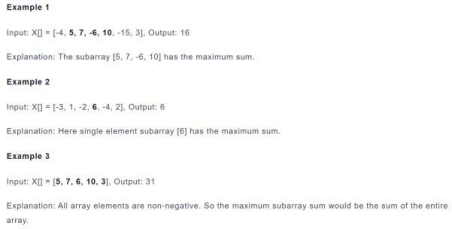
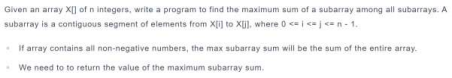
**DIVIDE AND CONQUER CODING PROBLEMS**

**1.**

****

#include <bits/stdc++.h>

using namespace std;

// Finds max sum crossing the midpoint

int maxCrossingSum(const vector<int>& arr, int l, int m, int h) {

int sum = 0, leftMax = INT\_MIN;

for (int i = m; i >= l; --i) {

sum += arr[i];

leftMax = max(leftMax, sum);

}

sum = 0;

int rightMax = INT\_MIN;

for (int i = m + 1; i <= h; ++i) {

sum += arr[i];

rightMax = max(rightMax, sum);

}

return leftMax + rightMax;

}

// Recursive function to find max subarray in arr[l…h]

int maxSubarrayDC(const vector<int>& arr, int l, int h) {

if (l == h) return arr[l]; // Base case: single element

int m = l + (h - l) / 2;

int leftMax = maxSubarrayDC(arr, l, m);

int rightMax = maxSubarrayDC(arr, m + 1, h);

int crossMax = maxCrossingSum(arr, l, m, h);

return max({leftMax, rightMax, crossMax});

}

// Wrapper for convenience

int maxSubarraySum(const vector<int>& arr) {

if (arr.empty()) return 0; // Or handle empty array as needed

return maxSubarrayDC(arr, 0, arr.size() - 1);

}

int main() {

vector<int> arr = {-4, 5, 7, -6, 10, -15, 3};

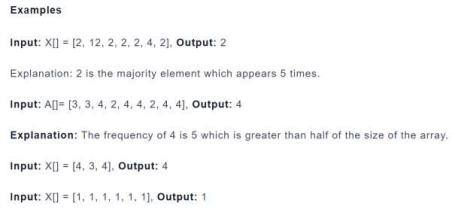
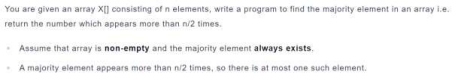
cout << "Max subarray sum = " << maxSubarraySum(arr) << "\n";

// Output: 16

return 0;

}

2.



#include <bits/stdc++.h>

using namespace std;

int majorityElement(const vector<int>& arr) {

int candidate = 0, count = 0;

for (int num : arr) {

if (count == 0) {

candidate = num;

count = 1;

} else if (num == candidate) {

count++;

} else {

count--;

}

}

// Optional validation if majority existence isn't guaranteed

// count = 0;

// for (int num : arr) if (num == candidate) count++;

// if (count > arr.size() / 2) return candidate;

return candidate;

}

int main() {

vector<int> X = {2, 12, 2, 2, 2, 4, 2};

cout << majorityElement(X) << "\n"; // Outputs 2

return 0;

}