NAME: Ashutosh Bhatt
ADM NO.- 22scse1011712
SECTION- 21
SUBMITTED TO: Aditya Trivedi
DATE: 27/04/2025

1. Prefix Sum Array and Range Sum Query

```
public class PrefixSum {
    public static int[] createPrefixSum(int[] arr) {
        int[] prefixSum = new int[arr.length];
        prefixSum[0] = arr[0];
        for (int i = 1; i < arr.length; i++)
            prefixSum[i] = prefixSum[i - 1] + arr[i];
        return prefixSum;
    }

    public static int rangeSum(int[] prefixSum, int L, int R) {
        if (L == 0)
            return prefixSum[R];
        return prefixSum[R] - prefixSum[L - 1];
    }
}
```

---

2. Equilibrium Index

```
public class EquilibriumIndex {
    public static int findEquilibriumIndex(int[] arr) {
        int total = 0, leftSum = 0;
        for (int num : arr) total += num;

        for (int i = 0; i < arr.length; i++) {
            total -= arr[i];
            if (leftSum == total)
                return i;
            leftSum += arr[i];
        }
        return -1;
    }
```

```
}
```

---

3. Split Array into Equal Prefix and Suffix

```
public class SplitEqualPrefixSuffix {
    public static boolean canBeSplit(int[] arr) {
        int total = 0;
        for (int num : arr) total += num;

        int prefix = 0;
        for (int i = 0; i < arr.length - 1; i++) {
            prefix += arr[i];
            if (prefix == total - prefix)
                return true;
        }
        return false;
    }
}
```

---

4. Maximum Sum Subarray of Size K (Sliding Window)\

```
public class MaxSumSubarray {
    public static int maxSum(int[] arr, int k) {
        int windowSum = 0;
        for (int i = 0; i < k; i++) windowSum += arr[i];

        int maxSum = windowSum;
        for (int i = k; i < arr.length; i++) {
            windowSum += arr[i] - arr[i - k];
            maxSum = Math.max(maxSum, windowSum);
        }
        return maxSum;
    }
}
```

---

5. Longest Substring Without Repeating Characters

```
import java.util.*;

public class LongestUniqueSubstring {
    public static int lengthOfLongestSubstring(String s) {
        Set<Character> set = new HashSet<>();
        int left = 0, maxLength = 0;
        for (int right = 0; right < s.length(); right++) {
            while (set.contains(s.charAt(right))) {
                set.remove(s.charAt(left++));
            }
            set.add(s.charAt(right));
            maxLength = Math.max(maxLength, right - left + 1);
        }
        return maxLength;
    }
}
```

---

6. Sliding Window Technique Explanation

```
int maxSum = 0, windowSum = 0;
for (int i = 0; i < k; i++) {
    windowSum += arr[i]; // First window
}
maxSum = windowSum;

for (int i = k; i < arr.length; i++) {
    windowSum += arr[i] - arr[i - k]; // Slide window
    maxSum = Math.max(maxSum, windowSum);
}
```

---

7. Longest Palindromic Substring

```
public class LongestPalindrome {
```

```
    public static String longestPalindrome(String s) {
        if (s == null || s.length() < 1) return "";
        int start = 0, end = 0;
        for (int i = 0; i < s.length(); i++) {
            int len1 = expand(s, i, i);
            int len2 = expand(s, i, i + 1);
            int len = Math.max(len1, len2);
            if (len > end - start) {
                start = i - (len - 1) / 2;
                end = i + len / 2;
            }
        }
        return s.substring(start, end + 1);
    }

    private static int expand(String s, int left, int right) {
        while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
            left--;
            right++;
        }
        return right - left - 1;
    }
}
```

---

8. Longest Common Prefix

```
public class LongestCommonPrefix {
    public static String findLCP(String[] strs) {
        if (strs.length == 0) return "";
        String prefix = strs[0];
        for (int i = 1; i < strs.length; i++) {
            while (!strs[i].startsWith(prefix)) {
                prefix = prefix.substring(0, prefix.length() - 1);
```

```
                if (prefix.isEmpty()) return "";
            }
        }
        return prefix;
    }
}
```

---

9. All Permutations of a String

```
public class StringPermutations {
    public static void generatePermutations(String str, String result) {
        if (str.length() == 0) {
            System.out.println(result);
            return;
        }
        for (int i = 0; i < str.length(); i++) {
            generatePermutations(str.substring(0, i) + str.substring(i + 1), result +
str.charAt(i));
        }
    }

    public static void main(String[] args) {
        generatePermutations("ABC", "");
    }
}
```

10. Generate all permutations of a given string. Write its algorithm, program. Find its time and space complexities.\

```
public class Permutations {
    public static void generatePermutations(String str) {
        permute(str.toCharArray(), 0);
    }

    private static void permute(char[] chars, int index) {
        if (index == chars.length) {
            System.out.println(String.valueOf(chars));
            return;
        }

        for (int i = index; i < chars.length; i++) {
```

```
            swap(chars, index, i);              // Fix a character
            permute(chars, index + 1);          // Recurse
            swap(chars, index, i);              // Backtrack
        }
    }

    private static void swap(char[] chars, int i, int j) {
        char temp = chars[i];
        chars[i] = chars[j];
        chars[j] = temp;
    }

    public static void main(String[] args) {
        String str = "abc";
        System.out.println("Permutations of " + str + ":");
        generatePermutations(str);
    }
}
```

11.Find two numbers in a sorted array that add up to a target. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
public class FindTwoNumbers {
public static int[] findTwoNumbers(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;

        while (left < right) {
            int sum = nums[left] + nums[right];
            if (sum == target) {
                return new int[] {nums[left], nums[right]};
            } else if (sum < target) {
                left++;
            } else {
                right--;
            }
        }
        return new int[] {};
    }
public static void main(String[] args) {
        int[] nums = {1, 2, 3, 4, 6, 8, 10};
        int target = 10;
        int[] result = findTwoNumbers(nums, target);
        if (result.length > 0) {
```

```
            System.out.println("Pair found: " + result[0] + " and " + result[1]);
        } else {
            System.out.println("No pair found.");
        }
    }
}
```

12.Rearrange numbers into the lexicographically next greater permutation. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
import java.util.Arrays;

public class NextPermutation {
public static void nextPermutation(int[] nums) {
        int n = nums.length;
        int pivot = -1;
        for (int i = n - 2; i >= 0; i--) {
            if (nums[i] < nums[i + 1]) {
                pivot = i;
                break;
            }
        }
        if (pivot == -1) {
            // If no pivot, reverse the array
            reverse(nums, 0, n - 1);
            return;
        }
        for (int i = n - 1; i > pivot; i--) {
            if (nums[i] > nums[pivot]) {
                // Step 3: Swap
                swap(nums, i, pivot);
                break;
            }
        }
        reverse(nums, pivot + 1, n - 1);
    }
    private static void reverse(int[] nums, int start, int end) {
        while (start < end) {
            swap(nums, start++, end--);
        }
    }
    private static void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }
```

```
    public static void main(String[] args) {
        int[] nums = {1, 2, 3};
        nextPermutation(nums);
        System.out.println(Arrays.toString(nums));
    }
}
```

13.How to merge two sorted linked lists into one sorted list. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
class ListNode {
 int val;
    ListNode next;
    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class MergeSortedLists {
    public static ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode dummy = new ListNode(-1); // Dummy node
        ListNode current = dummy;

        while (list1 != null && list2 != null) {
            if (list1.val <= list2.val) {
                current.next = list1;
                list1 = list1.next;
            } else {
                current.next = list2;
                list2 = list2.next;
            }
            current = current.next;
        }
        if (list1 != null) {
            current.next = list1;
        } else {
            current.next = list2;
        }
        return dummy.next;
    }
    public static void main(String[] args) {
        ListNode list1 = new ListNode(1);
        list1.next = new ListNode(3);
        list1.next.next = new ListNode(5);
        ListNode list2 = new ListNode(2);
```

```
        list2.next = new ListNode(4);
        list2.next.next = new ListNode(6);
        ListNode mergedList = mergeTwoLists(list1, list2);
        while (mergedList != null) {
            System.out.print(mergedList.val + " ");
            mergedList = mergedList.next;
        }
    }
}
```

Time Complexity –

Space Complexity-

14. Find the median of two sorted arrays using binary search. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
public class MedianOfTwoSortedArrays {
public static double findMedianSortedArrays(int[] nums1, int[] nums2) {
        if (nums1.length > nums2.length) {
            return findMedianSortedArrays(nums2, nums1);
        }

        int x = nums1.length;
        int y = nums2.length;
        int low = 0, high = x;

        while (low <= high) {
            int partitionX = (low + high) / 2;
            int partitionY = (x + y + 1) / 2 - partitionX;

            int maxLeftX = (partitionX == 0) ? Integer.MIN_VALUE : nums1[partitionX - 1];
            int minRightX = (partitionX == x) ? Integer.MAX_VALUE : nums1[partitionX];

            int maxLeftY = (partitionY == 0) ? Integer.MIN_VALUE : nums2[partitionY - 1];
            int minRightY = (partitionY == y) ? Integer.MAX_VALUE : nums2[partitionY];

            if (maxLeftX <= minRightY && maxLeftY <= minRightX) {
                if ((x + y) % 2 == 0) {
                    return ((double)Math.max(maxLeftX, maxLeftY) + Math.min(minRightX,
minRightY)) / 2;
                } else {
                    return (double)Math.max(maxLeftX, maxLeftY);
                }
            } else if (maxLeftX > minRightY) {
                high = partitionX - 1;
            } else {
                low = partitionX + 1;
```

```
            }
        }

        throw new IllegalArgumentException("Input arrays are not sorted.");
    }

    public static void main(String[] args) {
        int[] nums1 = {1, 3};
        int[] nums2 = {2};
        System.out.println("Median: " + findMedianSortedArrays(nums1, nums2));
    }
}
```

15. Find the k-th smallest element in a sorted matrix. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
public class KthSmallestElement {
public static int kthSmallest(int[][] matrix, int k) {
        int n = matrix.length;
        int low = matrix[0][0];
        int high = matrix[n - 1][n - 1];

        while (low < high) {
            int mid = low + (high - low) / 2;
            int count = countLessEqual(matrix, mid);

            if (count < k) {
                low = mid + 1;
            } else {
                high = mid;
            }
        }
        return low;
    }

 private static int countLessEqual(int[][] matrix, int target) {
        int n = matrix.length;
        int count = 0;
        int row = n - 1, col = 0;

        while (row >= 0 && col < n) {
            if (matrix[row][col] <= target) {
                count += row + 1;
                col++;
            } else {
                row--;
```

```
            }
        }
        return count;
    }

    public static void main(String[] args) {
        int[][] matrix = {
            {1, 5, 9},
            {10, 11, 13},
            {12, 13, 15}
        };
        int k = 8;
        System.out.println("K-th smallest element: " + kthSmallest(matrix, k));
    }
}
```

16. Find the majority element in an array that appears more than n/2 times. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
public class MajorityElement {
    public static int findMajorityElement(int[] nums) {
        int candidate = nums[0];
        int count = 0;

        for (int num : nums) {
            if (count == 0) {
                candidate = num;
            }
            count += (num == candidate) ? 1 : -1;
        }

        count = 0;
        for (int num : nums) {
            if (num == candidate) {
                count++;
            }
        }

        if (count > nums.length / 2) {
            return candidate;
        } else {
            throw new IllegalArgumentException("No majority element found.");
        }
    }

    public static void main(String[] args) {
```

```
        int[] nums = {2, 2, 1, 1, 1, 2, 2};
        System.out.println("Majority Element: " + findMajorityElement(nums));
    }
}
```

17. Calculate how much water can be trapped between the bars of a histogram. Write its algorithm, program. Find its time and space complexities. Explain with suitable example

```
public class TrappingRainWater {
    public static int trap(int[] height) {
        int left = 0, right = height.length - 1;
        int leftMax = 0, rightMax = 0;
        int water = 0;

        while (left < right) {
            if (height[left] < height[right]) {
                if (height[left] >= leftMax) {
                    leftMax = height[left];
                } else {
                    water += leftMax - height[left];
                }
                left++;
            } else {
                if (height[right] >= rightMax) {
                    rightMax = height[right];
                } else {
                    water += rightMax - height[right];
                }
                right--;
            }
        }
        return water;
    }

    public static void main(String[] args) {
        int[] height = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};
        int result = trap(height);
        System.out.println("Total water trapped: " + result);
    }
}
```

18. Maximum XOR of Two Numbers in an Array

Algorithm:

1. Use a Trie (binary tree) to insert all numbers in binary.
2. For each number, try to find a number in Trie that gives the maximum XOR with it.

Time Complexity:

- Time: O(n * 32) → 32 for max bit size
- Space: O(n * 32) for Trie

Java Code:

java
CopyEdit

```
class TrieNode {
    TrieNode[] children = new TrieNode[2];
}

public class MaxXOR {
    TrieNode root = new TrieNode();

    public void insert(int num) {
        TrieNode node = root;
        for (int i = 31; i >= 0; i--) {
            int bit = (num >>> i) & 1;
            if (node.children[bit] == null)
                node.children[bit] = new TrieNode();
            node = node.children[bit];
        }
    }

    public int findMaxXOR(int num) {
        TrieNode node = root;
        int maxXOR = 0;
        for (int i = 31; i >= 0; i--) {
            int bit = (num >>> i) & 1;
            int toggled = 1 - bit;
            if (node.children[toggled] != null) {
                maxXOR |= (1 << i);
                node = node.children[toggled];
            } else {
                node = node.children[bit];
            }
```

```
        }
        return maxXOR;
    }

    public int findMaximumXOR(int[] nums) {
        for (int num : nums) insert(num);
        int max = 0;
        for (int num : nums)
            max = Math.max(max, findMaxXOR(num));
        return max;
    }

    public static void main(String[] args) {
        int[] nums = {3, 10, 5, 25, 2, 8};
        MaxXOR obj = new MaxXOR();
        System.out.println("Maximum XOR: " + obj.findMaximumXOR(nums));
    }
}
```

Example:

- Input: [3, 10, 5, 25, 2, 8]
- Output: 28 (XOR of 5 and 25)

---

19. Maximum Product Subarray

Algorithm:

1. Track current max and min (because of negative values).
2. For each element, update max and min.
3. Track the global maximum.

Time & Space:

- Time: O(n)

- Space: O(1)

Java Code:

```
public class MaxProductSubarray {
    public static int maxProduct(int[] nums) {
        int max = nums[0], min = nums[0], result = nums[0];

        for (int i = 1; i < nums.length; i++) {
            int temp = max;
            max = Math.max(nums[i], Math.max(max * nums[i], min * nums[i]));
            min = Math.min(nums[i], Math.min(temp * nums[i], min * nums[i]));
            result = Math.max(result, max);
        }

        return result;
    }

    public static void main(String[] args) {
        int[] nums = {2, 3, -2, 4};
        System.out.println("Max Product Subarray: " + maxProduct(nums));
    }
}
```

Example:

- Input: [2, 3, -2, 4]
- Output: 6 (2 * 3)

---

20. Count Numbers with Unique Digits

Algorithm:

1. For n = 0, return 1.
2. Use combinatorics:
    - First digit: 9 choices (1–9)

- Second digit: 9 choices (0–9 excluding used one)
- Then 8, 7,... etc.

Time & Space:

- Time: O(n)
- Space: O(1)

Java Code:

```
public class UniqueDigits {
    public static int countNumbersWithUniqueDigits(int n) {
        if (n == 0) return 1;
        int result = 10, uniqueDigits = 9, available = 9;

        for (int i = 2; i <= n && available > 0; i++) {
            uniqueDigits *= available;
            result += uniqueDigits;
            available--;
        }

        return result;
    }

    public static void main(String[] args) {
        int n = 2;
        System.out.println("Count of numbers with unique digits for n = " + n + ": " +
countNumbersWithUniqueDigits(n));
    }
```

Q21. How to count the number of 1s in the binary representation of numbers from 0 to n. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
import java.util.Arrays;

public class CountSetBits {

        public static int[] countBits(int n) {

        int[] countBits = new int[n + 1];
```

```
        countBits[0] = 0;
        for (int i = 1; i <= n; i++) {
        countBits[i] = countBits[i >> 1] + (i & 1);
        }
        return countBits;
        }
        public static void main(String[] args) {
        int n = 5;
        int[] result = countBits(n);
        System.out.println("Number of 1s in binary from 0 to " + n + ": " + Arrays.toString(result));
        }
}
```

Q22. How to check if a number is a power of two using bit manipulation. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
public class PowerOfTwo {
        public static boolean isPowerOfTwo(int n) {
        if (n <= 0) return false;
        return (n & (n - 1)) == 0;
        }

        public static void main(String[] args) {
        int[] testNumbers = {1, 2, 3, 4, 5, 16, 18};
        for (int num : testNumbers) {
        System.out.println(num + " is power of 2?  " + isPowerOfTwo(num));
        }
```

```
        }
}
```

Q23. How to find the maximum XOR of two numbers in an array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
import java.util.HashSet;

public class MaxXORWithoutTrie {

        public static int findMaximumXOR(int[] nums) {

        int maxXOR = 0, mask = 0;

        for (int i = 31; i >= 0; i--) {

        mask |= (1 << i); // Update the mask to include the i-th bit

        HashSet<Integer> prefixes = new HashSet<>();

        for (int num : nums) {

            prefixes.add(num & mask);

        }

        int candidate = maxXOR | (1 << i);

        for (int prefix : prefixes) {

                if (prefixes.contains(prefix ^ candidate)) {

                maxXOR = candidate;

                break;

                }

        }

        }

        return maxXOR;

        }
```

```
    public static void main(String[] args) {
    int[] nums = {3, 10, 5, 25, 2, 8};
   System.out.println("Maximum XOR: " + findMaximumXOR(nums));
    }
}
```

Q24. Explain the concept of bit manipulation and its advantages in algorithm design.

Bit manipulation involves performing operations directly on the binary representation of numbers using bitwise operators. These operators include:

Operator	Symbol	Description
AND	&	1 if both bits are 1
OR	`	`
XOR	^	1 if bits are different
NOT	~	Flips all bits
Left Shift	<<	Shifts bits to the left (multiplies by 2^n)
Right Shift	>>	Shifts bits to the right (divides by 2^n)

Example :

```
int a = 5;        // binary: 0101
int b = 3;        // binary: 0011
```

System.out.println(a & b); // 1 (0001)

System.out.println(a | b); // 7 (0111)

System.out.println(a ^ b); // 6 (0110)

Advantages in Algorithm Design :

1. Efficiency:
    - Bitwise operations are faster than arithmetic or logical operations (single CPU instruction).
    - Useful in time-critical applications like cryptography, networking, or embedded systems.
2. Memory Optimization:
    - Compactly store and manipulate data (e.g., flags, masks, sets) using single integers.
    - Ideal for fixed-width data like 32-bit or 64-bit numbers.
3. Simpler Logic for Certain Problems:
    - Toggle a bit: x ^ (1 << i)
    - Check if power of two: x & (x - 1) == 0
    - Count set bits: Brian Kernighan's Algorithm
4. Real-World Applications:
    - Finding subsets using bitmasking.
    - Swapping values without temporary variable.
    - Optimization in DSA problems like max XOR, set operations, etc.

Example :

Problem: Check if a number is a power of 2

```
boolean isPowerOfTwo(int n) {

        return n > 0 && (n & (n - 1)) == 0;

}
```

Why it works:

- Power of two has only one bit set, so n & (n - 1) becomes 0.

Time & Space Complexity :

- Time Complexity: O(1) per operation (very fast).
- Space Complexity: O(1), only basic variables are used.

Q25. Solve the problem of finding the next greater element for each element in an array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

:

```
import java.util.Stack;

import java.util.Arrays;

public class NextGreaterElement {

        public static int[] nextGreaterElements(int[] nums) {

        int n = nums.length;

        int[] res = new int[n];

        Stack<Integer> stack = new Stack<>();

        for (int i = n - 1; i >= 0; i--) {

        while (!stack.isEmpty() && stack.peek() <= nums[i]) {

                stack.pop();

        }

        res[i] = stack.isEmpty() ? -1 : stack.peek();

        stack.push(nums[i]);

        }

        return res;

        }

        public static void main(String[] args) {
```

```
        int[] nums = {4, 5, 2, 10, 8};

      System.out.println("Next Greater Elements: " +
Arrays.toString(nextGreaterElements(nums)));

        }

}
```

Q26. Remove the n-th node from the end of a singly linked list. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
class ListNode {

        int val;

        ListNode next;

        ListNode(int x) { val = x; }

}


public class RemoveNthNode {

        public static ListNode removeNthFromEnd(ListNode head, int n) {

        ListNode dummy = new ListNode(0);

        dummy.next = head;

        ListNode fast = dummy;

        ListNode slow = dummy;


        for (int i = 0; i <= n; i++) {

        fast = fast.next;
```

```
}

while (fast != null) {
fast = fast.next;
slow = slow.next;
}

slow.next = slow.next.next;

return dummy.next;
}

public static void printList(ListNode head) {
while (head != null) {
System.out.print(head.val + " ");
head = head.next;
}
System.out.println();
}

public static void main(String[] args) {
ListNode head = new ListNode(1);
head.next = new ListNode(2);
head.next.next = new ListNode(3);
```

```
        head.next.next.next = new ListNode(4);
        head.next.next.next.next = new ListNode(5);

        int n = 2;
        head = removeNthFromEnd(head, n);

        System.out.print("List after removing " + n + "th node from end: ");
        printList(head);
        }
}
// 27. Intersection of two linked lists
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) {
        val = x;
        next = null;
    }
}

class LinkedListIntersection {
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        ListNode a = headA, b = headB;
        while (a != b) {
            a = (a == null) ? headB : a.next;
            b = (b == null) ? headA : b.next;
        }
        return a;
    }
}

// 28. Two stacks in a single array
class TwoStacks {
    int[] arr;
    int top1, top2;
    TwoStacks(int n) {
        arr = new int[n];
```

```
        top1 = -1;
        top2 = n;
    }
    void push1(int x) {
        if (top1 + 1 < top2) arr[++top1] = x;
    }
    void push2(int x) {
        if (top1 + 1 < top2) arr[--top2] = x;
    }
    int pop1() {
        return top1 >= 0 ? arr[top1--] : -1;
    }
    int pop2() {
        return top2 < arr.length ? arr[top2++] : -1;
    }
}

// 29. Integer palindrome without string conversion
class IntegerPalindrome {
    public boolean isPalindrome(int x) {
        if (x < 0 || (x % 10 == 0 && x != 0)) return false;
        int reversed = 0;
        while (x > reversed) {
            reversed = reversed * 10 + x % 10;
            x /= 10;
        }
        return x == reversed || x == reversed / 10;
    }
}

// 30. Concept of linked lists
/*
A linked list is a linear data structure where each element (node) contains a value and a
reference to the next node.
Applications: memory-efficient lists, dynamic data manipulation, stack/queue implementations,
graph adjacency.
*/

// 31. Max in every sliding window of size K using deque
import java.util.*;
class MaxSlidingWindow {
    public int[] maxSlidingWindow(int[] nums, int k) {
        Deque<Integer> deque = new LinkedList<>();
        int[] result = new int[nums.length - k + 1];
```

```
        for (int i = 0; i < nums.length; i++) {
            while (!deque.isEmpty() && deque.peek() < i - k + 1)
                deque.poll();
            while (!deque.isEmpty() && nums[deque.peekLast()] < nums[i])
                deque.pollLast();
            deque.offer(i);
            if (i >= k - 1) result[i - k + 1] = nums[deque.peek()];
        }
        return result;
    }
}

// 32. Largest rectangle in a histogram
class LargestRectangle {
    public int largestRectangleArea(int[] heights) {
        Stack<Integer> stack = new Stack<>();
        int max = 0, i = 0;
        while (i < heights.length) {
            if (stack.isEmpty() || heights[i] >= heights[stack.peek()])
                stack.push(i++);
            else {
                int h = heights[stack.pop()];
                int w = stack.isEmpty() ? i : i - stack.peek() - 1;
                max = Math.max(max, h * w);
            }
        }
        while (!stack.isEmpty()) {
            int h = heights[stack.pop()];
            int w = stack.isEmpty() ? i : i - stack.peek() - 1;
            max = Math.max(max, h * w);
        }
        return max;
    }
}

// 33. Sliding window technique in arrays
/*
Used to reduce the time complexity of problems involving subarrays or substrings.
Efficient for max/min/sum problems.
Examples: max sum subarray of size k, longest substring with k distinct characters, etc.
*/

// 34. Subarray sum equals k using hashing
class SubarraySumEqualsK {
```

```
    public int subarraySum(int[] nums, int k) {
        Map<Integer, Integer> map = new HashMap<>();
        map.put(0, 1);
        int sum = 0, count = 0;
        for (int num : nums) {
            sum += num;
            count += map.getOrDefault(sum - k, 0);
            map.put(sum, map.getOrDefault(sum, 0) + 1);
        }
        return count;
    }
}

// 35. K most frequent elements
class KMostFrequent {
    public int[] topKFrequent(int[] nums, int k) {
        Map<Integer, Integer> freqMap = new HashMap<>();
        for (int num : nums) freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);
        PriorityQueue<Integer> heap = new PriorityQueue<>((a, b) -> freqMap.get(a) -
freqMap.get(b));
        for (int key : freqMap.keySet()) {
            heap.offer(key);
            if (heap.size() > k) heap.poll();
        }
        int[] res = new int[k];
        for (int i = k - 1; i >= 0; i--) res[i] = heap.poll();
        return res;
    }
}

// 36. Generate all subsets of array
class SubsetsGenerator {
    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        backtrack(res, new ArrayList<>(), nums, 0);
        return res;
    }
    void backtrack(List<List<Integer>> res, List<Integer> temp, int[] nums, int start) {
        res.add(new ArrayList<>(temp));
        for (int i = start; i < nums.length; i++) {
            temp.add(nums[i]);
            backtrack(res, temp, nums, i + 1);
            temp.remove(temp.size() - 1);
        }
```

```
    }
}

// 37. Unique combinations summing to target
class CombinationSum {
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> res = new ArrayList<>();
        backtrack(res, new ArrayList<>(), candidates, target, 0);
        return res;
    }
    void backtrack(List<List<Integer>> res, List<Integer> temp, int[] candidates, int remain, int
start) {
        if (remain < 0) return;
        if (remain == 0) res.add(new ArrayList<>(temp));
        else {
            for (int i = start; i < candidates.length; i++) {
                temp.add(candidates[i]);
                backtrack(res, temp, candidates, remain - candidates[i], i);
                temp.remove(temp.size() - 1);
            }
        }
    }
}

// 38. Generate all permutations of an array
class Permutations {
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        backtrack(res, new ArrayList<>(), nums, new boolean[nums.length]);
        return res;
    }
    void backtrack(List<List<Integer>> res, List<Integer> temp, int[] nums, boolean[] used) {
        if (temp.size() == nums.length) res.add(new ArrayList<>(temp));
        else {
            for (int i = 0; i < nums.length; i++) {
                if (used[i]) continue;
                used[i] = true;
                temp.add(nums[i]);
                backtrack(res, temp, nums, used);
                used[i] = false;
                temp.remove(temp.size() - 1);
            }
        }
    }
```

```
}

// 39. Difference between subsets and permutations:
// Subsets are any combinations of elements (order doesn't matter), e.g., for {1, 2}, subsets: {},
{1}, {2}, {1,2}
// Permutations are ordered arrangements, e.g., for {1, 2}, permutations: {1,2}, {2,1}

// 40. Max Frequency Element
public static int maxFrequencyElement(int[] arr) {
    Map<Integer, Integer> freqMap = new HashMap<>();
    for (int num : arr) {
        freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);
    }
    int maxFreq = 0, result = -1;
    for (Map.Entry<Integer, Integer> entry : freqMap.entrySet()) {
        if (entry.getValue() > maxFreq) {
            maxFreq = entry.getValue();
            result = entry.getKey();
        }
    }
    return result;
    // Time: O(n), Space: O(n)
}

// 41. Kadane's Algorithm
public static int maxSubArraySum(int[] nums) {
    int maxSoFar = nums[0], currMax = nums[0];
    for (int i = 1; i < nums.length; i++) {
        currMax = Math.max(nums[i], currMax + nums[i]);
        maxSoFar = Math.max(maxSoFar, currMax);
    }
    return maxSoFar;
    // Time: O(n), Space: O(1)
}

// 42. Dynamic Programming Concept
// DP is solving a complex problem by breaking it into simpler subproblems and storing their
results.
// Used in Kadane’s Algorithm by building the solution from past optimal results.

// 43. Top K Frequent Elements
public static List<Integer> topKFrequent(int[] nums, int k) {
    Map<Integer, Integer> freqMap = new HashMap<>();
    for (int num : nums)
```

```
        freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);

    PriorityQueue<Map.Entry<Integer, Integer>> pq = new PriorityQueue<>((a, b) -> a.getValue()
- b.getValue());

    for (Map.Entry<Integer, Integer> entry : freqMap.entrySet()) {
        pq.offer(entry);
        if (pq.size() > k) pq.poll();
    }

    List<Integer> result = new ArrayList<>();
    while (!pq.isEmpty()) result.add(pq.poll().getKey());
    Collections.reverse(result);
    return result;
    // Time: O(n log k), Space: O(n)
}

// 44. Two Sum Using Hashing
public static 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);
    }
    return new int[]{};
    // Time: O(n), Space: O(n)
}

// 45. Priority Queues
// A priority queue retrieves elements by priority, not insertion order.
// Useful in Dijkstra's algorithm, A* search, Huffman coding, etc.

// 46. Longest Palindromic Substring
public static String longestPalindrome(String s) {
    if (s == null || s.length() < 1) return "";
    int start = 0, end = 0;
    for (int i = 0; i < s.length(); i++) {
        int len1 = expandAroundCenter(s, i, i);
        int len2 = expandAroundCenter(s, i, i + 1);
        int len = Math.max(len1, len2);
        if (len > end - start) {
```

```
            start = i - (len - 1) / 2;
            end = i + len / 2;
        }
    }
    return s.substring(start, end + 1);
}
private static int expandAroundCenter(String s, int left, int right) {
    while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
        left--;
        right++;
    }
    return right - left - 1;
    // Time: O(n^2), Space: O(1)
}

// 47. Histogram Problem
// Largest Rectangle in Histogram is a classic problem.
// Uses stack to store indices and calculate max area efficiently.
// Applications in skyline problems, image processing, etc.

// 48. Next Permutation
public static void nextPermutation(int[] nums) {
    int i = nums.length - 2;
    while (i >= 0 && nums[i] >= nums[i + 1]) i--;
    if (i >= 0) {
        int j = nums.length - 1;
        while (nums[j] <= nums[i]) j--;
        swap(nums, i, j);
    }
    reverse(nums, i + 1);
}

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

private static void reverse(int[] nums, int start) {
    int i = start, j = nums.length - 1;
    while (i < j) swap(nums, i++, j--);
    // Time: O(n), Space: O(1)
}
```

```
// 49. Intersection of Two Linked Lists
public static ListNode getIntersectionNode(ListNode headA, ListNode headB) {
    Set<ListNode> seen = new HashSet<>();
    while (headA != null) {
        seen.add(headA);
        headA = headA.next;
    }
    while (headB != null) {
        if (seen.contains(headB)) return headB;
        headB = headB.next;
    }
    return null;
    // Time: O(m + n), Space: O(m)
}

// 50. Equilibrium Index
public static int findEquilibriumIndex(int[] arr) {
    int totalSum = Arrays.stream(arr).sum();
    int leftSum = 0;
    for (int i = 0; i < arr.length; i++) {
        if (leftSum == totalSum - leftSum - arr[i]) return i;
        leftSum += arr[i];
    }
    return -1;
    // Time: O(n), Space: O(1)
}

class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; next = null; }
}
```