Two Pointers

[26. Remove Duplicates from Sorted Array]

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
     int i = 0;
    for (int j = 1; j < nums.size(); j++) {
        if (nums[i] != nums[j]) {
            i++;
            nums[i] = nums[j];
        }
    }
    return i + 1;
    }
};
```

Java

```
class Solution {
    public int removeDuplicates(int[] nums) {
       int i = 0;
    for (int j = 1; j < nums.length; j++) {
        if (nums[i] != nums[j]) {
            i++;
            nums[i] = nums[j];
        }
    }
    return i + 1;
    }
}
```

[27. Remove Element]

```
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
     int i = 0;
    for (int j = 0; j < nums.size(); j++) {
        if (nums[j] != val) {
            nums[i] = nums[j];
            i++;
        }
    }
    return i;
    }
};
```

```
class Solution {
    public int removeElement(int[] nums, int val) {
        int i = 0;
    for (int j = 0; j < nums.length; j++) {
        if (nums[j] != val) {
            nums[i] = nums[j];
            i++;
        }
    }
    return i;
    }
}
```

[283. Move Zeroes]

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
     int i = 0;
    for (int j = 0; j < nums.size(); j++) {
        if (nums[j] != 0) {
            swap(nums[i], nums[j]);
            i++;
        }
    }
    }
};
```

Java

```
class Solution {
    public void moveZeroes(int[] nums) {
       int i = 0;
    for (int j = 0; j < nums.length; j++) {
        if (nums[j] != 0) {
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
            i++;
        }
    }
    }
}
```

[344. Reverse String]

```
class Solution {
public:
    void reverseString(vector<char>& s) {
    int left = 0, right = s.size() - 1;
    while (left < right) {
        swap(s[left], s[right]);
        left++;
        right--;
    }
    }
};
```

Java

```
class Solution {
    public void reverseString(char[] s) {
    int left = 0, right = s.length - 1;
    while (left < right) {
        char temp = s[left];
        s[left] = s[right];
        s[right] = temp;
        left++;
        right--;
    }
    }
}
```

345. Reverse Vowels of a String

```
class Solution {
public:
bool isVowel(char c) {
    c = tolower(c);
    return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';
}
string reverseVowels(string s) {
      int left = 0, right = s.size() - 1;
    while (left < right) {
        if (!isVowel(s[left])) {
            left++;
        } else if (!isVowel(s[right])) {
            right--;
        } else {
            swap(s[left], s[right]);
            left++;
            right--;
        }
    }
    return s;
    }
};
```

```
class Solution {
    private boolean isVowel(char c) {
    c = Character.toLowerCase(c);
    return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';
}

    public String reverseVowels(String s) {
        char[] chars = s.toCharArray();
    int left = 0, right = chars.length - 1;
    while (left < right) {
        if (!isVowel(chars[left])) {
            left++;
        } else if (!isVowel(chars[right])) {
            right--;
        } else {
            char temp = chars[left];
            chars[left] = chars[right];
            chars[right] = temp;
            left++;
            right--;
        }
    }
    return new String(chars);
    }
}
```

125. Valid Palindrome

```
class Solution {
public:
    bool isPalindrome(string s) {
    int left = 0, right = s.size() - 1;
    while (left < right) {
        while (left < right && !isalnum(s[left])) left++;
        while (left < right && !isalnum(s[right])) right--;
        if (tolower(s[left]) != tolower(s[right])) return false;
        left++;
        right--;
    }
    return true;
    }
};
```

```
Java
class Solution {
   public boolean isPalindrome(String s) {
   int left = 0, right = s.length() - 1;
   while (left < right) {
      while (left < right && !Character.isLetterOrDigit(s.charAt(left))) left++;
      while (left < right && !Character.isLetterOrDigit(s.charAt(right))) right--;
      if (Character.toLowerCase(s.charAt(left)) != Character.toLowerCase(s.charAt(right))) {
         return false;
      }
      left++;
      right--;
   }
   return true;
}
}
```

28. Find the Index of the First Occurrence in a String

```
class Solution {
public:
   int strStr(string haystack, string needle) {
    int m = haystack.size(), n = needle.size();
   for (int i = 0; i <= m - n; i++) {
      if (haystack.substr(i, n) == needle) {
         return i;
      }
   }
   return -1;
   }
};
```

```
Java
class Solution {
   public int strStr(String haystack, String needle) {
   int m = haystack.length(), n = needle.length();
   for (int i = 0; i <= m - n; i++) {
      if (haystack.substring(i, i + n).equals(needle)) {
         return i;
      }
   }
   return -1;
   }
}
```

917. Reverse Only Letters

```
class Solution {
public:
    string reverseOnlyLetters(string s) {
    int left = 0, right = s.size() - 1;
    while (left < right) {
        if (!isalpha(s[left])) {
            left++;
        } else if (!isalpha(s[right])) {
            right--;
        } else {
            swap(s[left], s[right]);
            left++;
            right--;
        }
    }
    return s;
    }
};
```

```
class Solution {
    public String reverseOnlyLetters(String s) {
    char[] chars = s.toCharArray();
    int left = 0, right = chars.length - 1;
    while (left < right) {
        if (!Character.isLetter(chars[left])) {
            left++;
        } else if (!Character.isLetter(chars[right])) {
            right--;
        } else {
            char temp = chars[left];
            chars[left] = chars[right];
            chars[right] = temp;
            left++;
            right--;
        }
    }
    return new String(chars);
    }
}
```

922. Sort Array By Parity II

```
class Solution {
public:
    vector<int> sortArrayByParityII(vector<int>& nums) {
    int i = 0, j = 1; // i for even indices, j for odd indices
    int n = nums.size();
    while (i < n && j < n) {
        if (nums[i] % 2 == 0) {
            i += 2;
        } else if (nums[j] % 2 == 1) {
            j += 2;
        } else {
            swap(nums[i], nums[j]);
            i += 2;
            j += 2;
        }
    }
    return nums;
    }
};
```

```
class Solution {
    public int[] sortArrayByParityII(int[] nums) {
    int i = 0, j = 1; // i for even indices, j for odd indices
    int n = nums.length;
    while (i < n && j < n) {
        if (nums[i] % 2 == 0) {
            i += 2;
        } else if (nums[j] % 2 == 1) {
            j += 2;
        } else {
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
            i += 2;
            j += 2;
        }
    }
    return nums;
    }
}
```

541. Reverse String II

```
class Solution {
public:
    string reverseStr(string s, int k) {
    for (int i = 0; i < s.size(); i += 2 * k) {
        int left = i, right = min(i + k - 1, (int)s.size() - 1);
        while (left < right) {
            swap(s[left], s[right]);
            left++;
            right--;
        }
    }
    return s;
    }
};
```

Java

```
class Solution {
    public String reverseStr(String s, int k) {
      char[] chars = s.toCharArray();
    for (int i = 0; i < chars.length; i += 2 * k) {
        int left = i, right = Math.min(i + k - 1, chars.length - 1);
        while (left < right) {
            char temp = chars[left];
            chars[left] = chars[right];
            chars[right] = temp;
            left++;
            right--;
        }
    }
    return new String(chars);
    }
}
```

557. Reverse Words in a String III

```
class Solution {
public:
    string reverseWords(string s) {
   int start = 0;
    for (int end = 0; end <= s.size(); end++) {
        if (end == s.size() || s[end] == ' ') {
            reverse(s.begin() + start, s.begin() + end);
            start = end + 1;
        }
    }
    return s;

    }
};
```

```
class Solution {
    public String reverseWords(String s) {
    char[] chars = s.toCharArray();
    int start = 0;
    for (int end = 0; end <= chars.length; end++) {
        if (end == chars.length || chars[end] == ' ') {
            reverse(chars, start, end - 1);
            start = end + 1;
        }
    }
    return new String(chars);
    }
    private void reverse(char[] chars, int left, int right) {
    while (left < right) {
        char temp = chars[left];
        chars[left] = chars[right];
        chars[right] = temp;
        left++;
        right--;
    }
}
}
```

696. Count Binary Substrings

```
class Solution {
public:
    int countBinarySubstrings(string s) {
    int prev = 0, curr = 1, count = 0;
    for (int i = 1; i < s.size(); i++) {
        if (s[i] == s[i - 1]) {
            curr++;
        } else {
            count += min(prev, curr);
            prev = curr;
            curr = 1;
        }
    }
    return count + min(prev, curr);
    }
};
```

```
class Solution {
    public int countBinarySubstrings(String s) {
   int prev = 0, curr = 1, count = 0;
    for (int i = 1; i < s.length(); i++) {
        if (s.charAt(i) == s.charAt(i - 1)) {
            curr++;
        } else {
            count += Math.min(prev, curr);
            prev = curr;
            curr = 1;
        }
    }
    return count + Math.min(prev, curr);
    }
}
```

1089. Duplicate Zeros

```
class Solution {
public:
    void duplicateZeros(vector<int>& arr) {
      int n = arr.size(), countZeros = 0;
    for (int i = 0; i < n; i++) {
        if (arr[i] == 0) countZeros++;
    }
    int i = n - 1, j = n + countZeros - 1;
    while (i >= 0) {
        if (j < n) arr[j] = arr[i];
        if (arr[i] == 0) {
            j--;
            if (j < n) arr[j] = 0;
        }
        i--;
        j--;
    }
    }
};
```

```
class Solution {
    public void duplicateZeros(int[] arr) {
    int n = arr.length, countZeros = 0;
    for (int num : arr) {
        if (num == 0) countZeros++;
    }
    int i = n - 1, j = n + countZeros - 1;
    while (i >= 0) {
        if (j < n) arr[j] = arr[i];
        if (arr[i] == 0) {
            j--;
            if (j < n) arr[j] = 0;
        }
        i--;
        j--;
    }
    }
}
```

1332. Remove Palindromic Subsequences

```
class Solution {
public:
    int removePalindromeSub(string s) {
    int left = 0, right = s.size() - 1;
    while (left < right) {
        if (s[left] != s[right]) return 2;
        left++;
        right--;
    }
    return 1;
    }
};
```

```
class Solution {
    public int removePalindromeSub(String s) {
    int left = 0, right = s.length() - 1;
    while (left < right) {
        if (s.charAt(left) != s.charAt(right)) return 2;
        left++;
        right--;
    }
    return 1;
    }
}
```

1768. Merge Strings Alternately

```
class Solution {
public:
    string mergeAlternately(string word1, string word2) {
      string result;
    int i = 0, j = 0;
    while (i < word1.size() || j < word2.size()) {
        if (i < word1.size()) result += word1[i++];
        if (j < word2.size()) result += word2[j++];
    }
    return result;
    }
};
```

```
class Solution {
    public String mergeAlternately(String word1, String word2) {
    StringBuilder result = new StringBuilder();
    int i = 0, j = 0;
    while (i < word1.length() || j < word2.length()) {
        if (i < word1.length()) result.append(word1.charAt(i++));
        if (j < word2.length()) result.append(word2.charAt(j++));
    }
    return result.toString();
    }
}
```

2200. Find All K-Distant Indices in an Array

```
class Solution {
public:
    vector<int> findKDistantIndices(vector<int>& nums, int key, int k) {
   vector<int> result;
    for (int i = 0; i < nums.size(); i++) {
        for (int j = 0; j < nums.size(); j++) {
            if (nums[j] == key && abs(i - j) <= k) {
                result.push_back(i);
                break;
            }
        }
    }
    return result;
    }
};
```

```
class Solution {
    public List<Integer> findKDistantIndices(int[] nums, int key, int k) {
    List<Integer> result = new ArrayList<>();
    for (int i = 0; i < nums.length; i++) {
        for (int j = 0; j < nums.length; j++) {
            if (nums[j] == key && Math.abs(i - j) <= k) {
                result.add(i);
                break;
            }
        }
    }
    return result;
    }
}
```

2460. Apply Operations to an Array

```
class Solution {
public:
    vector<int> applyOperations(vector<int>& nums) {
    int n = nums.size();
    for (int i = 0; i < n - 1; i++) {
        if (nums[i] == nums[i + 1]) {
            nums[i] *= 2;
            nums[i + 1] = 0;
        }
    }
    int idx = 0;
    for (int i = 0; i < n; i++) {
        if (nums[i] != 0) {
            swap(nums[idx++], nums[i]);
        }
    }
    return nums;
    }
};
```

```
class Solution {
    public int[] applyOperations(int[] nums) {
    int n = nums.length;
    for (int i = 0; i < n - 1; i++) {
        if (nums[i] == nums[i + 1]) {
            nums[i] *= 2;
            nums[i + 1] = 0;
        }
    }
    int idx = 0;
    for (int i = 0; i < n; i++) {
        if (nums[i] != 0) {
            int temp = nums[idx];
            nums[idx++] = nums[i];
            nums[i] = temp;
        }
    }
    return nums;
    }
}
```

408. Valid Word Abbreviation

```
class Solution {
public:
    bool validWordAbbreviation(string word, string abbr) {
   int i = 0, j = 0;
    while (i < word.size() && j < abbr.size()) {
        if (isdigit(abbr[j])) {
            if (abbr[j] == '0') return false; // Leading zeros are invalid
            int num = 0;
            while (j < abbr.size() && isdigit(abbr[j])) {
                num = num * 10 + (abbr[j++] - '0');
            }
            i += num;
        } else {
            if (word[i++] != abbr[j++]) return false;
        }
    }
    return i == word.size() && j == abbr.size();
}

};


class Solution {
    public boolean validWordAbbreviation(String word, String abbr) {
   int i = 0, j = 0;
    while (i < word.length() && j < abbr.length()) {
        if (Character.isDigit(abbr.charAt(j))) {
            if (abbr.charAt(j) == '0') return false; // Leading zeros are invalid
            int num = 0;
            while (j < abbr.length() && Character.isDigit(abbr.charAt(j))) {
                num = num * 10 + (abbr.charAt(j++) - '0');
            }
            i += num;
        } else {
            if (i >= word.length() || word.charAt(i++) != abbr.charAt(j++)) return false;
        }
    }
    return i == word.length() && j == abbr.length();
    }
}
```

925. Long Pressed Name

```
class Solution {
public:
    bool isLongPressedName(string name, string typed) {
     int i = 0, j = 0;
    while (j < typed.size()) {
        if (i < name.size() && name[i] == typed[j]) {
            i++;
            j++;
        } else if (j > 0 && typed[j] == typed[j - 1]) {
            j++;
        } else {
            return false;
        }
    }
    return i == name.size();
    }
};


class Solution {
    public boolean isLongPressedName(String name, String typed) {
    int i = 0, j = 0;
    while (j < typed.length()) {
        if (i < name.length() && name.charAt(i) == typed.charAt(j)) {
            i++;
            j++;
        } else if (j > 0 && typed.charAt(j) == typed.charAt(j - 1)) {
            j++;
        } else {
            return false;
        }
    }
    return i == name.length();
    }
}
```

2108. Find First Palindromic String in the Array

```
class Solution {
public:
bool isPalindrome(const string& s) {
    int left = 0, right = s.size() - 1;
    while (left < right) {
        if (s[left] != s[right]) {
            return false;
        }
        left++;
        right--;
    }
    return true;
}

    string firstPalindrome(vector<string>& words) {
     for (const string& word : words) {
        if (isPalindrome(word)) {
            return word;
        }
    }
    return "";
    }
};


class Solution {
    public String firstPalindrome(String[] words) {
     for (String word : words) {
        if (isPalindrome(word)) {
            return word;
        }
    }
    return "";
    }
    private boolean isPalindrome(String s) {
    int left = 0, right = s.length() - 1;
    while (left < right) {
        if (s.charAt(left) != s.charAt(right)) {
            return false;
        }
        left++;
        right--;
    }
    return true;
}
}
```

2562. Find the Array Concatenation Value

```
class Solution {
public:
    long long findTheArrayConcVal(vector<int>& nums) {
     long long concatenationValue = 0;
    int left = 0, right = nums.size() - 1;
    while (left <= right) {
        if (left == right) {
            concatenationValue += nums[left];
        } else {
            string concat = to_string(nums[left]) + to_string(nums[right]);
            concatenationValue += stoll(concat);
        }
        left++;
        right--;
    }
    return concatenationValue;
    }
};


class Solution {
    public long findTheArrayConcVal(int[] nums) {
    long concatenationValue = 0;
    int left = 0, right = nums.length - 1;
    while (left <= right) {
        if (left == right) {
            concatenationValue += nums[left];
        } else {
            String concat = nums[left] + "" + nums[right];
            concatenationValue += Long.parseLong(concat);
        }
        left++;
        right--;
    }
    return concatenationValue;
    }
}
```

2903. Find Indices With Index and Value Difference I

```
class Solution {
public:
    vector<int> findIndices(vector<int>& nums, int indexDifference, int valueDifference) {
  int n = nums.size();

    // Check all possible pairs of indices
    for(int i = 0; i < n; i++) {
        for(int j = i + indexDifference; j < n; j++) {
            // Check if this pair satisfies both conditions
            if(abs(nums[i] - nums[j]) >= valueDifference) {
                return {i, j};
            }
        }
    }

    // If no valid pair is found
    return {-1, -1};

    }
};


class Solution {
    public int[] findIndices(int[] nums, int indexDifference, int valueDifference) {
    int n = nums.length;

    // Check all possible pairs of indices
    for(int i = 0; i < n; i++) {
        for(int j = i + indexDifference; j < n; j++) {
            // Check if pair satisfies value difference condition
            if(Math.abs(nums[i] - nums[j]) >= valueDifference) {
                return new int[]{i, j};
            }
        }
    }

    // If no valid pair is found
    return new int[]{-1, -1};
    }
}
```