📘 LeetCode 658 – Find K Closest Elements

✅ Problem Statement:

Given a sorted integer array `arr`, two integers `k` and `x`, return the `k` closest integers to `x` in the array.

The result should also be sorted in **ascending order**.

An integer `a` is closer to `x` than `b` if:

- `|a - x| < |b - x|`, or
- `|a - x| == |b - x|` and `a < b`

🔍 Approach: Using Max Heap (Priority Queue)

🔧 Idea:

- Traverse the array.
- For each element, calculate its absolute distance from `x`.
- Maintain a **max heap (priority queue)** of size `k`, where each element is stored as a pair: `{distance, value}`.
- If the heap size exceeds `k`, remove the top element (which is the farthest so far).
- Finally, extract all values from the heap, store them in a vector, sort it, and return it.

🧠 Why Max Heap?

- We want to **keep the k closest elements**, and remove the element with the **largest distance** whenever the size exceeds `k`.

- Max heap allows us to efficiently remove the element with the largest distance (`O(log k)`).

✅ Code (Your Original):

```
class Solution {
public:
    typedef pair<int,int> pi;
    vector<int> findClosestElements(vector<int>& arr, int k, int x) {
        priority_queue<pi> pq;
        for(int val: arr){
            int distance = abs(val - x);
            pq.push({distance, val});
            if(pq.size() > k){
                pq.pop();
            }
        }
        vector<int> ans;
        while(pq.size() > 0){
            ans.push_back(pq.top().second);
            pq.pop();
        }
        sort(ans.begin(), ans.end());
        return ans;
    }
};
```

📈 Time Complexity:

- Building the heap: `O(n * log k)`
- Extracting and sorting final `k` elements: `O(k * log k)`
- **Total:** `O(n * log k + k * log k)`

📦 Space Complexity:

- `O(k)` for the heap
- `O(k)` for the result vector
