🧠 Creating a min heap out of an Array:-

Ye code ek **array ko Min-Heap** me convert karta hai using a bottom-up approach.

Heap ek special **binary tree** hota hai jo array me store kiya ja sakta hai.

Is case me, tu **Min Heap** bana raha hai — jisme **parent node hamesha child se chhoti hoti hai**.

📌 Code Breakdown and Explanation

🔶 `int arr[] = {-1,10,2,14,11,1,4};`

- Tu ek array le raha hai:

```
Index:  0  1  2  3  4  5  6
Value: -1 10  2 14 11  1  4
```

- Note: Index 0 pe `1` rakha gaya hai just as **dummy**, kyunki heap ko 1-based indexing se build karna easy hota hai.

🔶 `int n = sizeof(arr)/sizeof(arr[0]);`

- Array ke elements ka count nikala ja raha hai → `n = 7` (indexes 0 to 6)

🔶 Heapify Function:

```
void heapify(int i, int arr[], int n)
```

- **Purpose:** `arr[i]` ko correct position pe le jao so that heap property maintain ho.
- Yeh function ye check karta hai ki kya parent node (`arr[i]`) apne left ya right child se bada hai.

- Agar bada hai, to swap kar ke neeche bhej deta hai (down-heapify or heapify-down).

📌 Heap Structure (Binary Tree logic):

Agar index `i` pe koi node hai:

- `left = 2*i`
- `right = 2*i + 1`

Ye tree structure me child pointers batate hain.

🔄 `for(int i = n/2; i >= 1; i--)`

- **Maqsad:** Last non-leaf node se lekar root tak heapify karna.

Kyun? Kyunki leaf nodes already heap hote hain, unko heapify karne ki zarurat nahi hoti.

Example:

`n = 7` → nodes at index `1` to `6`

So we run:

- `i = 3`
- `i = 2`
- `i = 1`

Is order me, bottom-up heapify hota hai.

🔁 Working Example:

Input array: `{10, 2, 14, 11, 1, 4}`

After heapify:

Heapified Tree (Min-Heap):

```
      1
     / \
```

```
    2     4
   / \   /
  14  11 10
```

Which corresponds to array:

```
arr[] = {-1, 1, 2, 4, 14, 11, 10}
```

🔚 `for(int i = 1; i < n; i++)`

- Ye loop Min-Heap banne ke baad usse print karta hai:

🔎 Output:

```
1 2 4 14 11 10
```

✅ Final Summary

Part	Explanation
Goal	Given array ko Min-Heap me convert karna
Method	Bottom-up heapify using complete binary tree logic
Heapify	Parent compare with child and swap down if needed
Print	Final array (min-heap) print hota hai

🔷 Mera `heapify()` Function Recap

```
void heapify(int i,int arr[],int n){
    while(true){
        int left = 2*i;
        int right = 2*i + 1;
```

```
        if (left >= n) break;

        if (right >= n){
            if(arr[i] > arr[left]){
                swap(arr[i], arr[left]);
                i = left;
                continue;
            }
            else break;
        }

        if(arr[left] < arr[right]){
            if(arr[i] > arr[left]){
                swap(arr[i], arr[left]);
                i = left;
            } else break;
        } else {
            if(arr[i] > arr[right]){
                swap(arr[i], arr[right]);
                i = right;
            } else break;
        }
    }
}
```

🎯 Example Array

```
int arr[] = {-1, 10, 2, 14, 11, 1, 4};  // 1-indexed, n = 7
```

Yani:

```
Index:  1   2   3   4   5   6
Value: 10   2  14  11   1   4
```

Step-by-Step Heapify Call

🔷 `i = 3` (first call from `for` loop)

- `arr[3] = 14`
- Left = 6 → `arr[6] = 4`
- Right = 7 → out of bounds

🧮 Comparison:

- Only left child exists.
- `14 > 4` → Swap
- New array: `{ -1, 10, 2, 4, 11, 1, 14 }`
- Continue heapifying at `i = 6` → no children, stop.

🔷 `i = 2`

- `arr[2] = 2`
- Left = 4 → `arr[4] = 11`
- Right = 5 → `arr[5] = 1`

🧮 Comparison:

- Left = 11, Right = 1 → right is smaller
- `2 > 1` → Swap with right
- New array: `{ -1, 10, 1, 4, 11, 2, 14 }`
- Continue at `i = 5` → no children, stop

🔷 `i = 1`

- `arr[1] = 10`
- Left = 2 → `arr[2] = 1`
- Right = 3 → `arr[3] = 4`

🧮 Comparison:

- Left = 1, Right = 4 → left is smaller
- `10 > 1` → Swap
- New array: `{ -1, 1, 10, 4, 11, 2, 14 }`
- Continue at `i = 2`

🔁 At `i = 2`:

- Left = 4 → `arr[4] = 11`
- Right = 5 → `arr[5] = 2`
- Right is smaller (2)
- `10 > 2` → Swap
- Final array: `{ -1, 1, 2, 4, 11, 10, 14 }`

✅ Final Min-Heap:

```
      1
     / \
    2   4
   / \   \
  11 10   14
```

Corresponding array:

```
arr[] = {-1, 1, 2, 4, 11, 10, 14}
```

🔥 Summary of Heapify Behavior:

- `heapify(i)` ensures that **subtree rooted at i** satisfies min-heap.
- It **swaps parent with the smaller child** if needed and continues down.
- Tu har node pe yeh check kar raha hai:

✅ Kya left ya right me koi child chhota hai?

✅ Agar haan, to swap kar aur neeche check karte jao.

📌 Initial Array (Before heapify):

```
Index:  1   2   3   4   5   6   7
Value: 10   2  14  11   1   4
```

🔁 Start Heapify (Bottom-Up from `i = n/2` to `1`)

🔷 Step 1: `i = 3`

- Node = `arr[3] = 14`
- Left = `arr[6] = 4`, Right = out of bounds

Comparison:

`14 > 4` → ✅ Swap

➡️ New array:

```
{-1, 10, 2, 4, 11, 1, 14}
```

📊 **Tree after** `heapify(3)`

```
      10
     /  \
    2    4
   / \  /
  11  1 14
```

🔷 Step 2: `i = 2`

- Node = `arr[2] = 2`

- Left = `arr[4] = 11`, Right = `arr[5] = 1`

Comparison:

Right is smaller (1)

`2 > 1` → ✅ Swap

➡️ New array:

```
{-1, 10, 1, 4, 11, 2, 14}
```

📊 **Tree after** `heapify(2)`

```
      10
     /   \
    1     4
   / \   /
  11  2 14
```

🔷 Step 3: `i = 1`

- Node = `arr[1] = 10`
- Left = `arr[2] = 1`, Right = `arr[3] = 4`

Comparison:

Left is smaller (1)

`10 > 1` → ✅ Swap

➡️ Now at `i = 2`

- Node = `10`, Left = `11`, Right = `2`

  Right is smaller → `10 > 2` → ✅ Swap again

➡️ Final array:

```
{-1, 1, 2, 4, 11, 10, 14}
```

📊 **Tree after** `heapify(1)`

```
      1
     / \
    2   4
   / \   \
  11  10  14
```

🎯 Final Output:

Array:

```
{-1, 1, 2, 4, 11, 10, 14}
```

Tree form:

```
      1
     / \
    2   4
   / \   \
  11  10  14
```

🔥 Summary Table:

Step	Heapify(i)	Swaps	Tree Root
1	i = 3	14 ↔ 4	10
2	i = 2	2 ↔ 1	10
3	i = 1	10 ↔ 1, then 10 ↔ 2	1
