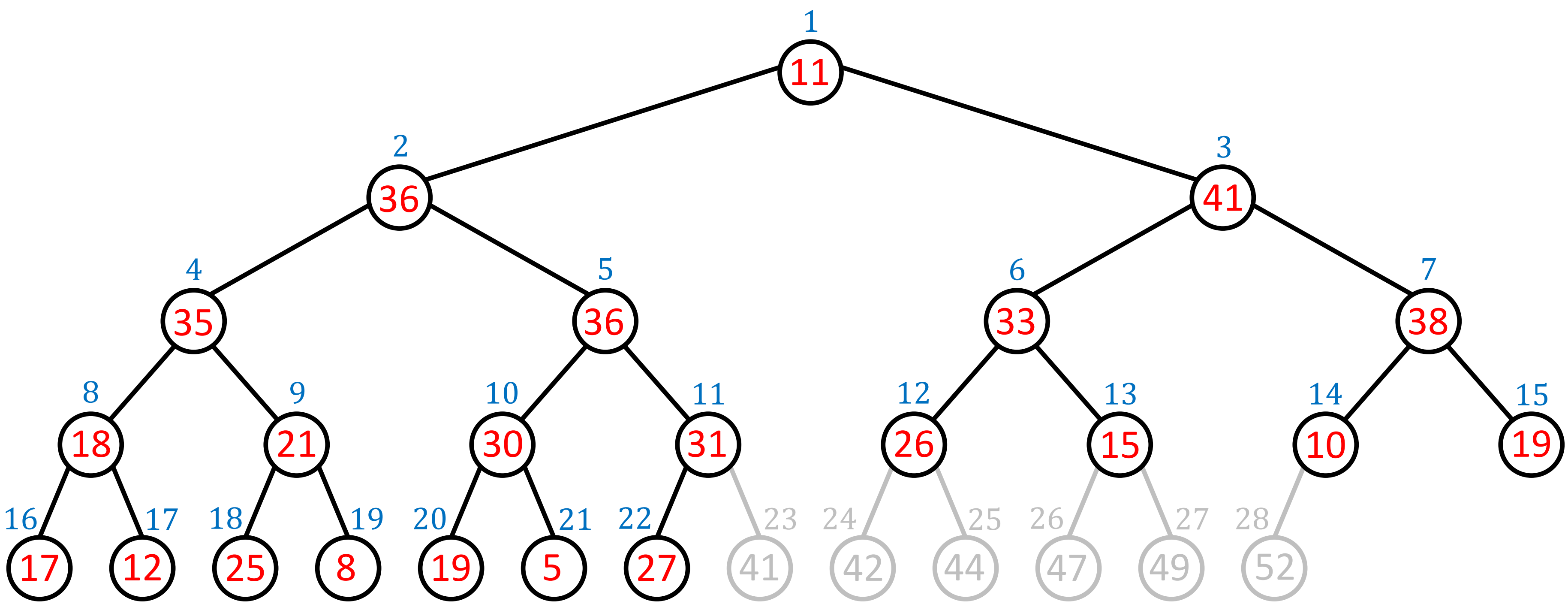


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	41	36	41	35	36	33	38	18	21	30	31	26	15	10	19	17	12	25	8	19	5	27	11	42	44	47	49	52

The Heapsort Algorithm

$A[1] = 41$ is a largest number in the max-heap $A[1..23]$.

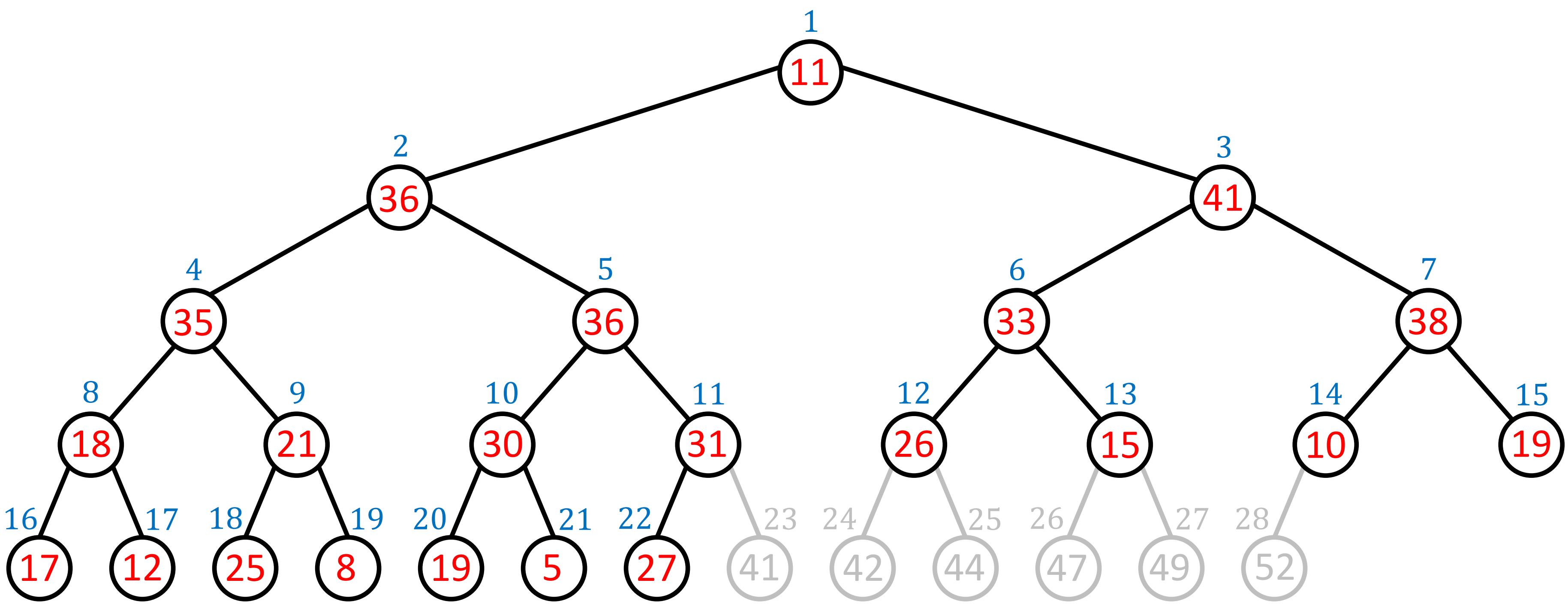
We swap $A[1]$ with $A[23]$ and treat $A[1..22]$ as the max-heap from now on.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	11	36	41	35	36	33	38	18	21	30	31	26	15	10	19	17	12	25	8	19	5	27	41	42	44	47	49	52

The Heapsort Algorithm

But $A[1..22]$ is not a valid max-heap!

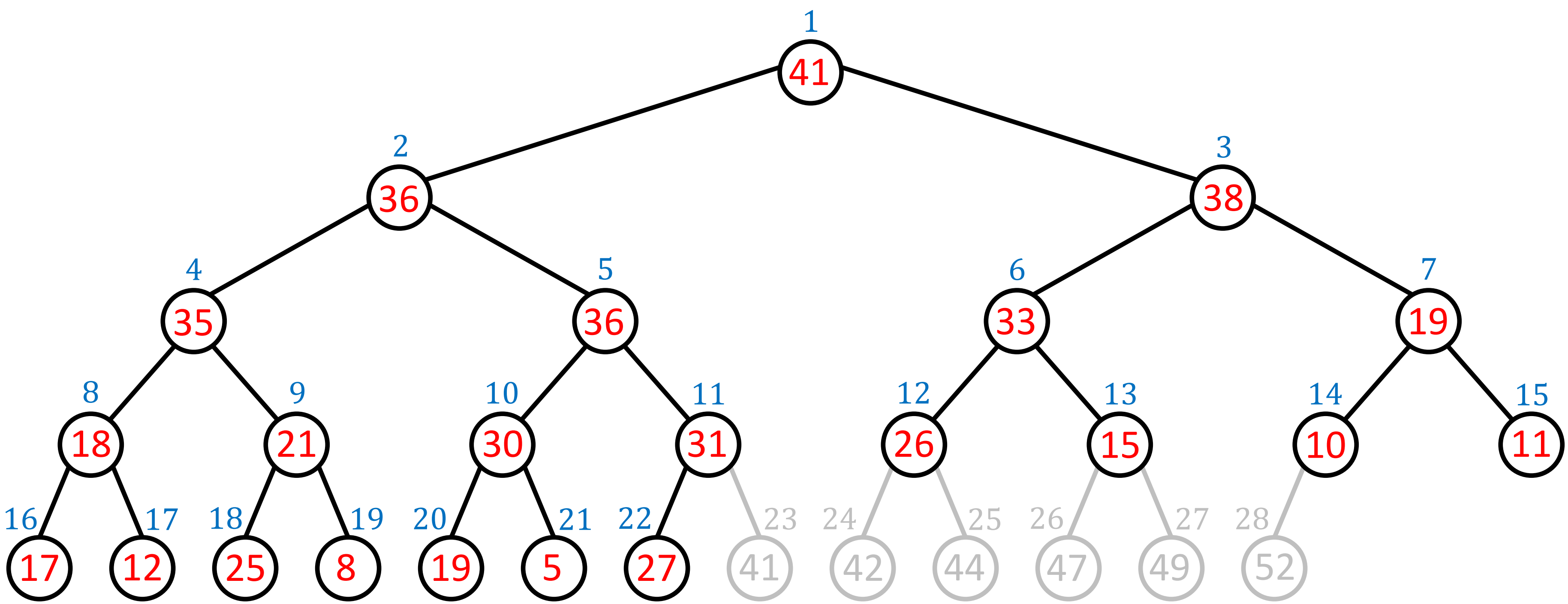


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	11	36	41	35	36	33	38	18	21	30	31	26	15	10	19	17	12	25	8	19	5	27	41	42	44	47	49	52

The Heapsort Algorithm

But $A[1..22]$ is not a valid max-heap!

We make $A[1..22]$ a valid max-heap by calling MAX-HEAPIFY(A, 1).

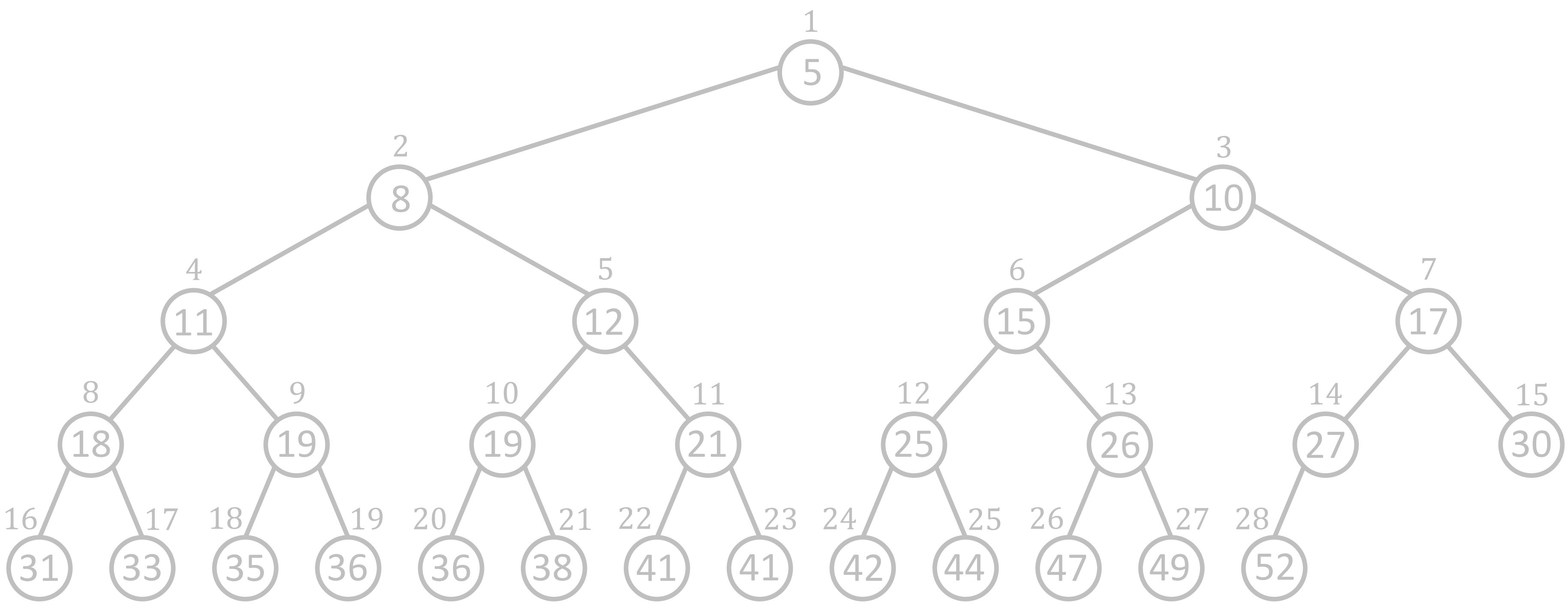


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	41	36	38	35	36	33	19	18	21	30	31	26	15	10	11	17	12	25	8	19	5	27	41	42	44	47	49	52

The Heapsort Algorithm

Keep going!

Finally, $A[1..28]$ will have its items sorted in nondecreasing order of value.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	5	8	10	11	12	15	17	18	19	19	21	25	26	27	30	31	33	35	36	36	38	41	41	42	44	47	49	52

The Heapsort Algorithm

Input: An array $A[\,1:\,n\,]$ of n numbers.

Output: Elements of $A[\,1:\,n\,]$ rearranged in non-decreasing order of value.

HEAPSORT (A)

1. BUILD-MAX-HEAP (A)
2. ***for*** $i = A.length$ ***downto*** 2
3. exchange $A[1]$ with $A[i]$
4. $A.heapsize = A.heapsize - 1$
5. MAX-HEAPIFY ($A, 1$)

Priority Queues

A *priority queue* is a data structure for maintaining a set S of elements, each with an associated value called a *key*.

A *max-priority queue* supports the following operations:

$\text{INSERT}(S, x)$ inserts the element x into the set S, which is equivalent to the operation $S = S \cup \{x\}$.

$\text{MAXIMUM}(S)$ returns the element of S with the largest key.

$\text{EXTRACT-MAX}(S)$ removes and returns the element of S with the largest key.

$\text{INCREASE-KEY}(S, x, k)$ increases the value of element x's key to the new value k, which is assumed to be at least as large as x's current key value.

A Max-Heap as a Max-Priority Queue

HEAP-MAXIMUM (A)

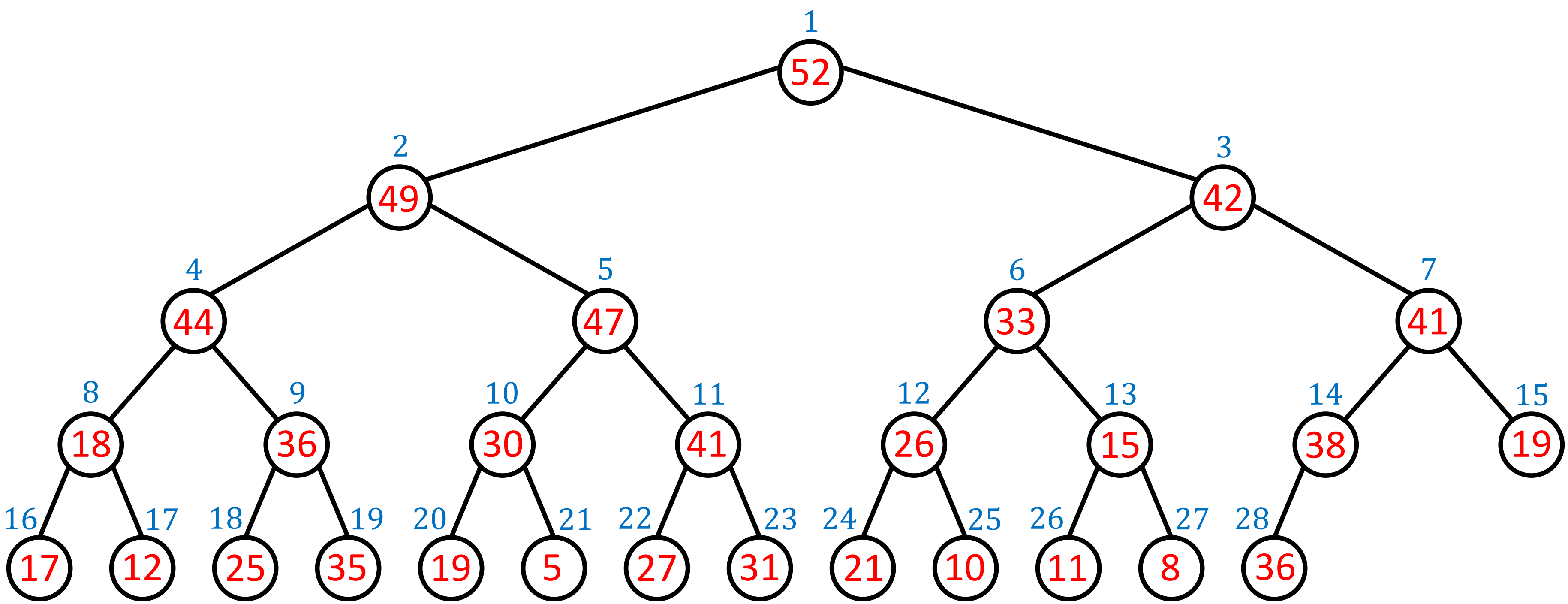
1. **return** $A[1]$

HEAP-EXTRACT-MAX (A)

1. **if** $A.heapsize < 1$
2. **error** "heap underflow"
3. $max = A[1]$
4. $A[1] = A[A.heapsize]$
5. $A.heapsize = A.heapsize - 1$
6. MAX-HEAPIFY (A, 1)
7. **return** max

Extracting the Maximum (Extract-Max)

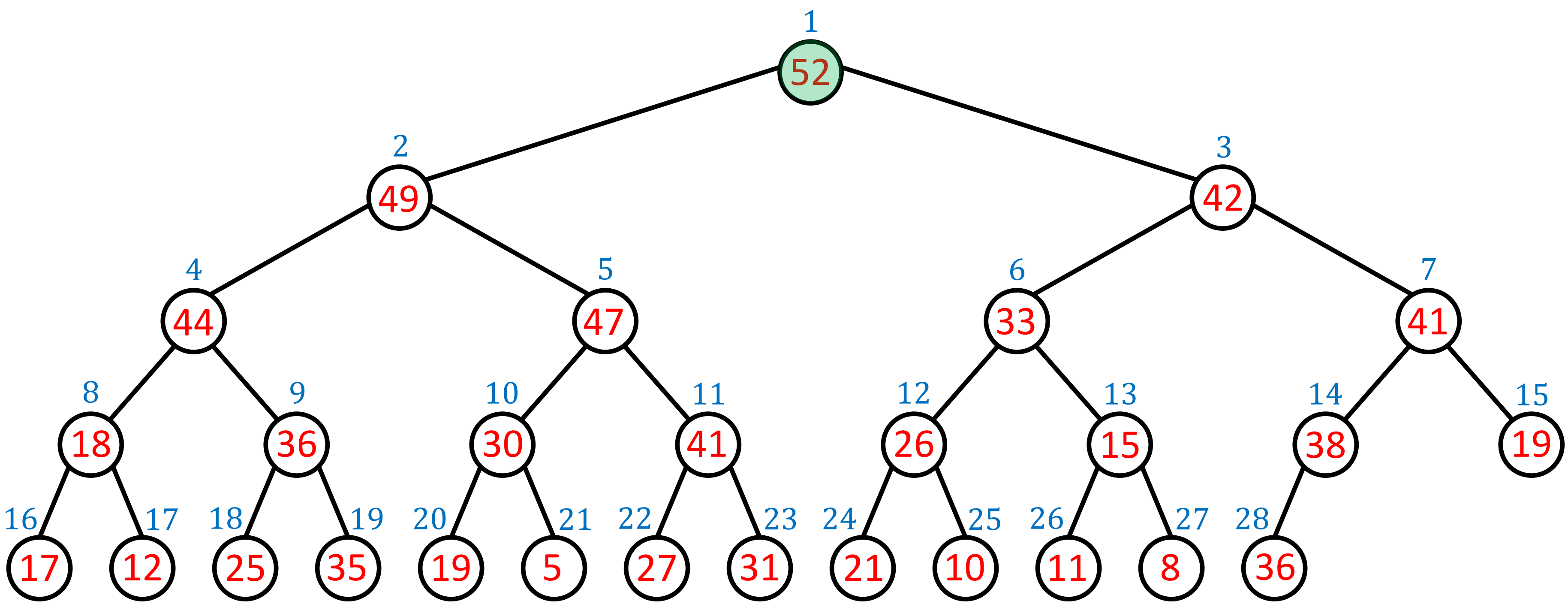
We want to extract the maximum item from the following max-heap.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	36

Extracting the Maximum (Extract-Max)

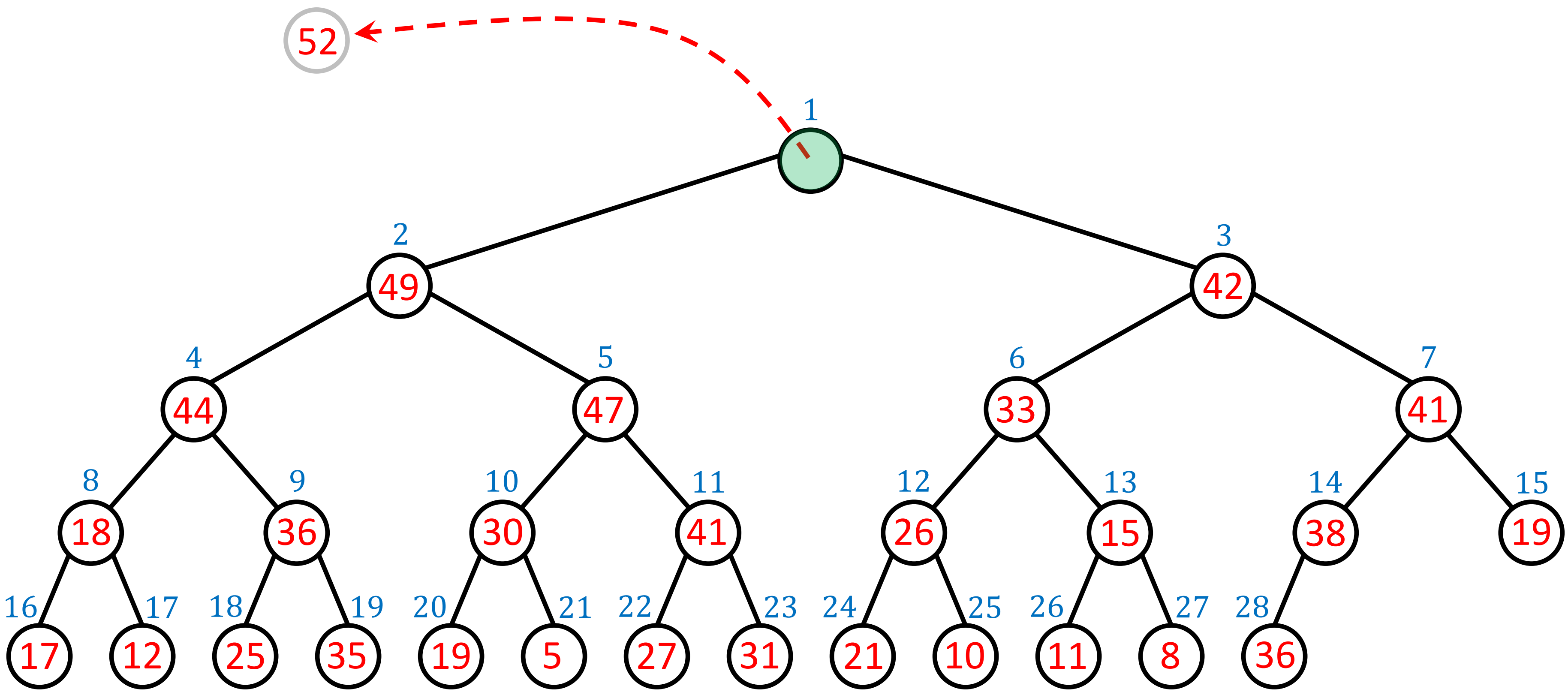
The maximum item is at $A[1]$. So, we remove that item and return.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	36

Extracting the Maximum (Extract-Max)

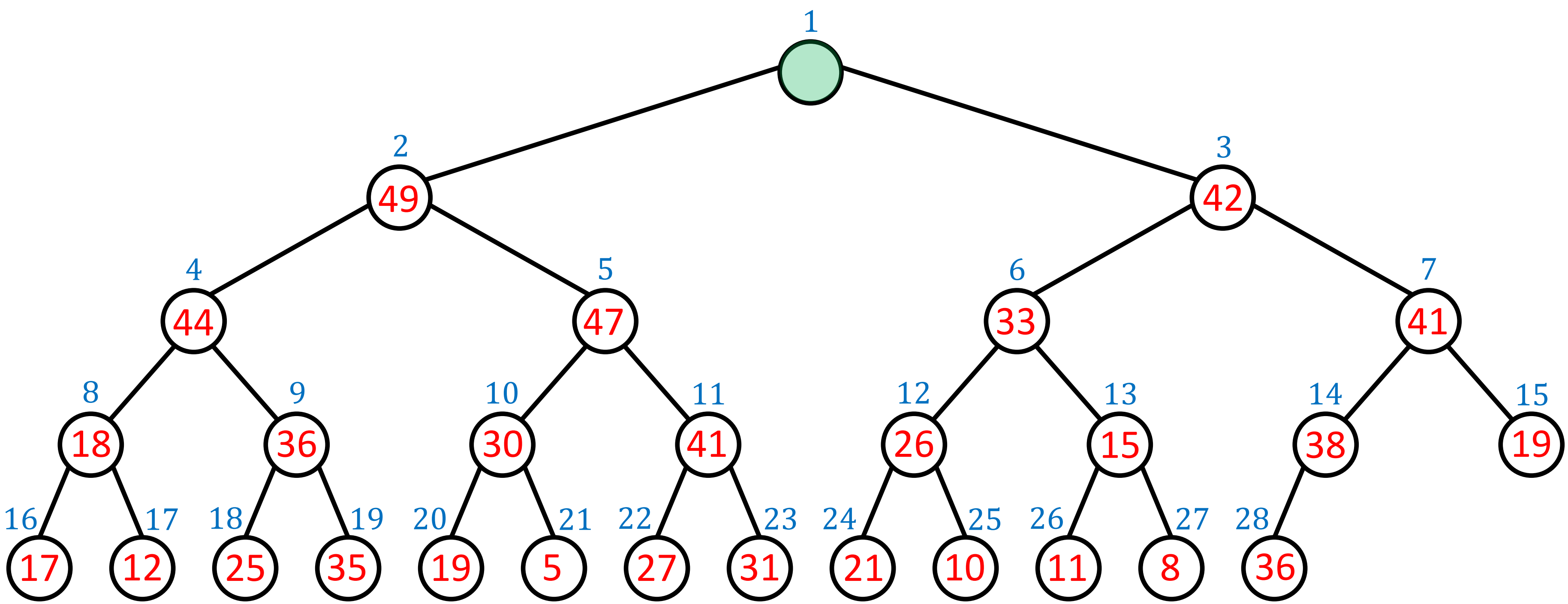
The maximum item is at $A[1]$. So, we remove that item and return.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A		49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	36

Extracting the Maximum (Extract-Max)

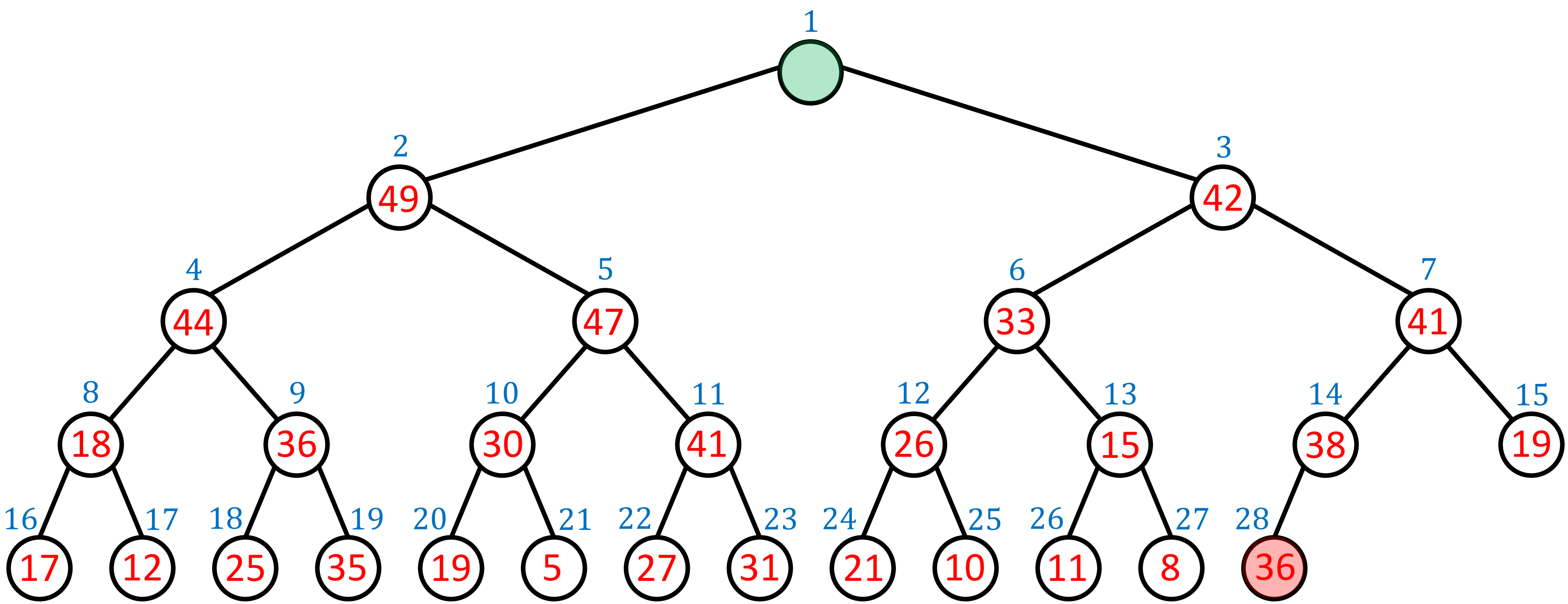
But the remaining items do not form a valid max-heap.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A		49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	36

Extracting the Maximum (Extract-Max)

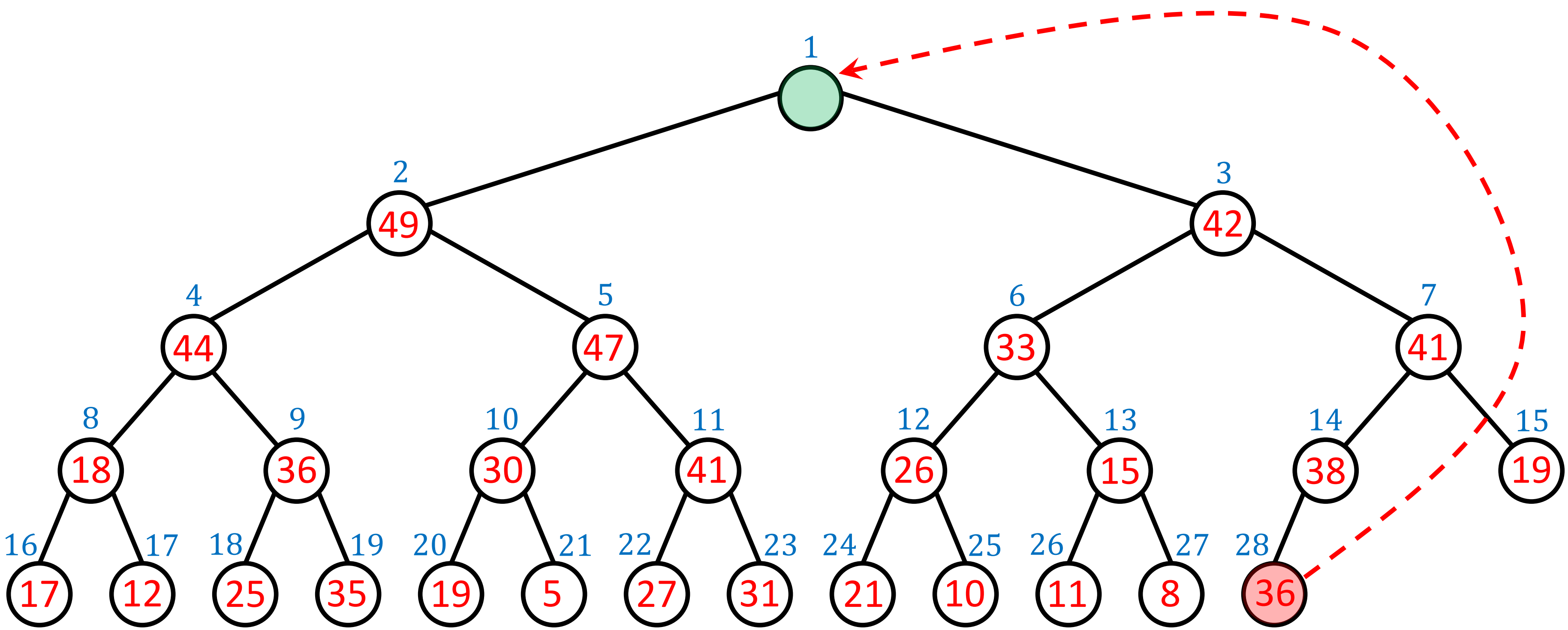
To fix the data structure we move the last item at $A[28]$ to $A[1]$.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A		49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	36

Extracting the Maximum (Extract-Max)

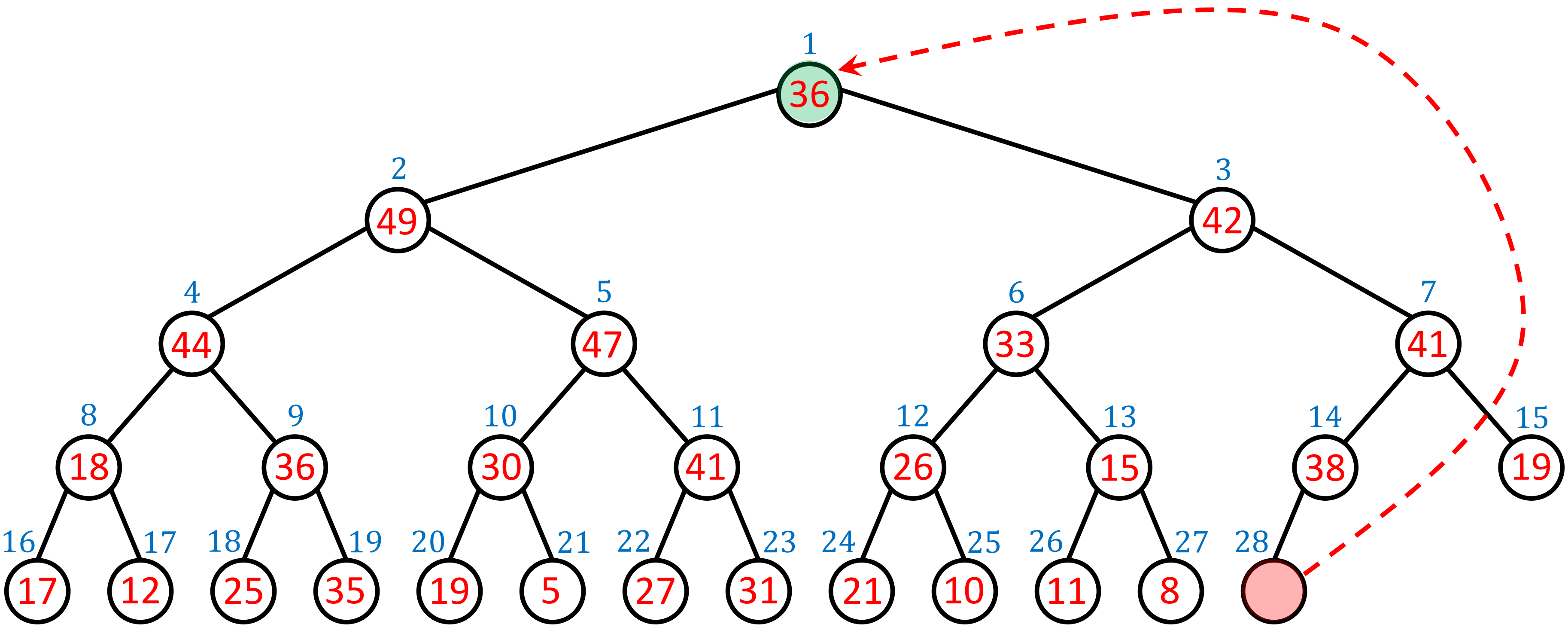
To fix the data structure we move the last item at $A[28]$ to $A[1]$.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A		49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	36

Extracting the Maximum (Extract-Max)

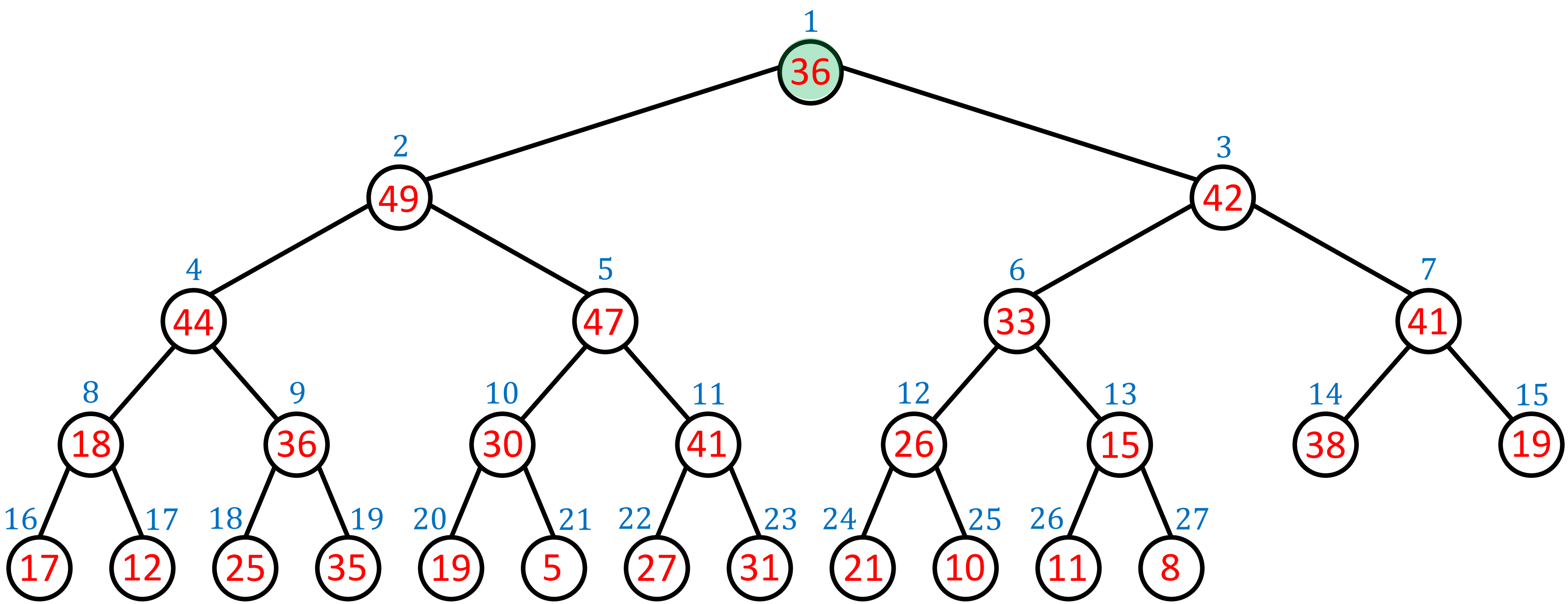
To fix the data structure we move the last item at $A[28]$ to $A[1]$.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	36	49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8	

Extracting the Maximum (Extract-Max)

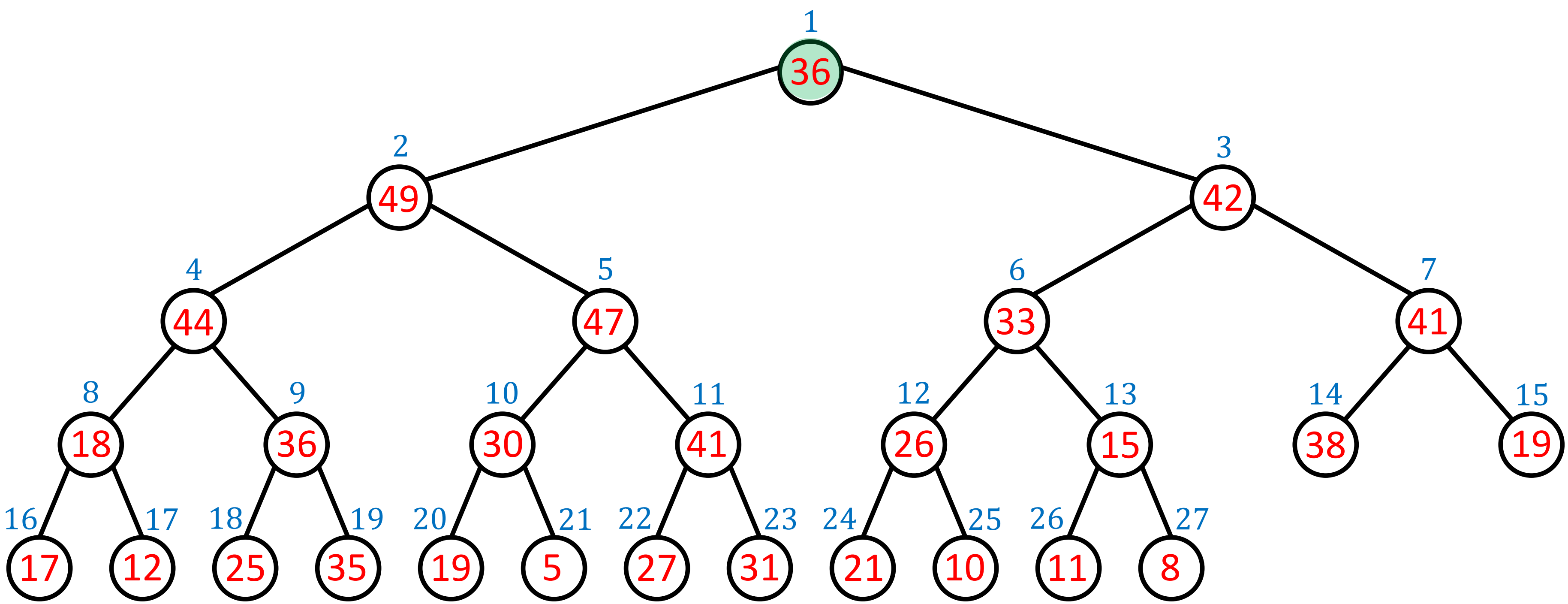
To fix the data structure we move the last item at $A[28]$ to $A[1]$.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
A	36	49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8

Extracting the Maximum (Extract-Max)

But the data structure is still not a valid max-heap.

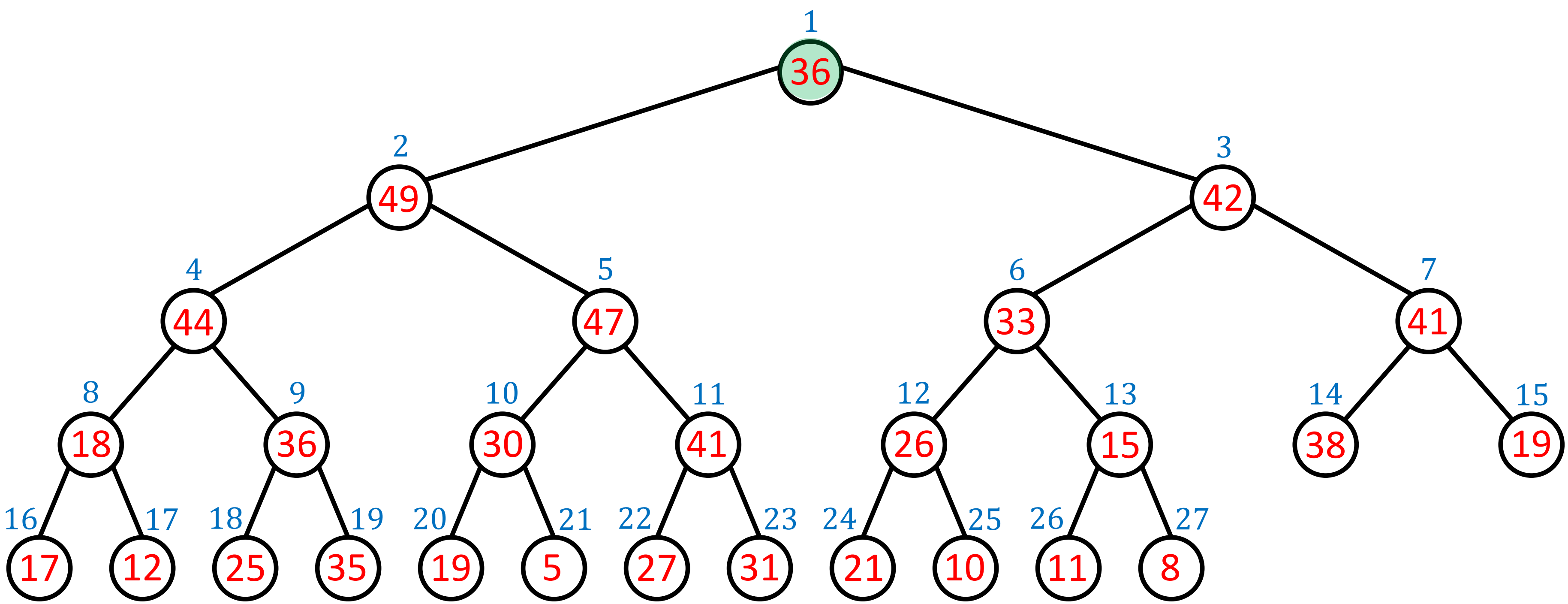


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
A	36	49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8

Extracting the Maximum (Extract-Max)

Though the subtrees rooted at $A[2]$ and $A[3]$ are both valid max-heaps, the item at $A[1]$ violates the max-heap property.

To fix it we simply call MAX-HEAPIFY(A, 1).

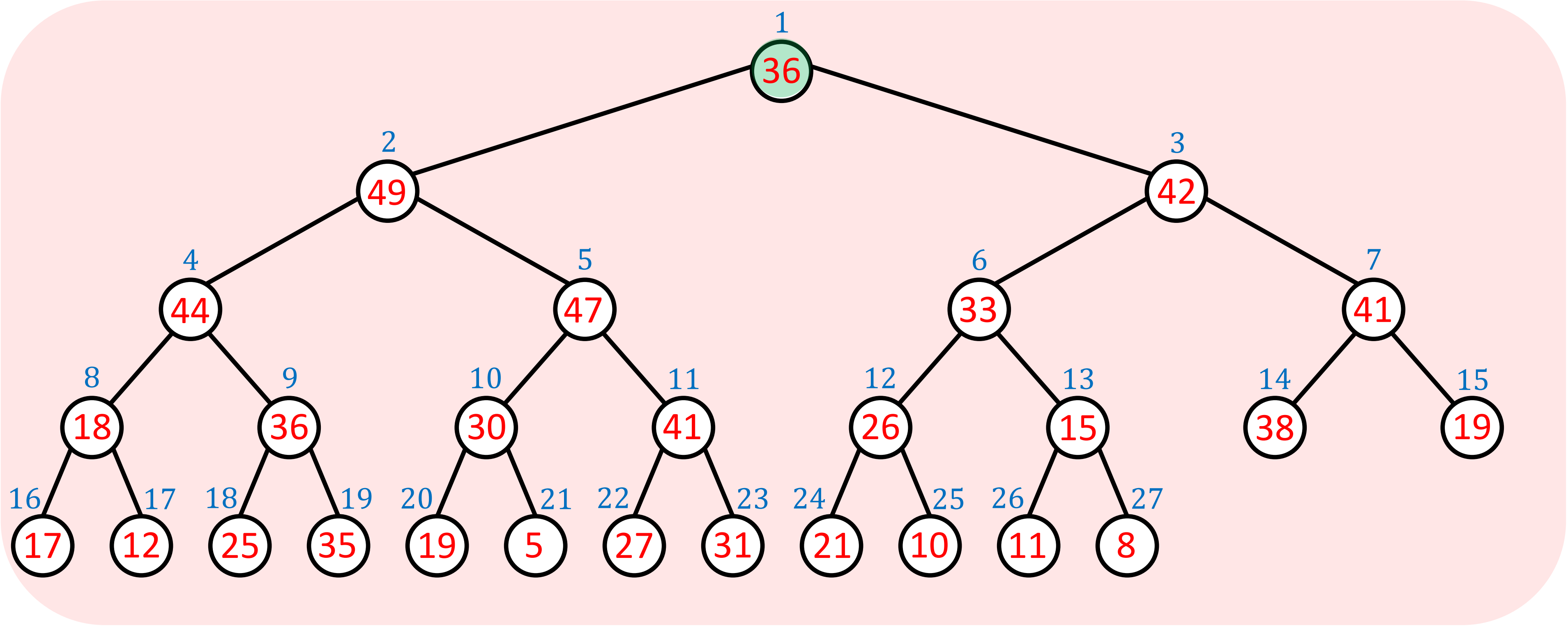


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
A	36	49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8

Extracting the Maximum (Extract-Max)

Though the subtrees rooted at $A[2]$ and $A[3]$ are both valid max-heaps, the item at $A[1]$ violates the max-heap property.

To fix it we simply call MAX-HEAPIFY(A, 1).

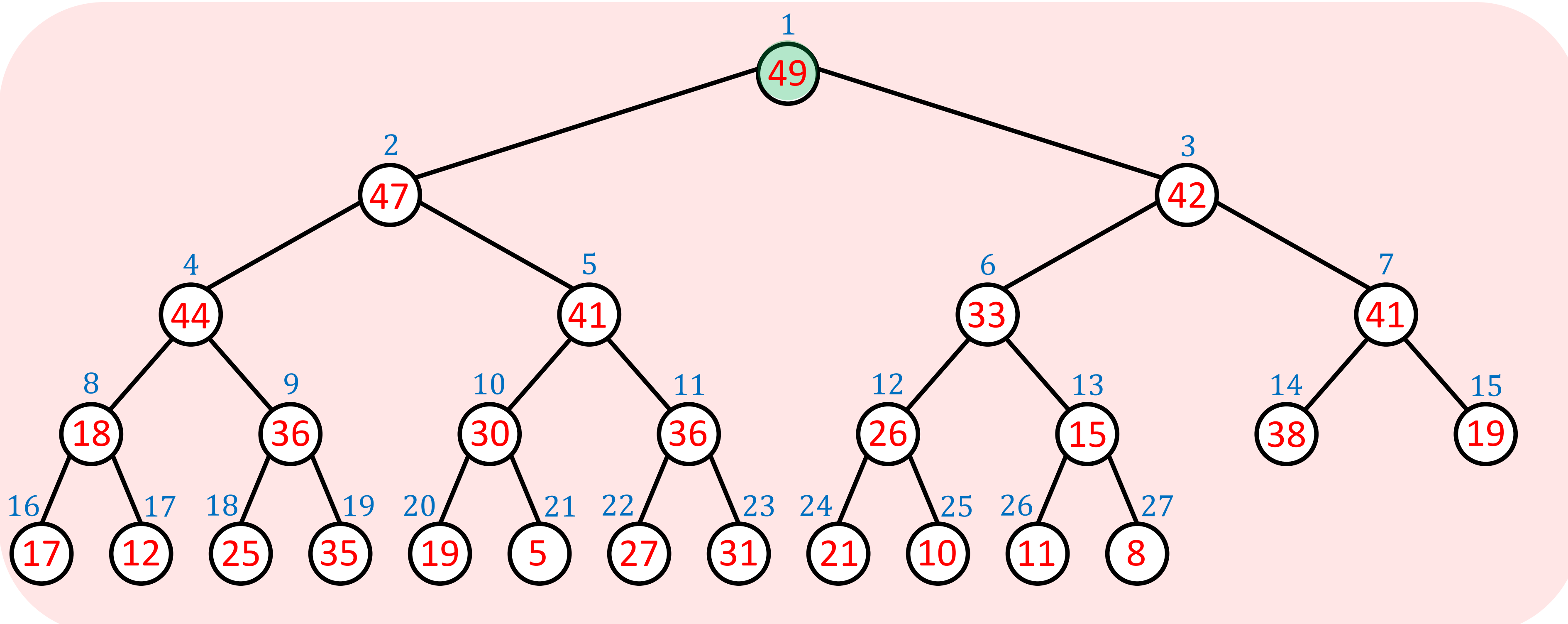


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
A	36	49	42	44	47	33	41	18	36	30	41	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8

Extracting the Maximum (Extract-Max)

Though the subtrees rooted at $A[2]$ and $A[3]$ are both valid max-heaps, the item at $A[1]$ violates the max-heap property.

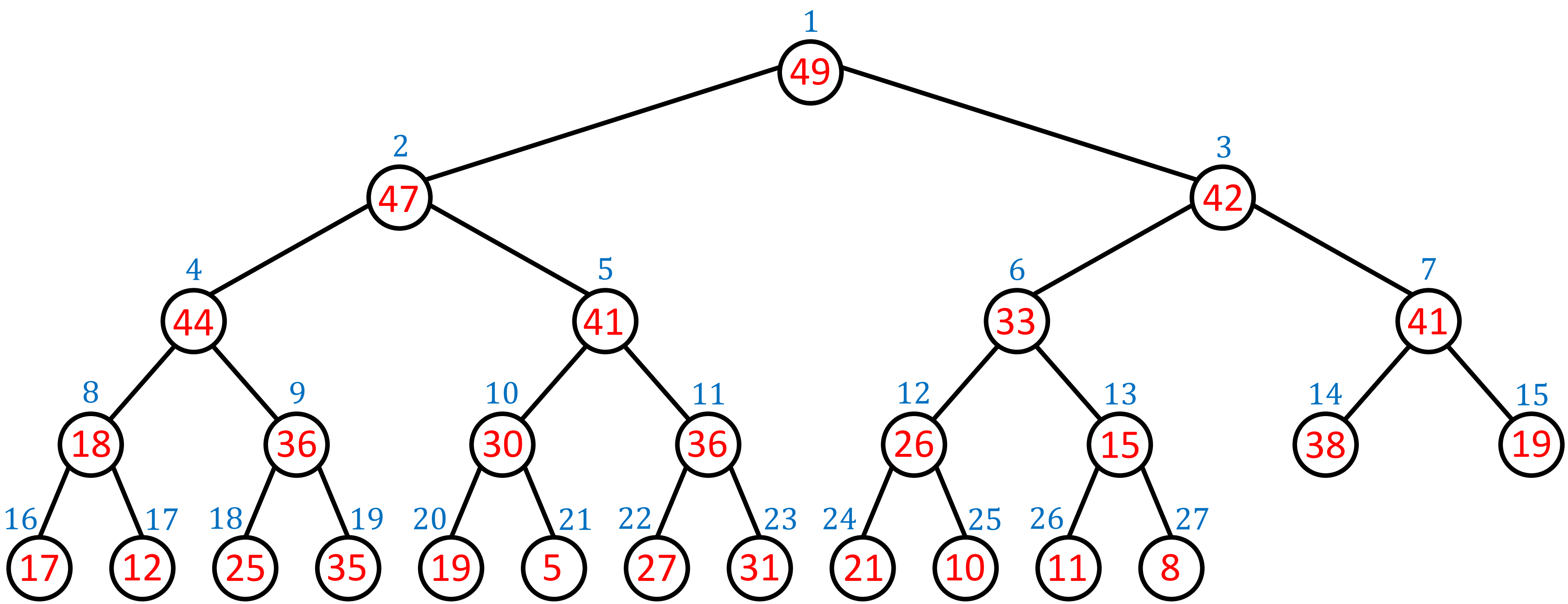
To fix it we simply call MAX-HEAPIFY(A, 1).



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
A	49	47	42	44	41	33	41	18	36	30	36	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8

Extracting the Maximum (Extract-Max)

This is now a valid max-heap:



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
A	49	47	42	44	41	33	41	18	36	30	36	26	15	38	19	17	12	25	35	19	5	27	31	21	10	11	8

A Max-Heap as a Max-Priority Queue

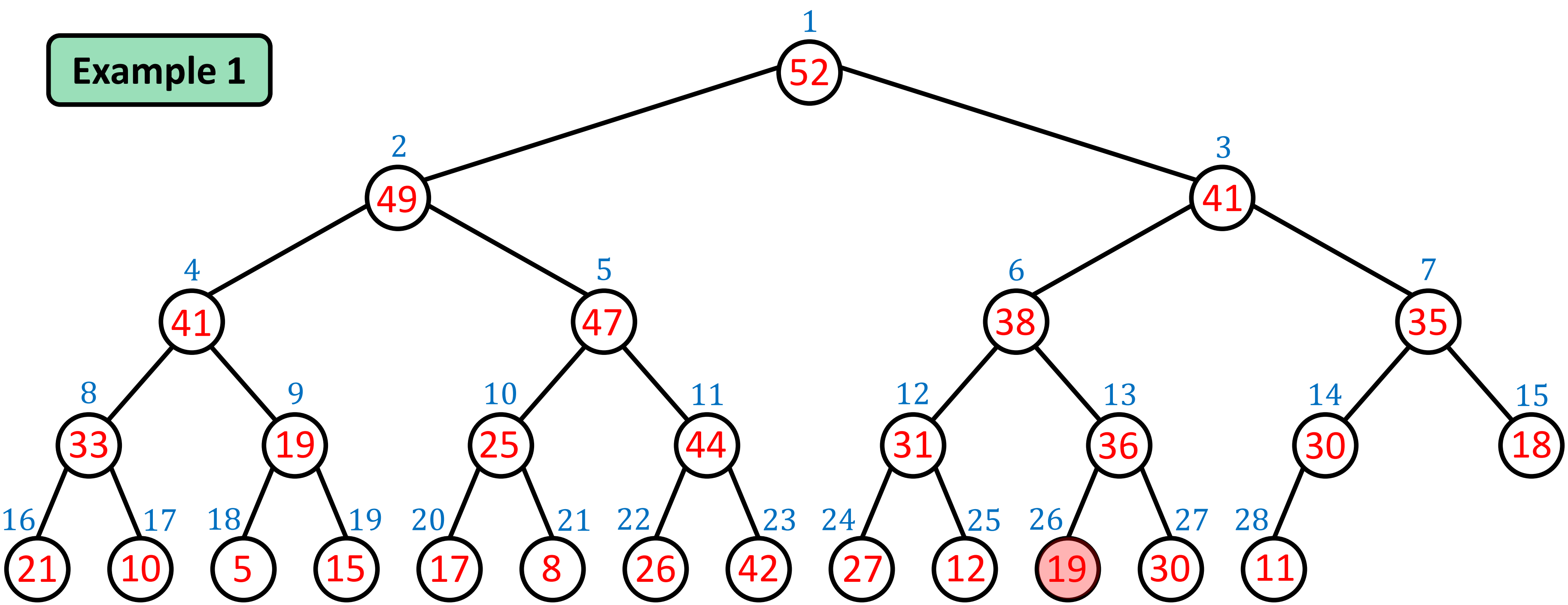
```
HEAP-INCREASE-KEY ( A, i, key )

if key < A[i]
    error "new key is smaller than current key"
A[i] = key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

Increasing a Key

Suppose we want to increase $A[26]$ from 19 to 47.

Example 1



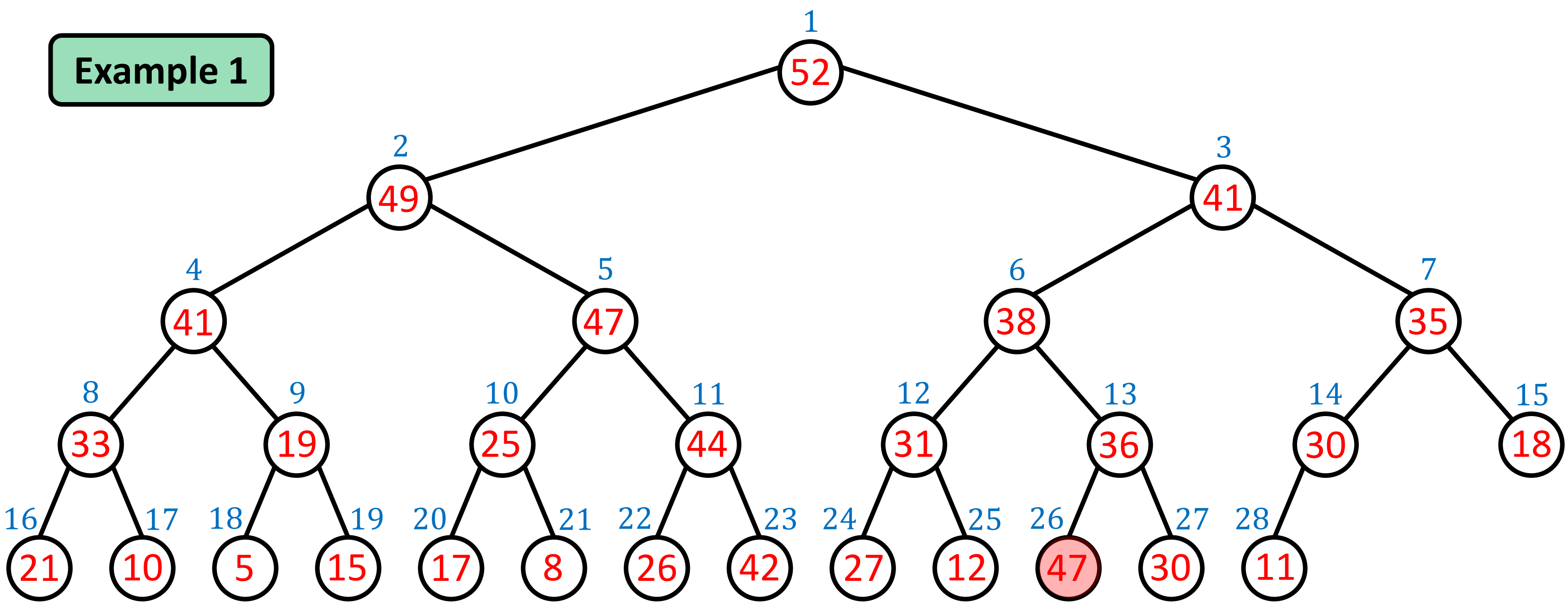
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	41	41	47	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11

Increasing a Key

Suppose we want to increase $A[26]$ from 19 to 47.

So, we set $A[26] = 47$.

Example 1

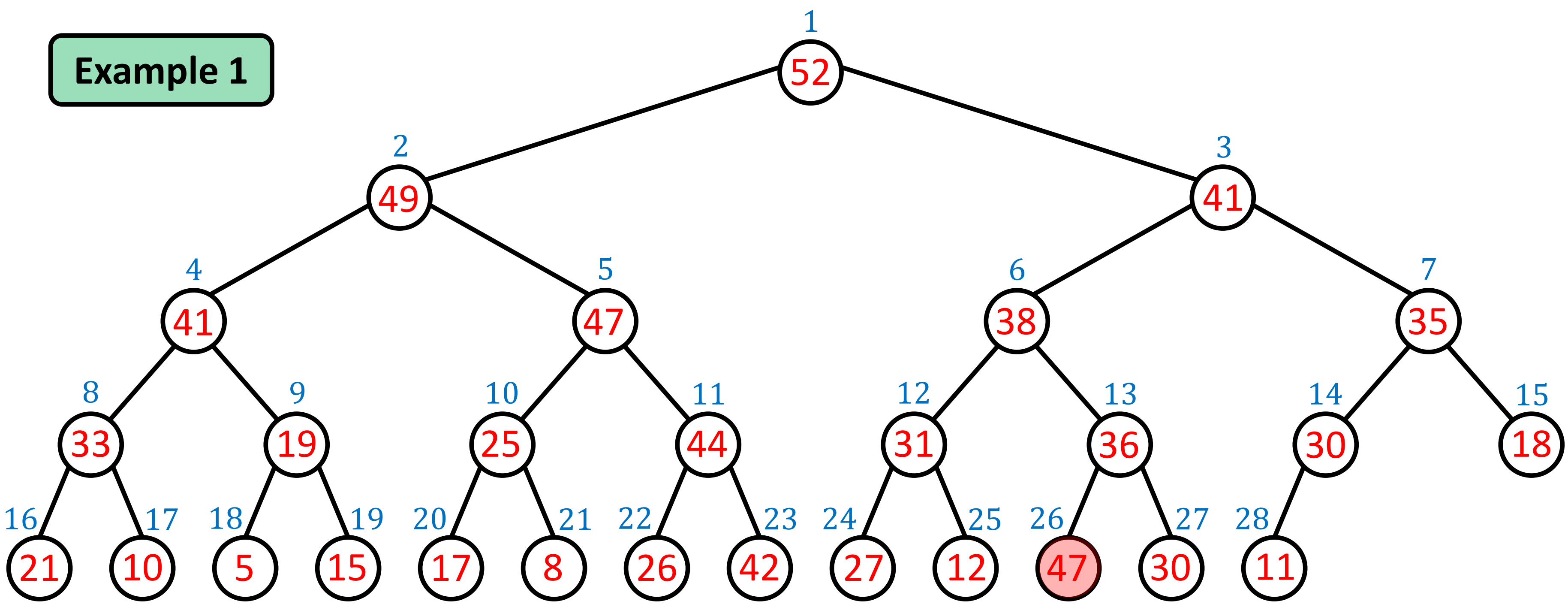


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	41	41	47	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	47	30	11

Increasing a Key

But $A[26] = 47$ violates the max-heap property because it is larger than its parent $A[13] = 36$.

Example 1



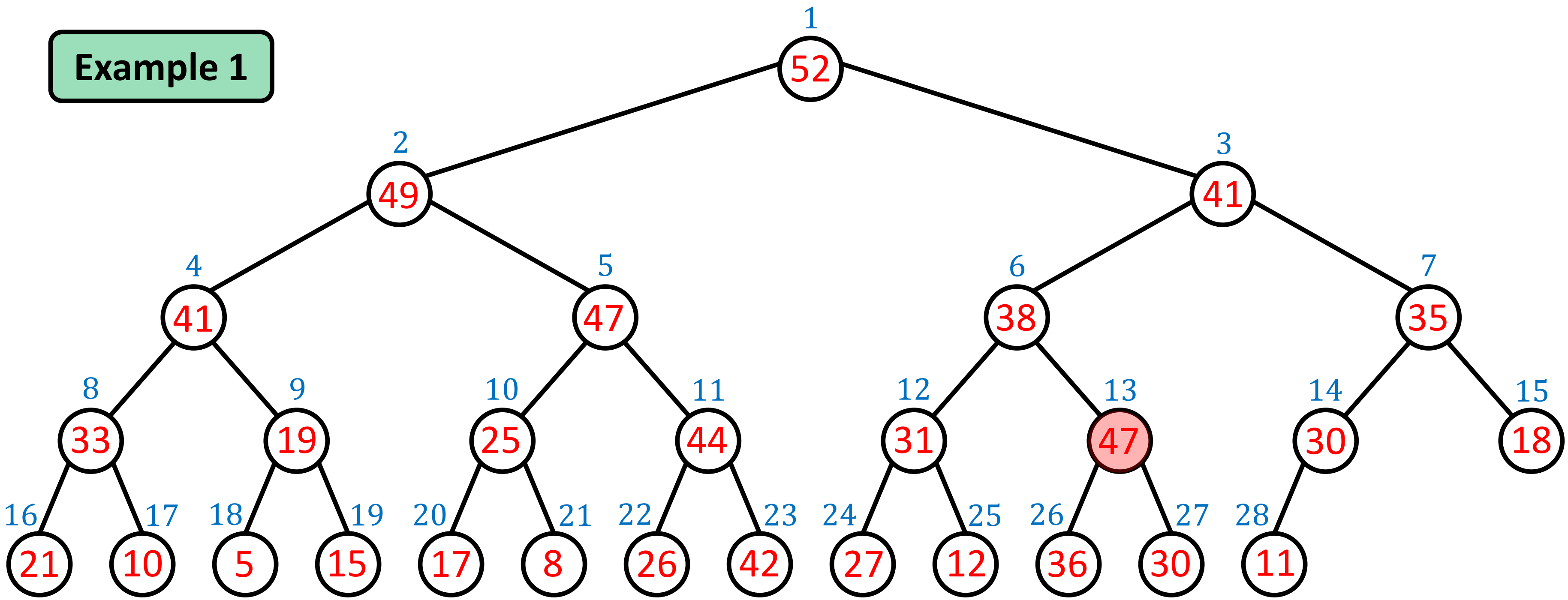
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	41	41	47	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	47	30	11

Increasing a Key

But $A[26] = 47$ violates the max-heap property because it is larger than its parent $A[13] = 36$.

We fix it by swapping $A[13]$ and $A[26]$.

Example 1

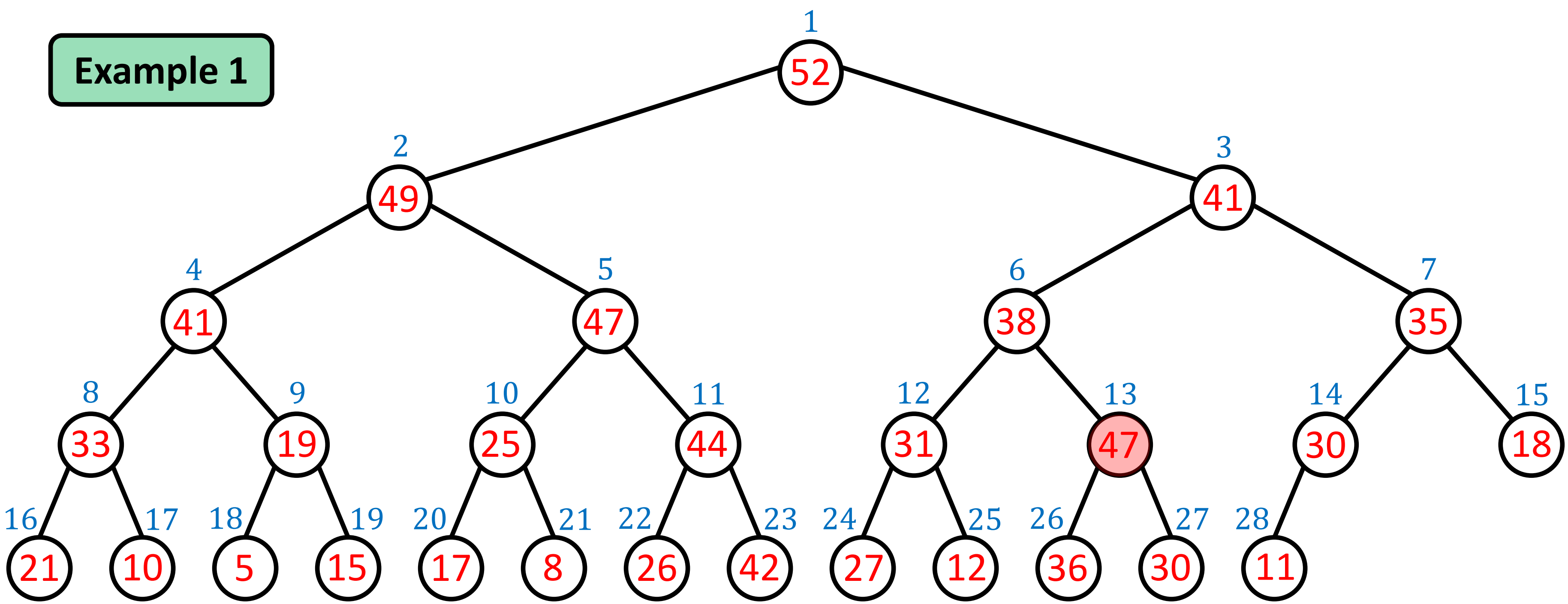


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	41	41	47	38	35	33	19	25	44	31	47	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

But now $A[13] = 47$ violates the max-heap property because it is larger than its parent $A[6] = 38$.

Example 1



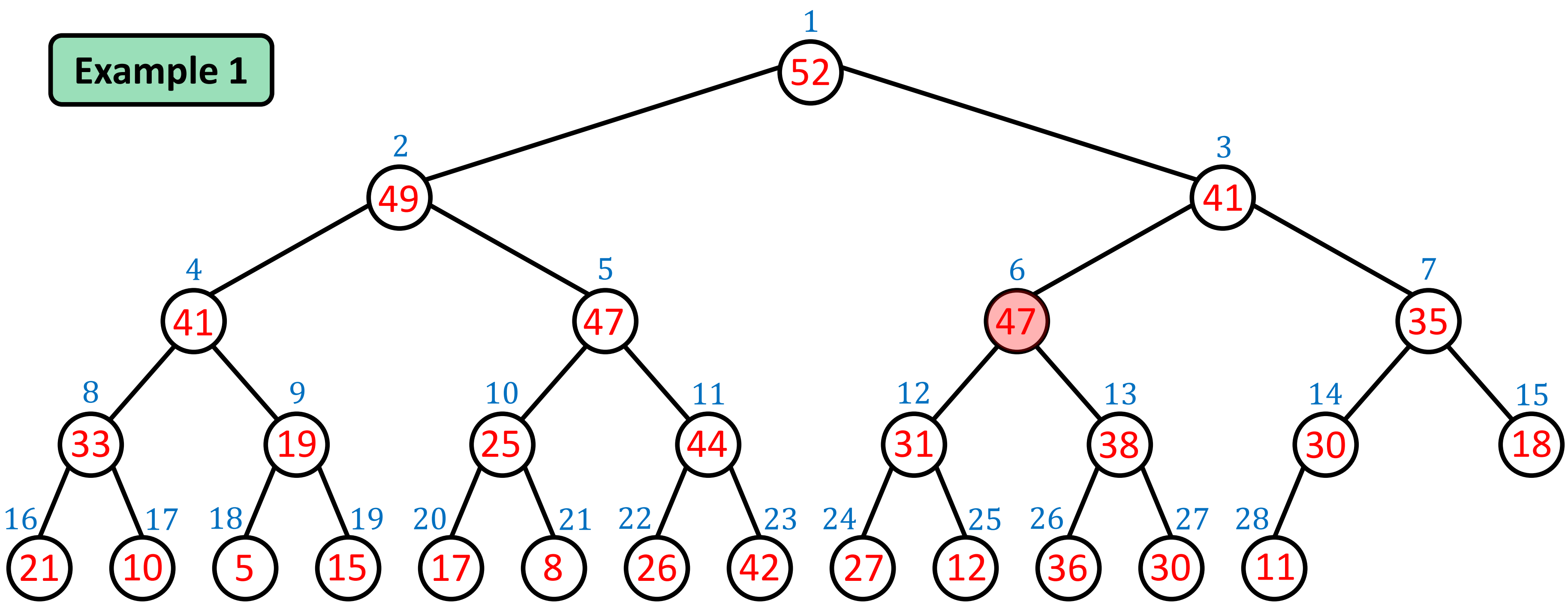
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	41	41	47	38	35	33	19	25	44	31	47	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

But now $A[13] = 47$ violates the max-heap property because it is larger than its parent $A[6] = 38$.

We fix it by swapping $A[6]$ and $A[13]$.

Example 1

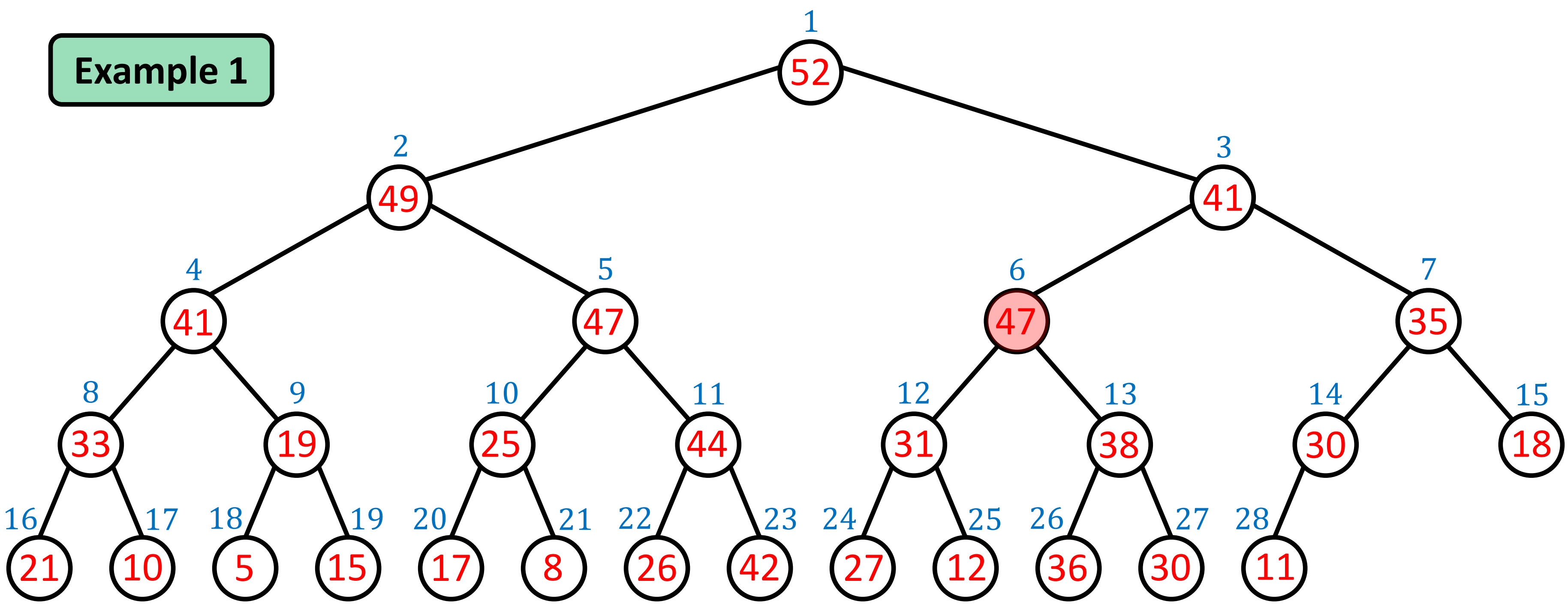


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	41	41	47	47	35	33	19	25	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

But now $A[6] = 47$ violates the max-heap property because it is larger than its parent $A[3] = 41$.

Example 1



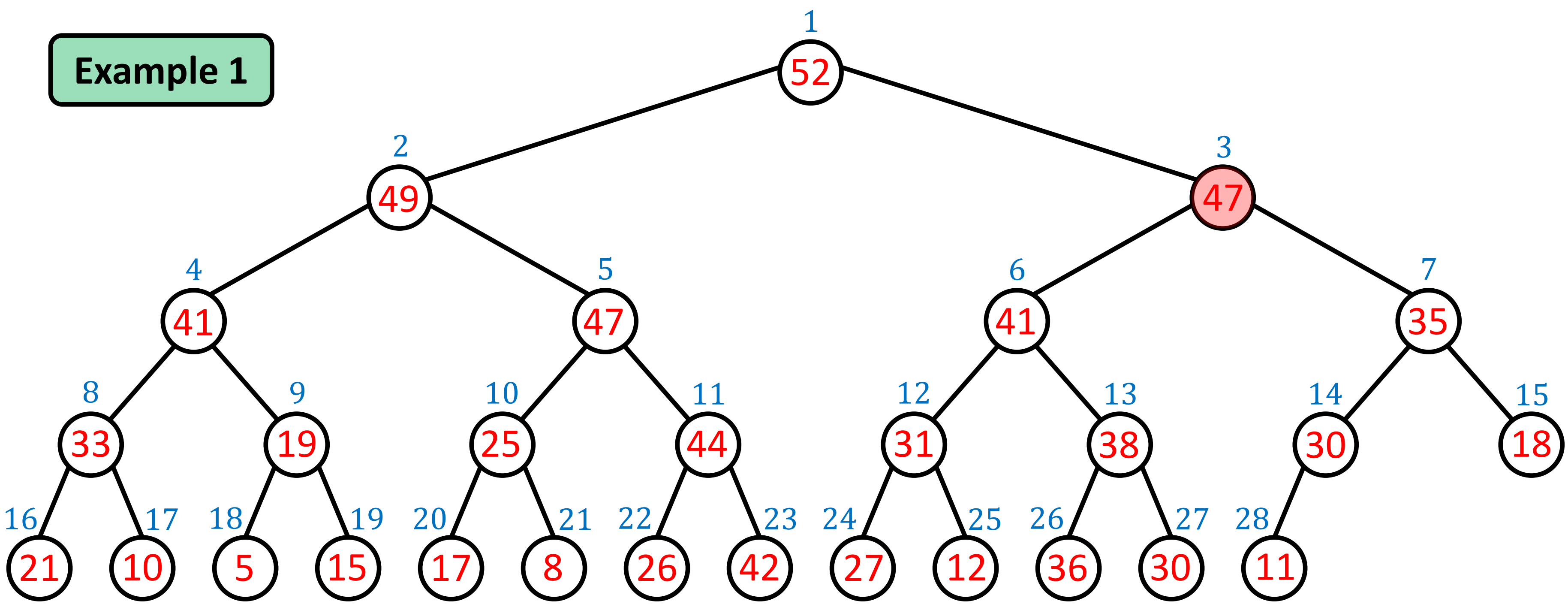
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	41	41	47	47	35	33	19	25	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

But now $A[6] = 47$ violates the max-heap property because it is larger than its parent $A[3] = 41$.

We fix it by swapping $A[3]$ and $A[6]$.

Example 1

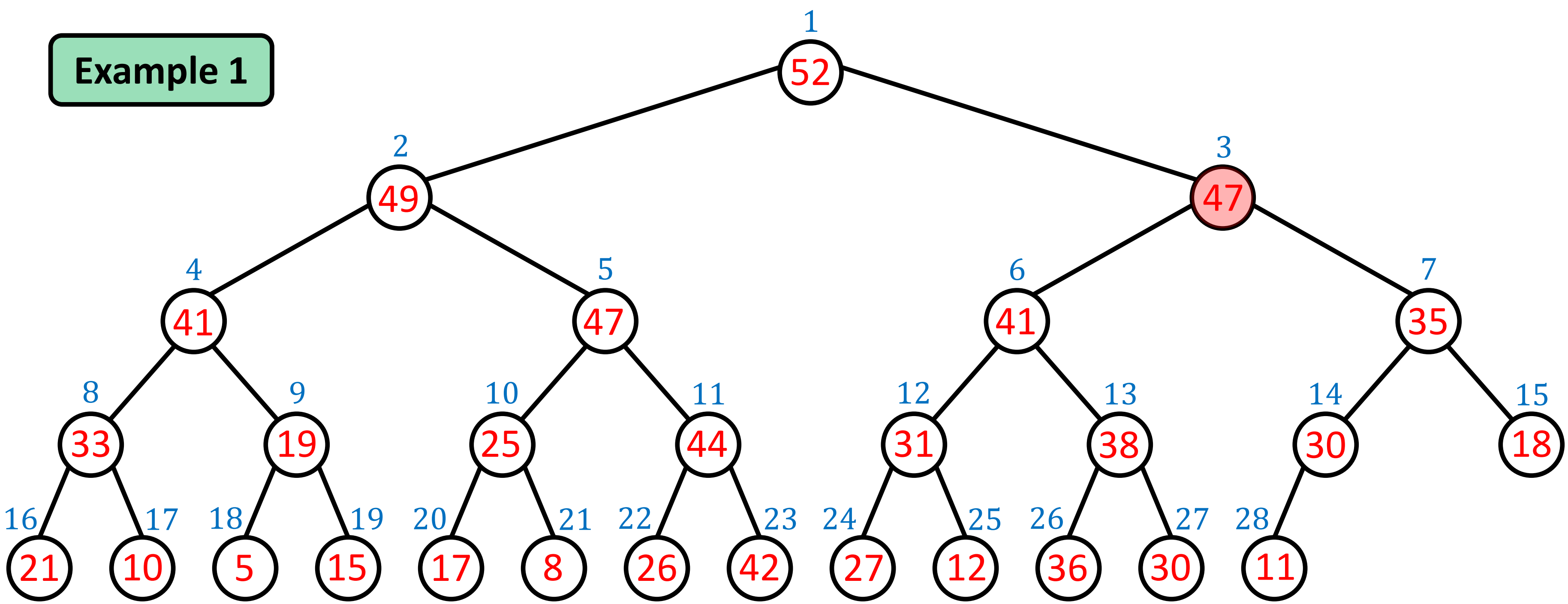


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	47	41	47	41	35	33	19	25	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

Now $A[3] = 47$ does not violate the max-heap property because it is not larger than its parent $A[1] = 52$.

Example 1



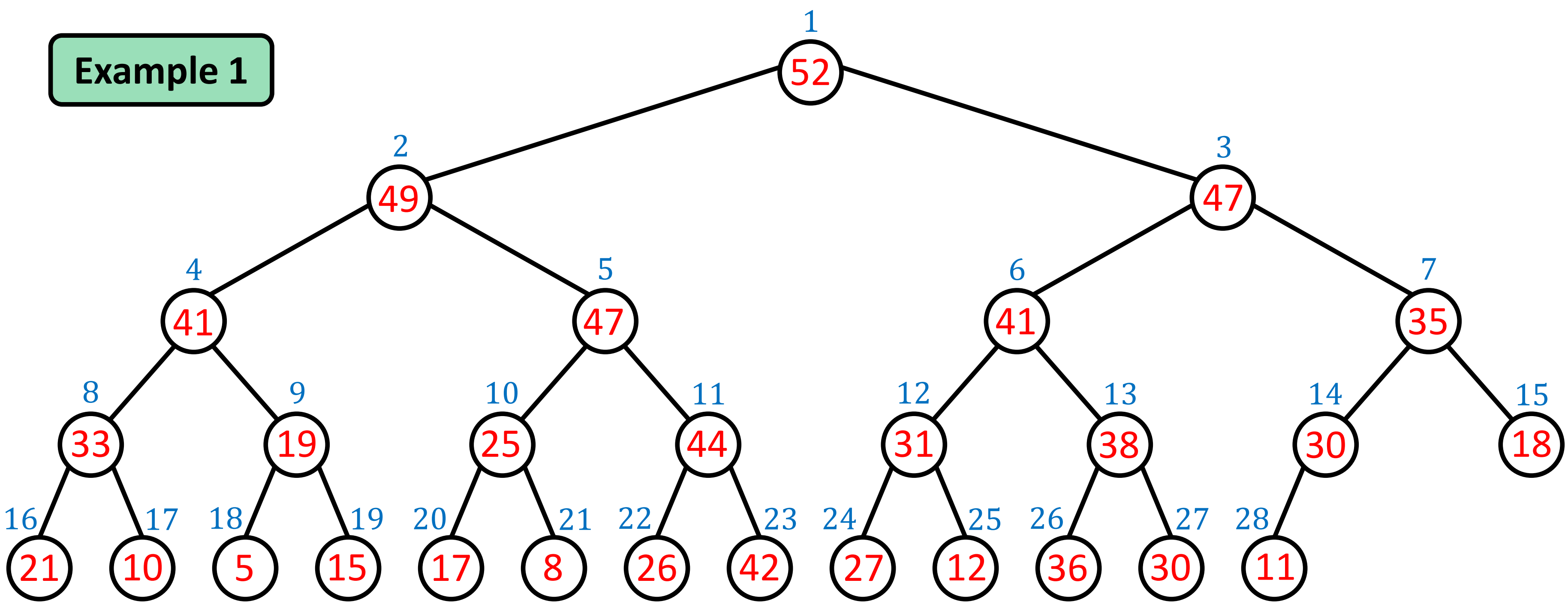
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	47	41	47	41	35	33	19	25	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

Now $A[3] = 47$ does not violate the max-heap property because it is not larger than its parent $A[1] = 52$.

So, we now have a valid max-heap.

Example 1

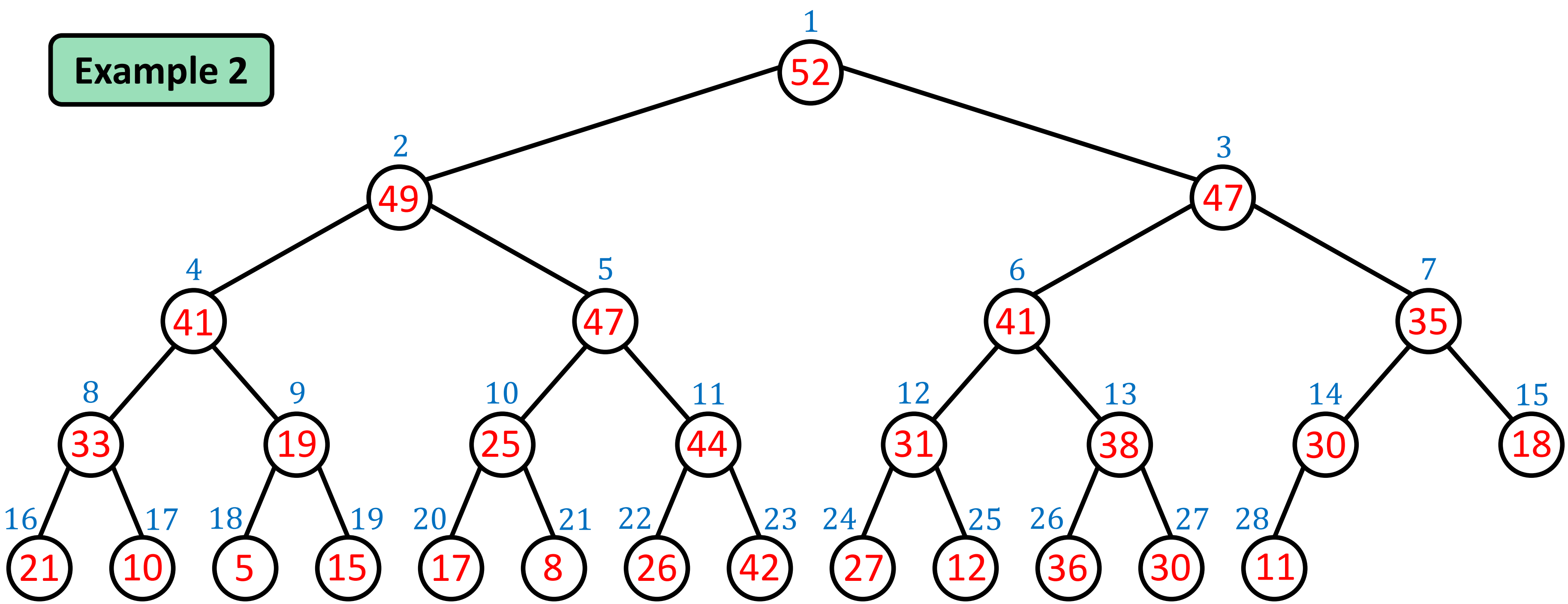


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	47	41	47	41	35	33	19	25	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

Now, suppose we want to increase $A[10]$ from 25 to 55.

Example 2



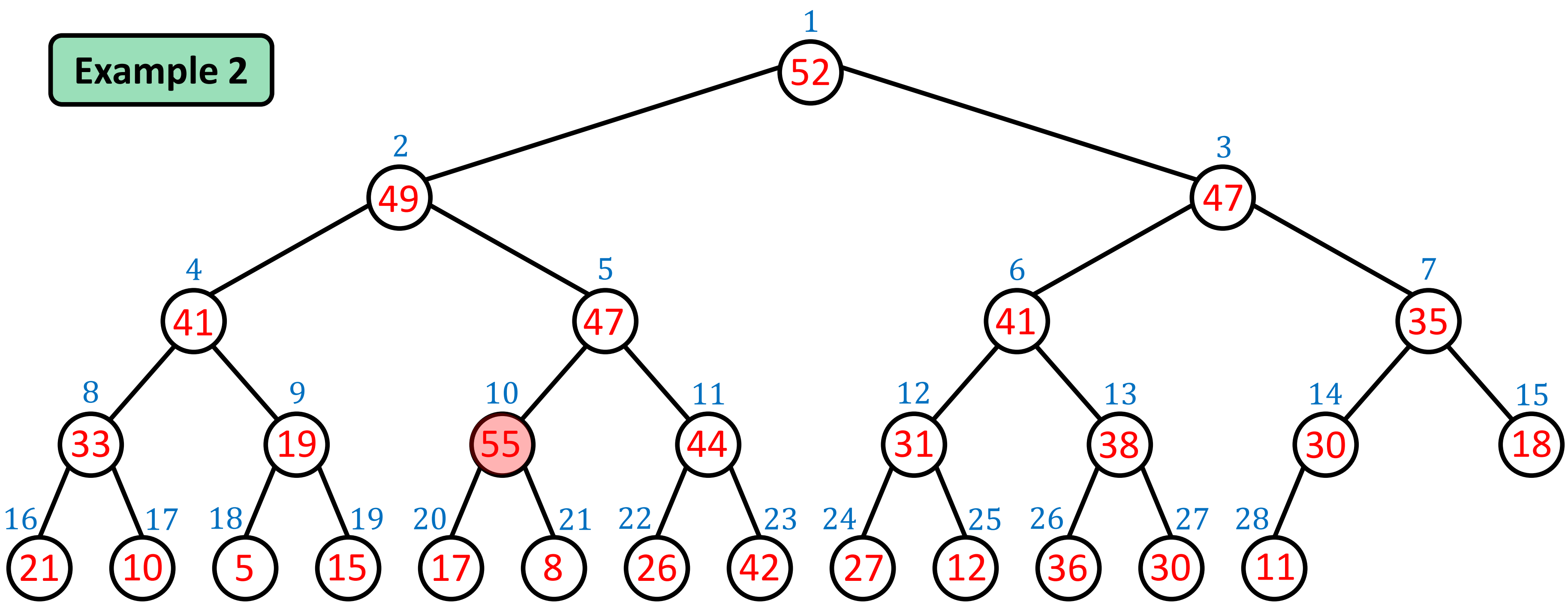
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	47	41	47	41	35	33	19	25	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

Now, suppose we want to increase $A[10]$ from 25 to 55.

So, we set $A[10] = 55$.

Example 2

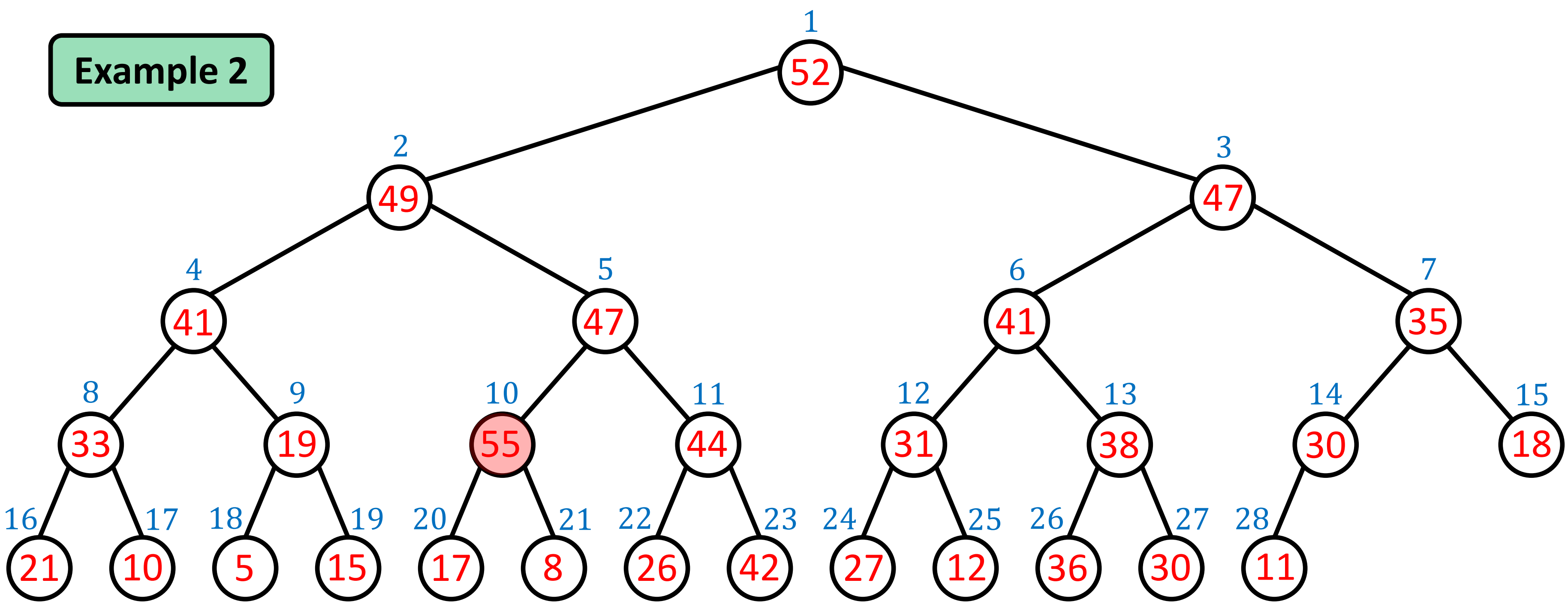


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	47	41	47	41	35	33	19	55	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

But $A[10] = 55$ violates the max-heap property because it is larger than its parent $A[5] = 47$.

Example 2



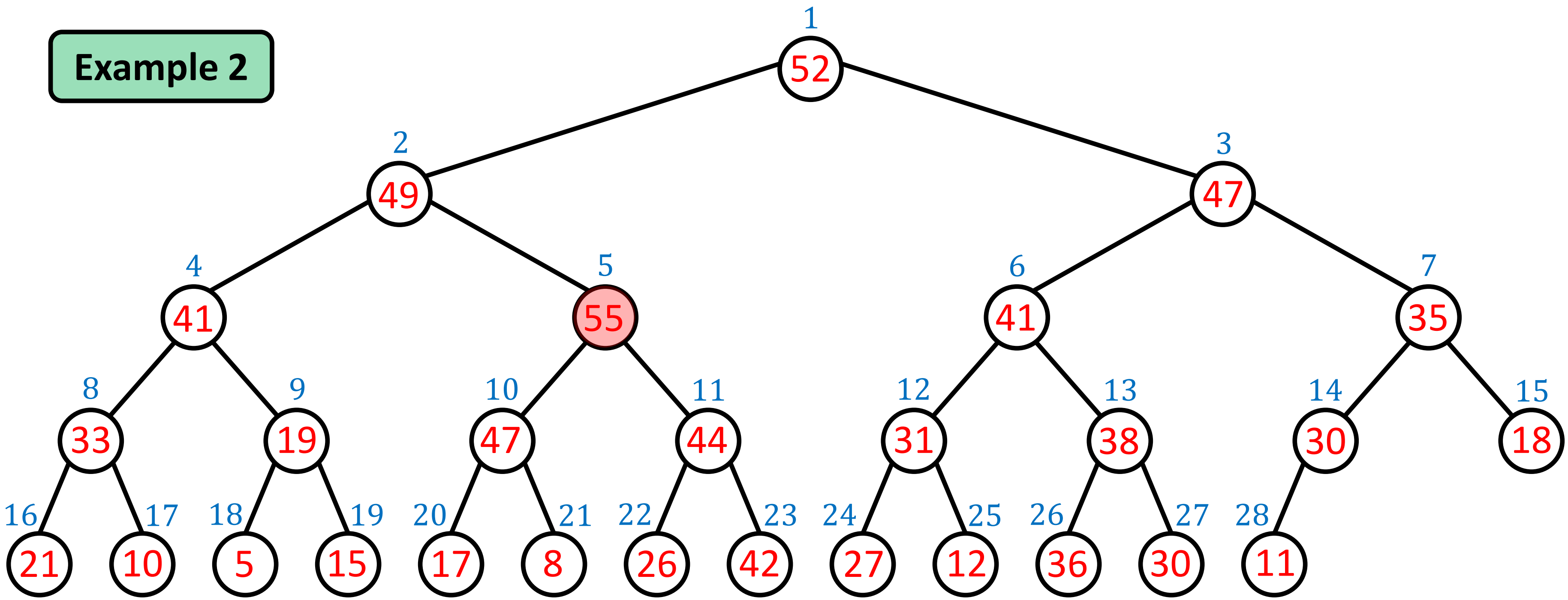
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	47	41	47	41	35	33	19	55	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

But $A[10] = 55$ violates the max-heap property because it is larger than its parent $A[5] = 47$.

We fix it by swapping $A[5]$ and $A[10]$.

Example 2

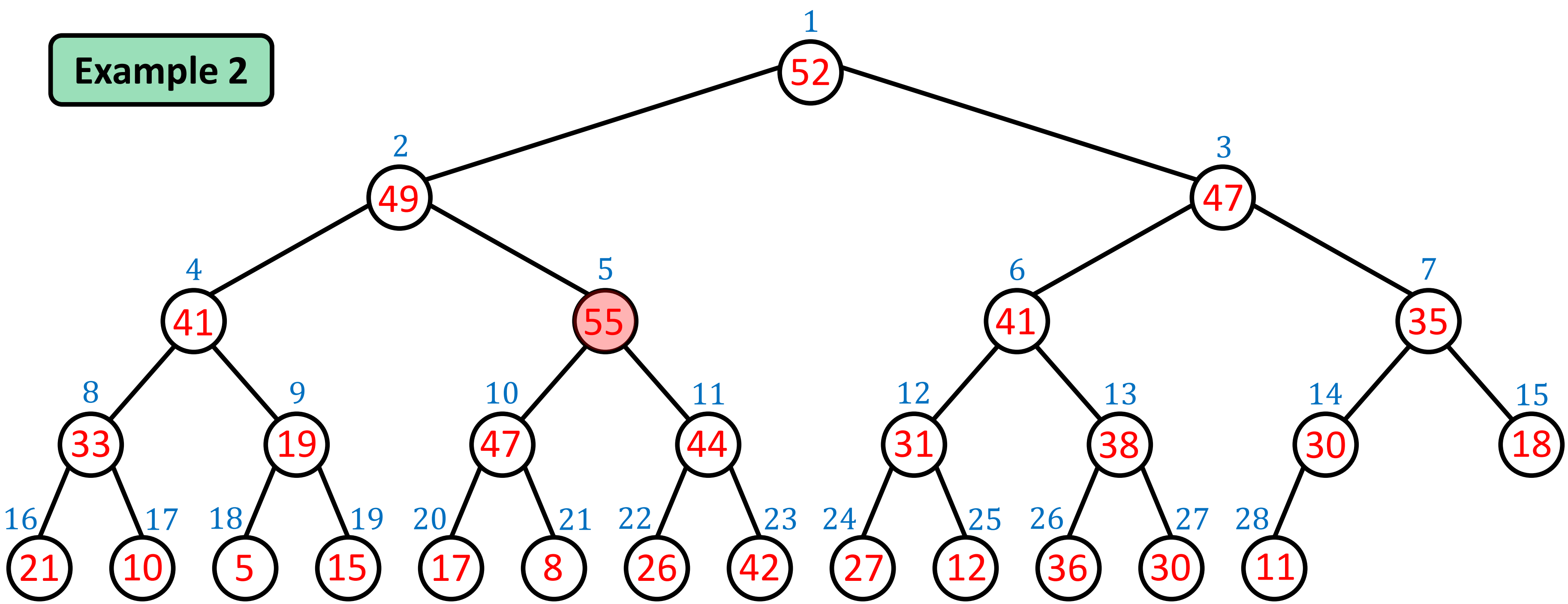


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	47	41	55	41	35	33	19	47	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

But now $A[5] = 55$ violates the max-heap property because it is larger than its parent $A[2] = 49$.

Example 2



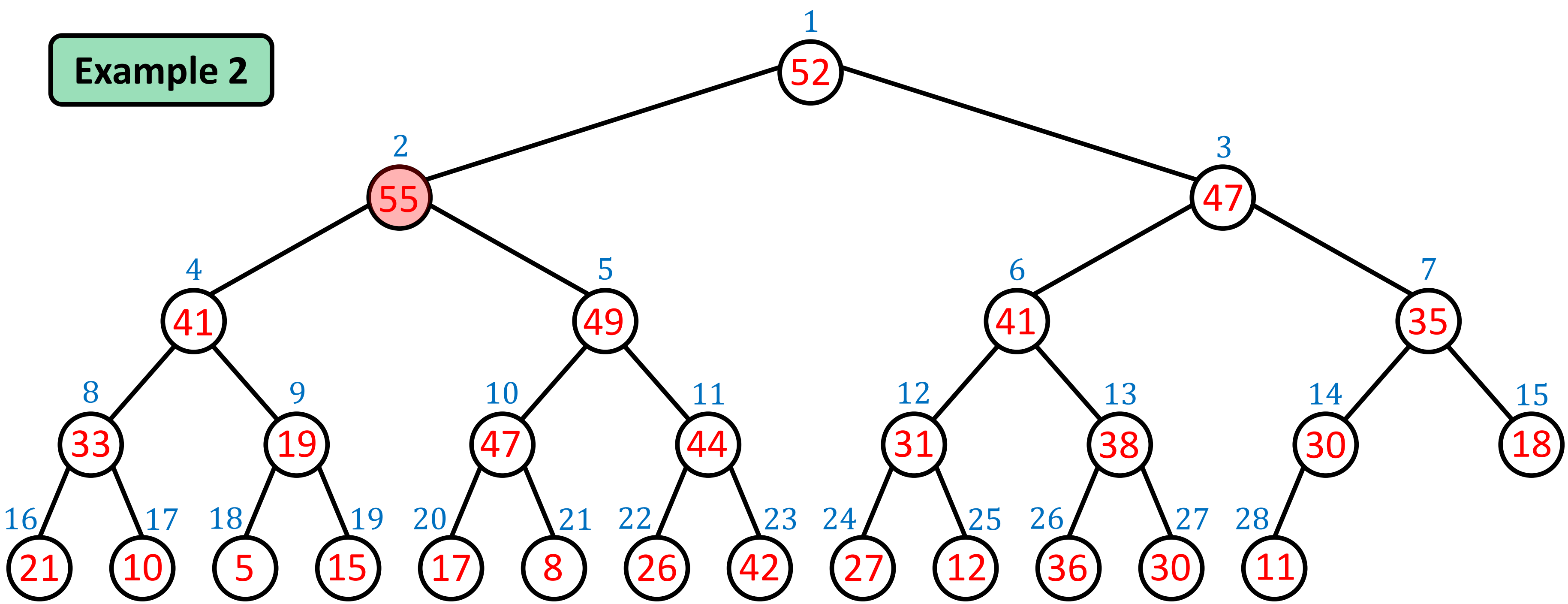
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	47	41	55	41	35	33	19	47	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

But now $A[5] = 55$ violates the max-heap property because it is larger than its parent $A[2] = 49$.

We fix it by swapping $A[2]$ and $A[5]$.

Example 2

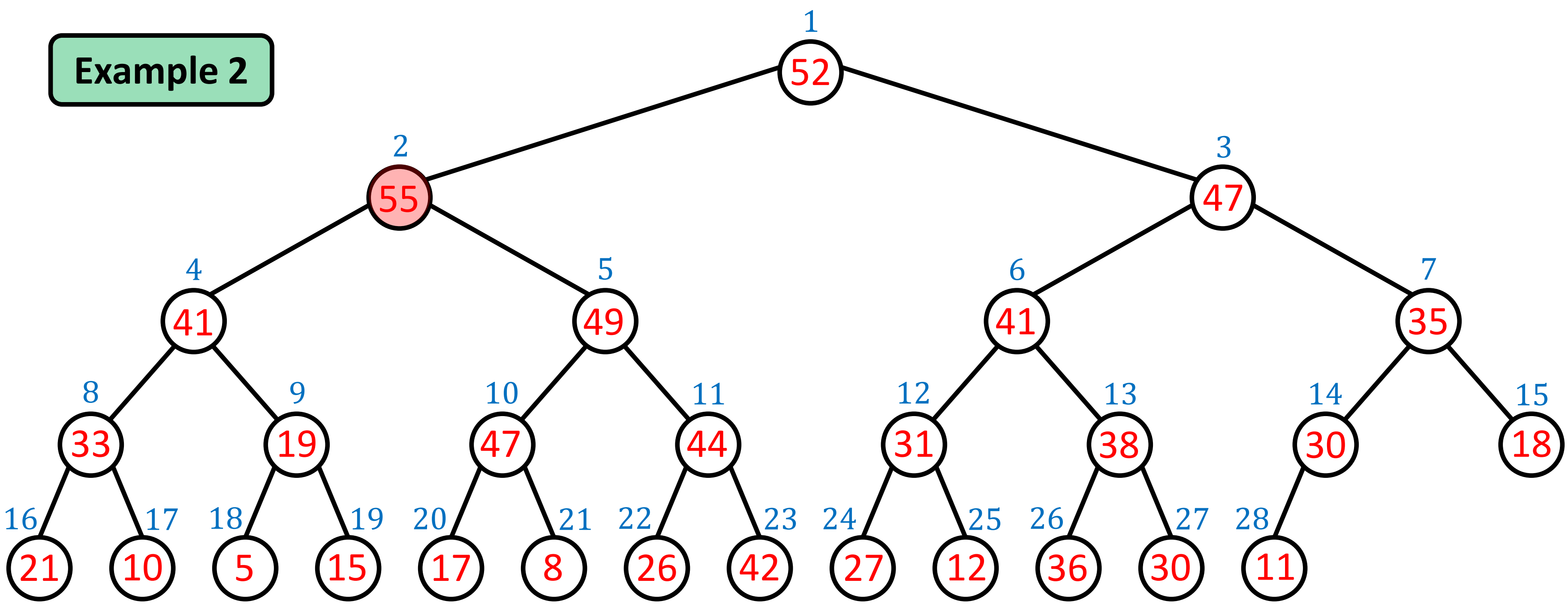


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	55	47	41	49	41	35	33	19	47	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

But now $A[2] = 55$ violates the max-heap property because it is larger than its parent $A[1] = 52$.

Example 2



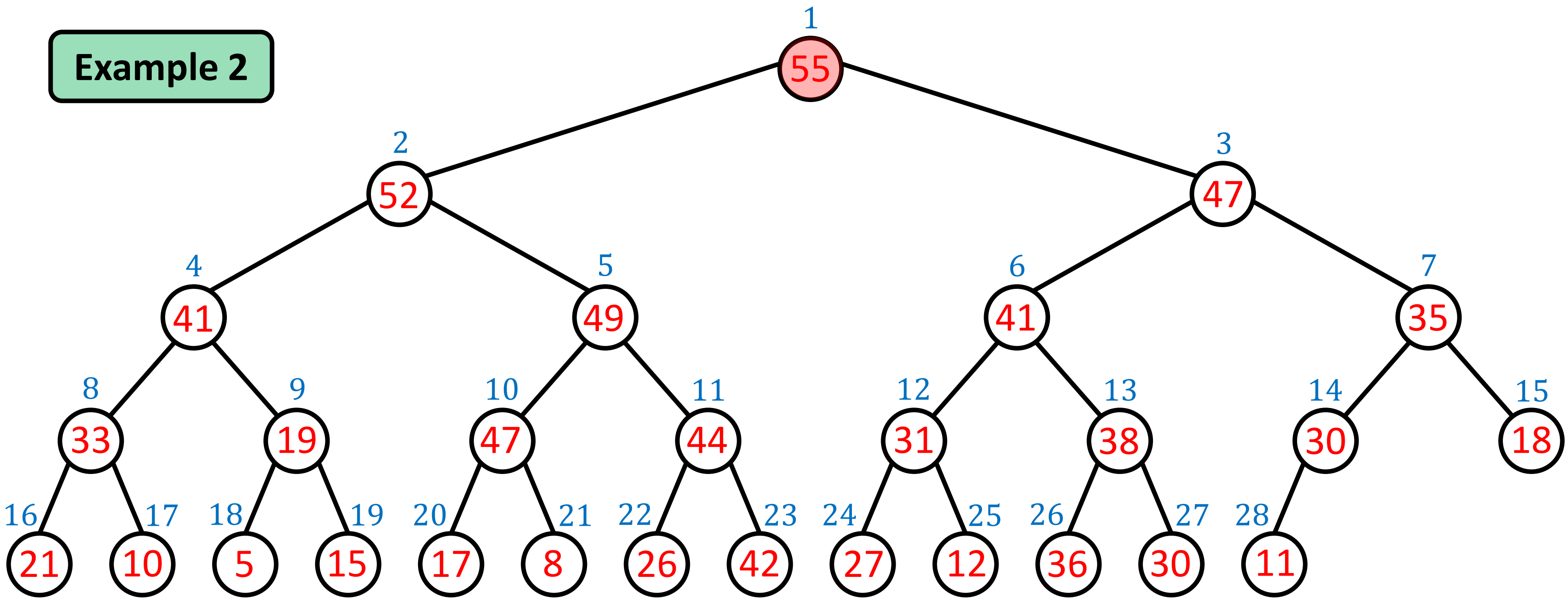
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	55	47	41	49	41	35	33	19	47	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

But now $A[2] = 55$ violates the max-heap property because it is larger than its parent $A[1] = 52$.

We fix it by swapping $A[1]$ and $A[2]$.

Example 2

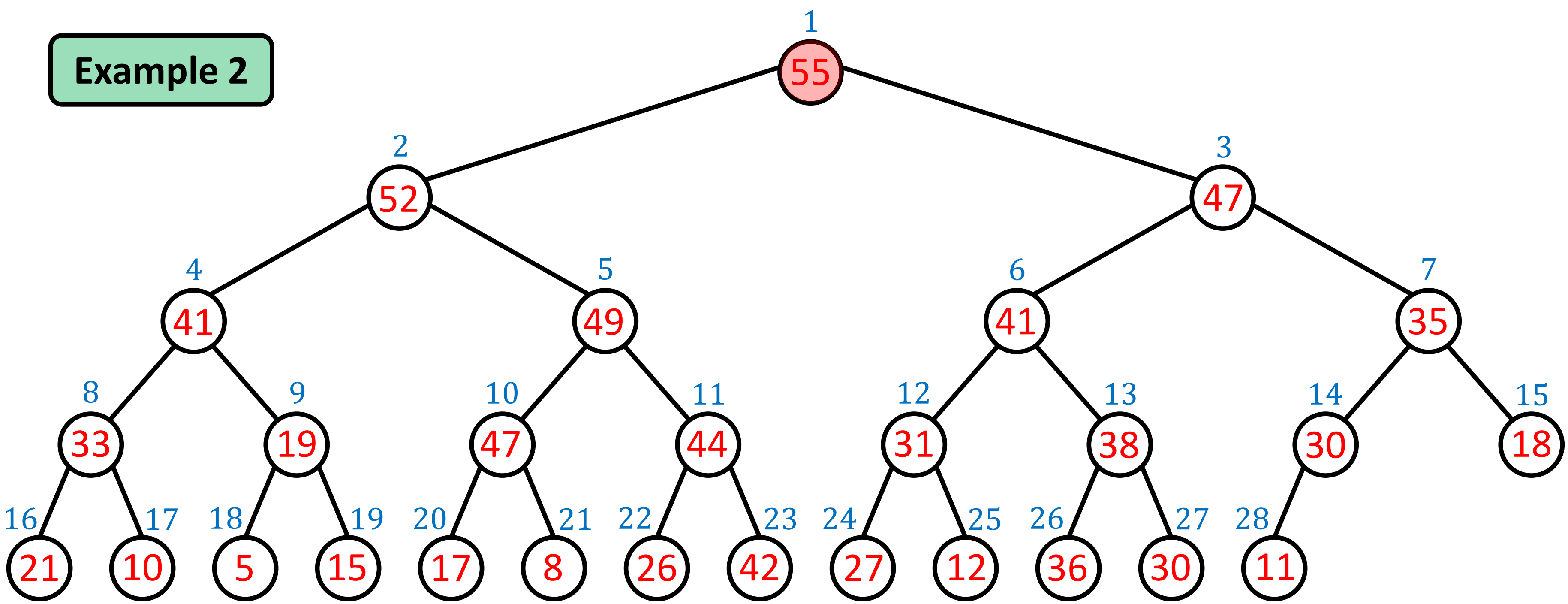


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	55	52	47	41	49	41	35	33	19	47	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

Now $A[1] = 55$ does not violate the max-heap property.

Example 2



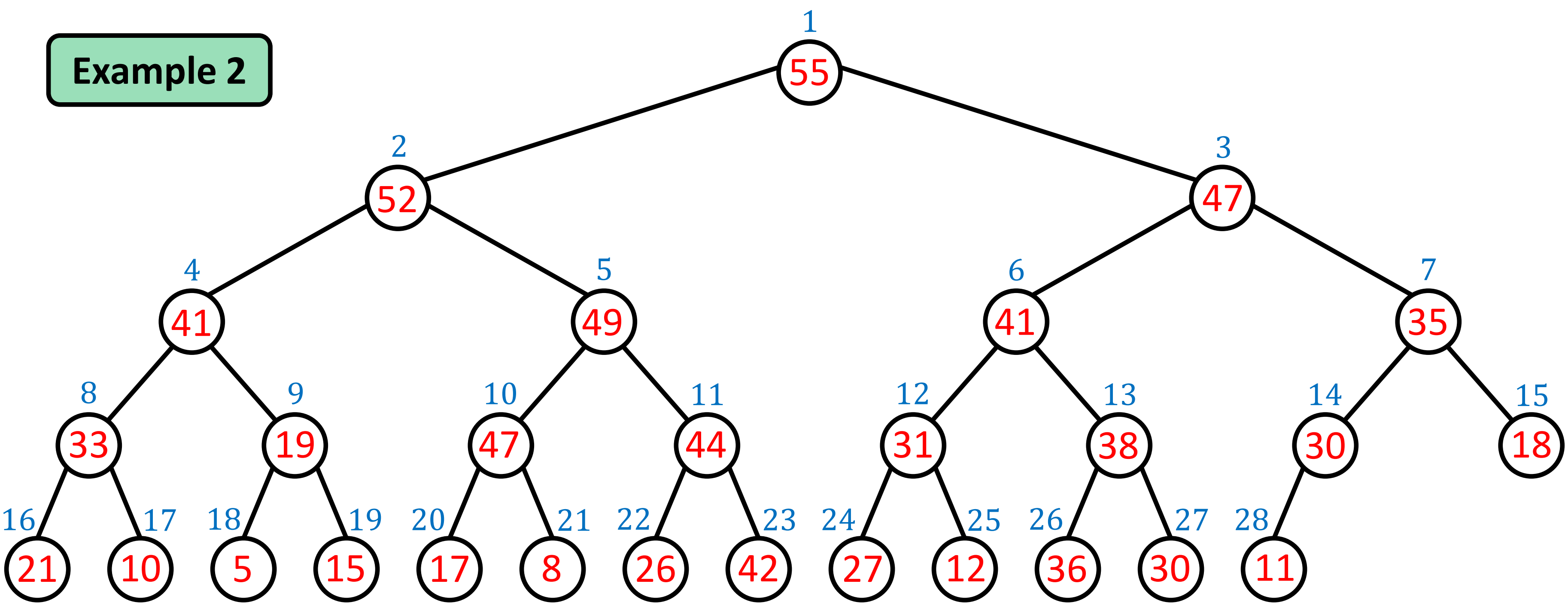
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	55	52	47	41	49	41	35	33	19	47	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

Increasing a Key

Now $A[1] = 55$ does not violate the max-heap property.

So, we now have a valid max-heap.

Example 2



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	55	52	47	41	49	41	35	33	19	47	44	31	38	30	18	21	10	5	15	17	8	26	42	27	12	36	30	11

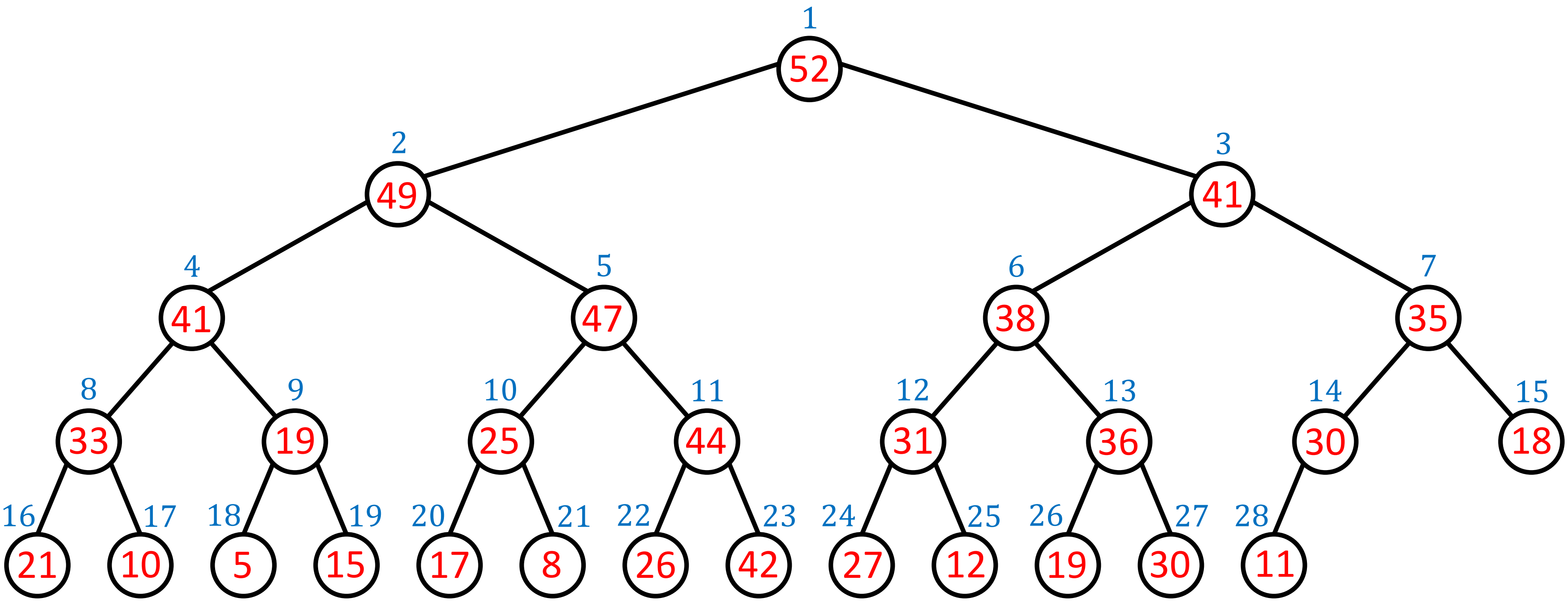
A Max-Heap as a Max-Priority Queue

MAX-HEAP-INSERT (*A*, *key*)

1. $A.heapsize = A.heapsize + 1$
2. $A[A.heapsize] = -\infty$
3. HEAP-INCREASE-KEY (A, $A.heapsize$, key)

Inserting a New Key

Suppose we want to insert a new key 50 into the following max-heap.

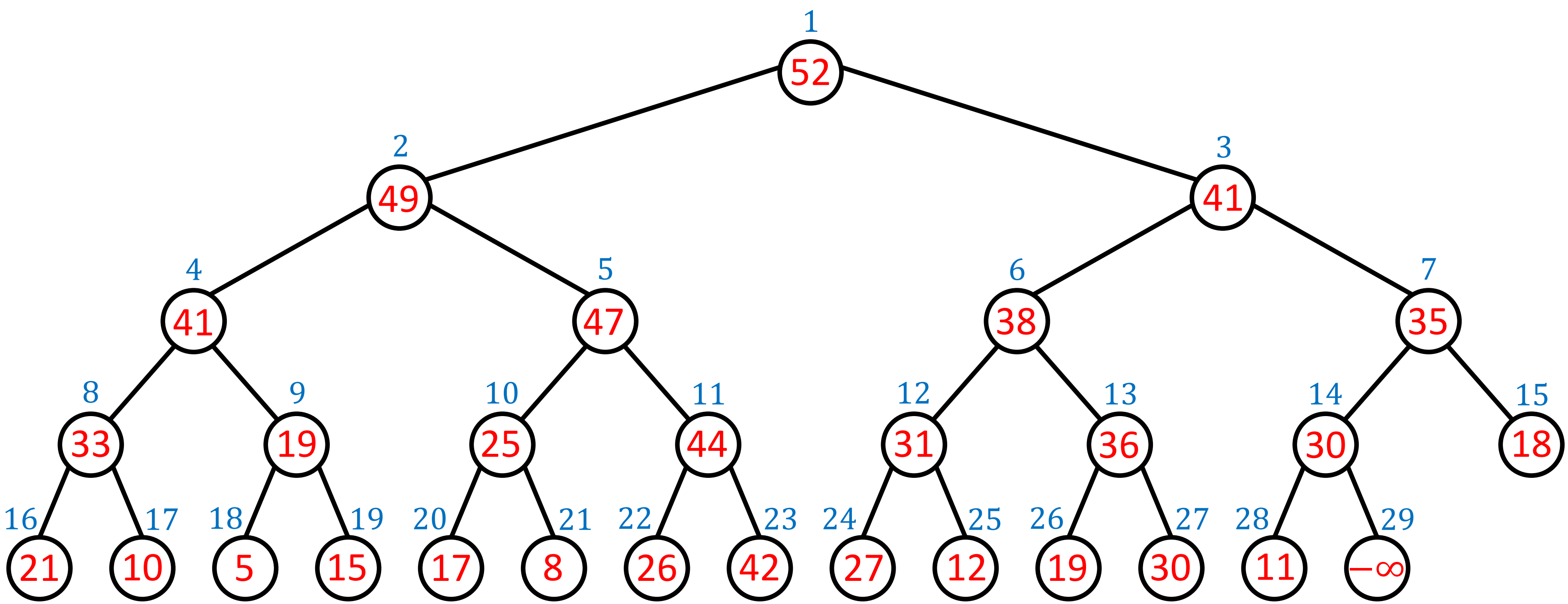


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
A	52	49	41	41	47	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11

Inserting a New Key

Suppose we want to insert a new key 50 into the following max-heap.

So, we create a new location $A[29]$ and set $A[29] = -\infty$.



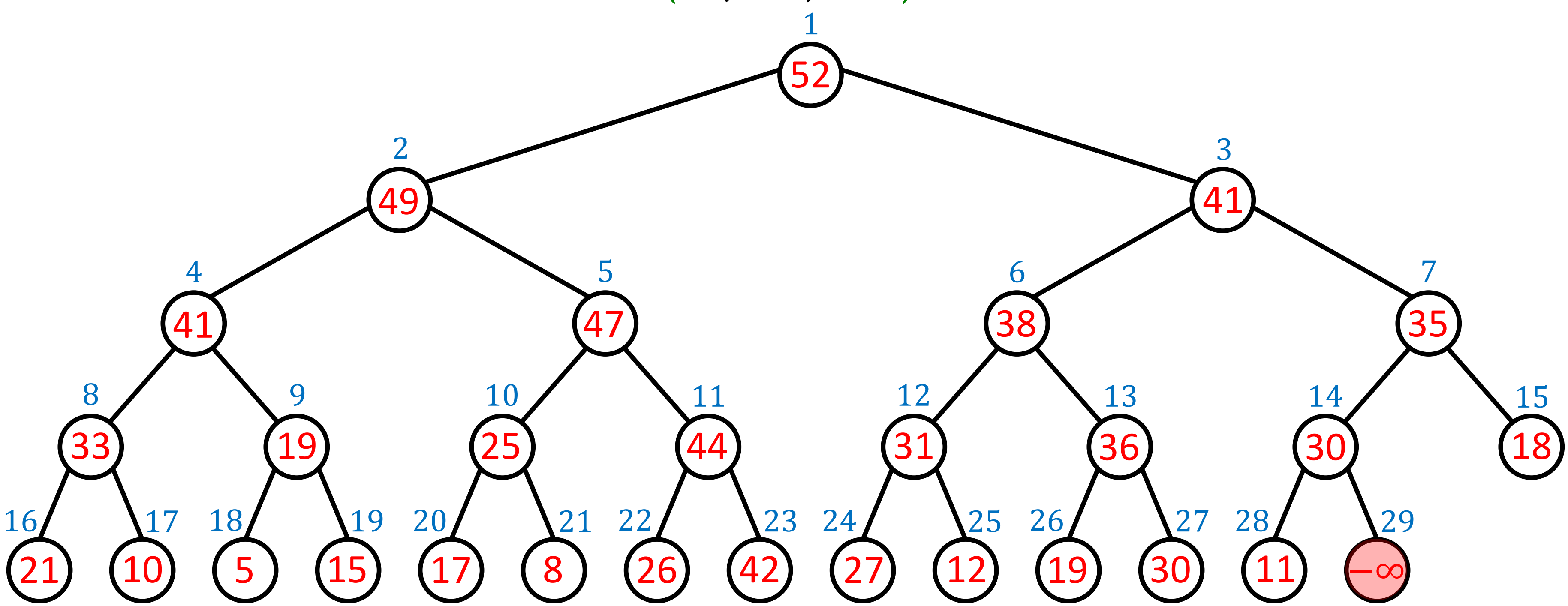
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
A	52	49	41	41	47	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11	$-\infty$

Inserting a New Key

Suppose we want to insert a new key 50 into the following max-heap.

So, we create a new location $A[29]$ and set $A[29] = -\infty$.

Then we call HEAP-INCREASE-KEY(A, 29, 50)



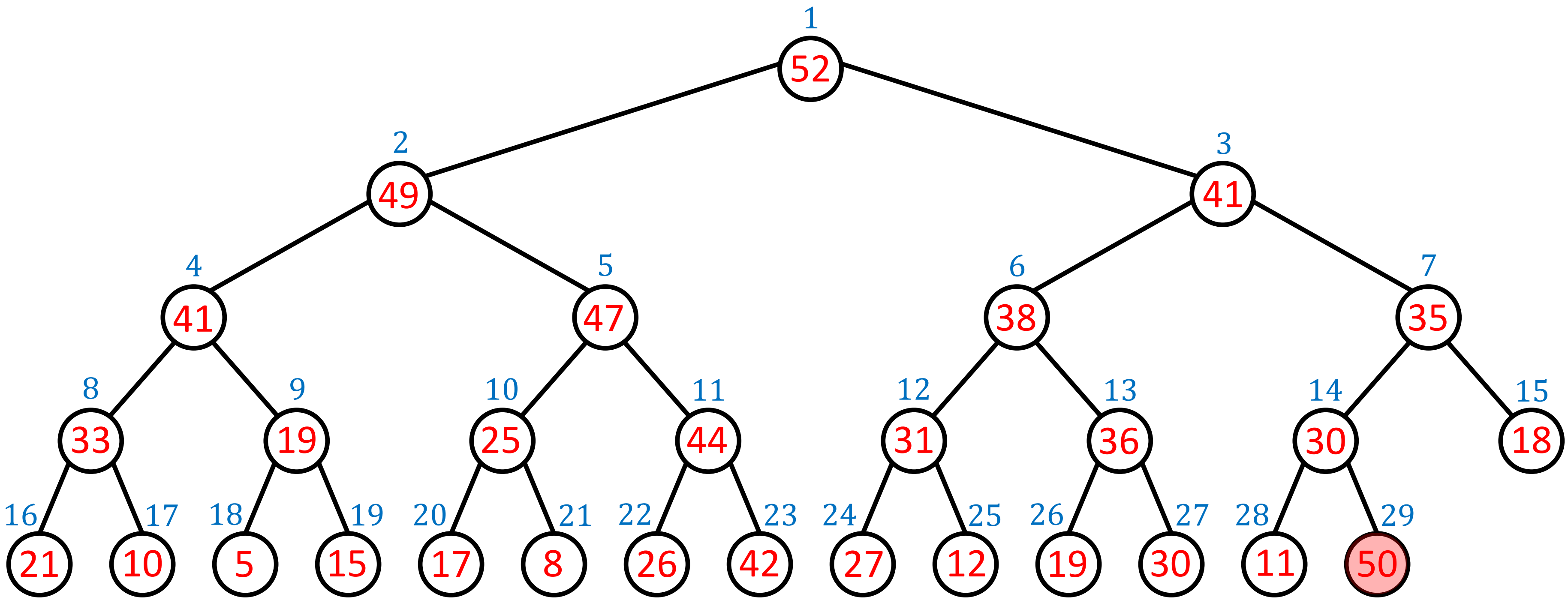
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
A	52	49	41	41	47	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11	$-\infty$

Inserting a New Key

Suppose we want to insert a new key 50 into the following max-heap.

So, we create a new location $A[29]$ and set $A[29] = -\infty$.

Then we call HEAP-INCREASE-KEY(A, 29, 50)



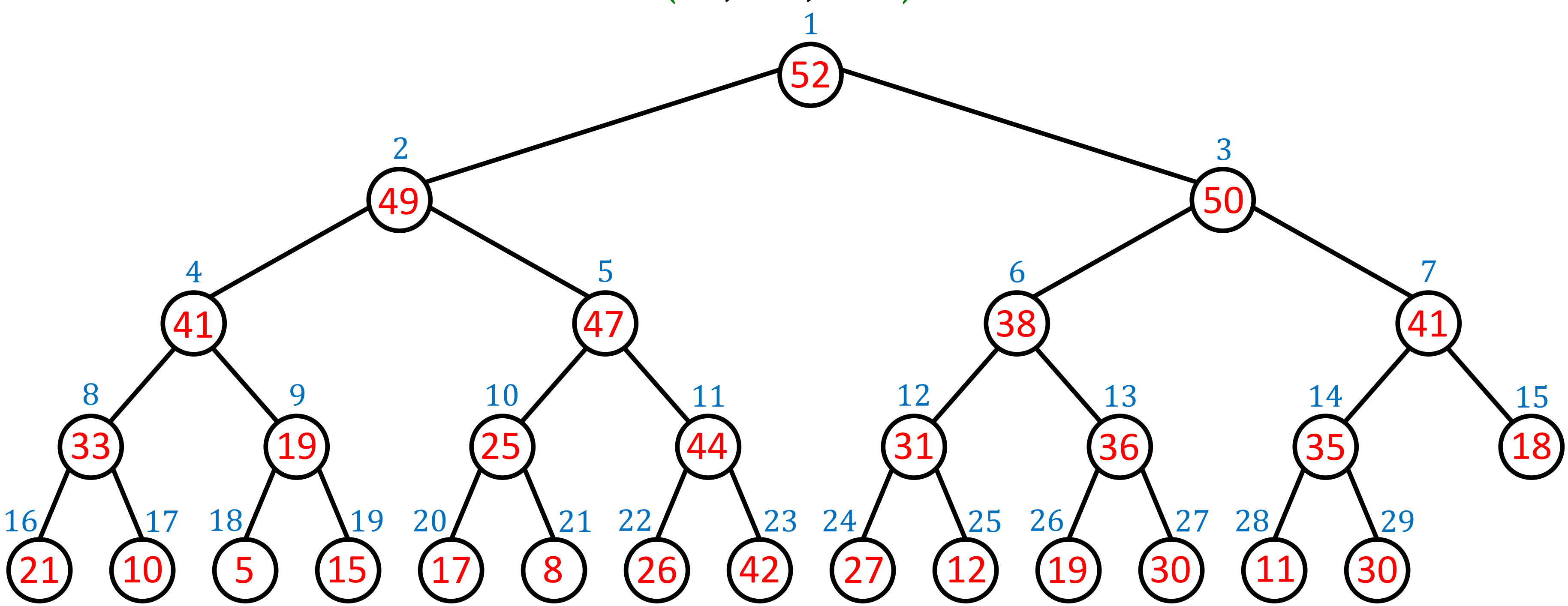
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
A	52	49	41	41	47	38	35	33	19	25	44	31	36	30	18	21	10	5	15	17	8	26	42	27	12	19	30	11	50

Inserting a New Key

Suppose we want to insert a new key 50 into the following max-heap.

So, we create a new location $A[29]$ and set $A[29] = -\infty$.

Then we call HEAP-INCREASE-KEY(A, 29, 50)



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
A	52	49	50	41	47	38	41	33	19	25	44	31	36	35	18	21	10	5	15	17	8	26	42	27	12	19	30	11	30

A Max-Heap as a Max-Priority Queue

Heap Operation	Max-Heap (worst-case)
BUILD-HEAP	$O(n)$
INSERT	$O(\log n)$
MAXIMUM	$\Theta(1)$
EXTRACT-MAX	$O(\log n)$
INCREASE-KEY	$O(\log n)$
DELETE	$O(\log n)$