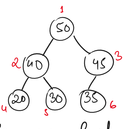
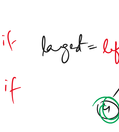


No of leaf nodes = $\frac{n}{2}+1$

No of non-leaf nodes = n - leaf nodes.

* When we convert an array into a heap, we just need to heapify the non-leaf nodes ($i = \frac{n}{2}$ to $i = 1$); because, the rest of the $\frac{n}{2}$ leaf nodes will automatically be converted to a min or max heap. (Important Note) *****



[Max Heap]

n = 5

(2 < 5)

a[largest] < a[l]

largest = l

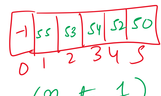
if

if (largest != i)

swap

heapify(arr, n)

50, 53, 55, 52, 50



($\frac{n}{2}$ to 1)

55	53	54	52	50
1	2	3	4	5

Heap Sort :→ $n \log n$

n → elements ?

$\log - n$

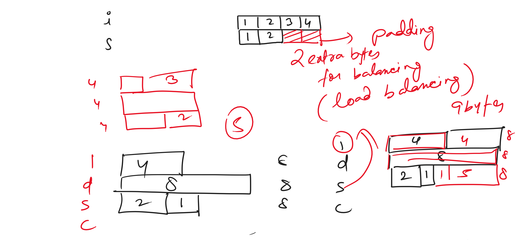

```
int size = n;
while (size > 1) {
    swap(arr[1], arr[size]);
    size--;
    heapify(arr, n, 1);
}
```

50	53	54	52	55
54	53	50	52	55
52	53	50	54	55
53	52	50	54	55
50	52	53	54	55
52	50			
50	52			

* Greedy Algorithms :→

concept → * Padding & Greedy Alignment

GFG → * Minimum No of Coins → GFG
* Activity Selection Problem
* Job Sequencing / Scheduling Problem
* Minimum Cost of Ropes → GFG
* Chocolate Distribution Problem | Minimum Absolute Difference
* Policemen Catching Thieves
* Huffman Encoding → GFG
* Fractional Knapsack | 0,1 Knapsack
* Nikunj & Donuts !



* When the compiler observes that there's imbalance of the size of the data types, it adds extra bytes as "padding" to balance the load.

* To save memory, we as users can implement descending order of data types to decrease the padding and make the code more efficient. This is called "Greedy Alignment".

* Minimum number of coins to get a particular value V.

Max 20

coins[] = {1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000}

coin(V = 91.

(4) [50, 20, 20, 1]

```
for (last → 0)
    while (V >= coins[i]) {
        V -= coins[i];
    }
return res;
```

91 - 50 = 41 - 20 = 21 - 20 = 1 = 1 - 1 = 0

(31) 20, 10, 1

Feedback :→ 12156