**Coding practice Problems 11-11-2024**

**1. 0/1 Knapsack Problem**

class Solution {

public:

int knapSack(int capacity, vector<int> &val, vector<int> &wt) {

int n = val.size();

vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));

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

for (int w = 1; w <= capacity; w++) {

if (wt[i - 1] <= w) dp[i][w] = max(dp[i - 1][w], val[i - 1] + dp[i - 1][w - wt[i - 1]]);

else dp[i][w] = dp[i - 1][w];

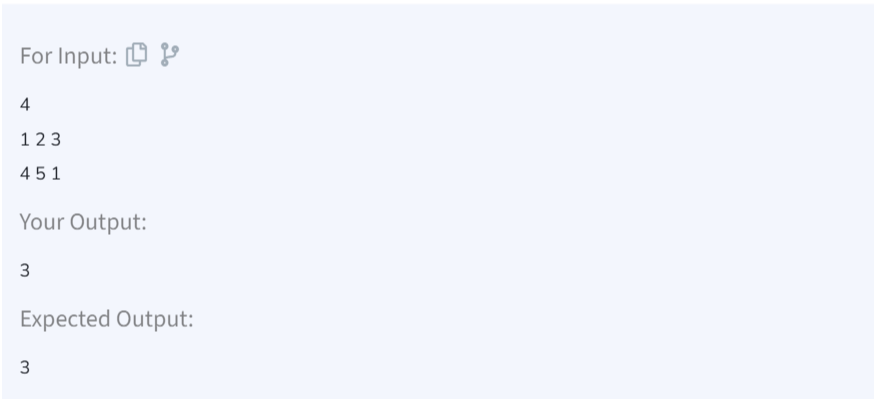
}

}

return dp[n][capacity];

}

};



TIME COMPLEXITY: O(n\*capacity)

SPACE COMPLEXITY: O(n\*capacity)

**2. Floor in Sorted Array**

class Solution {

public:

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

int left = 0, right = arr.size() - 1, floorIndex = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] <= k) {

floorIndex = mid;

left = mid + 1;

} else {

right = mid - 1;

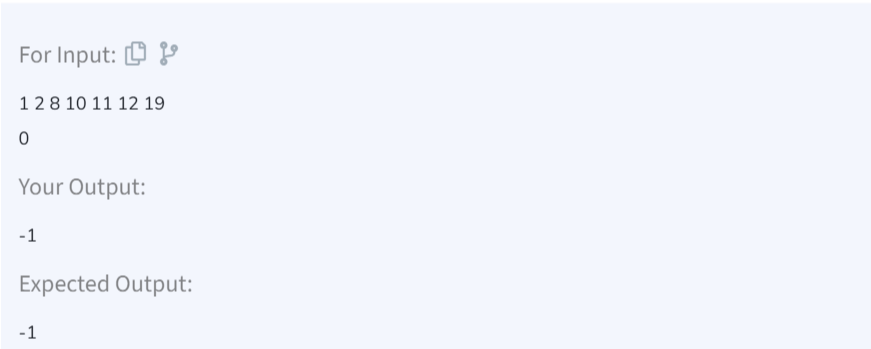
}

}

return floorIndex;

}

};



TIME COMPLEXITY: O(log n)

SPACE COMPLEXITY: O(1)

**3. Check Equal Arrays**

class Solution {

public:

bool check(vector<int>& arr1, vector<int>& arr2) {

if (arr1.size() != arr2.size()) return false;

unordered\_map<int, int> mp;

for (int num : arr1) mp[num]++;

for (int num : arr2) {

if (mp.find(num) == mp.end() || mp[num] == 0) return false;

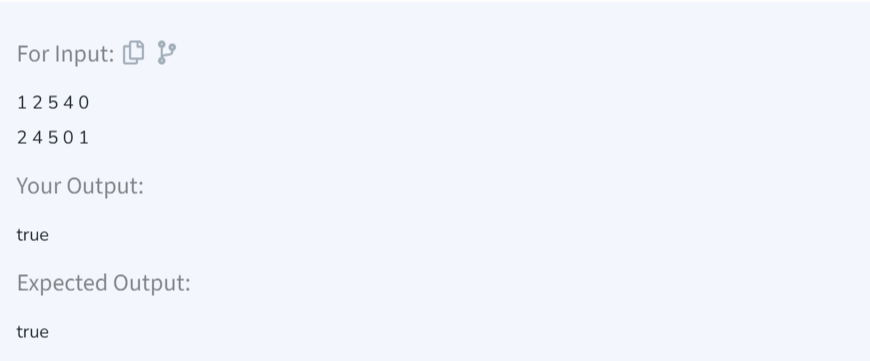
mp[num]--;

}

return true;

}

};



TIME COMPLEXITY: O(n)

SPACE COMPLEXITY: O(n)

**4. Palindrome Linked List**

struct Node {

int data;

Node \*next;

Node(int x) : data(x), next(NULL) {}

};

class Solution {

public:

bool isPalindrome(Node \*head) {

if (!head || !head->next) return true;

Node \*slow = head, \*fast = head;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

}

Node \*revHead = reverse(slow);

Node \*curr = head;

while (revHead) {

if (curr->data != revHead->data) return false;

curr = curr->next;

revHead = revHead->next;

}

return true;

}

private:

Node\* reverse(Node \*head) {

Node \*prev = NULL, \*next = NULL;

while (head) {

next = head->next;

head->next = prev;

prev = head;

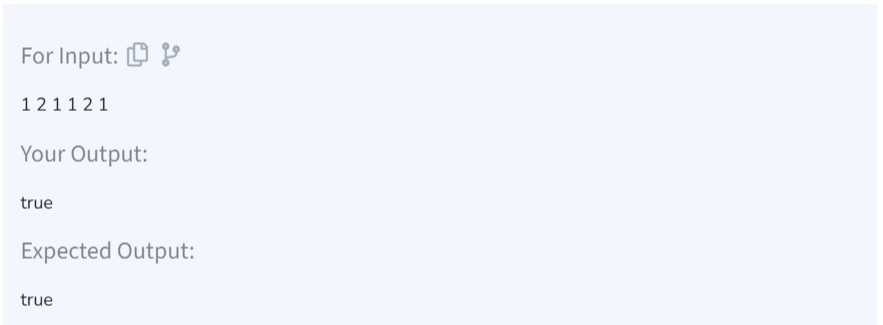
head = next;

}

return prev;

}

};



TIME COMPLEXITY: O(n)  
SPACE COMPLEXITY: O(1)

**5. Balanced Tree Check**

class Solution{

public:

bool isBalanced(Node \*root) {

return checkBalance(root) != -1;

}

private:

int checkBalance(Node \*root) {

if (!root) return 0;

int leftHeight = checkBalance(root->left);

if (leftHeight == -1) return -1;

int rightHeight = checkBalance(root->right);

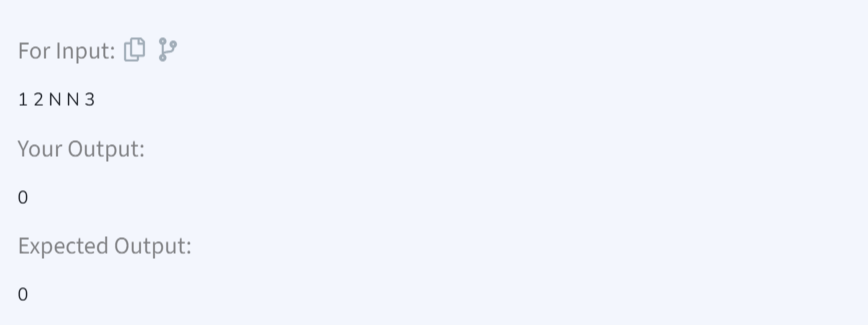
if (rightHeight == -1) return -1;

if (abs(leftHeight - rightHeight) > 1) return -1;

return max(leftHeight, rightHeight) + 1;

}

};

****

TIME COMPLEXITY: O(n)

SPACE COMPLEXITY: O(h)

**6. Triplet Sum Array**

class Solution {

public:

bool find3Numbers(int arr[], int n, int x) {

sort(arr, arr + n);

for (int i = 0; i < n - 2; ++i) {

int left = i + 1, right = n - 1;

while (left < right) {

int current\_sum = arr[i] + arr[left] + arr[right];

if (current\_sum == x) return true;

else if (current\_sum < x) left++;

else right--;

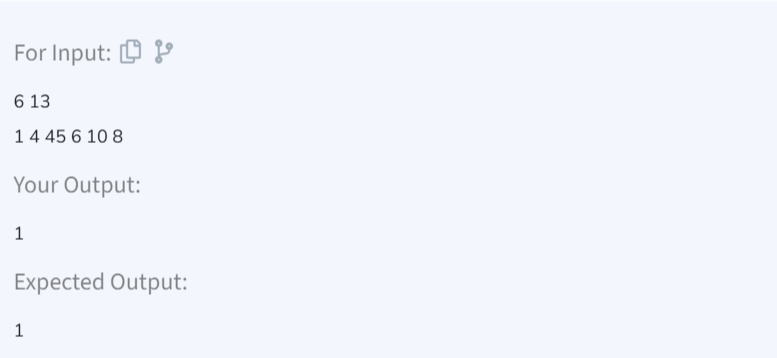
}

}

return false;

}

};

****

TIME COMPLEXITY: O(n^2)  
SPACE COMPLEXITY: O(1)