Heap sort

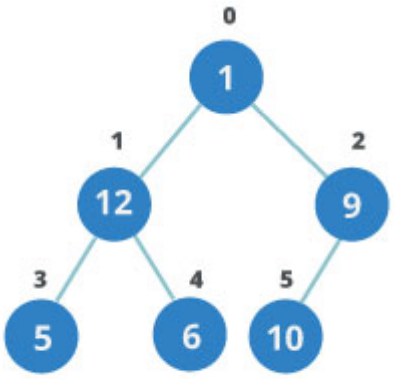
Heap sort is a comparison based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for remaining element.

In max-heaps, maximum element will always be at the root. Heap Sort uses this property of heap to sort the array.

What is Binary Heap?
Let us first define a Complete Binary Tree. A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible

A Binary Heap is a Complete Binary Tree where items are stored in a special order such that value in a parent node is greater(or smaller) than the values in its two children nodes. The former is called as max heap and the latter is called min heap. The heap can be represented by binary tree or array.

Why array based representation for Binary Heap?
Since a Binary Heap is a Complete Binary Tree, it can be easily represented as array and array based representation is space efficient. 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.



```
For an array 1 12 9 5 6 10

Left child of 1 (index 0)
= element in (2*0+1) index
= element in 1 index
= 12


Right child of 1
= element in (2*0+2) index
= element in 2 index
= 9

Similarly,
Left child of 12 (index 1)
= element in (2*1+1) index
= element in 3 index
= 5

Right child of 12
= element in (2*1+2) index
= element in 4 index
= 6
```

Heap Sort Algorithm for sorting in increasing order:

1. Build a max heap from the input data.
2. At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.
3. Repeat above steps while size of heap is greater than 1.

Heapify procedure can be applied to a node only if its children nodes are heapified. So the heapification must be performed in the bottom up order.

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 on all the elements including the root element.

In the case of complete tree, the first index of 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

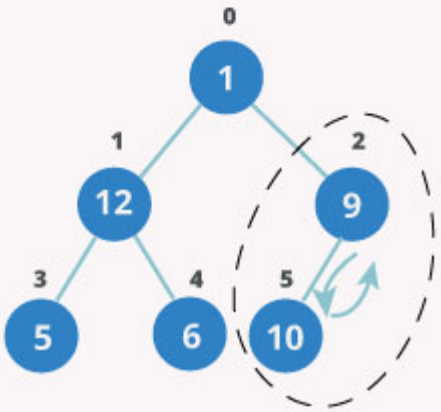
```
// Build heap (rearrange array)
for (int i = n / 2 - 1; i >= 0; i--)
   heapify(arr, n, i);
```

	0	1	2	3	4	5
arr	1	12	9	5	6	10

n 6

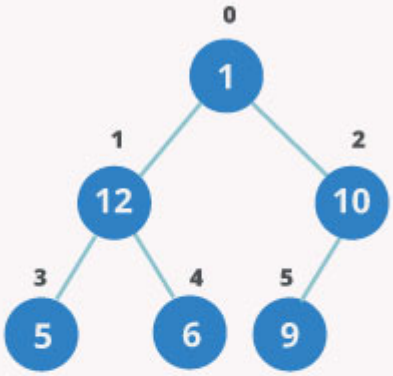
i = 6/2-1
= 2 -> 0

i = 2



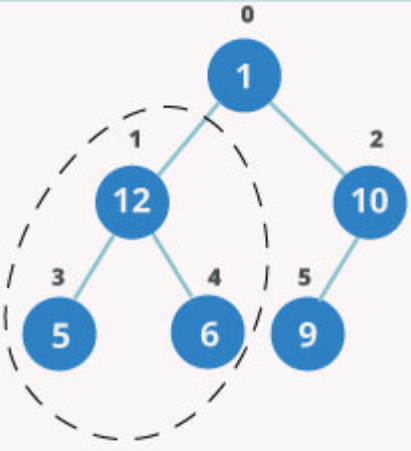
0	1	2	3	4	5
1	12	9	5	6	10

heapify(arr, 6, 2)



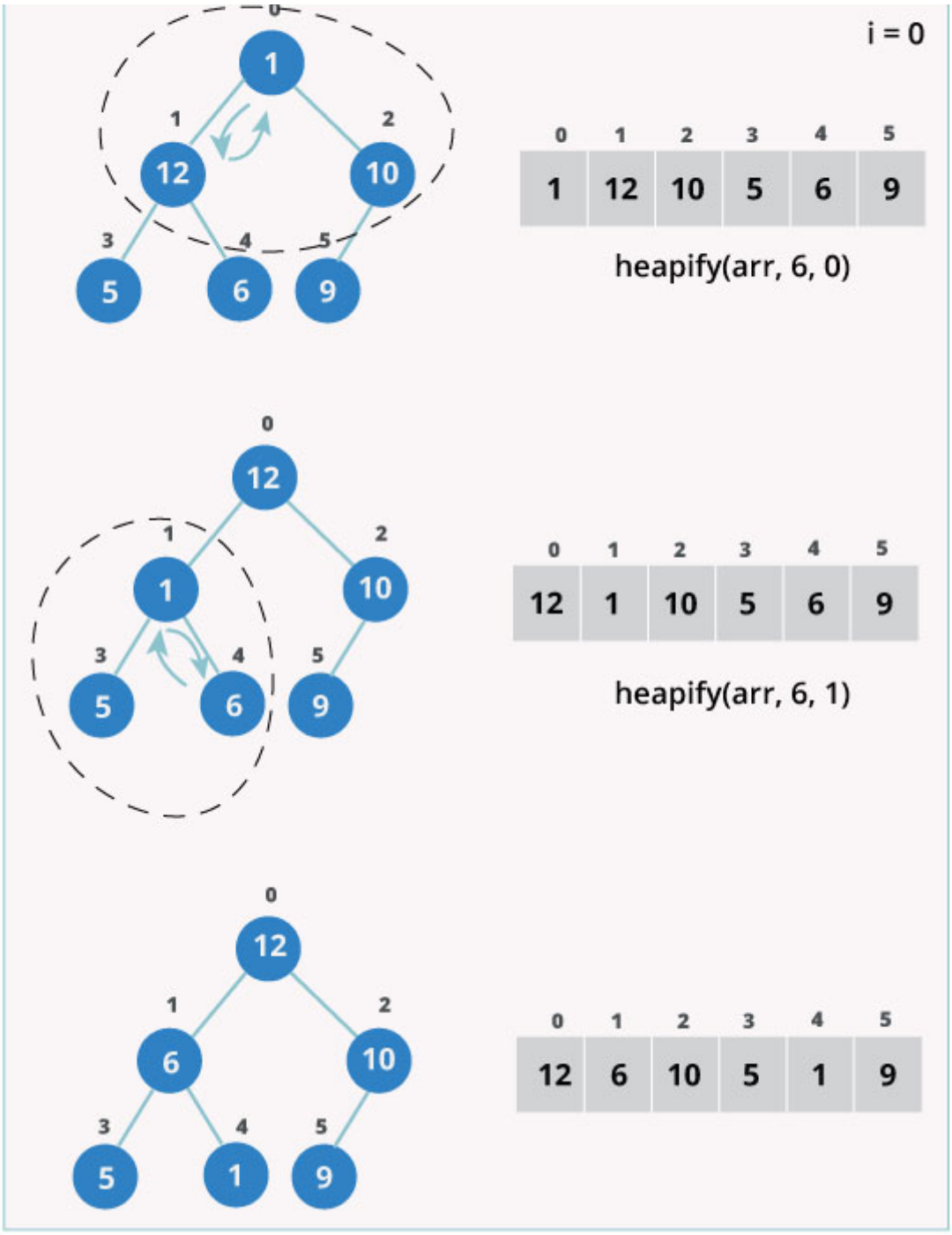
0	1	2	3	4	5
1	12	10	5	6	9

i = 1

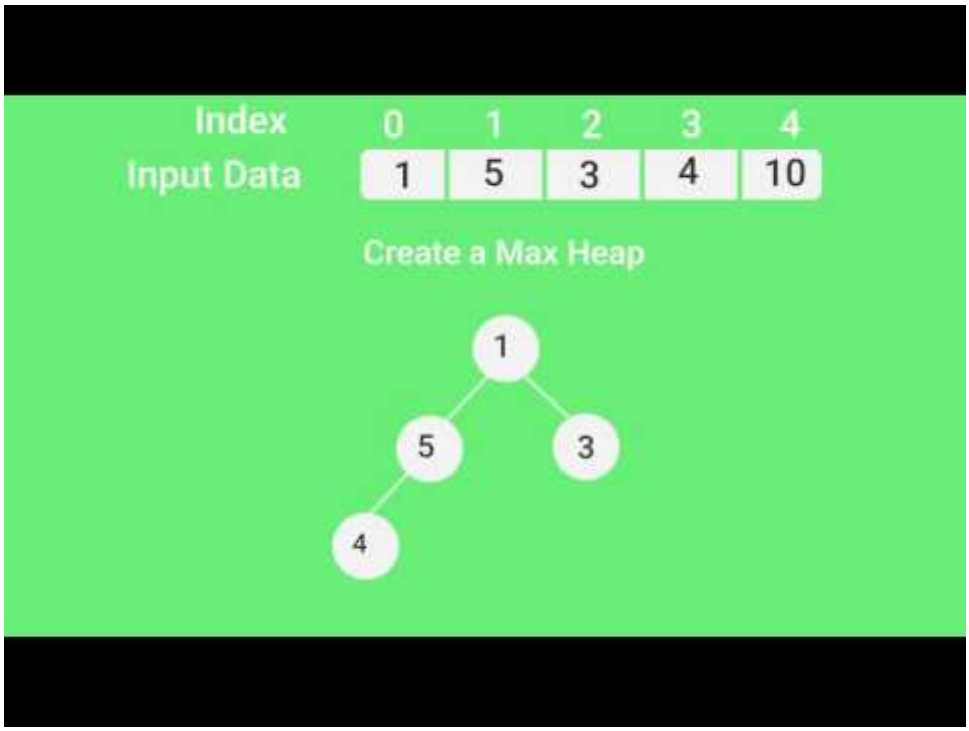


0	1	2	3	4	5
1	12	10	5	6	9

heapify(arr, 6, 1)



Working of HeapSort (video)



Performance

Heap Sort has O(n*logn) time complexities for all the cases (best case, average case and worst case).Let us understand the reason why. The height of a complete binary tree containing n elements is log(n)

As we have seen earlier, to fully heapify an element whose subtrees are already max-heaps, we need to keep comparing the element with its left and right children and pushing it downwards until it reaches a point where both its children are smaller than it. In the worst case scenario, we will need to move an element from the root to the leaf node making a multiple of log(n) comparisons and swaps.

During the build_max_heap stage, we do that for n/2 elements so the worst case complexity of the build_heap step is n/2*log(n) ~ nlogn.

During the sorting step, we exchange the root element with the last element and heapify the root element. For each element, this again takes logn worst time because we might have to bring the element all the way from the root to the leaf. Since we repeat this n times, the heap_sort step is also nlogn.

Also since the build_max_heap and heap_sort steps are executed one after another, the algorithmic complexity is not multiplied and it remains in the order of nlogn.

Also it performs sorting in O(1) space complexity. Comparing with Quick Sort, it has better worst case O(nlogn). Quick Sort has complexity $O(n^2)$ for worst case. But in other cases, Quick Sort is fast.

Introsort is an alternative to heapsort that combines quicksort and heapsort to retain advantages of both: worst case speed of heapsort and average speed of quicksort.

Application of Heap Sort

Systems concerned with security and embedded system such as Linux Kernel uses Heap Sort because of the O(nlogn) upper bound on Heapsort's running time and constant O(1) upper bound on its auxiliary storage.