**Kth Smallest Element**

class Solution {

public:

int kthSmallest(std::vector<int> &arr, int k) {

int max\_element = \*std::max\_element(arr.begin(), arr.end());

vector<int> freq(max\_element + 1, 0);

for (int num : arr) {

freq[num]++;

}

int count = 0;

for (int i = 0; i <= max\_element; i++) {

count += freq[i];

if (count >= k) {

return i;

}

}

return -1;

}

};  
  
**Minimize the Heights**

class Solution {

public:

int getMinDiff(int k, vector<int> &arr) {

if (arr.size() == 1) return 0;

std::sort(arr.begin(), arr.end());

int n = arr.size();

int difference = arr[n-1] - arr[0];

for (int i = 1; i < n; i++) {

int minHeight = std::min(arr[0] + k, arr[i] - k);

int maxHeight = std::max(arr[n-1] - k, arr[i-1] + k);

difference = std::min(difference, maxHeight - minHeight);

}

return difference;

}

};

**Parenthesis Checker**class Solution {

public:

bool isParenthesisBalanced(string& s) {

stack<char> st;

unordered\_map<char, char> match\_bracket = {{')', '('}, {'}', '{'}, {']', '['}};

for (char ch : s) {

if (ch == '(' || ch == '{' || ch == '[') {

st.push(ch);

} else if (ch == ')' || ch == '}' || ch == ']') {

if (st.empty() || st.top() != match\_bracket[ch]) {

return false;

}

st.pop();

}

}

return st.empty();

}

};  
  
**Equilibrium Point**  
  
class Solution {

public:

int equilibriumPoint(vector<int> &arr) {

long long tot\_sum = 0;

for (int num : arr) {

tot\_sum += num;

}

long long left\_sum = 0;

for (int i = 0; i < arr.size(); i++) {

tot\_sum -= arr[i];

if (left\_sum == tot\_sum) {

return i + 1;

}

left\_sum += arr[i];

}

return -1;

}

};  
  
**Binary Search**

class Solution {

public:

int binarysearch(vector<int> &arr, int k) {

int low = 0, high = arr.size() - 1;

int mid=0;

while (low <= high) {

mid = (low + high) / 2;

if (arr[mid] == k) {

return mid;

}

else if (arr[mid] > k) {

high = mid - 1;

}

else {

low = mid + 1;

}

}

return -1;

}

};

**Next Greater Element**class Solution {

public:

vector<int> nextLargerElement(vector<int>& arr) {

int n = arr.size();

vector<int> res(n, -1);

stack<int> st;

for (int i = n - 1; i >= 0; i--) {

while (!st.empty() && st.top() <= arr[i]) {

st.pop();

}

if (!st.empty()) {

res[i] = st.top();

}

st.push(arr[i]);

}

return res;

}

};  
  
**Union of Arrays with Duplicates**

class Solution {

public:

int findUnion(vector<int>& a, vector<int>& b) {

unordered\_set<int> unionSet;

for (int num : a) {

unionSet.insert(num);

}

for (int num : b) {

unionSet.insert(num);

}

return unionSet.size();

}

};