Heap Sort

One of the best Sorting Algorithm

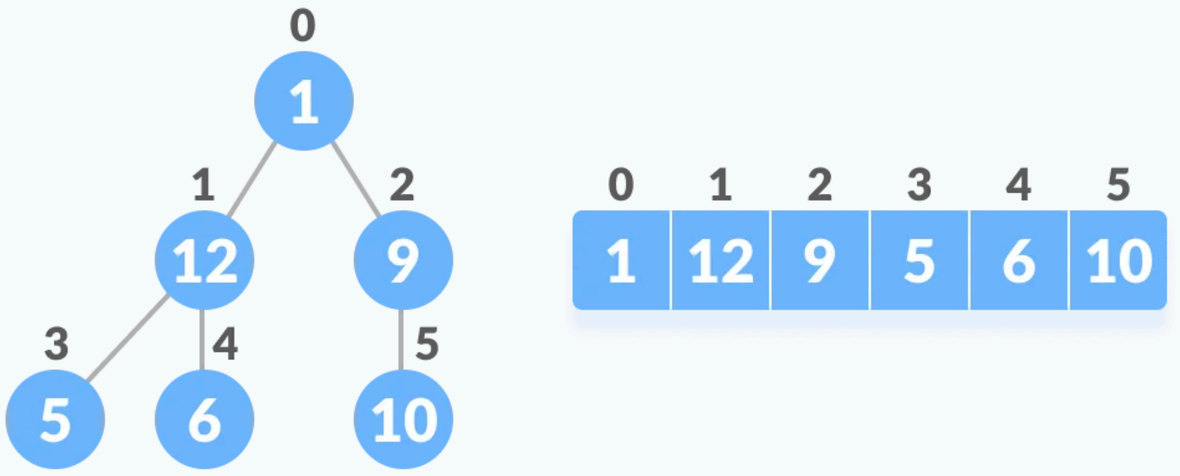
Sagor, 10 Jan 2022

Reference: Heap Sort Algorithm, https://www.programiz.com/dsa/heap-sort

Relationship between Array Indexes and Tree Elements

A complete binary tree has an interesting property that we can use to find the children and parents of any node.

If the index of any element in the array is `i`, the element in the index `2i+1` will become the left child and element in `2i+2` index will become the right child. Also, the parent of any element at index `i` is given by the lower bound of `(i-1)/2`.



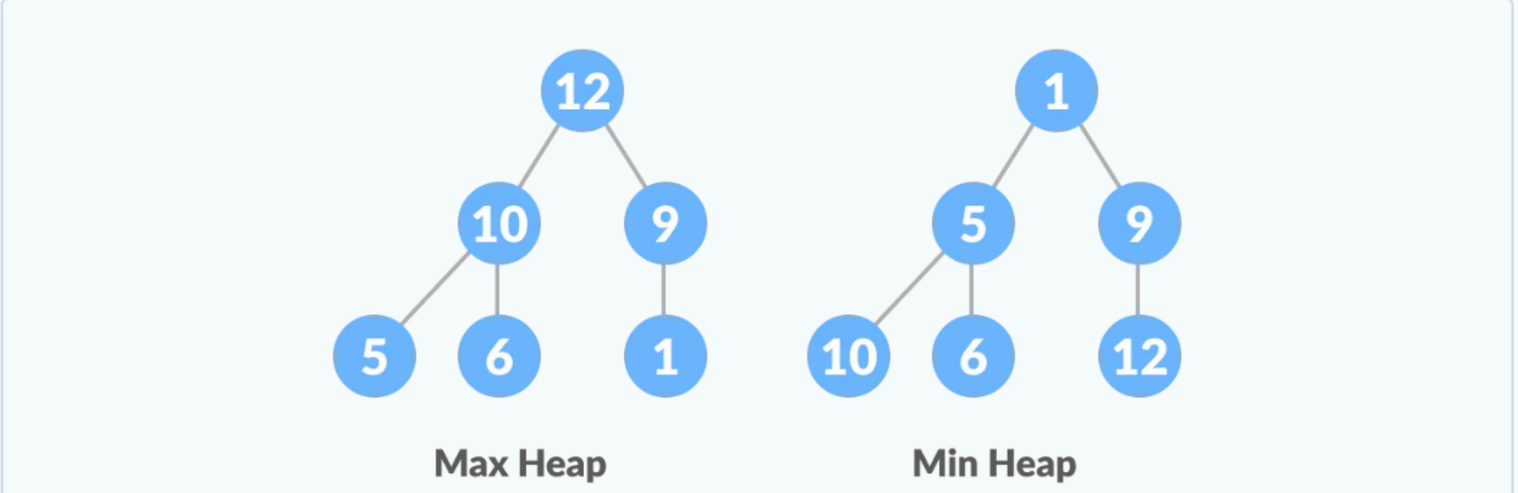
Relationship between array and heap indices

What is Heap Data Structure?

Heap is a special tree-based data structure. A binary tree is said to follow a heap data structure if

- it is a complete binary tree
- All nodes in the tree follow the property that they are greater than their children i.e. the largest element is at the root and both its children and smaller than the root and so on. Such a heap is called a max-heap. If instead, all nodes are smaller than their children, it is called a min-heap

The following example diagram shows Max-Heap and Min-Heap.



Max Heap and Min Heap

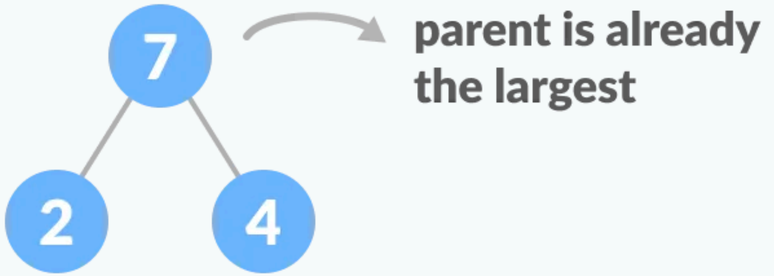
How to "heapify" a tree

Starting from a complete binary tree, we can modify it to become a Max-Heap by running a function called heapify on all the non-leaf elements of the heap.

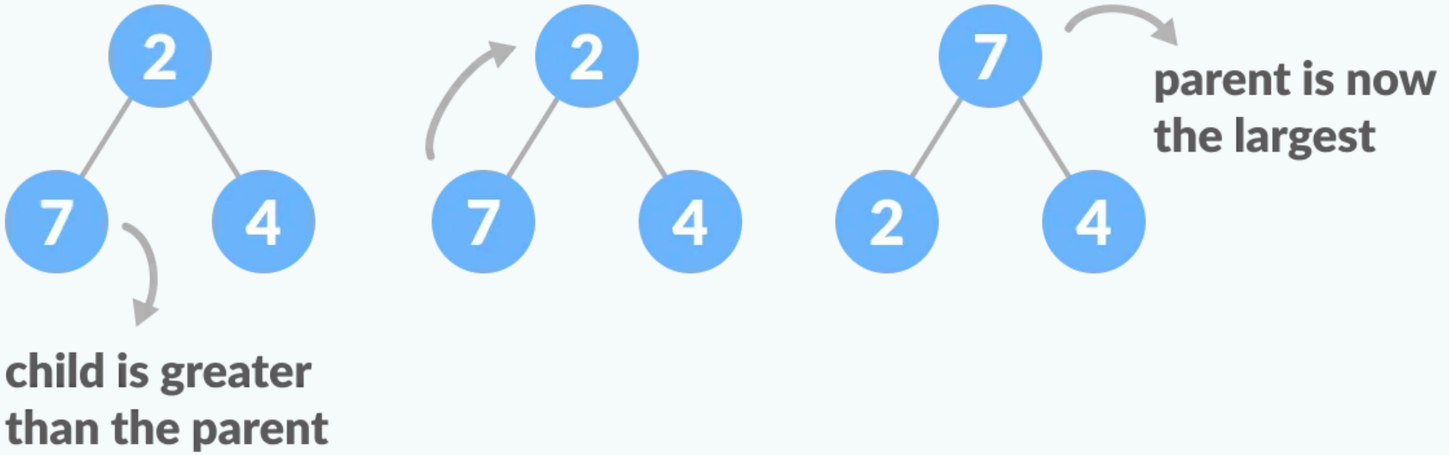
Since heapify uses recursion, it can be difficult to grasp. So let's first think about how you would heapify a tree with just three elements.

```
heapify(array)
    Root = array[0]
    Largest = largest( array[0] , array [2*0 + 1]. array[2*0+2])
    if(Root != Largest)
          Swap(Root, Largest)
```

Scenario-1



Scenario-2



Heapify base cases

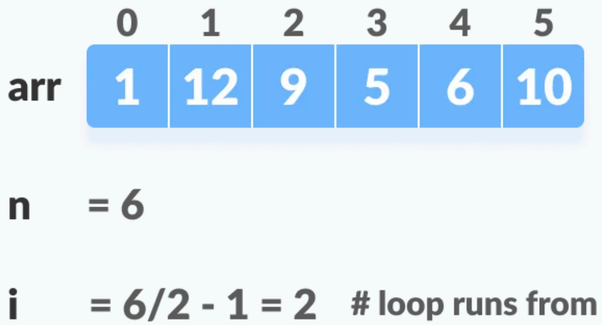
Build max-heap

To build a max-heap from any tree, we can thus start heapifying each sub-tree from the bottom up and end up with a max-heap after the function is applied to all the elements including the root element.

In the case of a complete tree, the first index of a non-leaf node is given by `n/2 - 1`. All other nodes after that are leaf-nodes and thus don't need to be heapified.

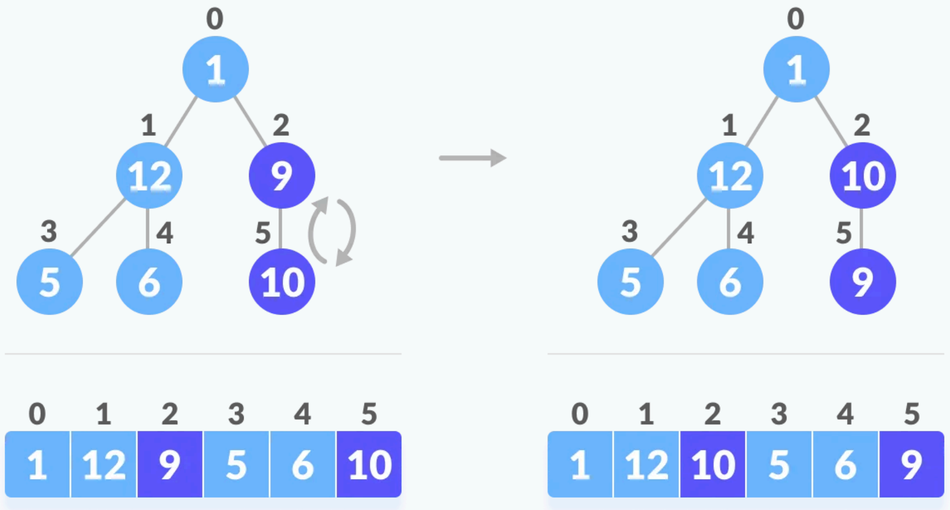
So, we can build a maximum heap as

```
// Build heap (rearrange array)
for (int i = n / 2 - 1; i >= 0; i--)
  heapify(arr, n, i);
```



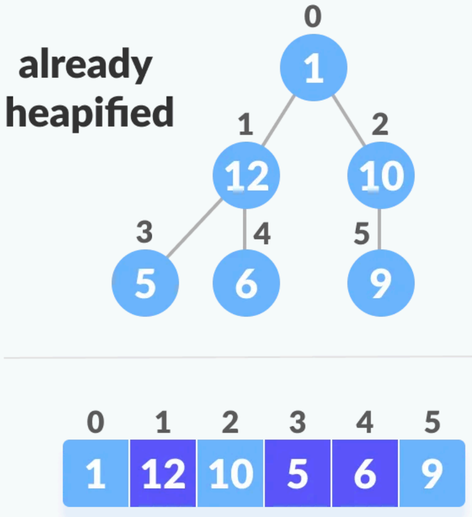
Create array and calculate i

i = 2 → heapify(arr, 6, 2)

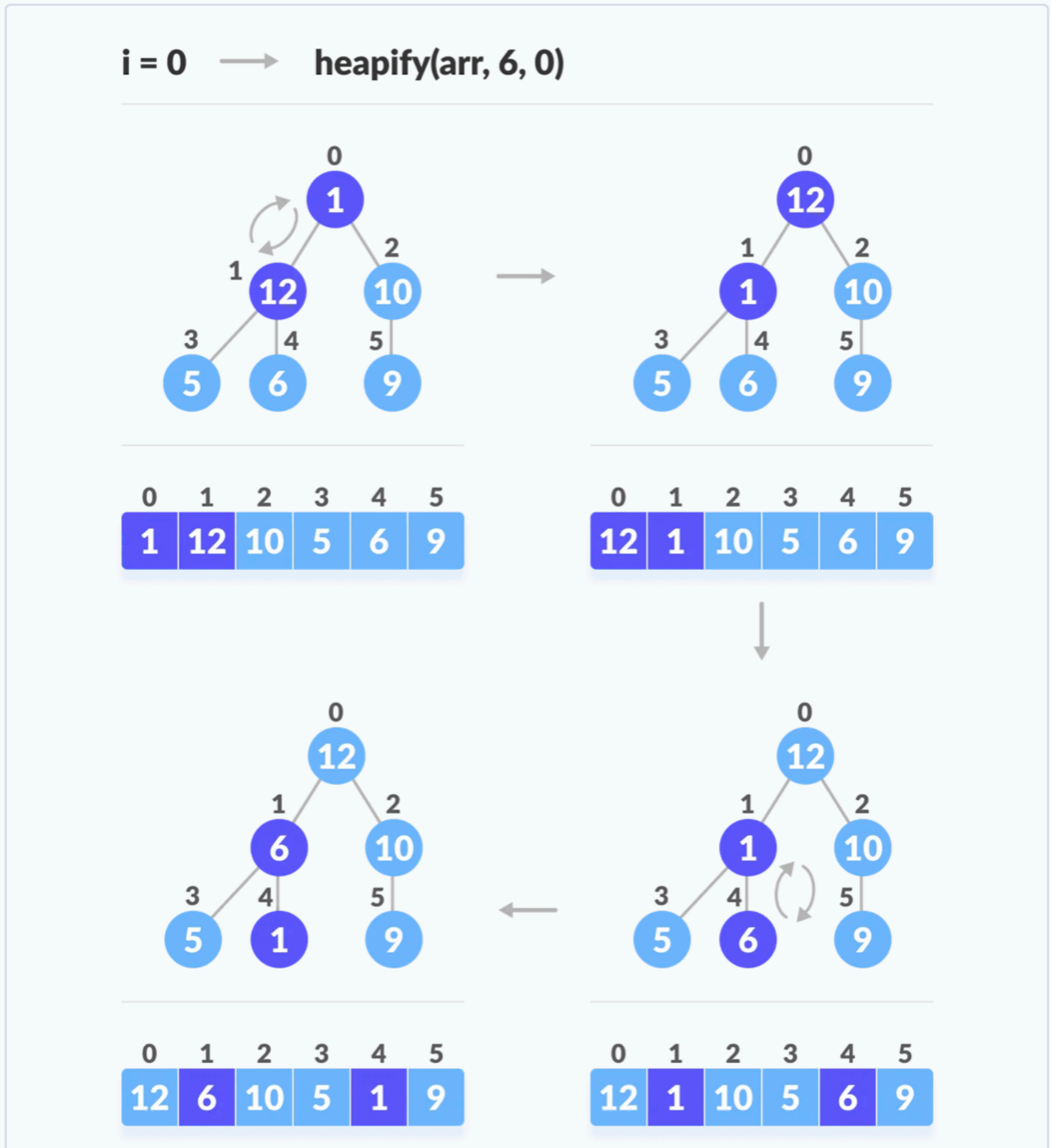


Steps to build max heap for heap sort

i = 1 → heapify(arr, 6, 1)



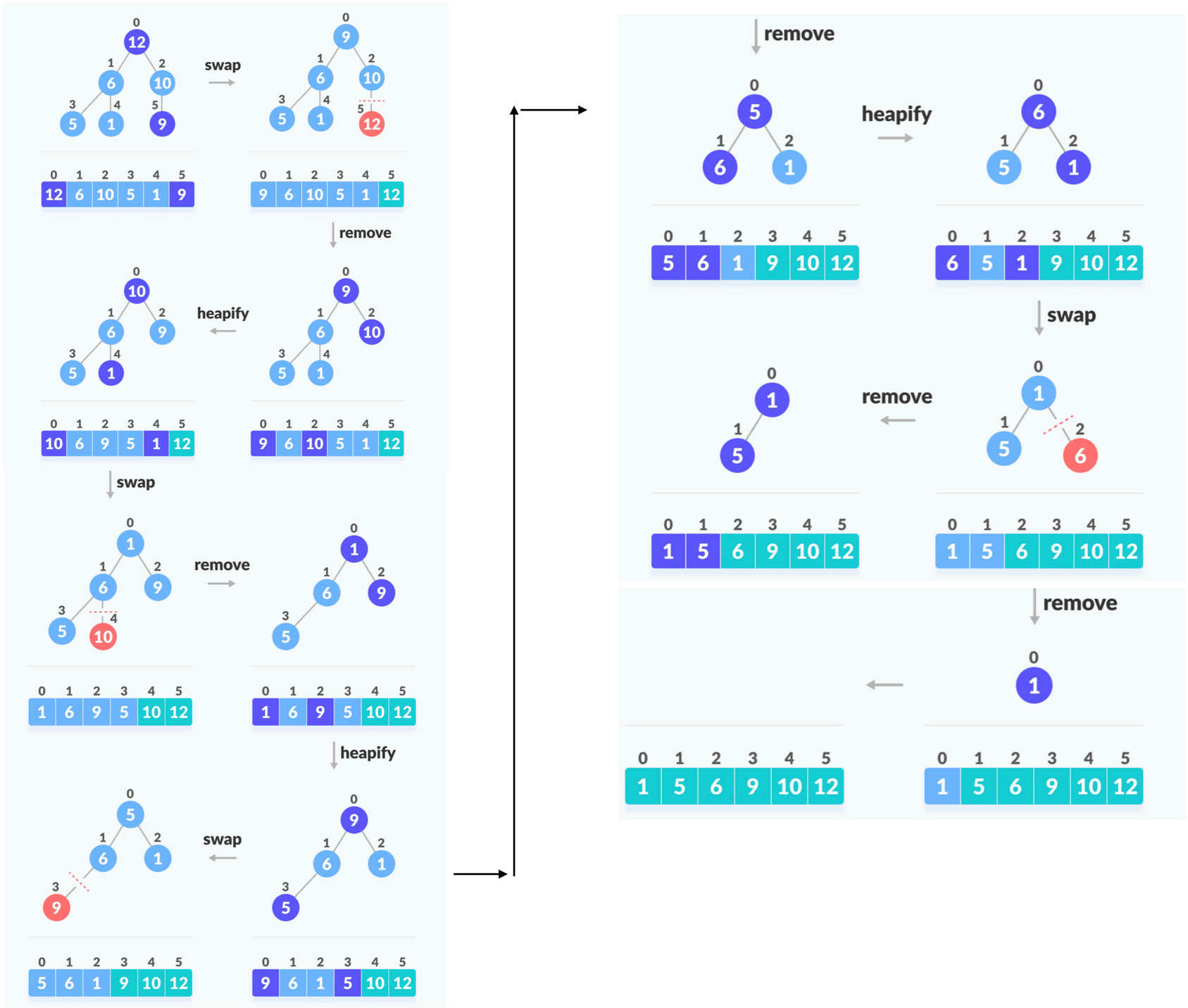
Steps to build max heap for heap sort



Steps to build max heap for heap sort

Working of Heap Sort

1. Since the tree satisfies Max-Heap property, then the largest item is stored at the root node.
2. **Swap:** Remove the root element and put at the end of the array (nth position) Put the last item of the tree (heap) at the vacant place.
3. **Remove:** Reduce the size of the heap by 1.
4. **Heapify:** Heapify the root element again so that we have the highest element at root.
5. The process is repeated until all the items of the list are sorted.


swap
remove
heapify
swap
remove
heapify
swap
remove
heapify
swap
remove
remove

Code in C++

```cpp
#include <iostream>
using namespace std;

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 root = n / 2 - 1; root >= 0; root--)
        heapify(arr, n, root);

    // 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);
    }
}
```

Heap Sort Complexity

Time Complexity	
Best	O(nlog n)
Worst	O(nlog n)
Average	O(nlog n)
Space Complexity	O(1)
Stability	No

END