51. N-Queens

C++ solution:

```
class Solution {
public:
    vector<vector<string>> solveNQueens(int n) {

        vector<string> current_result(n, string(n, '.'));
        vector<vector<string>> result;

        solve(current_result, n, 0, result);

        return result;
    }

    void solve(vector<string>& current_result, int n, int row, vector<vector<string>>&
 result)
    {
        if (row == n)
        {
            result.insert(result.end(), current_result);
        }

        int col = 0;
        for (; col < n; ++col)
        {
            if (isValid(current_result, n, row, col))
            {
                current_result[row][col] = 'Q';
                solve(current_result, n, row + 1, result);
                current_result[row][col] = '.';
            }
        }
    }

    bool isValid(vector<string>& current_result, int n, int row, int col)
    {
        for (int r = 0; r < n; ++r)
        {
            if (current_result[r][col] == 'Q')
            {
                return false;
            }
        }

        for (int c = 0; c < n; ++c)
        {
```

```
            if (current_result[row][c] == 'Q')
            {
                return false;
            }
        }

        for (int r = 0; r < n; ++r)
        {
            int c_temp1 = r + col - row;
            if (c_temp1 >= 0 && c_temp1 < n && current_result[r][c_temp1] == 'Q')
            {
                return false;
            }

            int c_temp2 = row + col - r;
            if (c_temp2 >= 0 && c_temp2 < n && current_result[r][c_temp2] == 'Q')
            {
                return false;
            }
        }

        return true;
    }

};
```

52. N-Queens II

C++ solution:

```
class Solution {
public:
    int totalNQueens(int n) {

        vector<string> current_result(n, string(n, '.'));
        int result = 0;

        solve(current_result, n, 0, result);

        return result;
    }

    void solve(vector<string>& current_result, int n, int row, int& result)
    {
        if (row == n)
        {
            ++result;
        }
```

```
    int col = 0;
    for (; col < n; ++col)
    {
        if (isValid(current_result, n, row, col))
        {
            current_result[row][col] = 'Q';
            solve(current_result, n, row + 1, result);
            current_result[row][col] = '.';
        }
    }
}

bool isValid(vector<string>& current_result, int n, int row, int col)
{
    for (int r = 0; r < n; ++r)
    {
        if (current_result[r][col] == 'Q')
        {
            return false;
        }
    }

    for (int c = 0; c < n; ++c)
    {
        if (current_result[row][c] == 'Q')
        {
            return false;
        }
    }

    for (int r = 0; r < n; ++r)
    {
        int c_temp1 = r + col - row;
        if (c_temp1 >= 0 && c_temp1 < n && current_result[r][c_temp1] == 'Q')
        {
            return false;
        }

        int c_temp2 = row + col - r;
        if (c_temp2 >= 0 && c_temp2 < n && current_result[r][c_temp2] == 'Q')
        {
            return false;
        }
    }

    return true;
}
```

```
};
```

53. Maximum Subarray

C++ solution:

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        return maxSum(nums, 0, nums.size() - 1);
    }

    int maxSum(vector<int>& nums, int left, int right) {

        if (left == right)
        {
            return nums[left];
        }

        if (left > right)
        {
            return INT_MIN;
        }

        int mid = (left + right) / 2;

        int result_left = maxSum(nums, left, mid - 1);
        int result_right = maxSum(nums, mid + 1, right);

        int result_mid = nums[mid], sum_mid = nums[mid];

        for (int i = mid - 1; i >= left; --i)
        {
            sum_mid += nums[i];
            if (sum_mid > result_mid)
            {
                result_mid = sum_mid;
            }
        }

        sum_mid = result_mid;

        for (int i = mid + 1; i <= right; ++i)
        {
            sum_mid += nums[i];
            if (sum_mid > result_mid)
            {
```

```
                result_mid = sum_mid;
            }
        }

        int result = max(result_left, result_right);
        result = max(result, result_mid);

        return result;
    }
};
```

54. Spiral Matrix

C++ solution:

```
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {

        vector<int> result;

        if (matrix.empty())
        {
            return result;
        }

        if (matrix[0].empty())
        {
            return result;
        }

        int rows = matrix.size();
        int cols = matrix[0].size();

        vector<vector<bool>> visited(rows, vector<bool>(cols, false));

        enum direction { up, down, left, right };

        int r = 0, c = 0;
        result.insert(result.end(), matrix[0][0]);
        visited[0][0] = true;
        direction dir = right;

        while (true)
        {
            switch (dir)
            {
                case up:
```

```
                if (r == 0 || visited[r - 1][c])
                {
                    ++c;
                    dir = right;
                }
                else
                {
                    --r;
                }
                break;

            case down:
                if (r == rows - 1 || visited[r + 1][c])
                {
                    --c;
                    dir = left;
                }
                else
                {
                    ++r;
                }
                break;

            case left:
                if (c == 0 || visited[r][c - 1])
                {
                    --r;
                    dir = up;
                }
                else
                {
                    --c;
                }
                break;

            case right:
                if (c == cols - 1 || visited[r][c + 1])
                {
                    ++r;
                    dir = down;
                }
                else
                {
                    ++c;
                }
                break;
        }
```

```
            if (r < 0 || r > rows - 1 || c < 0 || c > cols - 1)
            {
                break;
            }

            if (visited[r][c])
            {
                break;
            }

            result.insert(result.end(), matrix[r][c]);
            visited[r][c] = true;
        }

        return result;
    }
};
```

55. Jump Game

C++ solution:

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int n = nums.size();

        int first_index = n - 1;

        for (int position = n - 2; position >= 0; --position)
        {
            if (position + nums[position] >= first_index)
            {
                first_index = position;
            }
        }

        if (first_index == 0)
        {
            return true;
        }

        return false;
    }
};
```

56. Merge Intervals

C++ solution:

```
class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
        vector<vector<int>> sorted_intervals;

        for (vector<int> interval : intervals)
        {
            insert(sorted_intervals, interval);
        }

        return sorted_intervals;
    }

    void insert(vector<vector<int>>& intervals, vector<int>& newInterval) {

        int n = intervals.size();
        int current = 0;

        while (current < n && intervals[current][1] < newInterval[0])
        {
            ++current;
        }

        if (current == n)
        {
            intervals.insert(intervals.end(), newInterval);
            return;
        }

        if (intervals[current][0] <= newInterval[1])
        {
            newInterval[0] = min(newInterval[0], intervals[current][0]);
            newInterval[1] = max(newInterval[1], intervals[current][1]);
            intervals.erase(intervals.begin() + current);

            while (current < intervals.size() && intervals[current][0] <= newInterval[
1])
            {
                newInterval[1] = max(newInterval[1], intervals[current][1]);
                intervals.erase(intervals.begin() + current);
            }
        }

        intervals.insert(intervals.begin() + current, newInterval);
    }
};
```

57. Insert Interval

C++ solution:

```
class Solution {
public:
    vector<vector<int>> insert(vector<vector<int>>& intervals, vector<int>& newInterva
l) {

        vector<vector<int>> result;
        int n = intervals.size();
        int current = 0;

        while (current < n && intervals[current][1] < newInterval[0])
        {
            result.insert(result.end(), intervals[current]);
            ++current;
        }

        if (current == n)
        {
            result.insert(result.end(), newInterval);
            return result;
        }

        if (intervals[current][0] <= newInterval[1])
        {
            newInterval[0] = min(newInterval[0], intervals[current][0]);
            newInterval[1] = max(newInterval[1], intervals[current][1]);
            ++current;

            while (current < n && intervals[current][0] <= newInterval[1])
            {
                newInterval[1] = max(newInterval[1], intervals[current][1]);
                ++current;
            }
        }

        result.insert(result.end(), newInterval);

        while (current < n)
        {
            result.insert(result.end(), intervals[current]);
            ++current;
        }

        return result;
    }
};
```

58. Length of Last Word

C++ solution:

```
class Solution {
public:
    int lengthOfLastWord(string s) {
        int count = 0, current = s.length() - 1;

        while (current >= 0 && s[current] == ' ')
        {
            --current;
        }

        if (current == -1)
        {
            return 0;
        }

        while (current >= 0 && s[current] != ' ')
        {
            ++count;
            --current;
        }

        return count;
    }
};
```

59. Spiral Matrix II

C++ solution:

```
class Solution {
public:
    vector<vector<int>> generateMatrix(int n) {

        vector<vector<int>> result(n, vector<int>(n));

        vector<vector<bool>> visited(n, vector<bool>(n, false));

        enum direction { up, down, left, right };

        int r = 0, c = 0;
        result[0][0] = 1;
        visited[0][0] = true;
        direction dir = right;
        int num = 2, max_num = n * n;

        while (num <= max_num)
```

```
        {
            switch (dir)
            {
                case up:
                    if (r == 0 || visited[r - 1][c])
                    {
                        ++c;
                        dir = right;
                    }
                    else
                    {
                        --r;
                    }
                    break;

                case down:
                    if (r == n - 1 || visited[r + 1][c])
                    {
                        --c;
                        dir = left;
                    }
                    else
                    {
                        ++r;
                    }
                    break;

                case left:
                    if (c == 0 || visited[r][c - 1])
                    {
                        --r;
                        dir = up;
                    }
                    else
                    {
                        --c;
                    }
                    break;

                case right:
                    if (c == n - 1 || visited[r][c + 1])
                    {
                        ++r;
                        dir = down;
                    }
                    else
                    {
                        ++c;
```

```
                    }
                    break;
                }

                result[r][c] = num;
                visited[r][c] = true;

                ++num;
            }

            return result;
        }
};
```

60. Permutation Sequence

C++ solution:

```
class Solution {
public:
    string getPermutation(int n, int k) {

        int size = 1;
        vector<char> chars;
        for (int i = 1; i <= n; ++i)
        {
            size *= i;
            chars.insert(chars.end(), '0' + i);
        }

        int current_parts = n;
        k -= 1;

        string result;

        while (size > 1)
        {
            size /= current_parts;
            result.insert(result.end(), chars[k / size]);
            chars.erase(chars.begin() + k / size);
            --current_parts;
            k %= size;
        }

        result.insert(result.end(), chars[0]);

        return result;
    }
};
```