Summary of Coding Problems - 2025-07-01

Jump Game II

Brute Force Approach: Dynamic Programming O(n^2)

```
cpp
int jumpGameII(vector<int>& nums) {
 int n = nums.size();
 vector<int> dp(n, INT_MAX);
 dp[0] = 0;
 for(int i = 0; i < n; i++) {
 for(int j = 1; j <= nums[i] && i + j < n; j++) {
 dp[i + j] = min(dp[i + j], dp[i] + 1);
 }
 }
 return dp[n - 1];
}
```

Optimal Approach: Greedy O(n)

```
cpp
int jumpGameII(vector<int>& nums) {
 int jumps = 0, farthest = 0, end = 0;
 for (int i = 0; i < nums.size() - 1; i++) {
 farthest = max(farthest, i + nums[i]);
 if (i == end) {
 jumps++;
 end = farthest;
 }
 }
 return jumps;
}
```

Regular Expression Matching

Brute Force Approach: Recursive (TLE) **Optimal Approach:** Dynamic Programming O(m*n)

```
cpp
bool isMatch(string s, string p) {
 int m = s.length(), n = p.length();
 vector<vector<bool>> dp(m+1, vector<bool>(n+1, false));
 dp[0][0] = true;
 for (int j = 2; j <= n; j++) {
 if (p[j-1] == '*') dp[0][j] = dp[0][j-2];
 }
 for (int i = 1; i <= m; i++) {
 for (int j = 1; j <= n; j++) {
 if (p[j-1] == '*') {
 dp[i][j] = dp[i][j-2] || (dp[i-1][j] && (s[i-1] == p[j-2] || p[j-2] ==
'.'));
 } else {
 dp[i][j] = dp[i-1][j-1] && (s[i-1] == p[j-1] || p[j-1] == '.');
 }
 }
 }
 return dp[m][n];
}
```

N-Queens

Brute Force Approach: Backtracking (Standard) **Optimal Approach:** Bitmask Optimization

```
cpp
int solve(int row, int cols, int diag1, int diag2, int n) {
 if (row == n) return 1;
 int count = 0;
 int available = ((1 << n) - 1) & ~(cols | diag1 | diag2);
 while (available) {
 int pos = available & -available;
 available -= pos;
 count += solve(row + 1, cols | pos, (diag1 | pos) << 1, (diag2 | pos) >> 1, n);
 }
 return count;
}
int totalNQueens(int n) {
 return solve(0, 0, 0, 0, n);
}
```

Trapping Rain Water

Brute Force Approach: O(n^2)

```
cpp
int trap(vector<int>& height) {
 int n = height.size();
 int total = 0;
 for (int i = 0; i < n; i++) {
 int left = 0, right = 0;
 for (int j = 0; j <= i; j++) left = max(left, height[j]);
 for (int j = i; j < n; j++) right = max(right, height[j]);
 total += min(left, right) - height[i];
 }
 return total;
}
```

Optimal Approach: Two-pointer O(n)

First Missing Positive

Brute Force Approach: HashSet O(n) space

```
cpp
int firstMissingPositive(vector<int>& nums) {
 unordered_set<int> s;
 for (int x : nums) if (x > 0) s.insert(x);
 for (int i = 1; i <= nums.size() + 1; i++) {
 if (!s.count(i)) return i;
 }
 return nums.size() + 1;
}
```

Optimal Approach: In-place index sort O(n)

```
cpp
int firstMissingPositive(vector<int>& nums) {
 int n = nums.size();
 for (int i = 0; i < n; i++) {
 while (nums[i] > 0 && nums[i] <= n && nums[nums[i] - 1] != nums[i]) {
 swap(nums[i], nums[nums[i] - 1]);
 }
 }
 for (int i = 0; i < n; i++) {
 if (nums[i] != i + 1) return i + 1;
 }
 return n + 1;
}
```

Two Sum

Brute Force Approach: Double loop O(n^2)

```
cpp
vector<int> twoSum(vector<int>& nums, int target) {
 for (int i = 0; i < nums.size(); i++) {
 for (int j = i + 1; j < nums.size(); j++) {
 if (nums[i] + nums[j] == target)
 return {i, j};
 }
 }
 return {};
}
```

Optimal Approach: HashMap O(n)

```
cpp
vector<int> twoSum(vector<int>& nums, int target) {
 unordered_map<int, int> mp;
 for (int i = 0; i < nums.size(); i++) {
 int complement = target - nums[i];
 if (mp.count(complement)) return {mp[complement], i};
 mp[nums[i]] = i;
 }
 return {};
}
```