[31. Next Permutation]

C++ solution:

```
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        int n = nums.size();
        if (n < 2)
        {
            return;
        }

        int i = n - 2;
        for (; i >=0; --i)
        {
            if (nums[i] < nums[i + 1])
            {
                break;
            }
        }

        if (i >= 0)
        {
            int left = i + 1, right = n - 1, mid = left;
            while (left < right)
            {
                mid = (left + right) / 2;
                if (nums[mid] <= nums[i])
                {
                    right = mid - 1;
                }
                else
                {
                    left = mid + 1;
                }
            }

            while (nums[mid] <= nums[i])
            {
                --mid;
            }

            while (mid < n && nums[mid] > nums[i])
            {
                ++mid;
            }
```

```
            --mid;

            nums[i] = nums[i] ^ nums[mid];
            nums[mid] = nums[i] ^ nums[mid];
            nums[i] = nums[i] ^ nums[mid];
        }

        for (int j = i + 1; j <= (i + n) / 2; ++j)
        {
            if (nums[j] != nums[n + i - j])
            {
                nums[j] = nums[j] ^ nums[n + i - j];
                nums[n + i - j] = nums[j] ^ nums[n + i - j];
                nums[j] = nums[j] ^ nums[n + i - j];
            }
        }
    }
};
```

[32. Longest Valid Parentheses]

C++ solution:

```
class Solution {
public:
    int longestValidParentheses(string s) {

        int length = s.length();
        stack<char> characters;
        stack<int> indexes;
        int max_count = 0, count = 0;
        vector<pair<int, int>> left_right;

        for (int i = 0; i < length; ++i)
        {
            if (s[i] == '(')
            {
                characters.push('(');
                indexes.push(i);
            }
            else if ((!characters.empty()) && characters.top() == '(')
            {
                while ((!left_right.empty()) && left_right.back().first > indexes.top(
))
                {
                    left_right.pop_back();
                }
```

```
                left_right.insert(left_right.end(), pair<int, int>(indexes.top(), i));

                characters.pop();
                indexes.pop();
            }
        }

        if (left_right.size() == 0)
        {
            return 0;
        }

        if (left_right.size() == 1)
        {
            return left_right[0].second - left_right[0].first + 1;
        }

        for (int i = 0; i < left_right.size(); ++i)
        {
            count = left_right[i].second - left_right[i].first + 1;
            while (i < left_right.size() - 1 && left_right[i].second + 1 == left_right
[i + 1].first)
            {
                count += left_right[i + 1].second - left_right[i].second;
                left_right[i].second = left_right[i + 1].second;
                left_right.erase(left_right.begin() + i + 1);

            }

            if (count > max_count)
            {
                max_count = count;
            }
        }

        return max_count;
    }
};
```

33. Search in Rotated Sorted Array

C++ solution:

```
class Solution {
public:
    int binary_search(vector<int>& nums, int begin, int end, int target) {
        while (begin <= end)
        {
```

```
            int mid = (begin + end) / 2;
            if (nums[mid] == target)
            {
                return mid;
            }

            if (nums[mid] > target)
            {
                end = mid - 1;
            }
            else
            {
                begin = mid + 1;
            }
        }

        return -1;
    }

    int search(vector<int>& nums, int target) {

        int n = nums.size();
        if (n == 0)
        {
            return -1;
        }

        if (n == 1)
        {
            if (nums[0] == target)
            {
                return 0;
            }
            return -1;
        }

        int left = 0, right = n - 1, mid = 0;

        while (left <= right)
        {
            if (nums[left] < nums[right])
            {
                return binary_search(nums, left, right, target);
            }

            mid = (left + right) / 2;

            if (nums[mid] == target)
```

```
            {
                return mid;
            }

            if (nums[mid] < nums[left])
            {
                if (target > nums[mid] && target < nums[left])
                {
                    return binary_search(nums, mid + 1, right, target);
                }
                else
                {
                    right = mid - 1;
                }
            }
            else
            {
                if (target > nums[right] && target < nums[mid])
                {
                    return binary_search(nums, left, mid - 1, target);
                }
                else
                {
                    left = mid + 1;
                }
            }
        }

        return -1;
    }
};
```

[34. Find First and Last Position of Element in Sorted Array]

C++ solution:

```
class Solution {
public:
    int binary_search(vector<int>& nums, int begin, int end, int target) {
        while (begin <= end)
        {
            int mid = (begin + end) / 2;
            if (nums[mid] == target)
            {
                return mid;
            }

            if (nums[mid] > target)
```

```
            {
                end = mid - 1;
            }
            else
            {
                begin = mid + 1;
            }
        }

        return -1;
    }

    vector<int> searchRange(vector<int>& nums, int target) {
        int n = nums.size();
        vector<int> result(2, -1);

        if (n == 0)
        {
            return result;
        }

        int index = binary_search(nums, 0, n - 1, target);
        if (index == -1)
        {
            return result;
        }

        int begin = -1, end = -1;

        while (true)
        {
            int temp = binary_search(nums, 0, index - 1, target);
            if (temp == -1)
            {
                begin = index;
                break;
            }
            else
            {
                index = temp;
            }
        }

        while (true)
        {
            int temp = binary_search(nums, index + 1, n - 1, target);
            if (temp == -1)
            {
```

```
                end = index;
                break;
            }
            else
            {
                index = temp;
            }
        }

        result[0] = begin;
        result[1] = end;

        return result;
    }
};
```

35. Search Insert Position

C++ solution:

```
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        int left = 0, right = nums.size() - 1;

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

        int mid = 0;

        while (left <= right)
        {
            mid = (left + right) / 2;
            if (nums[mid] == target)
            {
                return mid;
            }

            if (nums[mid] > target)
            {
                right = mid - 1;
            }
            else
            {
                if (left == right)
                {
                    ++mid;
                }
                left = mid + 1;
            }
        }

        return mid;
    }
};
```

[36. Valid Sudoku]

C++ solution:

```
class Solution {
public:
    bool isValidSudoku(vector<vector<char>>& board) {

        int r = 0, c = 0, temp = 0;
```

```
        for (r = 0; r < 9; ++r)
        {
            vector<int> nums(9, 0);
            for (c = 0; c < 9; ++c)
            {
                if (board[r][c] > '0' && board[r][c] <= '9')
                {
                    temp = board[r][c] - '1';
                    if (nums[temp] == 1)
                    {
                        return false;
                    }
                    nums[temp] = 1;
                }

            }
        }

        for (c = 0; c < 9; ++c)
        {
            vector<int> nums(9, 0);
            for (r = 0; r < 9; ++r)
            {
                if (board[r][c] > '0' && board[r][c] <= '9')
                {
                    temp = board[r][c] - '1';
                    if (nums[temp] == 1)
                    {
                        return false;
                    }
                    nums[temp] = 1;
                }

            }
        }

        int start_r = 0, start_c = 0;
        for (int i = 0; i < 9; ++i)
        {
            start_r = i / 3 * 3;
            start_c = i % 3 * 3;
            vector<int> nums(9, 0);
            for (r = start_r; r < start_r + 3; ++r)
            {
                for (c = start_c; c < start_c + 3; ++c)
                {
                    if (board[r][c] > '0' && board[r][c] <= '9')
                    {
```

```
                        temp = board[r][c] - '1';
                        if (nums[temp] == 1)
                        {
                            return false;
                        }
                        nums[temp] = 1;
                    }
                }
            }
        }

        return true;
    }
};
```

37. Sudoku Solver

C++ solution:

```
class Solution {
public:
    void solveSudoku(vector<vector<char>>& board) {
        solve(board, 0, 0);
    }

    bool solve(vector<vector<char>>& board, int row, int col)
    {
        if (row == 9)
        {
            return true;
        }

        if (board[row][col] != '.')
        {
            if (col == 8)
            {
                return solve(board, row + 1, 0);
            }
            else
            {
                return solve(board, row, col + 1);
            }
        }
        else
        {
            for (char ch = '1'; ch <= '9'; ++ch)
            {
                if (isValid(board, row, col, ch))
```

```
            {
                board[row][col] = ch;
                if (solve(board, row, col))
                {
                    return true;
                }
                board[row][col] = '.';
            }
        }
    }

    return false;
}

bool isValid(vector<vector<char>>& board, int row, int col, char ch)
{

    int r = 0, c = 0;

    for (r = 0; r < 9; ++r)
    {
        if (board[r][col] == ch)
        {
            return false;
        }
    }

    for (c = 0; c < 9; ++c)
    {
        if (board[row][c] == ch)
        {
            return false;
        }
    }

    int start_r = row / 3 * 3;
    int start_c = col / 3 * 3;

    for (r = start_r; r < start_r + 3; ++r)
    {
        for (c = start_c; c < start_c + 3; ++c)
        {
            if (board[r][c] == ch)
            {
                return false;
            }
        }
    }
```

```
        return true;
    }
};
```

38. Count and Say

C++ solution:

```
class Solution {
public:
    string countAndSay(int n) {

        if (n == 1)
        {
            return "1";
        }

        string s = countAndSay(n - 1);
        int count = 0;
        char before = s[0];
        stringstream ss;

        for (char c : s)
        {
            if (count == 0 || c == before)
            {
                ++count;
            }
            else
            {
                ss << count << before;
                count = 1;
                before = c;
            }
        }
        ss << count << before;

        return ss.str();
    }
};
```

39. Combination Sum

C++ solution:

```
class Solution {
public:
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {

        int n = candidates.size();

        sort(candidates.begin(), candidates.end());

        return subCombinationSum(candidates, 0, target);
    }

    vector<vector<int>> subCombinationSum(vector<int>& sorted_candidates, int index, i
nt target) {

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

        for (int start_index = index; start_index < n; ++start_index)
        {

            if (sorted_candidates[start_index] > target)
            {
                break;
            }

            if (sorted_candidates[start_index] == target)
            {
                current = vector<int>(1, sorted_candidates[start_index]);
                result.insert(result.end(), current);
                break;
            }

            vector<vector<int>> behind = subCombinationSum(sorted_candidates, start_in
dex, target - sorted_candidates[start_index]);

            for (auto element : behind)
            {
                current = vector<int>(1, sorted_candidates[start_index]);
                current.insert(current.end(), element.begin(), element.end());
                result.insert(result.end(), current);
            }
        }

        return result;
    }
};
```

40. Combination Sum II

C++ solution:

```
class Solution {
public:
    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {

        int n = candidates.size();

        sort(candidates.begin(), candidates.end());

        return subCombinationSum(candidates, 0, target);
    }

    vector<vector<int>> subCombinationSum(vector<int>& sorted_candidates, int index, i
nt target) {

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

        for (int start_index = index; start_index < n; ++start_index)
        {

            if (start_index > index && sorted_candidates[start_index] == sorted_candid
ates[start_index - 1])
            {
                continue;
            }

            if (sorted_candidates[start_index] > target)
            {
                break;
            }

            if (sorted_candidates[start_index] == target)
            {
                current = vector<int>(1, sorted_candidates[start_index]);
                result.insert(result.end(), current);
                break;
            }

            vector<vector<int>> behind = subCombinationSum(sorted_candidates, start_in
dex + 1, target - sorted_candidates[start_index]);

            for (auto element : behind)
            {
```

```
                current = vector<int>(1, sorted_candidates[start_index]);
                current.insert(current.end(), element.begin(), element.end());
                result.insert(result.end(), current);
            }
        }

        return result;
    }
};
```