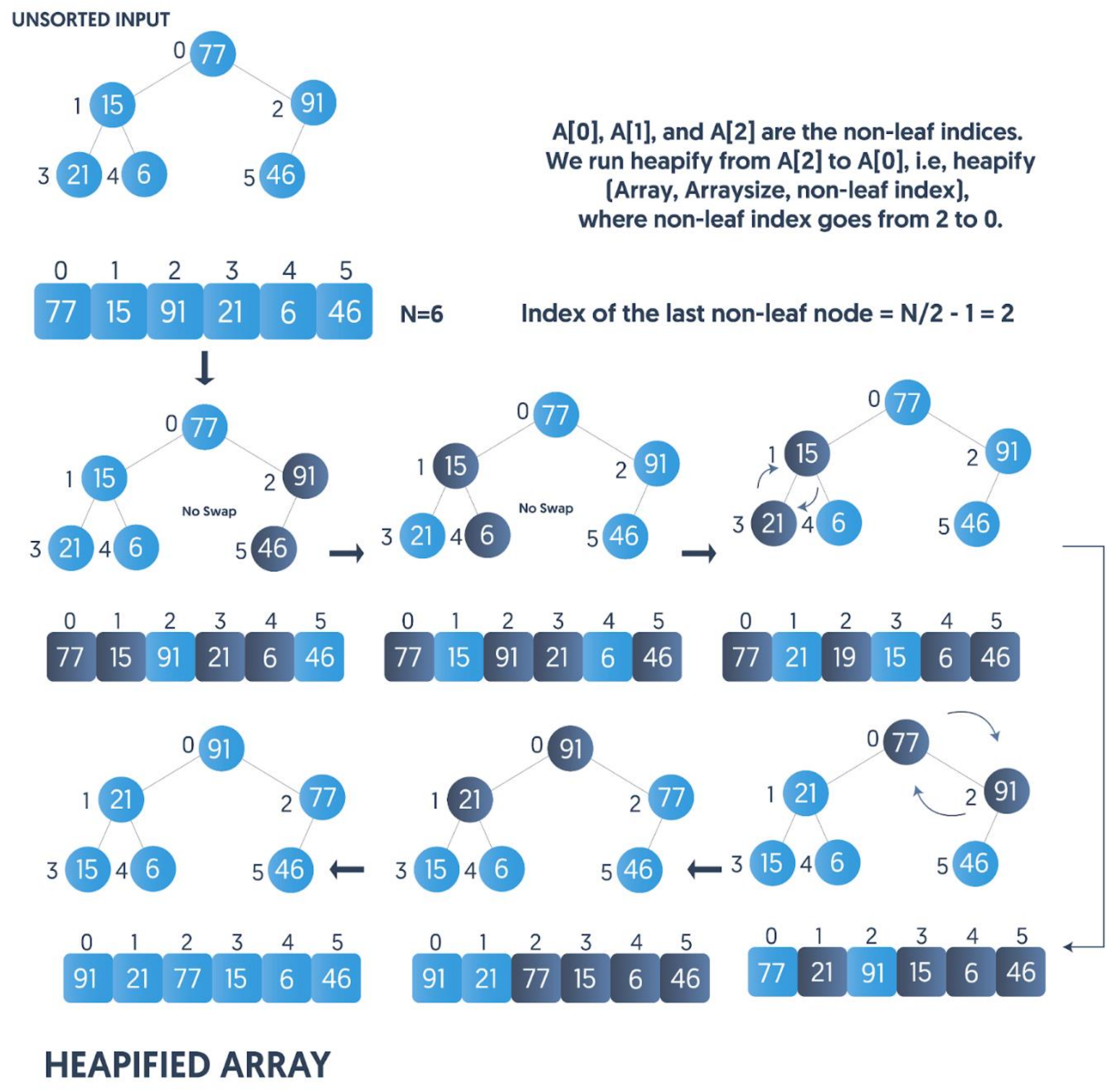
HEAP-SORT

Heap Sort Definition

Heap sort is an efficient comparison-based sorting algorithm that:

- Creates a heap from the input array.
- Then sorts the array by taking advantage of a heap's properties.

Heapify Method



UNSORTED INPUT
A[0], A[1], and A[2] are the non-leaf indices.
We run heapify from A[2] to A[0], i.e, heapify
(Array, Arraysize, non-leaf index),
where non-leaf index goes from 2 to 0.
N=6
Index of the last non-leaf node = N/2 - 1 = 2
No Swap
No Swap
HEAPIFIED ARRAY

- When sorting *in-place*, we can use a max heap to sort the array in ascending order and a min heap to sort the array in descending order.
- If sorting doesn't have to be *in-place*, we can use an auxiliary array to place the extracted element from the heap's top in its correct position, whether we use a min heap or a max heap for the sorting.

But even when sorting is not the aim, a min/max heap in itself is a useful construction:

- The root element of a max heap always contains the maximum element.
- The root element of a min heap always has the minimum element.

This quality of heaps can come in handy when we want to extract only the largest or smallest element from an array without sorting the remaining elements.

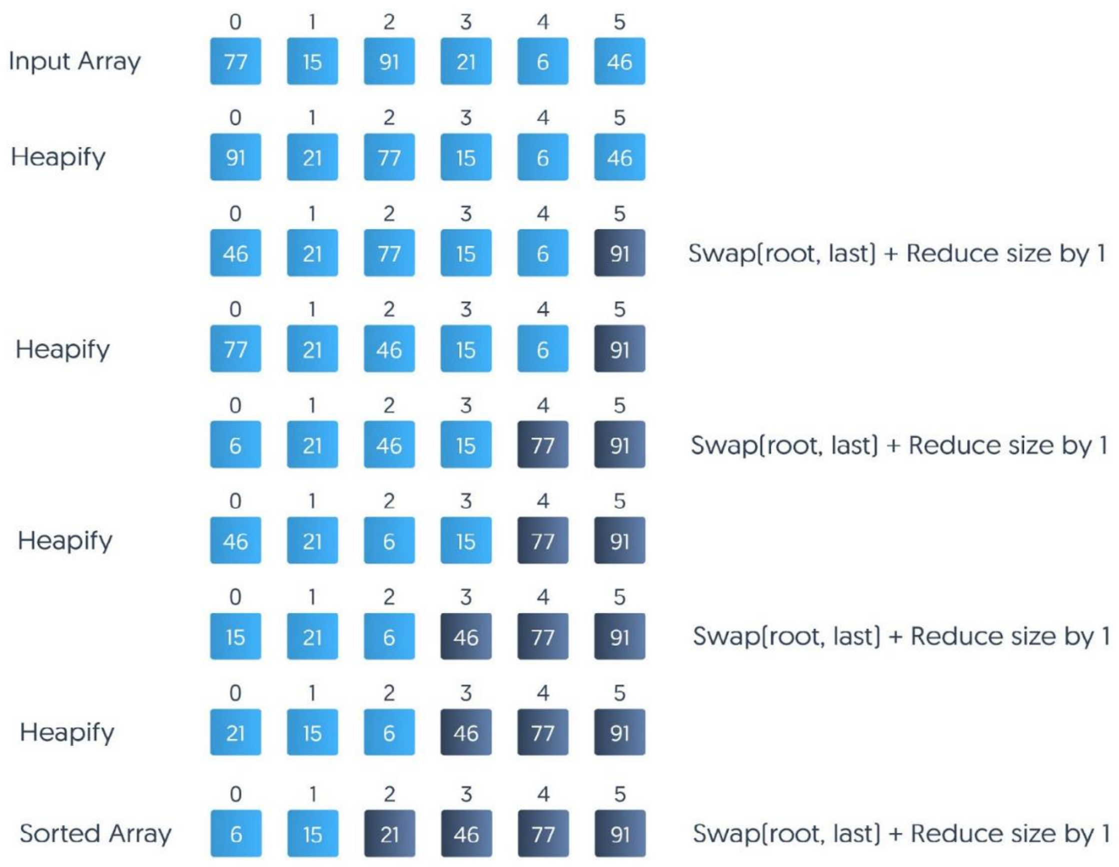
How Does Heap Sort Work?

After the *heap formation using the heapify method*, the sorting is done by:

1. **Swapping** the *root element* with the *last element of the array* and decreasing the length of the heap array by one. (In heap representation, it is equivalent to swapping the root with the bottom-most and right-most leaf and then deleting the leaf.)
2. **Restoring** heap properties (*reheapification*) *after each deletion*, where we need to apply the heapify method *only on the root node.* The subtree heaps will still have their heap properties intact at the beginning of the process.
3. **Repeating** this process until every element in the array is sorted: Root removal, its storage i*n the position of the highest index value used by the heap*, and heap length decrement.

- On a **max heap**, this process will sort the array in **ascending order.**
- On a **min heap**, this process will sort in **descending order**.

This process can be best illustrated using an example:



The process above ends when heap size = 2 because a two-element heap is always considered sorted.

So basically, the heap sort algorithm has two parts that run recursively till heap size >= 2:

- **Creating a heap** from the currently unsorted elements of the array.
- **Swapping the root element with the last element** of the heap (right-most leaf node)
- **Reducing heap size** by 1.

Code:

```
import java.util.*;

import java.io.*;
import java.lang.*;

public class Main
{
          public void buildHeap(int array[],int n){
       for (int i = n / 2 - 1; i >= 0; i--)
                        heapify(array, n, i);
   }

          public void sort(int array[])
          {
                    int n = array.length;

                    buildHeap(array,n);

                    for (int i=n-1; i>0; i--)
                    {

                              int temp = array[0];
                              array[0] = array[i];
                              array[i] = temp;

                              heapify(array, i, 0);
                    }
}

void heapify(int array[], int n, int i)
{
        int largest = i;
        int l = 2*i + 1;
        int r = 2*i + 2;

        if (l < n && array[l] > array[largest])
                largest = l;

        if (r < n && array[r] > array[largest])
                largest = r;

        if (largest != i)
        {
                int swap = array[i];
                array[i] = array[largest];
                array[largest] = swap;

                heapify(array, n, largest);
        }
}
```

```
    static void printArray(int array[])
    {
        int n = array.length;
        for (int i=0; i<n; ++i)
            System.out.print(array[i]+" ");
        System.out.println();
    }

    public static void main(String args[])
    {
        int array[] = {12, 11, 13, 5, 6, 7};
        int n = array.length;

        Main ob = new Main();
        ob.sort(array);

        System.out.println("Sorted array is");
        printArray(array);
    }
}
```