**Coding practice Problems 14-11-2024**

**1. Stock Buy and Sell**

class Solution {

public:

vector<vector<int>> stockBuySell(vector<int> A, int n) {

vector<vector<int>> result;

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

if (A[i] > A[i - 1]) {

vector<int> pair;

pair.push\_back(i - 1);

pair.push\_back(i);

result.push\_back(pair);

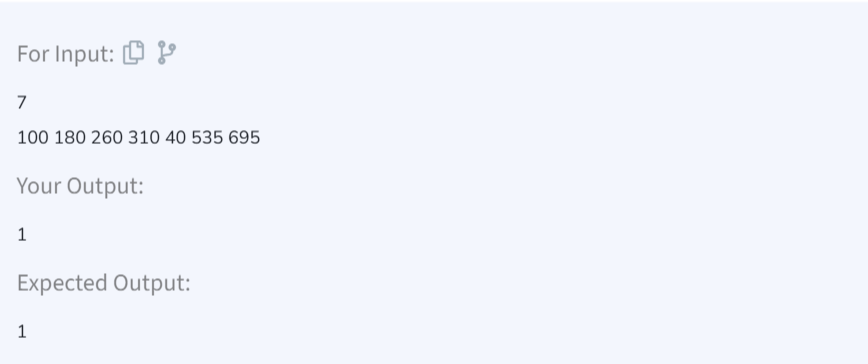
}

}

return result;

}

};



TIME COMPLEXITY: O(n)

SPACE COMPLEXITY: O(k)

**2. Coin Change**

class Solution {

public:

int countWays(int coins[], int n, int sum) {

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

dp[0] = 1;

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

for (int j = coins[i]; j <= sum; ++j) {

dp[j] += dp[j - coins[i]];

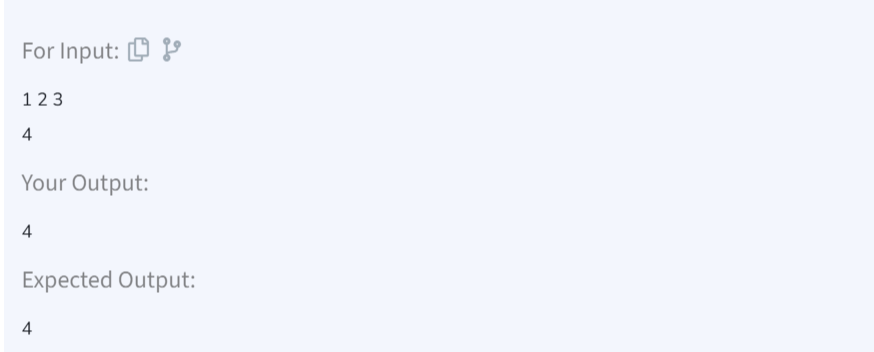
}

}

return dp[sum];

}

};



TIME COMPLEXITY: O(n\*sum)  
SPACE COMPLEXITY: O(sum)

**3. First and Last Occurences**

class Solution {

public:

vector<int> find(vector<int>& arr, int x) {

int n = arr.size();

vector<int> result(2, -1);

int low = 0, high = n - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == x) {

result[0] = mid;

high = mid - 1;

} else if (arr[mid] < x) {

low = mid + 1;

} else {

high = mid - 1;

}

}

low = 0, high = n - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == x) {

result[1] = mid;

low = mid + 1;

} else if (arr[mid] < x) {

low = mid + 1;

} else {

high = mid - 1;

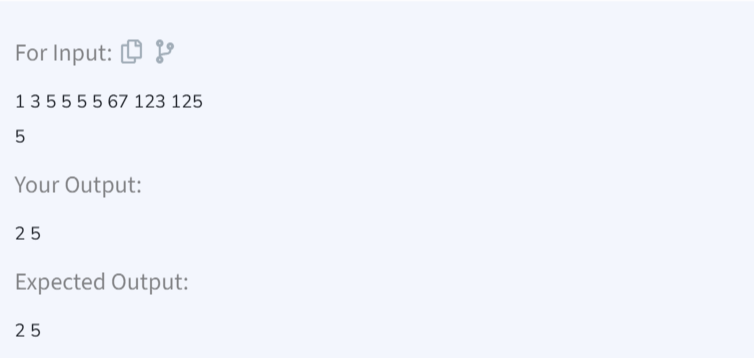
}

}

return result;

}

};



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

**4. Find Transition Point**

class Solution {

public:

int transitionPoint(vector<int>& arr) {

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

int result = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == 1) {

result = mid;

high = mid - 1;

} else {

low = mid + 1;

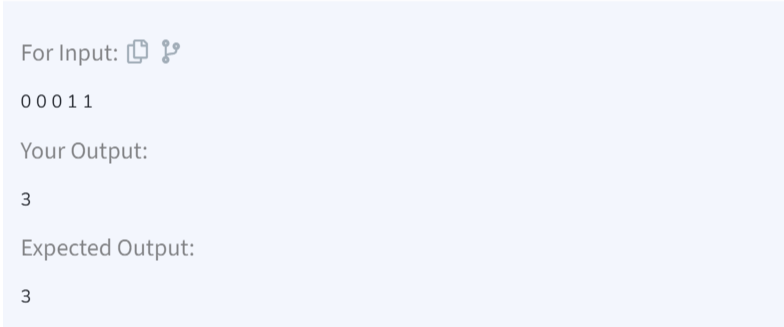
}

}

return result;

}

};



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

**5. First Repeating Element**

class Solution {

public:

int firstRepeated(vector<int>& arr) {

unordered\_map<int, int> seen;

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

if (seen.find(arr[i]) != seen.end()) {

return seen[arr[i]] + 1; }

seen[arr[i]] = i;

}

return -1;

}

};

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

**6. Remove Duplicates from Sorted Array**

class Solution {

public:

int remove\_duplicate(vector<int>& arr) {

if (arr.empty()) return 0;

int uniqueIndex = 1;

for (int i = 1; i < arr.size(); ++i) {

if (arr[i] != arr[i - 1]) {

arr[uniqueIndex] = arr[i];

++uniqueIndex;

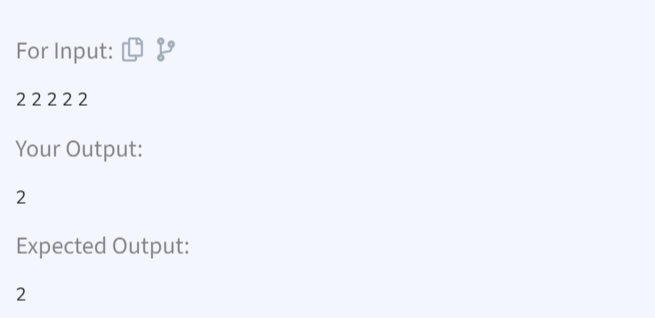
}

}

return uniqueIndex;

}

};



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

**7. Maximum Index**

class Solution {

public:

int maxIndexDiff(vector<int>& arr) {

int n = arr.size();

if (n <= 1) return 0;

vector<int> minArr(n), maxArr(n);

minArr[0] = arr[0];

maxArr[n-1] = arr[n-1];

for (int i = 1; i < n; ++i) minArr[i] = min(minArr[i-1], arr[i]);

for (int i = n-2; i >= 0; --i) maxArr[i] = max(maxArr[i+1], arr[i]);

int i = 0, j = 0, maxDiff = -1;

while (j < n && i < n) {

if (maxArr[j] >= minArr[i]) {

maxDiff = max(maxDiff, j - i);

j++;

} else {

i++;

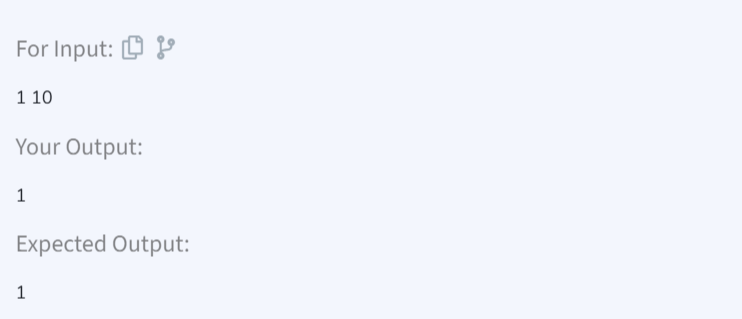
}

}

return maxDiff;

}

};



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

**8. Wave Array**

class Solution {

public:

void convertToWave(vector<int>& arr) {

int n = arr.size();

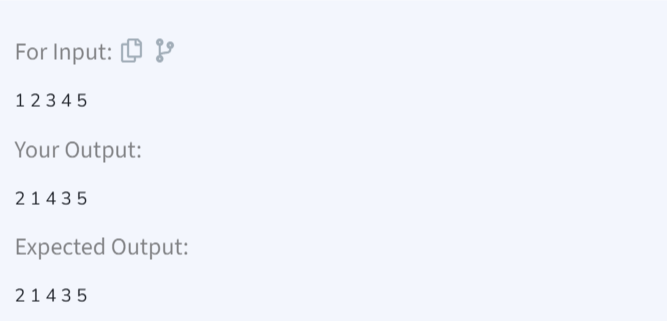
for (int i = 0; i < n - 1; i += 2) {

swap(arr[i], arr[i + 1]);

}

}

};

****

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