# Top LeetCode Interview Questions

## Most asked in Google Interview.

***Curated By*** **: Prakash Shukla**

## 1.Two Sum

**Link : https://leetcode.com/problems/two-sum/**

### Solution in Python

```python
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        hashmap = {}
        for i in range(len(nums)):
            complement = target - nums[i]
            if complement in hashmap:
                return [i, hashmap[complement]]
            hashmap[nums[i]] = i
```

### Solution in Java

```java
class Solution {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (map.containsKey(complement)) {
                return new int[] { map.get(complement), i };
            }
            map.put(nums[i], i);
        }
        // In case there is no solution, we'll just return null
        return null;
    }
}
```

# 2. Three Sum

**Link :** https://leetcode.com/problems/3sum/

## Solution in Java

```java
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        Set<List<Integer>> res = new HashSet<>();
        Set<Integer> dups = new HashSet<>();
        Map<Integer, Integer> seen = new HashMap<>();
        for (int i = 0; i < nums.length; ++i)
            if (dups.add(nums[i])) {
                for (int j = i + 1; j < nums.length; ++j) {
                    int complement = -nums[i] - nums[j];
                    if (seen.containsKey(complement) && seen.get(complement) == i) {
                        List<Integer> triplet = Arrays.asList(nums[i], nums[j],
complement);
                        Collections.sort(triplet);
                        res.add(triplet);
                    }
                    seen.put(nums[j], i);
                }
            }
        return new ArrayList(res);
    }
}
```

## Solution in Python

```python
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        res, dups = set(), set()
        seen = {}
        for i, val1 in enumerate(nums):
            if val1 not in dups:
                dups.add(val1)
                for j, val2 in enumerate(nums[i+1:]):
                    complement = -val1 - val2
                    if complement in seen and seen[complement] == i:
                        res.add(tuple(sorted((val1, val2, complement))))
                    seen[val2] = i
        return res
```

## Solution in C++

```cpp
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        set<vector<int>> res;
        unordered_set<int> dups;
        unordered_map<int, int> seen;
        for (int i = 0; i < nums.size(); ++i)
            if (dups.insert(nums[i]).second) {
                for (int j = i + 1; j < nums.size(); ++j) {
                    int complement = -nums[i] - nums[j];
                    auto it = seen.find(complement);
                    if (it != end(seen) && it->second == i) {
                        vector<int> triplet = {nums[i], nums[j], complement};
                        sort(begin(triplet), end(triplet));
                        res.insert(triplet);
                    }
                    seen[nums[j]] = i;
                }
            }
        return vector<vector<int>>(begin(res), end(res));
    }
};
```

# 3.Spiral Matrix

**Link : https://leetcode.com/problems/spiral-matrix/**

**Solution in Python**

```python
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        result = []
        rows, columns = len(matrix), len(matrix[0])
        up = left = 0
        right = columns - 1
        down = rows - 1

        while len(result) < rows * columns:
            # Traverse from left to right.
            for col in range(left, right + 1):
                result.append(matrix[up][col])

            # Traverse downwards.
            for row in range(up + 1, down + 1):
                result.append(matrix[row][right])

            # Make sure we are now on a different row.
            if up != down:
                # Traverse from right to left.
                for col in range(right - 1, left - 1, -1):
                    result.append(matrix[down][col])

            # Make sure we are now on a different column.
            if left != right:
                # Traverse upwards.
                for row in range(down - 1, up, -1):
                    result.append(matrix[row][left])

            left += 1
            right -= 1
            up += 1
            down -= 1

        return result
```

## Solution in Java

```java
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> result = new ArrayList<>();
        int rows = matrix.length;
        int columns = matrix[0].length;
        int up = 0;
        int left = 0;
        int right = columns - 1;
        int down = rows - 1;

        while (result.size() < rows * columns) {
            // Traverse from left to right.
            for (int col = left; col <= right; col++) {
                result.add(matrix[up][col]);
            }
            // Traverse downwards.
            for (int row = up + 1; row <= down; row++) {
                result.add(matrix[row][right]);
            }
            // Make sure we are now on a different row.
            if (up != down) {
                // Traverse from right to left.
                for (int col = right - 1; col >= left; col--) {
                    result.add(matrix[down][col]);
                }
            }
            // Make sure we are now on a different column.
            if (left != right) {
                // Traverse upwards.
                for (int row = down - 1; row > up; row--) {
                    result.add(matrix[row][left]);
                }
            }
            left++;
            right--;
            up++;
            down--;
        }

        return result;}}
```

# 4. Next Permutation

**Link : https://leetcode.com/problems/next-permutation/**

**Solution in Java**

```java
public class Solution {
    public void nextPermutation(int[] nums) {
        int i = nums.length - 2;
        while (i >= 0 && nums[i + 1] <= nums[i]) {
            i--;
        }
        if (i >= 0) {
            int j = nums.length - 1;
            while (nums[j] <= nums[i]) {
                j--;
            }
            swap(nums, i, j);
        }
        reverse(nums, i + 1);
    }

    private void reverse(int[] nums, int start) {
        int i = start, j = nums.length - 1;
        while (i < j) {
            swap(nums, i, j);
            i++;
            j--;
        }
    }

    private void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }
}
```

# 5.Longest substring without repeating characters

**Link :** https://leetcode.com/problems/longest-substring-without-repeating-characters/

## Solution in C++

```cpp
 class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        unordered_map<char, int> chars;

        int left = 0;
        int right = 0;

        int res = 0;
        while (right < s.length()) {
            char r = s[right];
            chars[r]++;

            while (chars[r] > 1) {
                char l = s[left];
                chars[l]--;
                left++;
            }

            res = max(res, right - left + 1);

            right++;
        }

        return res;
    }
};
```

## Solution in Java

```java
public class Solution {
    public int lengthOfLongestSubstring(String s) {
        Map<Character, Integer> chars = new HashMap();

        int left = 0;
        int right = 0;

        int res = 0;
        while (right < s.length()) {
            char r = s.charAt(right);
            chars.put(r, chars.getOrDefault(r,0) + 1);
```

```
            while (chars.get(r) > 1) {
                char l = s.charAt(left);
                chars.put(l, chars.get(l) - 1);
                left++;
            }

            res = Math.max(res, right - left + 1);

            right++;
        }
        return res;
    }
}
```

## Solution in Python

```
from collections import Counter
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        chars = Counter()

        left = right = 0

        res = 0
        while right < len(s):
            r = s[right]
            chars[r] += 1

            while chars[r] > 1:
                l = s[left]
                chars[l] -= 1
                left += 1

            res = max(res, right - left + 1)

            right += 1
        return res
```

# 6.Linked List Cycle

**Link : https://leetcode.com/problems/linked-list-cycle/**

## Solution in Python

```python
class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        if head is None:
            return False
        slow = head
        fast = head.next
        while slow != fast:
            if fast is None or fast.next is None:
                return False
            slow = slow.next
            fast = fast.next.next
        return True
```

## Solution in Java

```java
public class Solution {
    public boolean hasCycle(ListNode head) {
        if (head == null) {
            return false;
        }

        ListNode slow = head;
        ListNode fast = head.next;
        while (slow != fast) {
            if (fast == null || fast.next == null) {
                return false;
            }
            slow = slow.next;
            fast = fast.next.next;
        }
        return true;
    }
}
```

# 7.Middle of  Linked List

**Link : https://leetcode.com/problems/middle-of-the-linked-list/**

## Solution in Java

```java
class Solution {
    public ListNode middleNode(ListNode head) {
        ListNode slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
}
```

## Solution in Python

```python
class Solution:
    def middleNode(self, head):
        slow = fast = head
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next
        return slow
```

## Solution in C++

```cpp
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head;
        while (fast != NULL && fast->next != NULL) {
            slow = slow->next;
            fast = fast->next->next;
        }
        return slow;
    }
};
```

# 8.Reverse Linked List

**Link : https://leetcode.com/problems/reverse-linked-list/**

## Solution in Java

```java
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode prev = null;
        ListNode curr = head;
        while (curr != null) {
            ListNode nextTemp = curr.next;
            curr.next = prev;
            prev = curr;
            curr = nextTemp;
        }
        return prev;
    }
}
```

## Solution in C++

```cpp
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr;
        ListNode* curr = head;
        while (curr) {
            ListNode* nextTemp = curr->next;
            curr->next = prev;
            prev = curr;
            curr = nextTemp;
        }
        return prev;
    }
};
```

**Solution in Python**

```python
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        prev = None
        curr = head
        while curr:
            next_temp = curr.next
            curr.next = prev
            prev = curr
            curr = next_temp

        return prev
```

# 9.Palindrome Linked List

**Link : https://leetcode.com/problems/palindrome-linked-list/**

**Solution in Java**

```java
class Solution {
    public boolean isPalindrome(ListNode head) {
        if (head == null) return true;
        // Find the end of first half and reverse second half.
        ListNode firstHalfEnd = endOfFirstHalf(head);
        ListNode secondHalfStart = reverseList(firstHalfEnd.next);

        // Check whether or not there is a palindrome.
        ListNode p1 = head;
        ListNode p2 = secondHalfStart;
        boolean result = true;
        while (result && p2 != null) {
            if (p1.val != p2.val) result = false;
            p1 = p1.next;
            p2 = p2.next;
        }

        // Restore the list and return the result.
        firstHalfEnd.next = reverseList(secondHalfStart);
```

```
        return result;
    }

    // Taken from https://leetcode.com/problems/reverse-linked-list/solution/
    private ListNode reverseList(ListNode head) {
        ListNode prev = null;
        ListNode curr = head;
        while (curr != null) {
            ListNode nextTemp = curr.next;
            curr.next = prev;
            prev = curr;
            curr = nextTemp;
        }
        return prev;
    }

    private ListNode endOfFirstHalf(ListNode head) {
        ListNode fast = head;
        ListNode slow = head;
        while (fast.next != null && fast.next.next != null) {
            fast = fast.next.next;
            slow = slow.next;
        }
        return slow;
    }
}
```

## Solution in Python

```
class Solution:
    def isPalindrome(self, head: ListNode) -> bool:
        if head is None:
            return True

        # Find the end of first half and reverse second half.
        first_half_end = self.end_of_first_half(head)
        second_half_start = self.reverse_list(first_half_end.next)

        # Check whether or not there's a palindrome.
        result = True
```

```
        first_position = head
        second_position = second_half_start
        while result and second_position is not None:
            if first_position.val != second_position.val:
                result = False
            first_position = first_position.next
            second_position = second_position.next

        # Restore the list and return the result.
        first_half_end.next = self.reverse_list(second_half_start)
        return result

    def end_of_first_half(self, head):
        fast = head
        slow = head
        while fast.next is not None and fast.next.next is not None:
            fast = fast.next.next
            slow = slow.next
        return slow

    def reverse_list(self, head):
        previous = None
        current = head
        while current is not None:
            next_node = current.next
            current.next = previous
            previous = current
            current = next_node
        return previous
```

# 10.Remove Linked List Elements

**Link :** https://leetcode.com/problems/remove-linked-list-elements/

## Solution in C++

```cpp
class Solution {
  public:
  ListNode* removeElements(ListNode* head, int val) {
    ListNode* sentinel = new ListNode(0);
    sentinel->next = head;

    ListNode *prev = sentinel, *curr = head, *toDelete = nullptr;
    while (curr != nullptr) {
      if (curr->val == val) {
        prev->next = curr->next;
        toDelete = curr;
      } else prev = curr;

      curr = curr->next;

      if (toDelete != nullptr) {
        delete toDelete;
        toDelete = nullptr;
      }
    }

    ListNode *ret = sentinel->next;
    delete sentinel;
    return ret;
  }
};
```

## Solution in Java

```java
class Solution {
  public ListNode removeElements(ListNode head, int val) {
    ListNode sentinel = new ListNode(0);
    sentinel.next = head;

    ListNode prev = sentinel, curr = head;
    while (curr != null) {
      if (curr.val == val) prev.next = curr.next;
      else prev = curr;
      curr = curr.next;
    }
    return sentinel.next;
  }
```

```
}
```

## Solution in Python

```python
class Solution:
    def removeElements(self, head: ListNode, val: int) -> ListNode:
        sentinel = ListNode(0)
        sentinel.next = head

        prev, curr = sentinel, head
        while curr:
            if curr.val == val:
                prev.next = curr.next
            else:
                prev = curr
            curr = curr.next

        return sentinel.next
```

# Video solution of above problems

**If you want the proper solutions and explanations, then you can go through the below link.**

| | |
|---|---|
| **Spiral Matrix** | **https://opnr.app/yt/k6srf3ffk** |
| **Two Sum** | **https://opnr.app/yt/xftby1twd** |
| **3Sum** | **https://opnr.app/yt/nizlcqyno** |
| **Next Permutation** | **https://opnr.app/yt/j6gfsidwt** |
| **Longest substring without repeating characters** | **https://opnr.app/yt/d34zurn13** |
| **Linked List Cycle** | **https://opnr.app/yt/zp9l9riz5** |
| **Middle of  Linked List** | **https://opnr.app/yt/tn9epdmow** |
| **Reverse Linked List** | **https://opnr.app/yt/6kfoeuh24** |
| **Palindrome Linked List** | **https://opnr.app/yt/mn6ldwc96** |
| **Remove Linked List Elements** | **https://opnr.app/yt/w2f5111iz** |