DCA3702

MinHeap

ivanovitch.silva@ufrn.br

Properties of a Min-Heap

P1: A Min-Heap is a complete binary tree, meaning all levels of the tree are fully filled except possibly for the last level, which is filled from left to right.

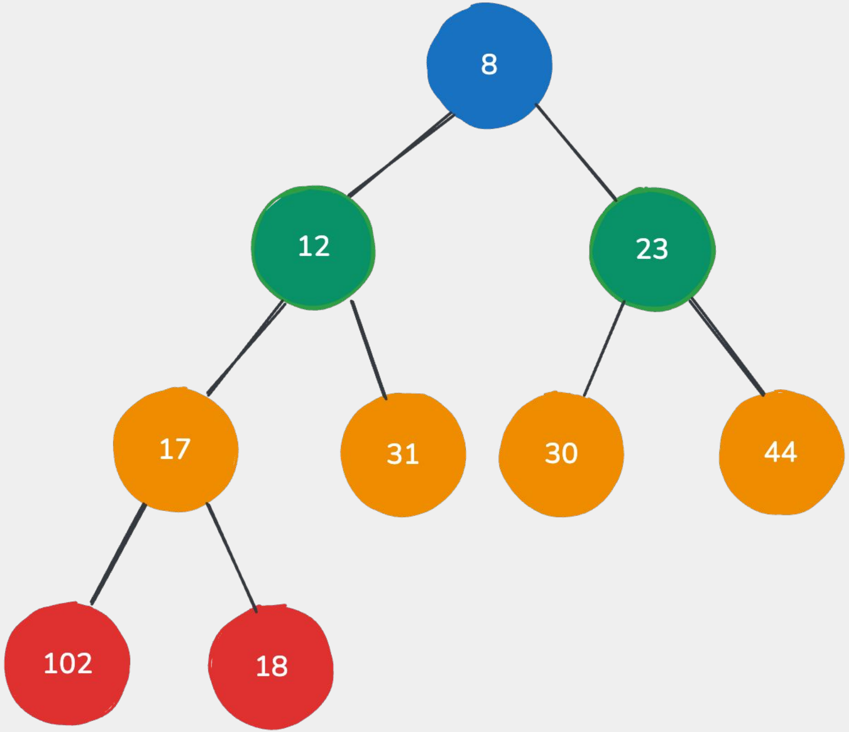
P2: Every node must be less than or equal to its children.

P3: Array Representation:
A Min-Heap can be efficiently represented using an array, where:

- The parent of a node at index *i* is located at *(i - 1) // 2*
- The left child is at *2 * i + 1*
- The right child is at *2 * i + 2*

P4: Main Operations

- **Build Heap:** Constructs the heap from an unsorted array.
- **Sift Down:** Reorganizes the heap after removing or replacing the top element.
- **Sift Up:** Reorganizes the heap after inserting a new element.
- **Insert:** Adds a new element to the heap and restores the heap property.
- **Remove:** Removes the smallest element (the root) and restructures the heap.



Index (i)	0	1	2	3	4	5	6	7	8

Array [8, 12, 23, 17, 31, 30, 44, 102, 18]

Properties of a Min-Heap

P1: A Min-Heap is a complete binary tree, meaning all levels of the tree are fully filled except possibly for the last level, which is filled from left to right.

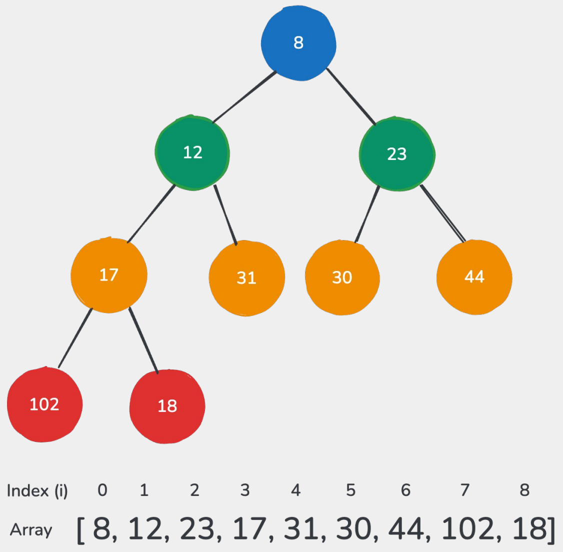
P2: Every node must be less than or equal to its children.

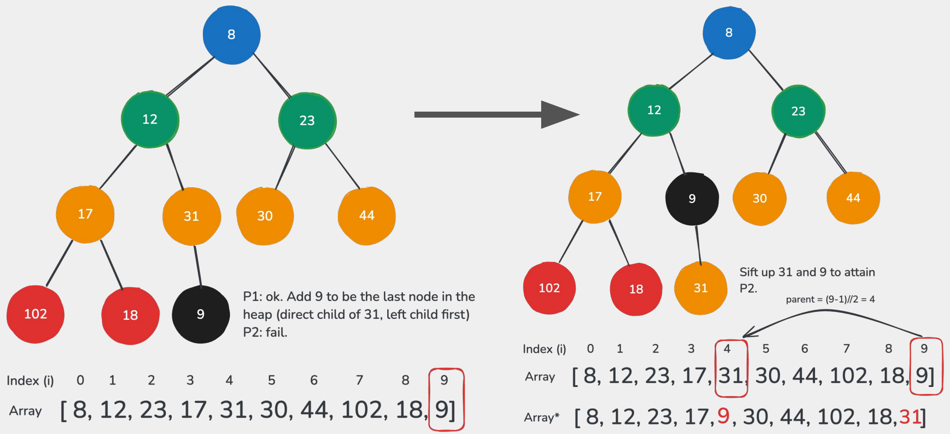
P3: Array Representation:
A Min-Heap can be efficiently represented using an array, where:

- The parent of a node at index *i* is located at *(i - 1) // 2*
- The left child is at *2 * i + 1*
- The right child is at *2 * i + 2*

P4: Main Operations

- **Build Heap:** Constructs the heap from an unsorted array.
- **Sift Down:** Reorganizes the heap after removing or replacing the top element.
- **Sift Up:** Reorganizes the heap after inserting a new element.
- **Insert:** Adds a new element to the heap and restores the heap property.
- **Remove:** Removes the smallest element (the root) and restructures the heap.





Properties of a Min-Heap

P1: A Min-Heap is a complete binary tree, meaning all levels of the tree are fully filled except possibly for the last level, which is filled from left to right.

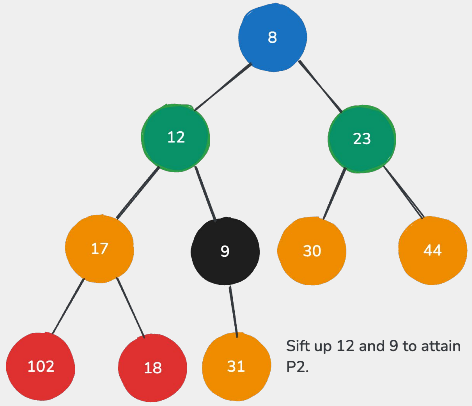
P2: Every node must be less than or equal to its children.

P3: Array Representation:
A Min-Heap can be efficiently represented using an array, where:

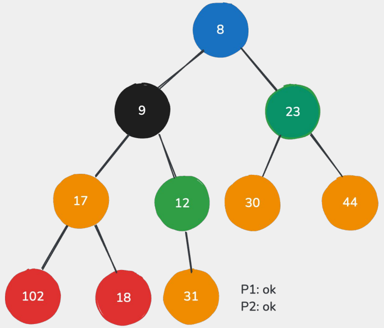
- The parent of a node at index *i* is located at *(i - 1) // 2*
- The left child is at *2 * i + 1*
- The right child is at *2 * i + 2*

P4: Main Operations

- **Build Heap:** Constructs the heap from an unsorted array.
- **Sift Down:** Reorganizes the heap after removing or replacing the top element.
- **Sift Up:** Reorganizes the heap after inserting a new element.
- **Insert:** Adds a new element to the heap and restores the heap property.
- **Remove:** Removes the smallest element (the root) and restructures the heap.



Index (i) 0 1 2 3 4 5 6 7 8 9
Array [8, 12, 23, 17, 9, 30, 44, 102, 18, 31]



Index (i) 0 1 2 3 4 5 6 7 8 9
Array [8, 12, 23, 17, 9, 30, 44, 102, 18, 31]

parent = (4-1)//2 = 1

Array* [8, 9, 23, 17, 12, 30, 44, 102, 18, 31]

Time Complexity - **O(log n)**
Space Complexity - **O(1)**

Properties of a Min-Heap

P1: A Min-Heap is a complete binary tree, meaning all levels of the tree are fully filled except possibly for the last level, which is filled from left to right.

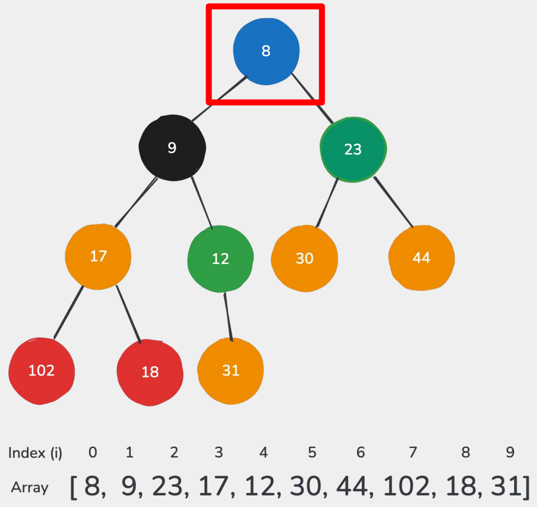
P2: Every node must be less than or equal to its children.

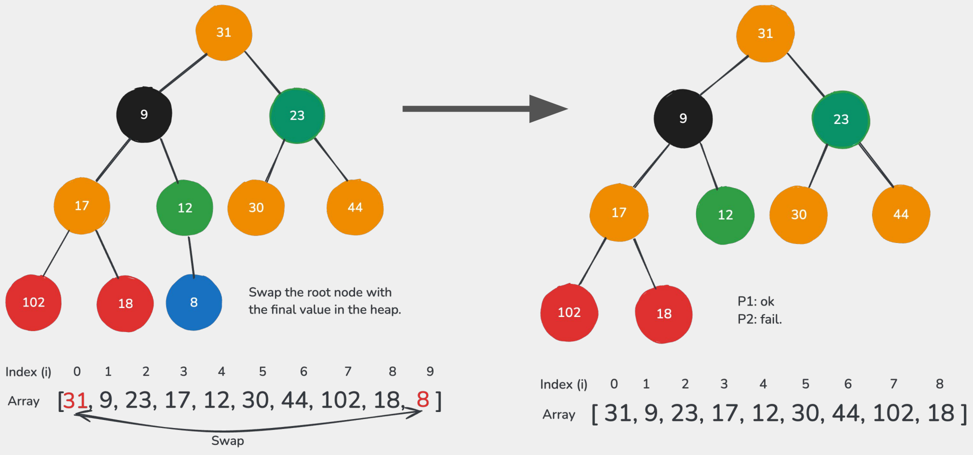
P3: Array Representation:
A Min-Heap can be efficiently represented using an array, where:

- The parent of a node at index *i* is located at *(i - 1) // 2*
- The left child is at *2 * i + 1*
- The right child is at *2 * i + 2*

P4: Main Operations

- **Build Heap:** Constructs the heap from an unsorted array.
- **Sift Down:** Reorganizes the heap after removing or replacing the top element.
- **Sift Up:** Reorganizes the heap after inserting a new element.
- **Insert:** Adds a new element to the heap and restores the heap property.
- **Remove:** Removes the smallest element (the root) and restructures the heap.





Properties of a Min-Heap

P1: A Min-Heap is a complete binary tree, meaning all levels of the tree are fully filled except possibly for the last level, which is filled from left to right.

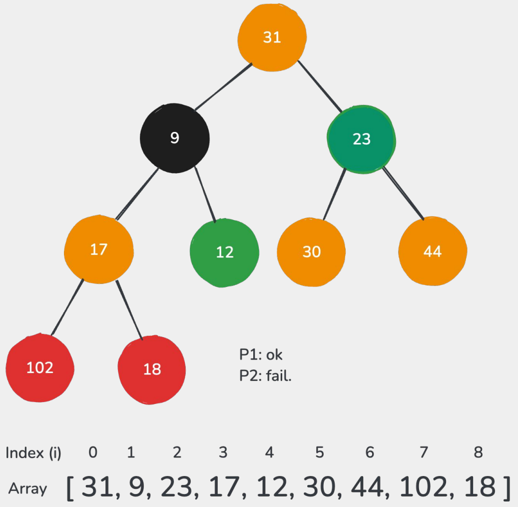
P2: Every node must be less than or equal to its children.

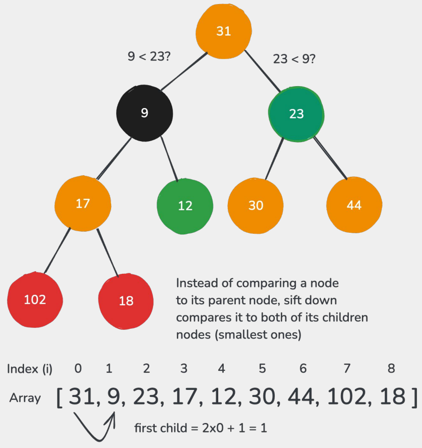
P3: Array Representation:
A Min-Heap can be efficiently represented using an array, where:

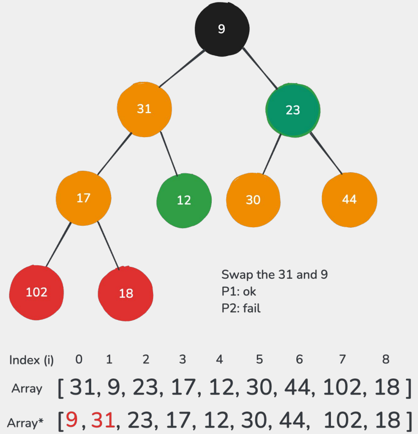
- The parent of a node at index *i* is located at *(i - 1) // 2*
- The left child is at *2 * i + 1*
- The right child is at *2 * i + 2*

P4: Main Operations

- **Build Heap:** Constructs the heap from an unsorted array.
- **Sift Down:** Reorganizes the heap after removing or replacing the top element.
- **Sift Up:** Reorganizes the heap after inserting a new element.
- **Insert:** Adds a new element to the heap and restores the heap property.
- **Remove:** Removes the smallest element (the root) and restructures the heap.







Properties of a Min-Heap

P1: A Min-Heap is a complete binary tree, meaning all levels of the tree are fully filled except possibly for the last level, which is filled from left to right.

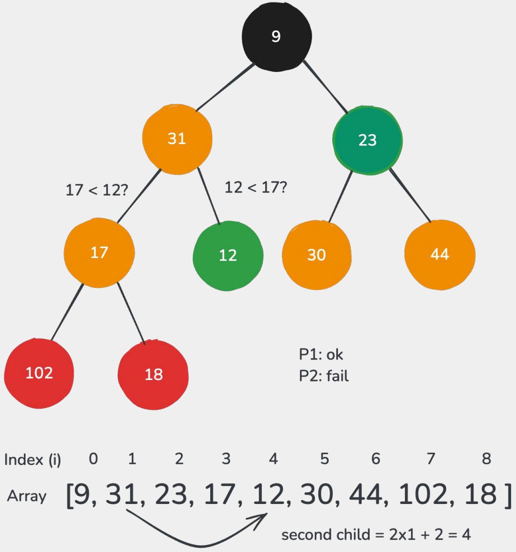
P2: Every node must be less than or equal to its children.

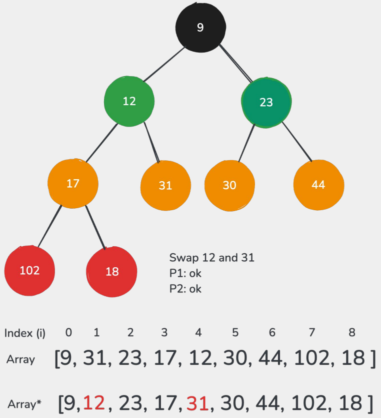
P3: Array Representation:
A Min-Heap can be efficiently represented using an array, where:

- The parent of a node at index *i* is located at *(i - 1) // 2*
- The left child is at *2 * i + 1*
- The right child is at *2 * i + 2*

P4: Main Operations

- **Build Heap:** Constructs the heap from an unsorted array.
- **Sift Down:** Reorganizes the heap after removing or replacing the top element.
- **Sift Up:** Reorganizes the heap after inserting a new element.
- **Insert:** Adds a new element to the heap and restores the heap property.
- **Remove:** Removes the smallest element (the root) and restructures the heap.





Time Complexity - **O(log n)**
Space Complexity - **O(1)**

How to calculate the time complexity?

$$N \rightarrow \frac{N}{2} \rightarrow \frac{N}{2^2} \rightarrow \frac{N}{2^3} \rightarrow \dots \rightarrow \frac{N}{2^k}$$

In the worst case scenario:

$$\frac{N}{2^k} = 1$$

$$2^k = N$$

$$log2^k = logN$$

$$k = logN$$

$$O(logN)$$

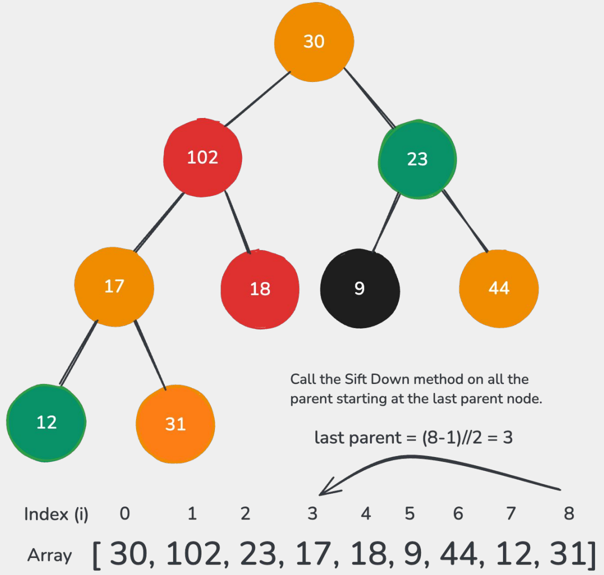
Properties of a Min-Heap

P1: A Min-Heap is a complete binary tree, meaning all levels of the tree are fully filled except possibly for the last level, which is filled from left to right.

P2: Every node must be less than or equal to its children.

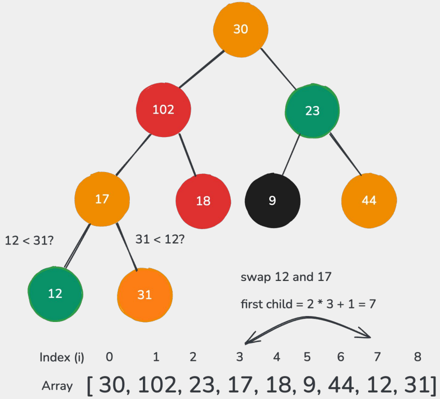
P3: Array Representation:
A Min-Heap can be efficiently represented using an array, where:

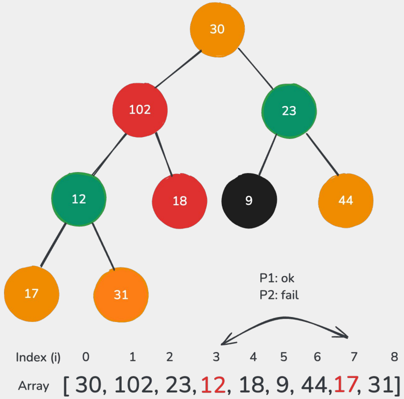
- The parent of a node at index *i* is located at *(i - 1) // 2*
- The left child is at *2 * i + 1*
- The right child is at *2 * i + 2*



P4: Main Operations

- **Build Heap:** Constructs the heap from an unsorted array.
- **Sift Down:** Reorganizes the heap after removing or replacing the top element.
- **Sift Up:** Reorganizes the heap after inserting a new element.
- **Insert:** Adds a new element to the heap and restores the heap property.
- **Remove:** Removes the smallest element (the root) and restructures the heap.





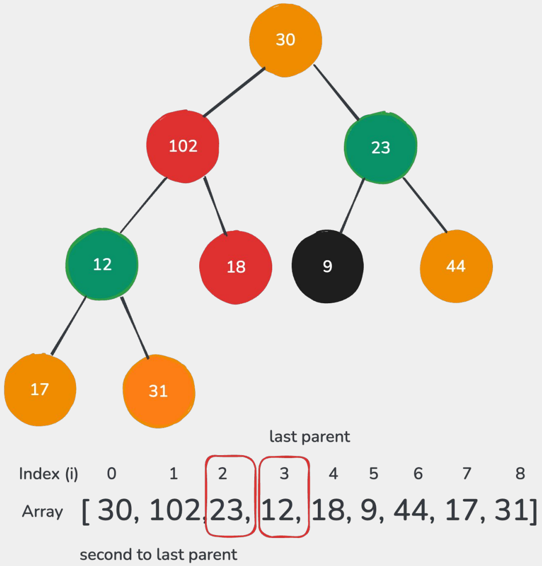
Properties of a Min-Heap

P1: A Min-Heap is a complete binary tree, meaning all levels of the tree are fully filled except possibly for the last level, which is filled from left to right.

P2: Every node must be less than or equal to its children.

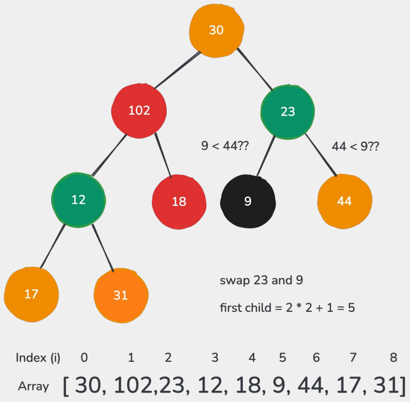
P3: Array Representation:
A Min-Heap can be efficiently represented using an array, where:

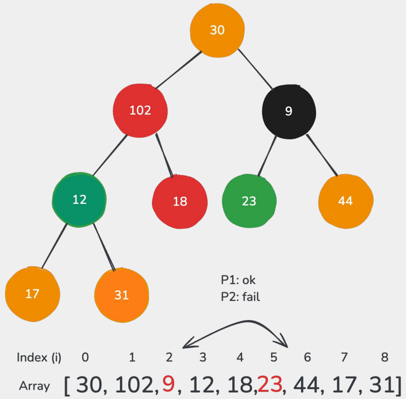
- The parent of a node at index *i* is located at *(i - 1) // 2*
- The left child is at *2 * i + 1*
- The right child is at *2 * i + 2*



P4: Main Operations

- **Build Heap:** Constructs the heap from an unsorted array.
- **Sift Down:** Reorganizes the heap after removing or replacing the top element.
- **Sift Up:** Reorganizes the heap after inserting a new element.
- **Insert:** Adds a new element to the heap and restores the heap property.
- **Remove:** Removes the smallest element (the root) and restructures the heap.





Properties of a Min-Heap

P1: A Min-Heap is a complete binary tree, meaning all levels of the tree are fully filled except possibly for the last level, which is filled from left to right.

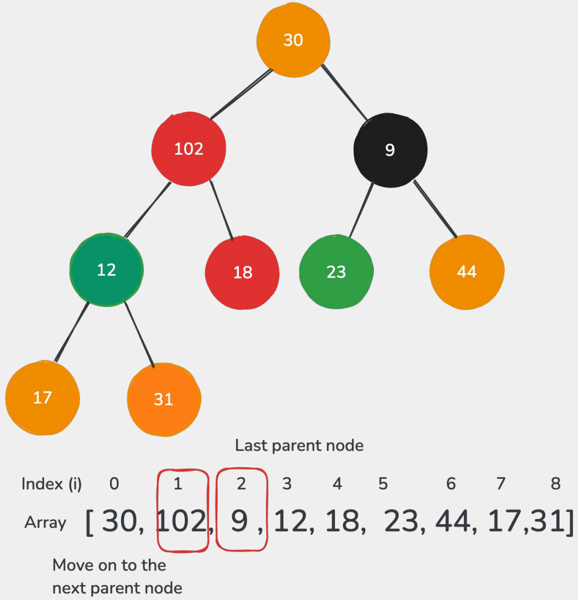
P2: Every node must be less than or equal to its children.

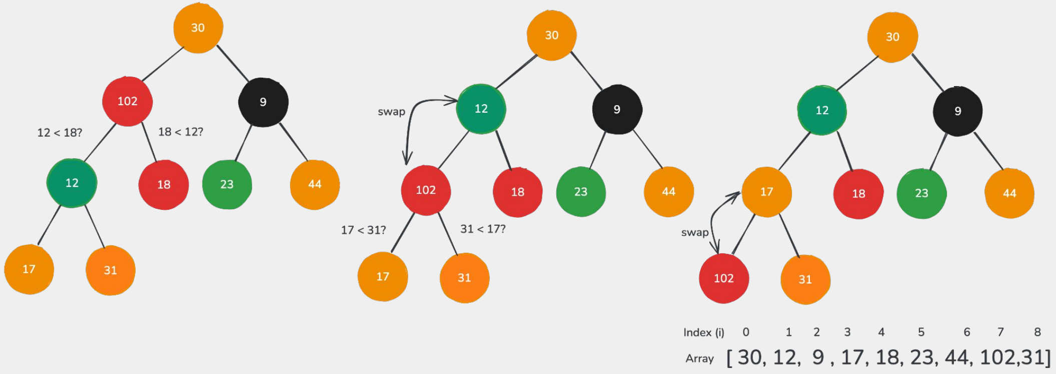
P3: Array Representation:
A Min-Heap can be efficiently represented using an array, where:

- The parent of a node at index *i* is located at *(i - 1) // 2*
- The left child is at *2 * i + 1*
- The right child is at *2 * i + 2*

P4: Main Operations

- **Build Heap:** Constructs the heap from an unsorted array.
- **Sift Down:** Reorganizes the heap after removing or replacing the top element.
- **Sift Up:** Reorganizes the heap after inserting a new element.
- **Insert:** Adds a new element to the heap and restores the heap property.
- **Remove:** Removes the smallest element (the root) and restructures the heap.





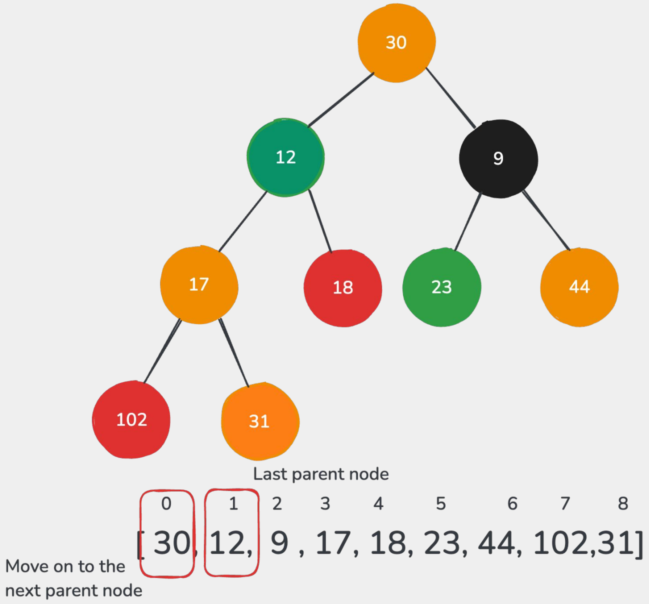
Properties of a Min-Heap

P1: A Min-Heap is a complete binary tree, meaning all levels of the tree are fully filled except possibly for the last level, which is filled from left to right.

P2: Every node must be less than or equal to its children.

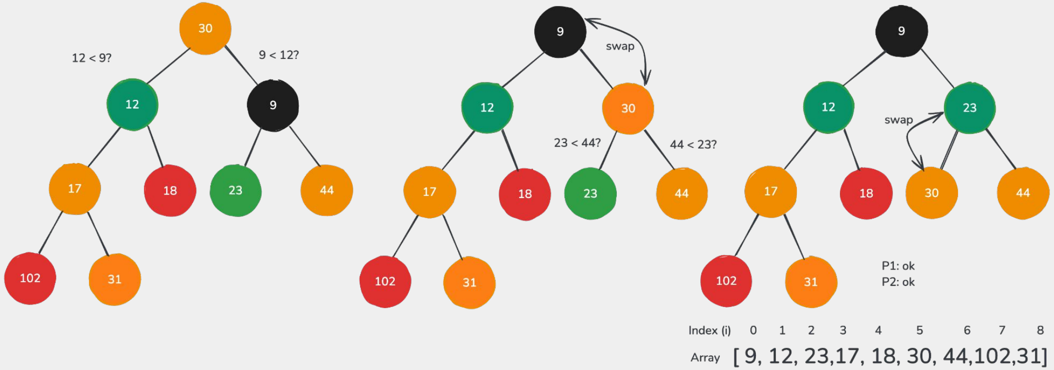
P3: Array Representation:
A Min-Heap can be efficiently represented using an array, where:

- The parent of a node at index *i* is located at *(i - 1) // 2*
- The left child is at *2 * i + 1*
- The right child is at *2 * i + 2*



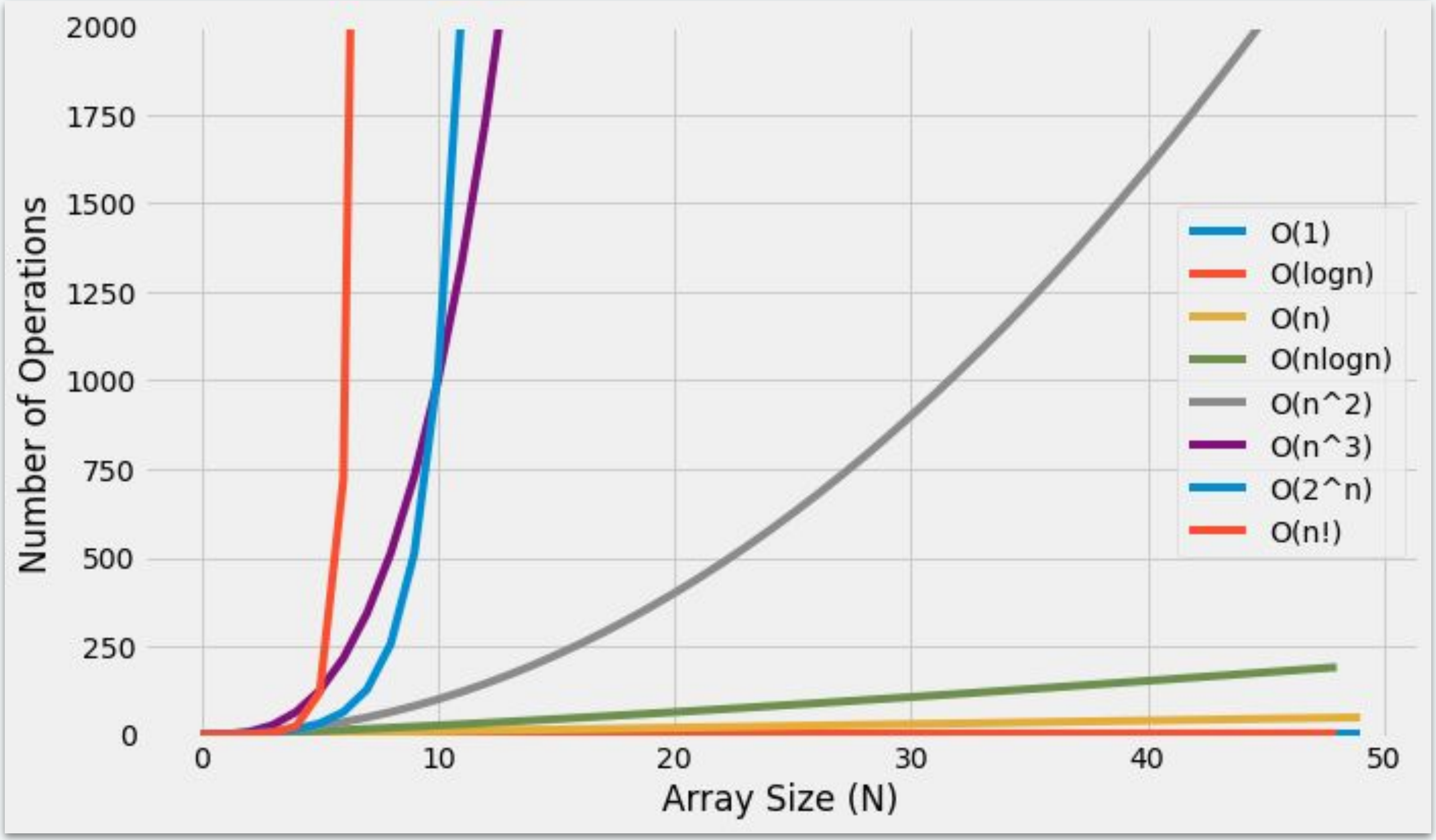
P4: Main Operations

- **Build Heap:** Constructs the heap from an unsorted array.
- **Sift Down:** Reorganizes the heap after removing or replacing the top element.
- **Sift Up:** Reorganizes the heap after inserting a new element.
- **Insert:** Adds a new element to the heap and restores the heap property.
- **Remove:** Removes the smallest element (the root) and restructures the heap.



Time and Space Complexity Analysis of MinHeap Operations

Method	Time Complexity	Space Complexity	Note
buildHeap	O(n)	O(1)	Use bottom-up heapify
siftDown	O(log n)	O(1)	Moves node downward
siftUp	O(log n)	O(1)	Moves node upward
insert	O(log n)	O(1)	Append and sift up
remove	O(log n)	O(1)	Swap root with last, pop, and sift down
peek	O(1)	O(1)	Direct access to index 0



$$O(1) < O(logn) < O(n) < O(nlogn) < O(n^2) < O(2^n) < O(n!)$$