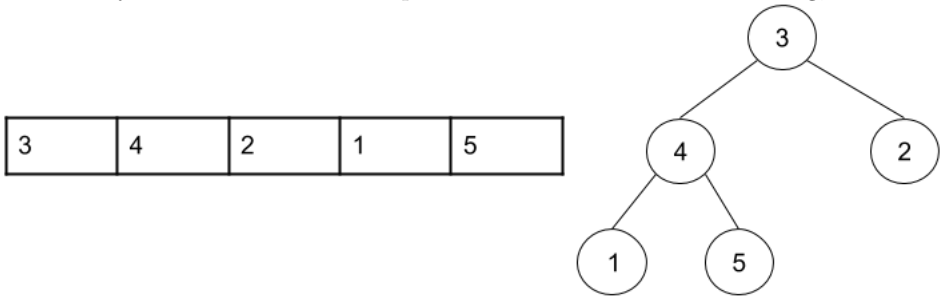
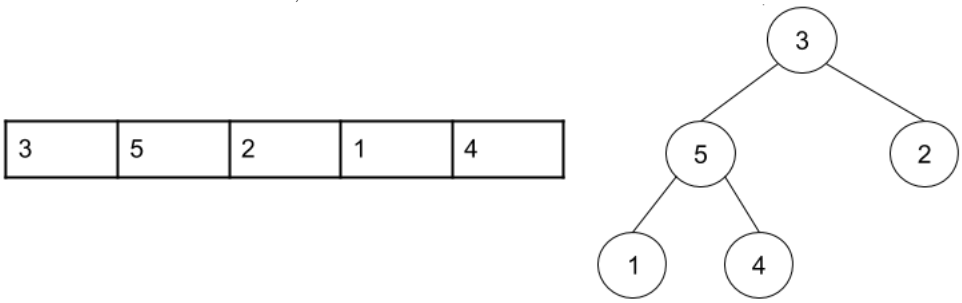
Heapsort

1 Heapsort

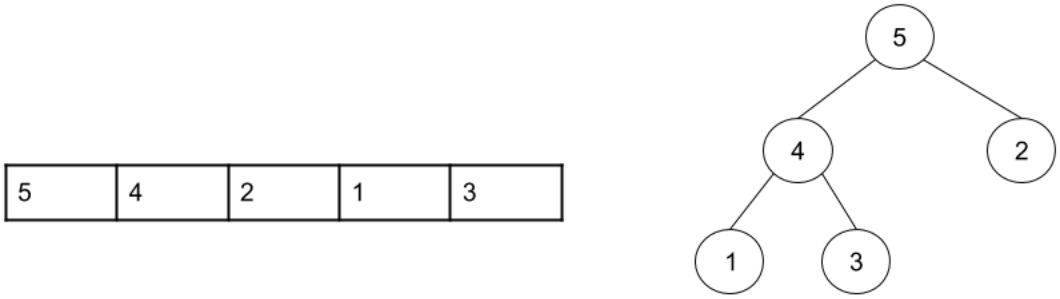
We learned about the heap data structure earlier in this book. We will now see how we can use this data structure in order to sort a list in a method called ***Heapsort***. We will create 2 subarrays within the array. One will be the heap and one will be the elements that are sorted. We will first "heapify" all our elements into a max heap then we will then perform n delete the max operations. We will put this item in the back of the array. Let's look at an example. Before we do the actual sorting, we will discuss heapifying.



We will start off with this unsorted array. To the right of it is its corresponding heap. In order to make this into a valid heap, we will sink items in *array reverse level order*. This means that we will start at the lowest level and move to the left, then we will sink the item if needed.

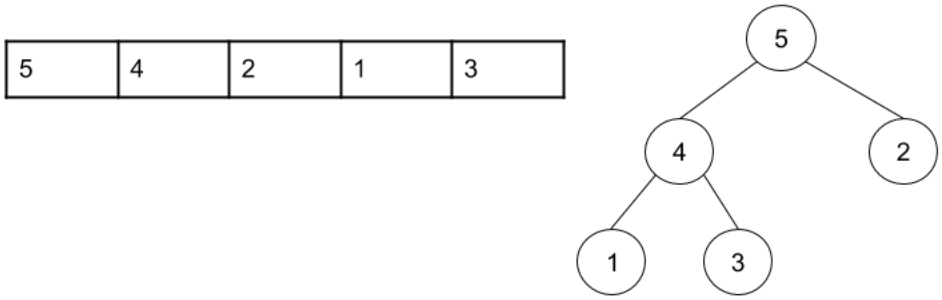


We cannot sink the first elements, 2, 1, or 5, but on the 4th element we check, 4, we realize it can be sinked since it is less than 5. We swap 4 and 5 and then we get our current heap.

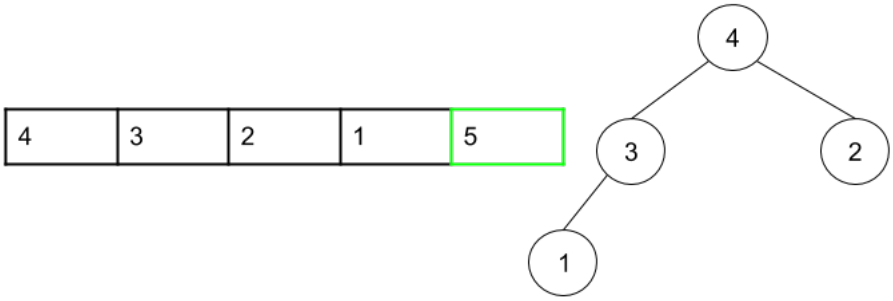


We move to the first index and then realize that we can swap 3 with 5, since 5 is bigger. We then see that 3 is less than 4 so we swap those elements as well.

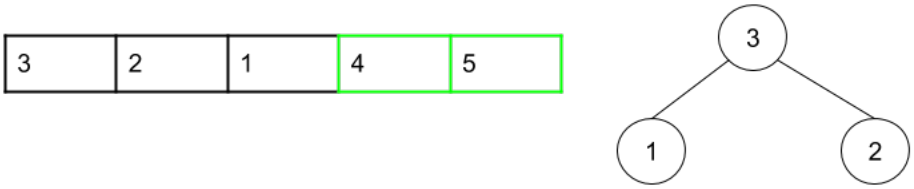
Now that we have our elements in the form of our max heap, we are ready to actually sort our array.



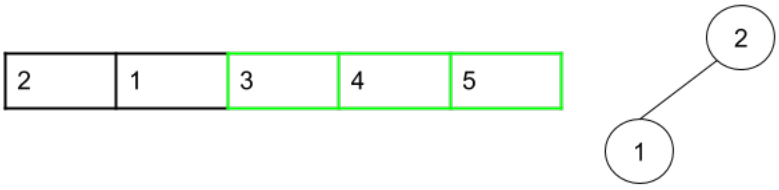
We will start off with the same heap that we did in the previous example.



Now we have deleted our max element, 5. This results in 4 getting swapped to the top. The total amount of elements in our heap array has decreased by 1. We put the old max element in the index of the array that once has the last element.



We do a similar process with the next max element, 4. Once again, the size of the array goes down by 1. We put the item 4 where the old last element was.



We now delete the max once more and move 3 to the 3rd index, which has now been freed up. Our heap is now only 2 elements.

We repeat this process and delete the max element, 2, adjust our heap and put 2 in the 2nd index. We only have one more element left in our heap, 1.

After we delete the max one more time, our heap is completely empty. At this point, our array is totally sorted and our work is complete.

The time complexity of heapsort is $O(Nlog(N))$. As we know, constructing a heap takes $O(Nlog(N))$ time. We perform N delete the max operations, one for each element in the list. Each of these delete the max operations takes $log(N)$ time. As a result, our running time is $O(Nlog(N) + Nlog(N) = 2Nlog(N)$, simplified this is $O(Nlog(N)))$.

The space complexity of heapsort is $O(1)$ because everything occurs in the same array.

Heapsort is an unstable sort because when placing items into heaps, it is not guaranteed that the item in the correct order will swim or sink.