Lesson:

Heap

Pre-Requisites

- Binary tree
- Complete binary tree

List of Concepts Involved

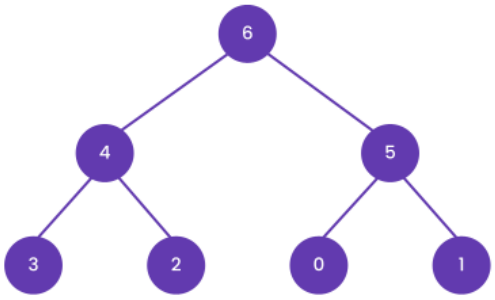
- Binary heap
- Representation of a binary tree using an array
- Insertion in a min heap
- Insertion in a max heap
- Deletion in a min heap
- Deletion in a max heap
- Heapify
- Using heapify to generate a min heap
- Using heapify to generate a max heap.
- Heap sort
- Sort in ascending order using heap sort
- Sort in descending order using heap sort

Topic 1: Binary Heap

A Binary Heap is a Binary Tree that follows the following properties-

1. It's a complete binary tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible).
2. A Binary Heap is either Min Heap or Max Heap.
   a. In a Min Binary Heap, the value of each parent node is smaller than the value of its child nodes.
   b. In a Max Binary Heap, the value of each parent node is greater than the value of its child nodes.

For example



It is a complete binary tree in which each parent node has a value greater than its children nodes.

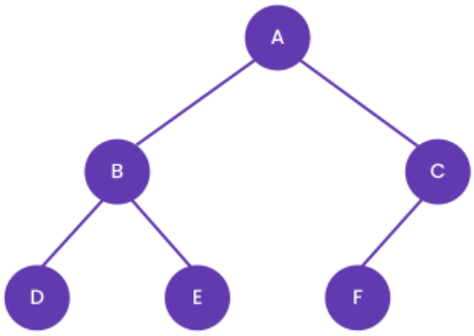
Topic 2: Representation of a heap using an array

In array representation of a binary tree, each index of the array represents a node of the tree-

1. The root of the binary tree is represented by the index 1 of the array.
2. If a node is present at index 'i', then-

a. Its left child will be at the index 2 * i.
b. Its right child will be at the index 2 * i + 1.
3. If a node is at index 'i', then its parent will be at index i / 2.

For example,



To represent it in the form of an array, let's take an array of size 8 i.e. the indices will range from 0 to 7. Initialize all the indices of the array with the value -1(you can take any value here) to represent empty locations.
We start by putting the root node at index 1 of the array.

A	-1	-1	-1	-1	-1	-1

B and C are the children of A.
=>We will put B at the index 2(2 * 1) and C at the index 3(2 * 1 + 1).
We can also confirm here that if we divide the indices of B and C by 2 i.e. 2 / 2 and 3 / 2, we get 1, the index of their parent.
Note: the indices should be given in the correct ordering i.e. left node should be given the index 2 * i and right child should be given the index 2 * i + 1.

A	B	C	-1	-1	-1	-1

D and E are the children of B.
=>We will put D at the index 4(2 * 2) and E at the index 5(2 * 2 + 1).

A	B	C	D	E	-1	-1

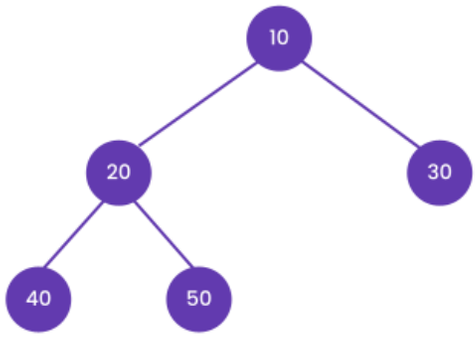
F is the left child of C.
=>We will put it at the index 6(2 * 3).

A	B	C	D	E	F	-1

The last index will remain empty as there is no right child of the node C.

Topic 3: Insertion in a min heap

To understand the insertion in a min heap, let's take an example,

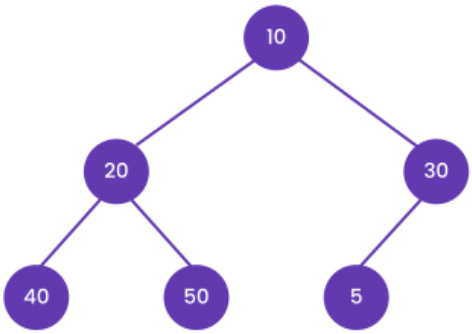


10	20	30	40	50

In this heap, let's say we are inserting an element with value 5. Since 5 is the smallest value in the heap, people think that we should insert it at the root and shift all other elements accordingly. However, we don't do it that way.

We need to make sure that the properties of the heap must be satisfied after the insertion. So we follow the following steps-

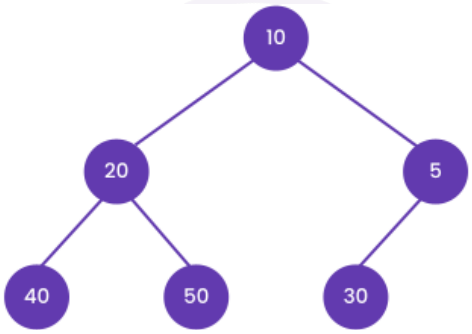
1. Insert the element at the last position available in the array.



10	20	30	40	50	5

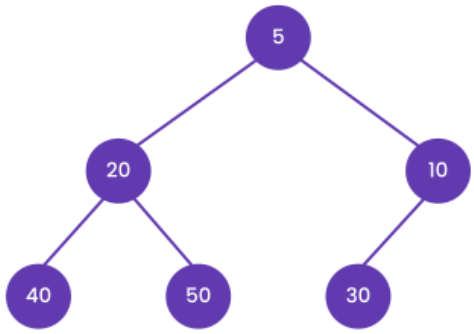
This ensures that the new heap being formed is also a complete binary tree.

2. Now we need to make sure that each node has a value smaller than its children. To do this we will compare the newly inserted node with its parent and if the parent has a larger value, we will swap the nodes.Index of parent of 5 = (Index of 5) / 2 = 6 / 2 = 3



10	20	5	40	50	30

3. Continue to do step 2 till either the newly inserted node becomes the root of the tree or it encounters a node with smaller value as its parent.



5	20	10	40	50	30

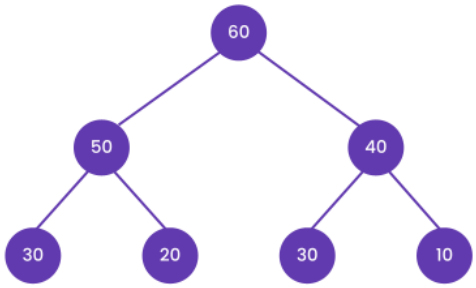
At this point we can say that the new element has been successfully inserted into our heap.

Now let's analyze the time complexity of the insertion.
In the worst case scenario, the newly inserted node will have to travel from the leaf position to the root node as seen in the above example. In such a case the number of swaps will be equal to the height of the binary tree.
We know that the height of a binary tree is log(n).
=>Time complexity of insertion in a heap in the worst case is O(log(n)).

Code
https://pastebin.com/C9dwUhvw

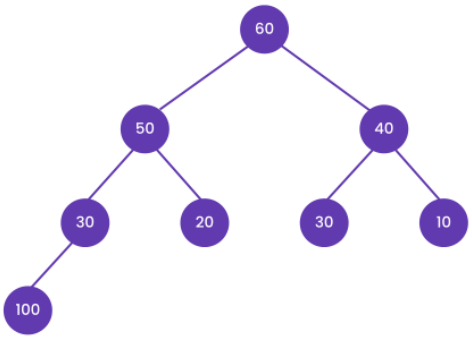
Topic 4: Insertion in a max heap

Insertion in a max heap is similar to insertion in a min heap. The only change that occurs here is the comparison between the parent and the child node. Let's take an example to understand this-

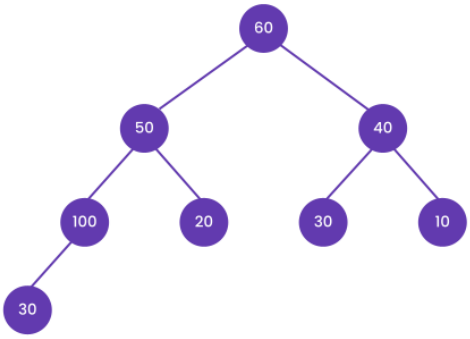


In this heap, let's say we are inserting an element with value 100.

1. Insert the element at the last position available in the array.

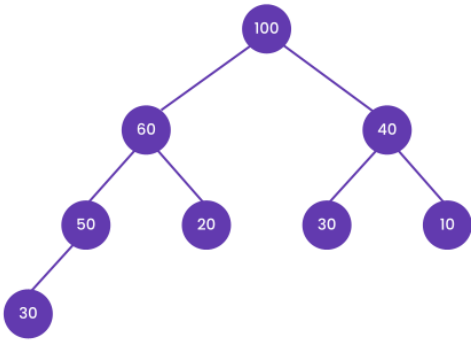


2. Compare its value with its parent. If its value is larger, swap the values.



60	50	40	100	20	30	10	30

3. Continue to do step 2 till either 100 reaches the root or the parent value is greater than 100.

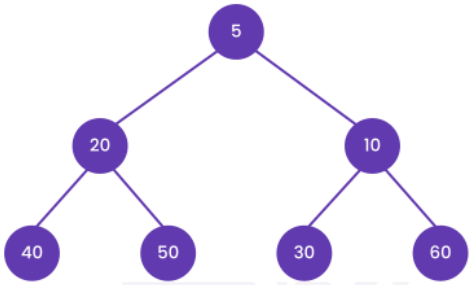


100	60	40	50	20	30	10	30

Code
https://pastebin.com/gjqgieAQ

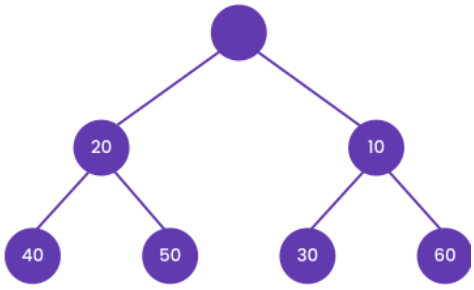
Topic 5: Deletion in a min heap

There is one thing that we need to note- **deletion in a heap can only be done from the root node.**
Now let's see how elements are deleted from the min heap using an example-



5	20	10	40	50	30	60

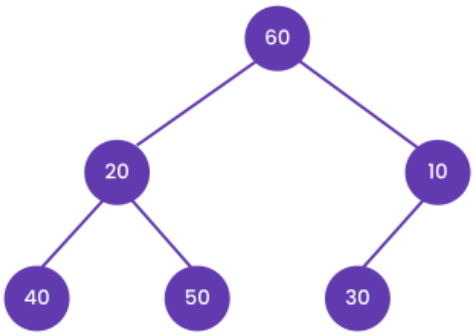
Let's remove the root element from the heap-



	20	10	40	50	30	60

Now going by the same principle in the insertion, we can't just bring any element as its replacement to the top. We need to ensure that the properties of a heap are being satisfied. So we do the following steps-

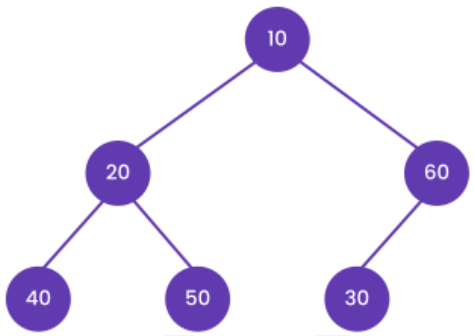
1. Bring the last element to the root node.



60	20	10	40	50	30

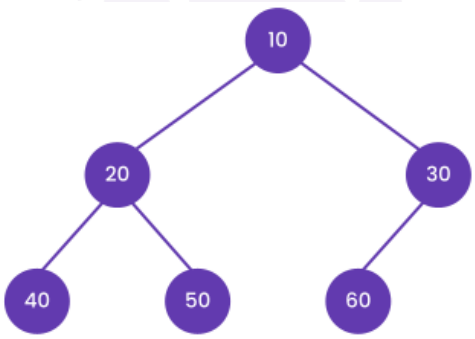
This ensures that the heap remains a complete binary tree after deletion of the root node.

2. Compare the values of its new children. Find the minimum between them. In our case, it's the node with value 10. If there is only one child then that is the child with min value.
3. If the value of the min child is less than the value of our current node. Swap the values.



10	20	60	40	50	30

4. Repeat the steps 2-3 till the value of the node is smaller than its children or you reach the leaf node.

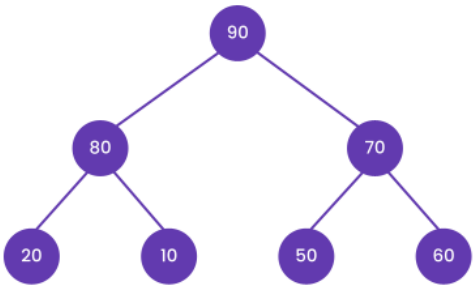


10	20	30	40	50	60

Code
https://pastebin.com/5RT8UhMT

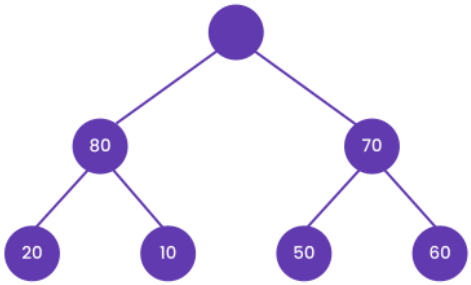
Topic 6: Deletion in a max heap

Deletion in a max heap is similar to deletion in a min heap. The only change that occurs here is the comparison between the parent and the child node. Let's take an example to understand this-



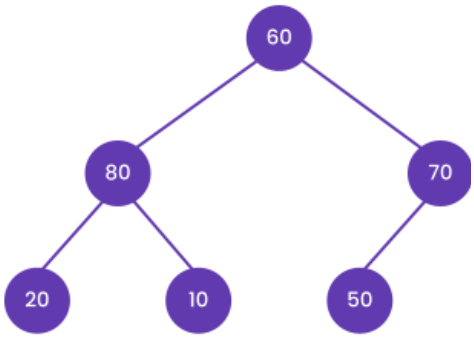
90	80	70	20	10	50	60

Let's remove the root element from the heap-



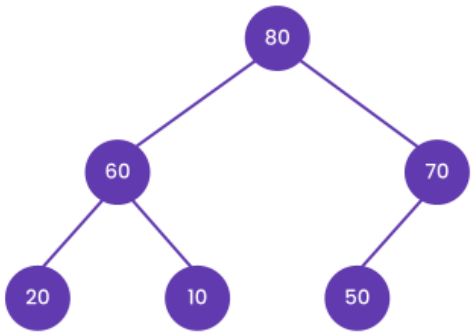
	80	70	20	10	50	60

1. Bring the last element to the top.



60	80	70	20	10	50

2. Compare the values of its new children. Find the maximum between them. In our case, it's the node with value 80. If there is only one child then that is the child with max value.
3. If the value of the max child is more than the value of our current node. Swap the values.



80	60	70	20	10	50

4. Repeat the steps 2-3 till the value of the node is more than its children or you reach the leaf node. In our case the first condition is true, so no more changes will be made to the heap.

Code
https://pastebin.com/nqm8tqy1

Topic 7: Heapify

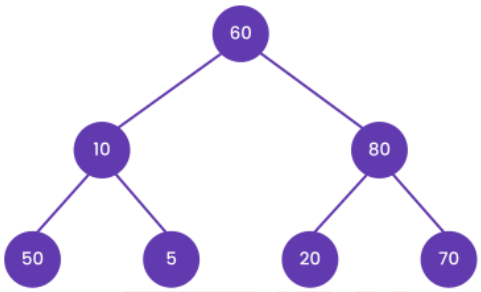
Heapify is the process of creating a heap data structure from a binary tree represented using an array. It is used to create Min-Heap or Max-heap.

Topic 8: Using heapify to generate a min heap

Min-heapify is a process of arranging the nodes in correct order so that they follow min-heap properties.

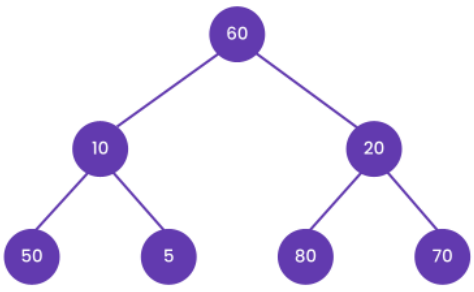
1. Start from the first non-leaf node of the heap whose index is given by n / 2.
2. Find the child of the current node whose value is minimum, let's call it min child. If there is only one element then it is the min child.
3. If the value of the min child is smaller than the value of the current node, swap the values.
4. Make the min child as the current node and repeat the steps 2-3.
5. Repeat the steps 2-4 for every other subtree in the heap.

For example



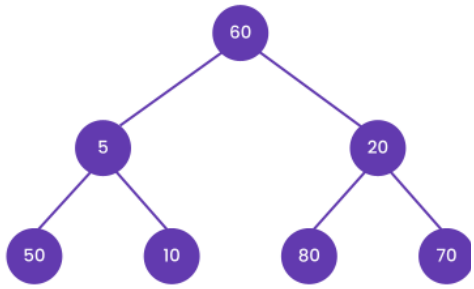
60	10	80	50	5	20	70

1. There are n = 7 nodes in the heap.
   n / 2 = 7 / 2 = 3 node with value 80.
   Min child for node 3 is node 6 i.e. with value 20. 20 is smaller than 80. So we will swap the two values.



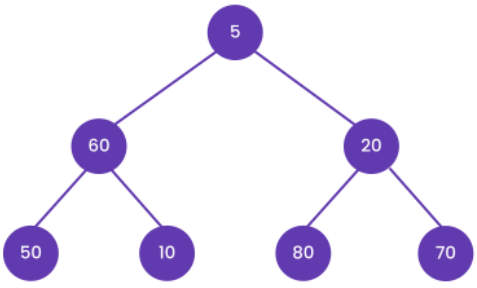
60	10	20	50	5	80	70

2. Now move to node 2. Node 2 has value 10. The min child for node 2 is node 5 i.e. with value 5. 5 is smaller than 10. So we will swap the two values.



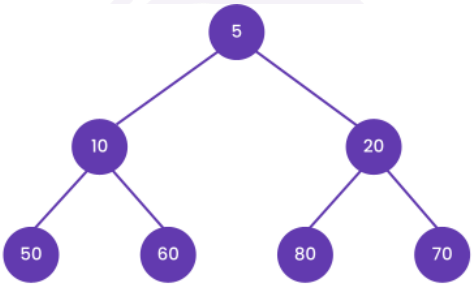
60	5	20	50	10	80	70

3. Now move to node 1. It has a value of 60. The min child for this node is node 2 i.e. with value 5. 5 is smaller than 60. So swap them.



5	60	20	50	10	80	70

It is still not done. We will again find the min child of node with value 60 i.e. node 1 now. The min child is node 5 i.e. with value 10. 10 is smaller than 60. So we will swap them.



5	10	20	50	60	80	70

Now we have got our min heap.

Code:
https://pastebin.com/RZHjZtc1

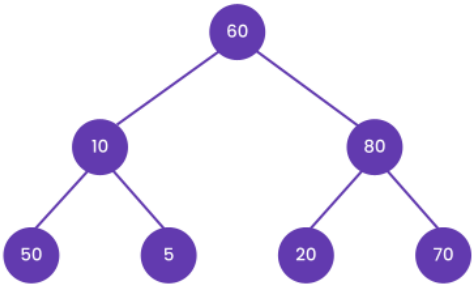
Topic 9: Using heapify to generate a max heap

The process to generate a max heap is the same as that of min heap except for the comparison operation-

1. Start from the first non-leaf node of the heap whose index is given by n / 2.
2. Find the child of the current node whose value is maximum, let's call it max child. If there is only one element then it is the max child.
3. If the value of the max child is greater than the value of the current node, swap the values.

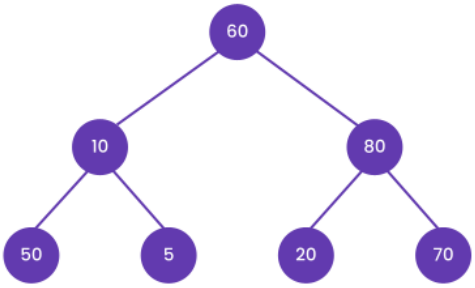
4. Make the max child as the current node and repeat the steps 2-3.
5. Repeat the steps 2-4 for every other subtree in the heap.

For example



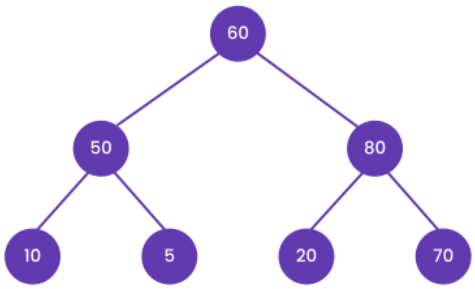
60	10	80	50	5	20	70

1. There are n = 7 nodes in the heap.
   n / 2 = 7 / 2 = 3 node with value 80.
   Max child for node 3 is node 7 i.e. with value 70. 70 is smaller than 80. So no swaps will be made.



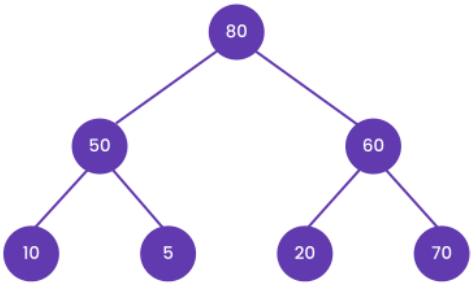
60	10	80	50	5	20	70

2. Now move to node 2. Node 2 has value 10. The max child for node 2 is node 4 i.e. with value 50. 50 is greater than 10. So we will swap the two values.



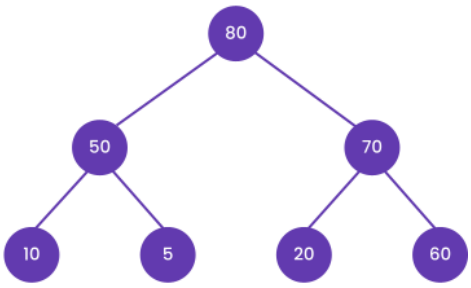
60	50	80	10	5	20	70

3. Now move to node 1. It has a value of 60. The min child for this node is node 3 i.e. with value 80. 80 is greater than 60. So swap them.



80	50	60	10	5	20	70

It is still not done. We will again find the max child of node with value 60 i.e. node 3 now. The max child is node 7 i.e. with value 70. 70 is greater than 60. So we will swap them.



80	50	70	10	5	20	60

Now we have got our max heap.

Code:
https://pastebin.com/MQ6fEdYH

Topic 10: Introduction to heapsort

Heap sort is a comparison-based sorting technique that uses Binary Heap data structure. At core it only has 2 **steps-**

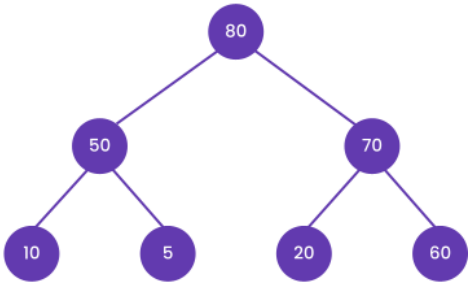
1. Convert the array into heap data structure using heapify.
2. One by one delete the root node of the heap and replace it with the last node in the heap and then heapify the root of the heap. Repeat this process till the size of the heap is greater than 1.

Topic 11: Sort in ascending order using heap sort.

To sort the elements of the array in ascending order, we will need to convert our array into a max heap.
For example, let's say we have the following array of 7 elements.

60	10	80	50	5	20	70

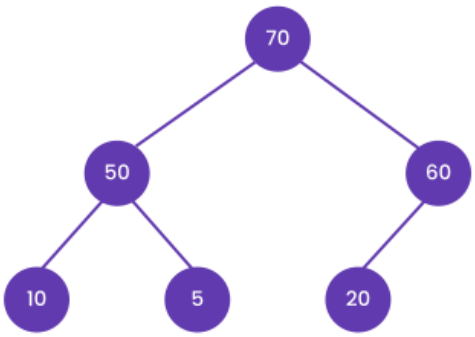
In the last lecture we have already seen how we can create a max heap from this array.
After creating the max heap, we will have-



80	50	70	10	5	20	60

Now we will remove the elements from this heap one by one. We already know how deletion works in heaps-

1. Remove 80 from the heap. After removal, we will get-



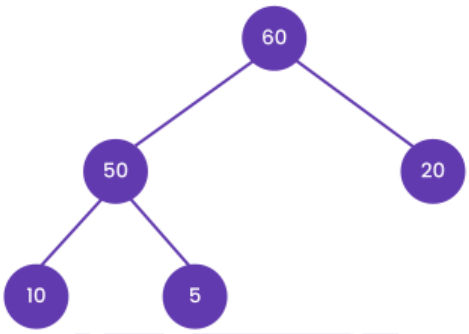
70	50	60	10	5	20	

In the above array, we can see that our heap now only extends to index 6 i.e. index 7 is empty. So we place the element that we have taken out at that location.

70	50	60	10	5	20	80

Although we have placed 80 at the last index, we still won't consider it as a part of our heap.

2. Remove 70 from the heap.

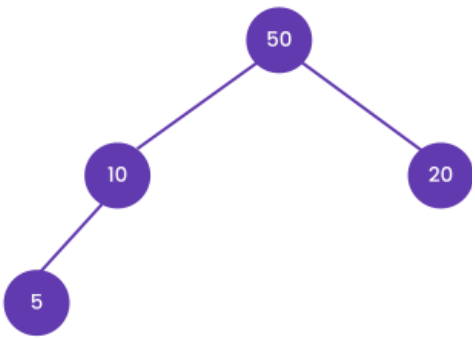


60	50	20	10	5		80

Again place the removed character at the place which is now empty in the array.

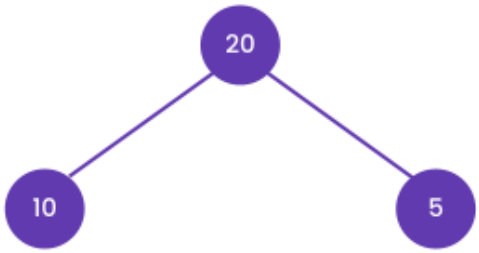
60	50	20	10	5	70	80

3. Repeat the same for the array till the heap is empt
   a. Removing 60



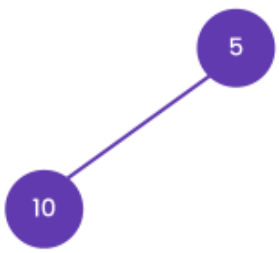
50	10	20	5	60	70	80

b. Removing 50



20	10	5	50	60	70	80

c. Removing 20



10	5	20	50	60	70	80

d. Removing 10

5	10	20	50	60	70	80

e. Removing 5

5	10	20	50	60	70	80

Our array is sorted.

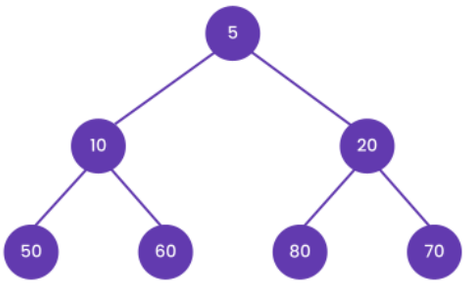
Code
https://pastebin.com/AHG5awXi

Topic 12: Sort in descending order using heap sort.

To sort the elements of the array in descending order, we will need to convert our array into a min heap.
For example, let's say we have the following array of 7 elements.

60	10	80	50	5	20	70

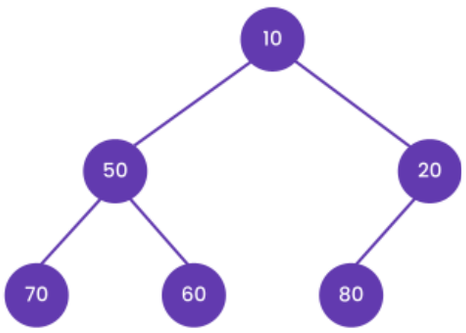
In the last lecture we have already seen how we can create a min heap from this array.
After creating the min heap, we will have-



5	10	20	50	60	80	70

Now we will remove the elements from this heap one by one. We already know how deletion works in heaps-

1. Remove 5 from the heap. After removal, we will get

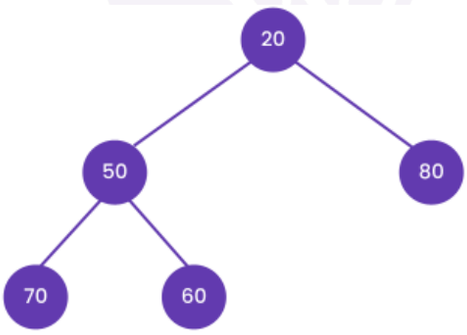


10	50	20	70	60	80	

Similar to the case of min heap, we will add the removed element at the empty space in the array.

10	50	20	70	60	80	5

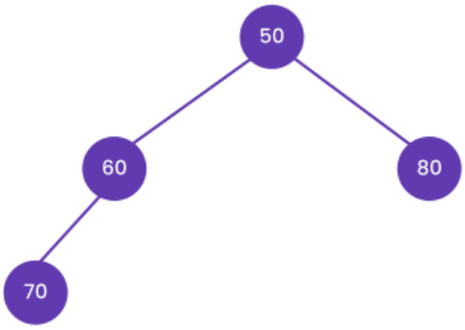
2. Remove 10 from the heap.



Again place the removed character at the place which is now empty in the array.

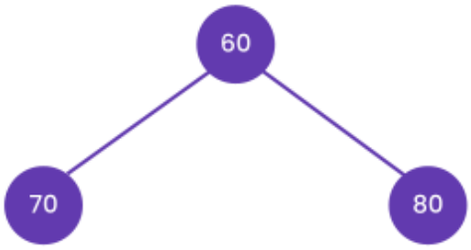
20	50	80	70	60	10	5

3. Repeat the same for the array till the heap is empty
   a. Removing 20



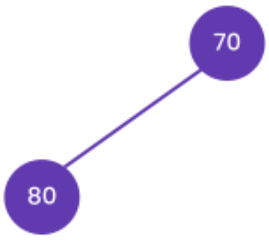
50	60	80	70	20	10	5

b. Removing 50



60	70	80	50	20	10	5

c. Removing 60



70	80	60	50	20	10	5

d. Removing 70

80	70	60	50	20	10	5

e. Removing 80

80	70	60	50	20	10	5

Our array is sorted.

Code
https://pastebin.com/hf8KD1wY

Q1. Given an array arr[] and an integer K where K is smaller than size of array, the task is to find the Kth smallest element in the given array. It is given that all array elements are distinct.
Note :- l and r denotes the starting and ending index of the array.
Input: N = 6
arr[] = 7 10 4 3 20 15
K = 3
Output : 7
Code link: https://pastebin.com/U4Z9G9nM
Explanation:

- The swap function is a utility function used to swap two integers. It takes the addresses of two integers as arguments and performs the swap operation.
- The MinHeap class is defined, representing a min heap data structure. It has private member variables: harr

(pointer to the array of elements in the heap), capacity (maximum possible size of the heap), and heap_size (current number of elements in the heap).

- The MinHeap constructor is defined. It takes an array a and its size size as arguments. The constructor initializes the heap_size to the given size and assigns the array a to the harr member variable. It then performs the MinHeapify operation on each non-leaf node of the heap.
- The parent, left, and right member functions are defined to calculate the indices of the parent, left child, and right child nodes, respectively, given an index i.
- The extractMin member function is defined to remove and return the minimum element (root) from the min heap. It first checks if the heap is empty and returns INT_MAX if it is. Then, it stores the minimum value (root) in a variable root. If there are more than one element in the heap, it replaces the root with the last element in the heap and calls MinHeapify to restore the heap property. Finally, it decreases the heap_size by 1 and returns the stored minimum value.
- The MinHeapify member function is a recursive function to heapify a subtree rooted at a given index i. It compares the value at index i with its left and right child nodes (if they exist) to find the smallest value among the three. If the smallest value is not at index i, it swaps the values and recursively calls MinHeapify on the affected subtree.
- The kthSmallest function is defined to find the kth smallest element in an array. It takes the input array arr, the size N of the array, and the value of K indicating the kth smallest element to find. Inside the function, a min heap mh is created using the MinHeap constructor. Then, it performs the extractMin operation K-1 times to extract the smallest elements from the heap, effectively finding the Kth smallest element. The function returns the root (minimum) element of the heap, which is the kth smallest element.

Output:

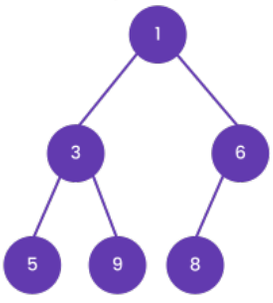
```
K'th smallest element is 5

...Program finished with exit code 0
Press ENTER to exit console.
```

Q2. Given an array representing min heap. Convert it into max heap.

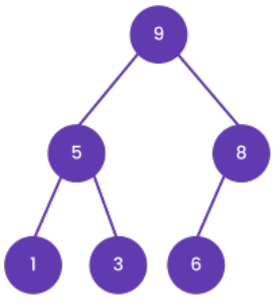
Input:

Min heap: 1, 3, 6, 5, 9, 8



Output:

Max heap: 9, 5, 8, 1, 3, 6



A2. Code: https://pastebin.com/MssUPnZc

Explanation:

1. Start from the bottommost and rightmost internal node in the min heap.
2. Heapify all internal nodes in a bottom-up manner using the maxHeapify() function.
3. By doing this, we are ensuring that every parent node in the heap has a greater value than its children nodes.
4. After heapifying all internal nodes, the resulting heap will be a max heap.

In summary, the idea is to swap the child nodes with their parent nodes until the parent node is greater than its children, ensuring that every parent node in the heap has a greater value than its children nodes, resulting in a max heap.

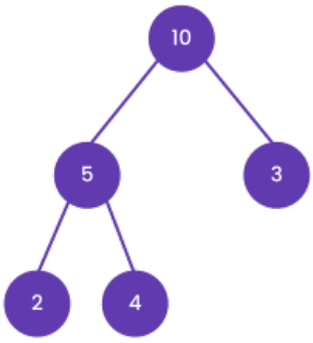
Time complexity: The time complexity of this code is O(n) because the *convertMaxHeap* function iterates through each internal node of the heap once and performs a constant amount of work per iteration.

Space complexity: The space complexity of this code is O(1) because the algorithm modifies the input array in place and does not use any additional data structures that scale with the input size.

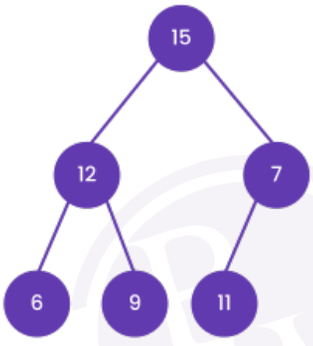
Q3. Given two binary max heaps as arrays, the task is to merge the given heaps

Input:

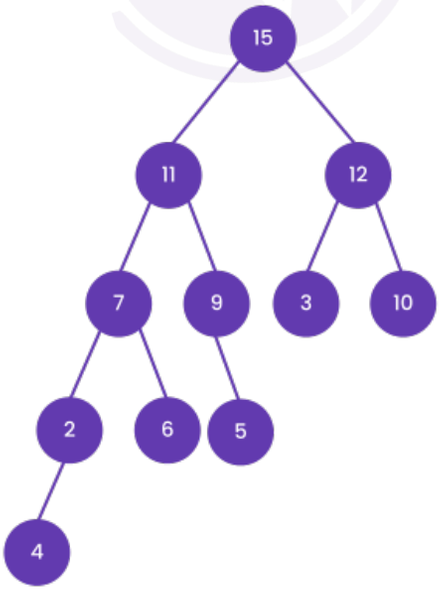
Max heap-1



Max heap-2



Output:



A3. Solution Code: https://pastebin.com/S6eCUqbU

Explanation:

1. Create a new array that can hold all the elements of the two heaps.
2. Copy the elements of the two heaps into the new array.
3. Build a max heap from the new array by repeatedly swapping elements downwards to satisfy the max heap

property. Start at the last parent node and work backwards towards the root.
4. The new array now represents the merged heap.

Essentially, we are merging the two heaps by combining their elements into a new array, and then rebuilding the heap from scratch to satisfy the max heap property. The key insight is that we can use the same algorithm for building a heap from an array to build a heap from the merged array.

Time complexity: The time complexity of the program to merge two max heaps using the array representation is O(n log n), where n is the total number of elements in both heaps. This is because we first create a new array that can hold all the elements of the two heaps, which takes O(n) time. We then build a max heap from the new array using the heapify algorithm, which has a time complexity of O(n log n). Therefore, the overall time complexity of the program is O(n log n).

Space complexity: O(n), where n is the total number of elements in both heaps. This is because we create a new array that can hold all the elements of the two heaps, which requires O(n) space. Therefore, the program has a linear space complexity in terms of the total number of elements in both heaps.

Q4. Given an array A[] of N positive integers and two positive integers K1 and K2. Find the sum of all elements between K1th and K2th smallest elements of the array. It may be assumed that (1 <= k1 < k2 <= n).

Input:

N = 7
A[] = {20, 8, 22, 4, 12, 10, 14}
K1 = 3, K2 = 6

Output:

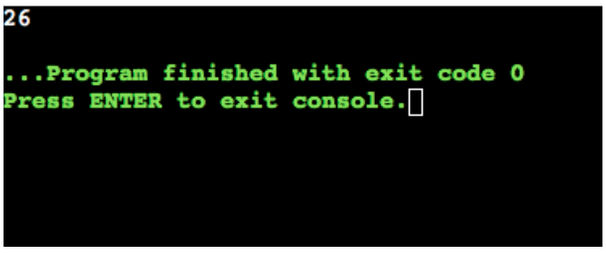
26

Explanation:

1. The minheapify function is defined. It takes three arguments: the array a, the index of the current node index, and the size of the array n. This function is used to maintain the min heap property by comparing the value at index with its left and right child nodes (if they exist) and swapping if necessary. It then recursively calls minheapify on the affected subtree.
2. In the main function, an array a is initialized with some values, and the size of the array n is calculated.
3. The for loop starting from (n / 2) - 1 to 0 is used to build the initial min heap by calling minheapify on each node in reverse order.
4. The value of k1 and k2 is decreased by 1 to convert them to 0-based indexing.
5. The first step is to extract the minimum element k1 times. This is achieved by swapping the minimum element at index 0 with the last element, reducing the size of the array, and calling minheapify on the root. This step is performed using a for loop from 0 to k1.
6. The second step is to extract the minimum element k2 - k1 - 1 times and calculate the sum of these elements. This step is similar to the first step but performed from k1 + 1 to k2 - 1. The sum of the extracted elements is accumulated in the ans variable.
7. Finally, the value of ans is printed, which represents the sum of elements between the k1th and k2th smallest elements in the array.

Code link: https://pastebin.com/ZXejyAHT

Output:



Upcoming lecture

- Priority queue