Day 135/180 Stack More Problems

1: Asteroid Collision

```
vector<int> asteroidCollision(int n, vector<int> &a) {
    vector<int> ans;  // Vector to store the final result
    stack<int> s1;     // Stack to simulate asteroid collisions

    // Loop through each asteroid
    for(int i = 0; i < n; i++) {
        // If the asteroid is moving to the right (positive value), push it onto
the stack
        if(a[i] > 0) {
            s1.push(a[i]);
            continue; // Move to the next asteroid
        }

        // If the asteroid is moving to the left (negative value)
        if(!s1.empty() && s1.top() > 0) {
            // Check for collisions
            while(!s1.empty() && s1.top() < abs(a[i]) && s1.top() > 0) {
                // If the top asteroid in stack is smaller in size, it gets
destroyed
                s1.pop();
            }

            // If there's a collision and both asteroids are of equal size, destroy
both
            if(!s1.empty() && s1.top() == abs(a[i])) {
                s1.pop();
            } else {
                // If the top asteroid in stack is moving to the left, push the
current asteroid
                // Otherwise, continue moving to the next asteroid
                if(!s1.empty()) {
                    if(s1.top() < 0)
                        s1.push(a[i]);
                } else
```

```
                    s1.push(a[i]);
                }
            } else {
                // If the stack is empty or the top asteroid is also moving left, push
the current asteroid
                s1.push(a[i]);
            }
        }

    // After processing all asteroids, transfer remaining asteroids from stack to
the result vector
    while(!s1.empty()) {
        ans.push_back(s1.top());
        s1.pop();
    }

    // Reverse the result vector to get the correct order of asteroids
    reverse(ans.begin(), ans.end());

    // Return the final result
    return ans;
}

TC :- O(n), SC :- O(n)
```

2: Baseball Game

```
class Solution {
public:
    int calPoints(vector<string>& ops) {
        stack<int> ans; // Stack to keep track of valid points

        // Iterate through each operation
        for(int i = 0; i < ops.size(); i++) {
            // If the current operation is "+"
            if(ops[i] == "+") {
                stack<int> s1; // Temporary stack to store values for calculating
sum
                int sum = 0; // Initialize sum variable
                int t = 2; // Number of elements to consider for the sum (last two
valid points)

                // Pop the top two elements from the main stack and calculate
their sum
                while(t-- > 0) {
                    int x = ans.top();
                    ans.pop();
                    sum += x;
                    s1.push(x); // Push the popped element onto the temporary
stack
                }

                // Restore the popped elements back to the main stack
                while(s1.size() != 0) {
                    int x = s1.top();
                    s1.pop();
                    ans.push(x);
                }

                // Push the sum of the last two valid points onto the main stack
                ans.push(sum);
            }
            // If the current operation is "C" (cancel the last valid point)
            else if(ops[i] == "C") {
                ans.pop(); // Pop the last valid point from the stack
            }
            // If the current operation is "D" (double the last valid point)
            else if(ops[i] == "D") {
                int x = ans.top(); // Get the last valid point
                ans.push(2 * x); // Double the last valid point and push onto the 
stack
            }
            // If the current operation is a number (valid point)
            else {
                ans.push(stoi(ops[i])); // Push the numeric value onto the stack
            }
        }

        int sum = 0; // Initialize sum variable to calculate total score

        // Calculate the total score by adding up all the valid points left in the 
stack
        while(ans.size() != 0) {
            int x = ans.top();
            ans.pop();
            sum += x;
        }

        return sum; // Return the total score
    }
};
```

3: Remove K Digits

```
string removeKdigits(string s, int k) {
    // If the length of the string is less than or equal to k, return "0" since
all digits will be removed
    if(s.size() <= k)
        return "0";

    string ans = ""; // Initialize an empty string to store the result
    bool flag = false; // Flag to track if leading zeros are encountered

    // Iterate through each character in the input string
    for(auto x : s) {
        // While the result string is not empty, the current character is
smaller than the last character in the result,
        // and there are still removals left (k > 0), remove the last character
from the result string
        while(ans.size() > 0 && ans.back() > x && k > 0) {
            ans.pop_back();
            k--;
        }

        // If the result string is empty and the current character is '0', set
the flag to true
        if(ans == "") {
            if(x != '0')
                ans += x; // Append non-zero characters to the result string
            else
                flag = true; // Set the flag to true indicating leading zeros
        } else {
            ans += x; // Append the current character to the result string
        }
    }

    // After processing all characters, if there are still removals left (k >
0), remove characters from the end of the result string
    while(ans != "" && k-- > 0)
        ans.pop_back();
    // If leading zeros were encountered and the result string is empty, return
"0"
    if(flag && ans == "")
        return "0";

    return ans; // Return the resulting string
}
```

4: Next Greater

```
class Solution {
public:
    vector<int> nextGreaterElement(vector<int>& nums1, vector<int>& nums2) {
        int n = nums1.size(); // Size of nums1 array
        map<int, int> m; // Map to store each element of nums1 along with its
index
        // Populate the map with elements of nums1 and their corresponding
indices
        for (int i = 0; i < n; i++)
            m[nums1[i]] = i;

        vector<int> ans(n, -1); // Initialize result vector with -1

        stack<int> s1; // Stack to track next greater elements

        // Traverse nums2 array from right to left
        for (int i = nums2.size() - 1; i >= 0; i--) {
            // Pop elements from stack if they are smaller than or equal to the
current element
            while (!s1.empty() && s1.top() <= nums2[i])
                s1.pop();

            // If the current element of nums2 is present in nums1 and there is
a greater element in stack
            if (m.count(nums2[i]) && !s1.empty()) {
                // Update the next greater element for the corresponding element
in nums1
                ans[m[nums2[i]]] = s1.top();
            }

            // Push the current element of nums2 onto the stack
            s1.push(nums2[i]);
        }

        return ans; // Return the result vector
    }
};
```

5: Next Greater Element 2

```
vector<int> nextGreaterElement(int n, vector<int>& arr) {
    // Initialize a vector to store the next greater element for each element
in arr
    vector<int> ans(n, -1);

    // Stack to track elements whose next greater element is not yet found
    stack<int> st;

    // Traverse arr from right to left
    for(int i = n - 1; i >= 0; i--) {
        // Pop elements from the stack that are smaller than or equal to the
current element
        while(!st.empty() && st.top() <= arr[i]) {
            st.pop();
        }
        // Push the current element onto the stack
        st.push(arr[i]);
    }

    // Traverse arr from right to left again to find the next greater element
for each element
    for(int i = n - 1; i >= 0; i--) {
        // Pop elements from the stack that are smaller than or equal to the
current element
        while(!st.empty() && st.top() <= arr[i]) {
            st.pop();
        }
        // If there is a next greater element in the stack, update the
corresponding element in ans
        if(!st.empty()) {
            ans[i] = st.top();
        }
        // Push the current element onto the stack
        st.push(arr[i]);
    }

    // Return the vector containing the next greater elements for each element
in arr
    return ans;
}
```