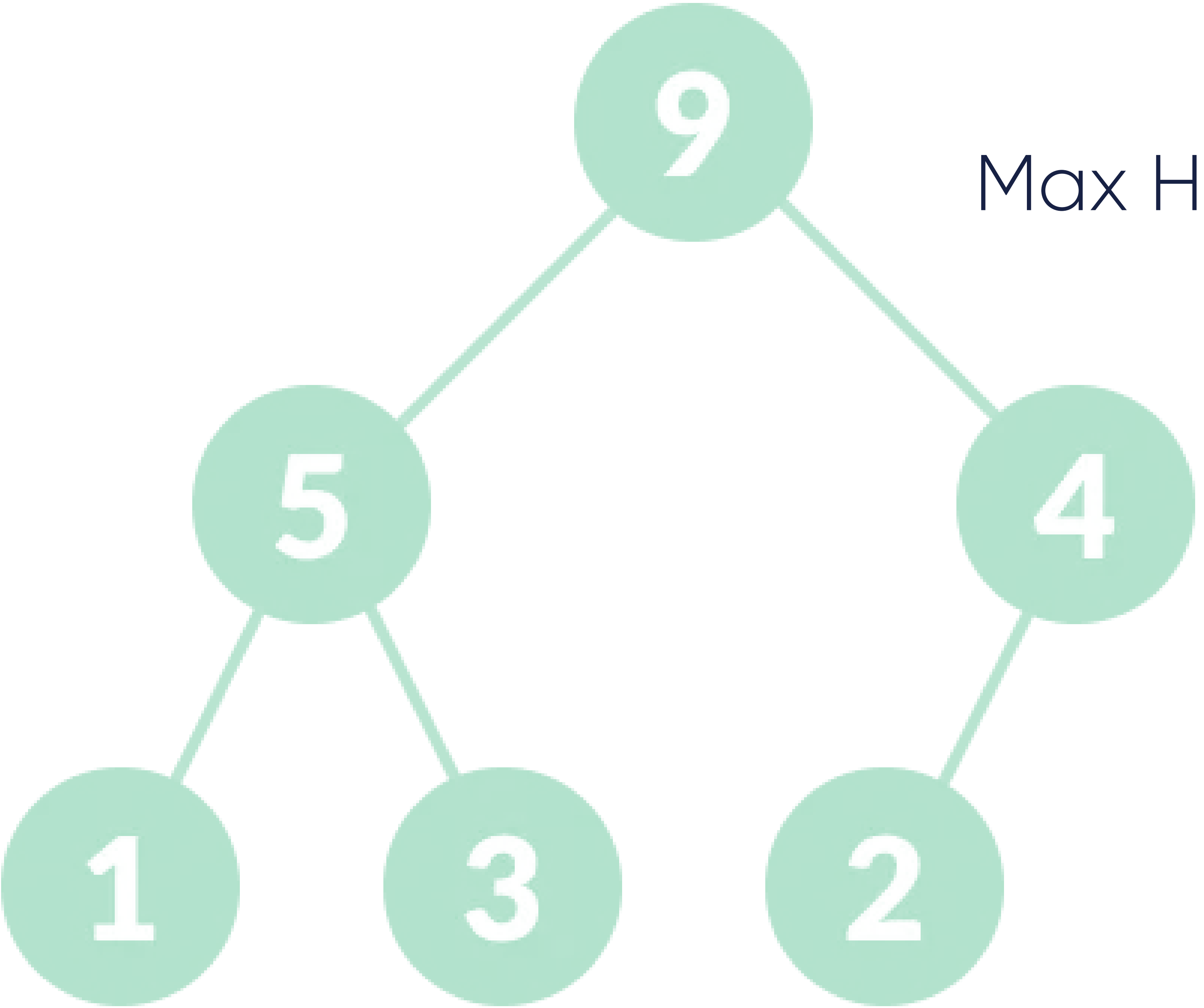
Heap Animation

Heap

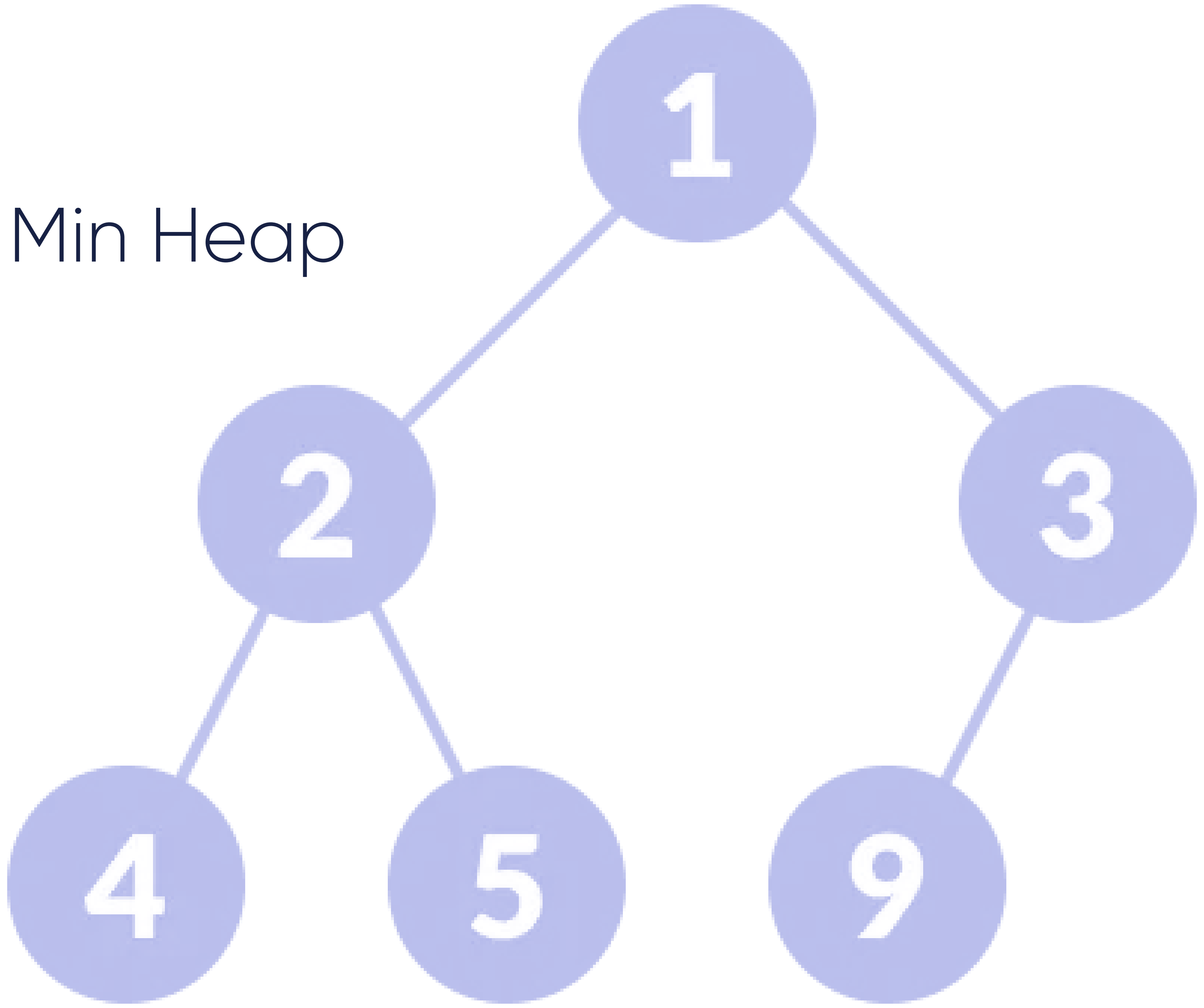
Heap data structure is a complete binary tree that satisfies the heap property, where any given node is:

- always greater than its child node/s and the key of the root node is the largest among all other nodes. This property is also called max heap property.
- always smaller than the child node/s and the key of the root node is the smallest among all other nodes. This property is also called min heap property.



Max Heap

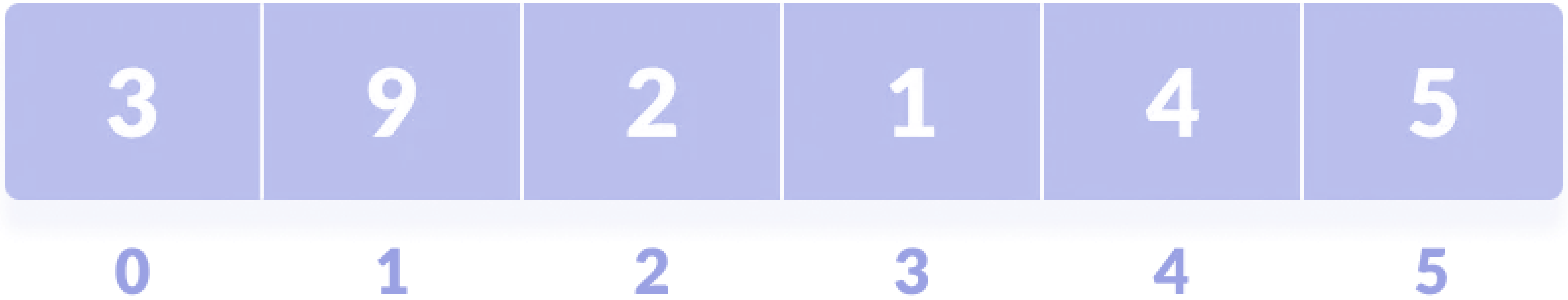
Min Heap



Heap Operations

Heapify

1- Heapify is the process of creating a heap data structure from a binary tree. It is used to create a Min-Heap or a Max-Heap.

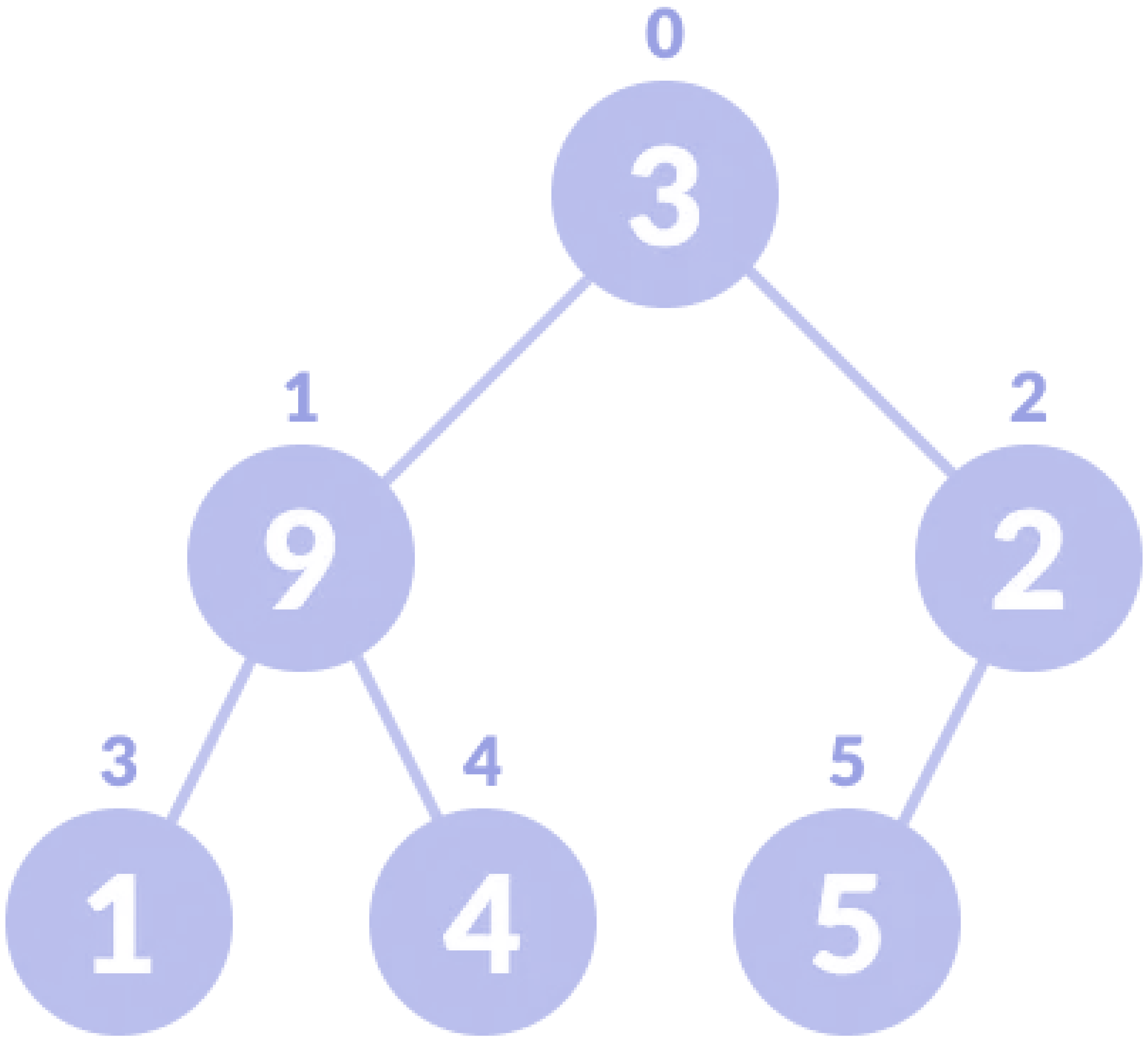


3	9	2	1	4	5
0	1	2	3	4	5

Heap Operations

Complete Binary Tree

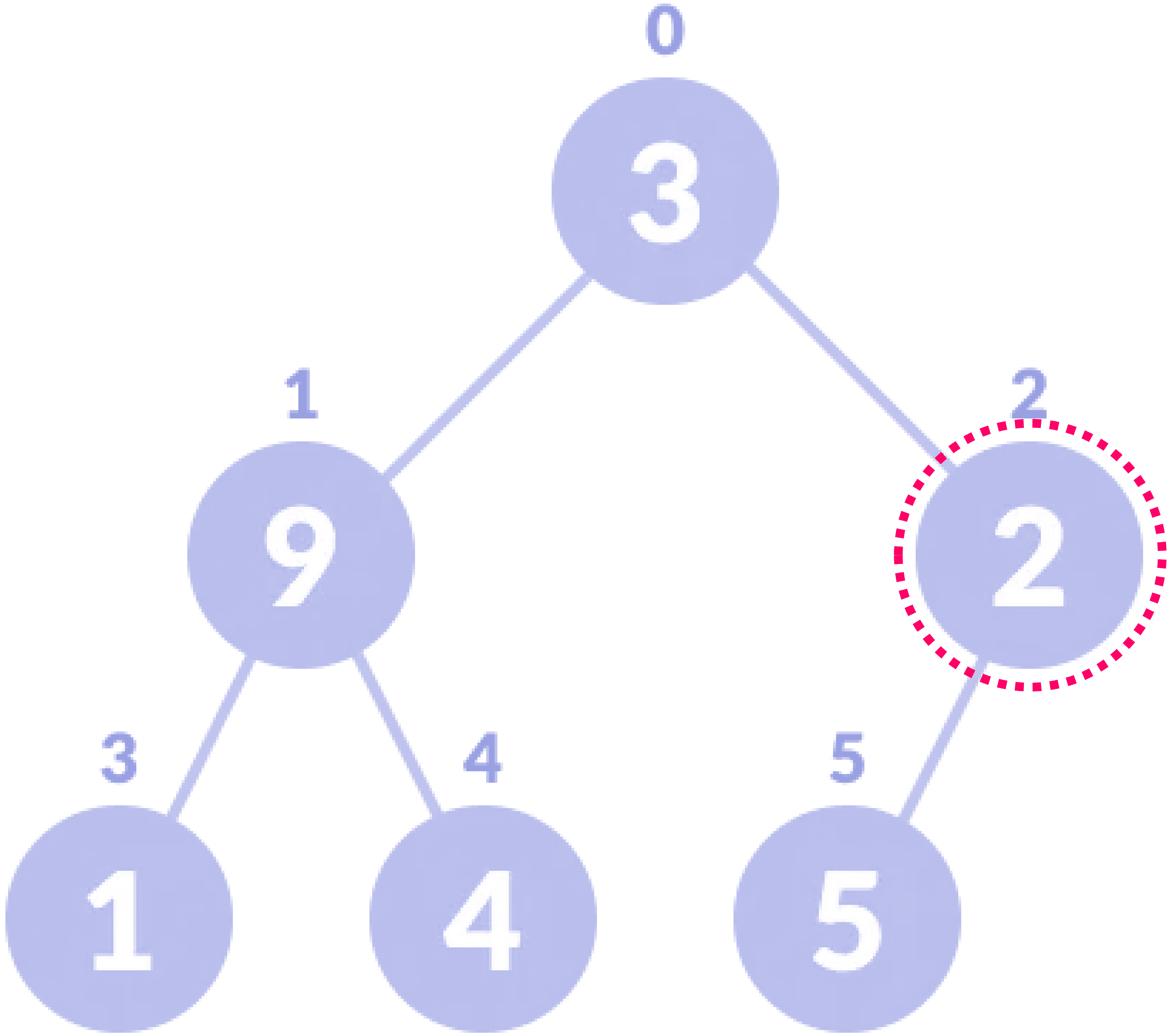
2- Create a complete binary tree from the array



Heap Operations

Complete Binary Tree

3- Start from the first index of non-leaf node whose index is given by n/2 - 1



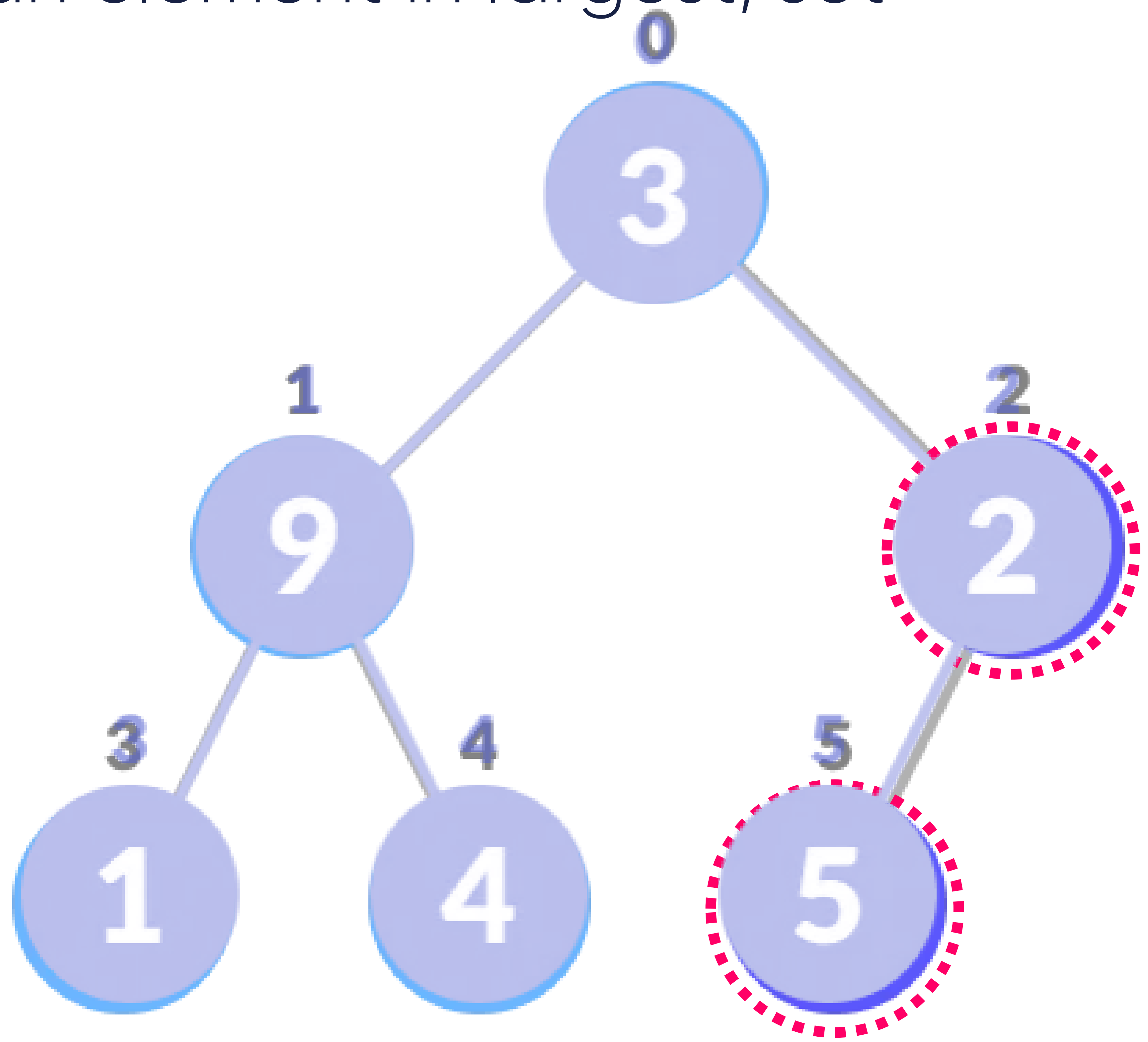
Heap Operations

4- Set current element i as largest.

5- The index of left child is given by 2i + 1 and the right child is given by 2i + 2.
If leftChild is greater than currentElement (i.e. element at ith index), set leftChildIndex as largest. If rightChild is greater than element in largest, set rightChildIndex as largest.

6- Swap largest with currentElement

7- Repeat steps 3-7 until the subtrees are also heapified.



Heap in C

```
#include <stdio.h>
int size = 0;

void swap(int *a, int *b)
{
  int temp = *b;
  *b = *a;
  *a = temp;
}


void heapify(int array[], int size, int i)
{
  if (size == 1)
  {
    printf("Single element in the heap");
  }
  else
  {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < size && array[l] > array[largest])
      largest = l;
    if (r < size && array[r] > array[largest])
      largest = r;
    if (largest != i)
    {
      swap(&array[i], &array[largest]);
      heapify(array, size, largest);
    }
  }
}
```

```
void insert(int array[], int newNum)
{
  if (size == 0)
  {
    array[0] = newNum;
    size += 1;
  }
  else
  {
    array[size] = newNum;
    size += 1;
    for (int i = size / 2 - 1; i >= 0; i--)
    {
      heapify(array, size, i);
    }
  }
}


void deleteRoot(int array[], int num)
{
  int i;
  for (i = 0; i < size; i++)
  {
    if (num == array[i])
      break;
  }

  swap(&array[i], &array[size - 1]);
  size -= 1;
  for (int i = size / 2 - 1; i >= 0; i--)
  {
    heapify(array, size, i);
  }
}
void printArray(int array[], int size)
{
  for (int i = 0; i < size; ++i)
    printf("%d ", array[i]);
  printf("\n");
}
```

```
int main()
{
  int array[10];

  insert(array, 3);
  insert(array, 4);
  insert(array, 9);
  insert(array, 5);
  insert(array, 2);

  printf("Max-Heap array: ");
  printArray(array, size);

  deleteRoot(array, 4);

  printf("After deleting an element: ");

  printArray(array, size);
}
```

Heap in Java

Heap Sort Algorithm Animation

Heap Sort Algorithm Complexity

Time Complexity

Best	O(nlog n)
Worst	O(nlog n)
Average	O(nlog n)

Space Complexity O(1)

Stability No

Heap Sort Algorithm in C

```c
#include <stdio.h>

// Function to swap the position of two elements
void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}

void heapify(int arr[], int n, int i) {
  // Find largest among root, left child and right child
  int largest = i;
  int left = 2 * i + 1;
  int right = 2 * i + 2;

  if (left < n && arr[left] > arr[largest])
    largest = left;

  if (right < n && arr[right] > arr[largest])
    largest = right;

  // Swap and continue heapifying if root is not largest
  if (largest != i) {
    swap(&arr[i], &arr[largest]);
    heapify(arr, n, largest);
  }
}

// Main function to do heap sort
void heapSort(int arr[], int n) {
  // Build max heap
  for (int i = n / 2 - 1; i >= 0; i--)
    heapify(arr, n, i);

  // Heap sort
  for (int i = n - 1; i >= 0; i--) {
    swap(&arr[0], &arr[i]);

    // Heapify root element to get highest
    // element at root again
    heapify(arr, i, 0);
  }
}

// Print an array
void printArray(int arr[], int n) {
  for (int i = 0; i < n; ++i)
    printf("%d ", arr[i]);
  printf("\n");
}

// Driver code
int main() {
  int arr[] = {1, 12, 9, 5, 6, 10};
  int n = sizeof(arr) / sizeof(arr[0]);

  heapSort(arr, n);

  printf("Sorted array is \n");
  printArray(arr, n);
}
```

Heap Sort Algorithm in Java