Q.1   
  
#include <iostream>

#include <vector>

#include <climits>

using namespace std;

int main() {

int n;

cin >> n;

vector<int> prices(n);

for (int i = 0; i < n; i++) {

cin >> prices[i];

}

int minLoss = INT\_MAX;

for (int i = 0; i < n; i++) {

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

if (prices[i] > prices[j]) {

int loss = prices[i] - prices[j];

if (loss < minLoss) {

minLoss = loss;

}

}

}

}

cout << minLoss << endl;

return 0;

}

q.2

#include <iostream>

#include <vector>

using namespace std;

int main() {

int n, m;

cin >> n;

vector<int> arr(n);

for (int i = 0; i < n; i++) {

cin >> arr[i];

}

cin >> m;

int maxRemainder = 0;

for (int i = 0; i < n; i++) {

int sum = 0;

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

sum += arr[j];

int remainder = sum % m;

if (remainder > maxRemainder) {

maxRemainder = remainder;

}

}

}

cout << maxRemainder << endl;

return 0;

}

q.3

#include <iostream>

#include <vector>

using namespace std;

void countingSort(vector<int>& arr) {

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

vector<int> count(maxVal + 1, 0);

for (int num : arr) {

count[num]++;

}

int index = 0;

for (int i = 0; i <= maxVal; i++) {

while (count[i] > 0) {

arr[index++] = i;

count[i]--;

}

}

}

int main() {

int n;

cin >> n;

vector<int> arr(n);

for (int i = 0; i < n; i++) {

cin >> arr[i];

}

countingSort(arr);

for (int num : arr) {

cout << num << " ";

}

return 0;

}

q.4

#include <iostream>

#include <vector>

using namespace std;

int knapsack(int W, vector<int>& weights, vector<int>& values, int N) {

vector<vector<int>> dp(N + 1, vector<int>(W + 1, 0));

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

for (int j = 1; j <= W; j++) {

if (weights[i - 1] <= j) {

dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weights[i - 1]] + values[i - 1]);

} else {

dp[i][j] = dp[i - 1][j];

}

}

}

return dp[N][W];

}

int main() {

int N, W;

cin >> N >> W;

vector<int> weights(N), values(N);

for (int i = 0; i < N; i++) {

cin >> weights[i] >> values[i];

}

cout << knapsack(W, weights, values, N) << endl;

return 0;

}

q.5

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

int minCostOfEncoding(vector<int>& frequencies) {

priority\_queue<int, vector<int>, greater<int>> minHeap;

for (int freq : frequencies) {

minHeap.push(freq);

}

int totalCost = 0;

while (minHeap.size() > 1) {

int first = minHeap.top(); minHeap.pop();

int second = minHeap.top(); minHeap.pop();

int cost = first + second;

totalCost += cost;

minHeap.push(cost);

}

return totalCost;

}

int main() {

int N;

cin >> N;

vector<int> frequencies(N);

for (int i = 0; i < N; i++) {

cin >> frequencies[i];

}

cout << minCostOfEncoding(frequencies) << endl;

return 0;

}

q.6

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

struct Node {

int freq;

Node\* left;

Node\* right;

Node(int f) : freq(f), left(nullptr), right(nullptr) {}

};

struct Compare {

bool operator()(Node\* a, Node\* b) {

return a->freq > b->freq;

}

};

void calculateDepths(Node\* root, int depth, int& totalCost) {

if (!root->left && !root->right) {

totalCost += root->freq \* depth;

return;

}

calculateDepths(root->left, depth + 1, totalCost);

calculateDepths(root->right, depth + 1, totalCost);

}

int weightedSumOfEncodedLengths(vector<int>& frequencies) {

priority\_queue<Node\*, vector<Node\*>, Compare> minHeap;

for (int freq : frequencies) {

minHeap.push(new Node(freq));

}

while (minHeap.size() > 1) {

Node\* left = minHeap.top(); minHeap.pop();

Node\* right = minHeap.top(); minHeap.pop();

Node\* merged = new Node(left->freq + right->freq);

merged->left = left;

merged->right = right;

minHeap.push(merged);

}

Node\* root = minHeap.top();

int totalCost = 0;

calculateDepths(root, 0, totalCost);

return totalCost;

}

int main() {

int N;

cin >> N;

vector<int> frequencies(N);

for (int i = 0; i < N; i++) {

cin >> frequencies[i];

}

cout << weightedSumOfEncodedLengths(frequencies) << endl;

return 0;

}