DSA Assignment for MTE

Name – Piyush Chaudhary

Admission No - 22SCSE1012120

1. Explain the concept of a prefix sum array and its applications.

A prefix sum array is an array where each element at index i contains the sum of all elements from index 0 to i in the original array.

Applications:
- Efficient range sum queries
- Histogram problems
- Array transformations
- Subarray sum problems

2. Write a program to find the sum of elements in a given range [L, R] using a prefix sum array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
1. Build prefix sum array where prefix[i] = prefix[i-1] + arr[i]
2. To find sum from L to R: result = prefix[R] - prefix[L - 1] (if L > 0)

```
#include <iostream>
#include <vector>
using namespace std;

vector<int> buildPrefixSum(const vector<int>& arr) {
    vector<int> prefix(arr.size());
    prefix[0] = arr[0];
    for (int i = 1; i < arr.size(); i++)
        prefix[i] = prefix[i - 1] + arr[i];
    return prefix;
}

int rangeSum(const vector<int>& prefix, int L, int R) {
    if (L == 0) return prefix[R];
    return prefix[R] - prefix[L - 1];
}

int main() {
    vector<int> arr = {1, 2, 3, 4, 5};
    vector<int> prefix = buildPrefixSum(arr);
    cout << "Sum from 1 to 3 is " << rangeSum(prefix, 1, 3);
    return 0;
}
```

Time Complexity: O(n) for preprocessing, O(1) for query
Space Complexity: O(n)

3. Solve the problem of finding the equilibrium index in an array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
1. Calculate total sum of array.
2. Initialize leftSum = 0
3. For each index i, check if leftSum == totalSum - leftSum - arr[i]

```
#include <iostream>
#include <vector>
using namespace std;

int findEquilibriumIndex(const vector<int>& arr) {
    int total = 0, leftSum = 0;
    for (int num : arr) total += num;
    for (int i = 0; i < arr.size(); i++) {
        if (leftSum == total - leftSum - arr[i])
            return i;
        leftSum += arr[i];
    }
    return -1;
}

int main() {
    vector<int> arr = {-7, 1, 5, 2, -4, 3, 0};
    cout << "Equilibrium Index: " << findEquilibriumIndex(arr);
    return 0;
}
```

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

4. Check if an array can be split into two parts such that the sum of the prefix equals the sum of the suffix. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
1. Compute total sum.
2. Traverse the array and keep prefix sum.
3. If at any point prefix sum == total - prefix sum, return true.

```
#include <iostream>
#include <vector>
using namespace std;

bool canSplitEqualSum(const vector<int>& arr) {
    int total = 0, prefix = 0;
    for (int num : arr) total += num;
    for (int i = 0; i < arr.size(); i++) {
        prefix += arr[i];
        if (prefix * 2 == total) return true;
    }
    return false;
}

int main() {
    vector<int> arr = {1, 2, 3, 3};
    cout << (canSplitEqualSum(arr) ? "Yes" : "No");
    return 0;
}
```

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

5. Find the maximum sum of any subarray of size K in a given array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
1. Use sliding window of size K.
2. Calculate sum of first window.
3. Slide the window and update max sum.

```
#include <iostream>
#include <vector>
using namespace std;

int maxSumSubarrayK(const vector<int>& arr, int k) {
    int windowSum = 0, maxSum = 0;
    for (int i = 0; i < k; i++) windowSum += arr[i];
    maxSum = windowSum;
    for (int i = k; i < arr.size(); i++) {
        windowSum += arr[i] - arr[i - k];
        maxSum = max(maxSum, windowSum);
    }
    return maxSum;
}

int main() {
    vector<int> arr = {100, 200, 300, 400};
    int k = 2;
    cout << "Max sum of subarray of size " << k << " is " <<
maxSumSubarrayK(arr, k);
    return 0;
}
```

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

6. Find the length of the longest substring without repeating characters. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
1. Use sliding window and hash map to store last seen indices.
2. Move left pointer when duplicate character is found.

```
#include <iostream>
#include <unordered_map>
using namespace std;

int lengthOfLongestSubstring(string s) {
    unordered_map<char, int> lastIndex;
    int maxLength = 0, start = 0;
    for (int end = 0; end < s.size(); end++) {
        if (lastIndex.count(s[end]))
            start = max(start, lastIndex[s[end]] + 1);
        lastIndex[s[end]] = end;
        maxLength = max(maxLength, end - start + 1);
    }
    return maxLength;
}

int main() {
    string s = "abcabcbb";
    cout << "Length of longest substring: " <<
lengthOfLongestSubstring(s);
    return 0;
}
```

Time Complexity: O(n)
Space Complexity: O(n)

7. Explain the sliding window technique and its use in string problems.

Sliding window is a technique to reduce nested loops by maintaining a window of elements and sliding it over the data.

Use in string problems:
- Longest substring without repeating characters
- Anagram detection
- Pattern matching
- Substrings of fixed size

8. Find the longest palindromic substring in a given string. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
1. Expand around center for each character and its pair.
2. Track longest found palindrome.

```
#include <iostream>
using namespace std;
```

```
string expandAroundCenter(string s, int l, int r) {
    while (l >= 0 && r < s.size() && s[l] == s[r]) {
        l--; r++;
    }
    return s.substr(l + 1, r - l - 1);
}

string longestPalindrome(string s) {
    string res;
    for (int i = 0; i < s.size(); i++) {
        string odd = expandAroundCenter(s, i, i);
        string even = expandAroundCenter(s, i, i + 1);
        if (odd.size() > res.size()) res = odd;
        if (even.size() > res.size()) res = even;
    }
    return res;
}

int main() {
    string s = "babad";
    cout << "Longest palindrome is " << longestPalindrome(s);
    return 0;
}
```

Time Complexity: O(n^2)
Space Complexity: O(1)

9. Find the longest common prefix among a list of strings. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
1. Assume first string is the prefix.

2. Compare character-by-character with each string and shorten prefix when mismatch occurs.

```
#include <iostream>
#include <vector>
using namespace std;

string longestCommonPrefix(vector<string>& strs) {
    if (strs.empty()) return "";
    string prefix = strs[0];
    for (int i = 1; i < strs.size(); i++) {
        while (strs[i].find(prefix) != 0)
            prefix = prefix.substr(0, prefix.size() - 1);
        if (prefix.empty()) return "";
    }
    return prefix;
}

int main() {
    vector<string> strs = {"flower", "flow", "flight"};
    cout << "Longest common prefix: " <<
longestCommonPrefix(strs);
    return 0;
}
```

Time Complexity: O(n * m)
Space Complexity: O(1)

10. Generate all permutations of a given string. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

Algorithm:
1. Use recursion with swapping.
2. At each level, swap current index with all indices >= current.

```
#include <iostream>
#include <string>
using namespace std;

void generatePermutations(string& s, int l, int r) {
    if (l == r) {
        cout << s << endl;
        return;
    }
    for (int i = l; i <= r; i++) {
        swap(s[l], s[i]);
        generatePermutations(s, l + 1, r);
        swap(s[l], s[i]); // backtrack
    }
}

int main() {
    string s = "abc";
    generatePermutations(s, 0, s.size() - 1);
    return 0;
}
```

Time Complexity: O(n!)
Space Complexity: O(n)

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.

```
#include <iostream>
#include <vector>
using namespace std;
pair<int, int> twoSum(vector<int>& nums, int target) {
    int left = 0, right = nums.size() - 1;
    while (left < right) {
        int sum = nums[left] + nums[right];
        if (sum == target) return {left, right};
        else if (sum < target) left++;
        else right--;
    }
    return {-1, -1};
}
int main() {
    vector<int> nums = {1, 2, 3, 4, 6};
    int target = 6;
    auto res = twoSum(nums, target);
    cout << res.first << " " << res.second;
    return 0;
}
```

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

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
void nextPermutation(vector<int>& nums) {
    int i = nums.size() - 2;
    while (i >= 0 && nums[i] >= nums[i + 1]) i--;
    if (i >= 0) {
        int j = nums.size() - 1;
        while (nums[j] <= nums[i]) j--;
        swap(nums[i], nums[j]);
    }
    reverse(nums.begin() + i + 1, nums.end());
}
```

```
int main() {
   vector<int> nums = {1, 2, 3};
   nextPermutation(nums);
   for (int n : nums) cout << n << " ";
   return 0;
}
```

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.

```
#include <iostream>
using namespace std;
struct ListNode {
   int val;
   ListNode* next;
   ListNode(int x) : val(x), next(NULL) {}
};
ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
   if (!l1) return l2;
   if (!l2) return l1;
   if (l1->val < l2->val) {
      l1->next = mergeTwoLists(l1->next, l2);
      return l1;
   } else {
      l2->next = mergeTwoLists(l1, l2->next);
      return l2;
   }
}
int main() {
   ListNode* a = new ListNode(1);
   a->next = new ListNode(3);
   ListNode* b = new ListNode(2);
   b->next = new ListNode(4);
   ListNode* res = mergeTwoLists(a, b);
```

```
    while (res) {
        cout << res->val << " ";
        res = res->next;
    }
    return 0;
}
```

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.

```
#include <iostream>
#include <vector>
using namespace std;
double findMedianSortedArrays(vector<int>& A, vector<int>&
B) {
    if (A.size() > B.size()) return findMedianSortedArrays(B, A);
    int x = A.size(), y = B.size();
    int low = 0, high = x;
    while (low <= high) {
        int partitionX = (low + high) / 2;
        int partitionY = (x + y + 1) / 2 - partitionX;
        int maxLeftX = (partitionX == 0) ? INT_MIN : A[partitionX -
1];
        int minRightX = (partitionX == x) ? INT_MAX : A[partitionX];
        int maxLeftY = (partitionY == 0) ? INT_MIN : B[partitionY -
1];
        int minRightY = (partitionY == y) ? INT_MAX : B[partitionY];
        if (maxLeftX <= minRightY && maxLeftY <= minRightX) {
            if ((x + y) % 2 == 0)
                return (max(maxLeftX, maxLeftY) + min(minRightX,
minRightY)) / 2.0;
            else
                return max(maxLeftX, maxLeftY);
        } else if (maxLeftX > minRightY) {
            high = partitionX - 1;
        } else {
            low = partitionX + 1;
        }
    }
    return 0.0;
}
int main() {
    vector<int> A = {1, 3};
    vector<int> B = {2};
    cout << "Median: " << findMedianSortedArrays(A, B);
    return 0;
}
```

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.

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
int kthSmallest(vector<vector<int>>& matrix, int k) {
    int n = matrix.size();
    priority_queue<int> pq;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < matrix[0].size(); j++) {
            pq.push(matrix[i][j]);
            if (pq.size() > k) pq.pop();
        }
    return pq.top();
}
int main() {
    vector<vector<int>> matrix = {{1,5,9}, {10,11,13}, {12,13,15}};
    int k = 8;
    cout << "K-th smallest: " << kthSmallest(matrix, k);
    return 0;
}
```

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.

```
#include <iostream>
#include <vector>
using namespace std;
int majorityElement(vector<int>& nums) {
    int count = 0, candidate = 0;
    for (int num : nums) {
        if (count == 0) candidate = num;
        count += (num == candidate) ? 1 : -1;
    }
    return candidate;
}
int main() {
    vector<int> nums = {2,2,1,1,1,2,2};
    cout << "Majority Element: " << majorityElement(nums);
    return 0;
}
```

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.

```
#include <iostream>
#include <vector>
using namespace std;
int trap(vector<int>& height) {
    int n = height.size(), left = 0, right = n - 1, res = 0;
    int leftMax = 0, rightMax = 0;
```

```
   while (left < right) {
      if (height[left] < height[right]) {
         height[left] >= leftMax ? (leftMax = height[left]) : res +=
(leftMax - height[left]);
         left++;
      } else {
         height[right] >= rightMax ? (rightMax = height[right]) :
res += (rightMax - height[right]);
         right--;
      }
   }
   return res;
}
int main() {
   vector<int> height = {0,1,0,2,1,0,1,3,2,1,2,1};
   cout << "Water Trapped: " << trap(height);
   return 0;
}
```

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

```
#include <iostream>
#include <vector>
using namespace std;
int findMaximumXOR(vector<int>& nums) {
   int maxXor = 0, mask = 0;
   for (int i = 31; i >= 0; i--) {
      mask |= (1 << i);
      unordered_set<int> s;
      for (int num : nums)
         s.insert(num & mask);
      int temp = maxXor | (1 << i);
      for (int prefix : s) {
            if (s.count(temp ^ prefix)) {
                maxXor = temp;
                break;
            }
        }
    }
    return maxXor;
}
int main() {
    vector<int> nums = {3,10,5,25,2,8};
    cout << "Max XOR: " << findMaximumXOR(nums);
    return 0;
}
```

19. How to find the maximum product subarray. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
#include <vector>
using namespace std;
int maxProduct(vector<int>& nums) {
    int maxProd = nums[0], minProd = nums[0], result = nums[0];
    for (int i = 1; i < nums.size(); i++) {
        if (nums[i] < 0) swap(maxProd, minProd);
        maxProd = max(nums[i], maxProd * nums[i]);
        minProd = min(nums[i], minProd * nums[i]);
        result = max(result, maxProd);
    }
    return result;
}
int main() {
    vector<int> nums = {2,3,-2,4};
    cout << "Max Product: " << maxProduct(nums);
    return 0;
}
```

20. Count all numbers with unique digits for a given number of digits. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
using namespace std;
int countNumbersWithUniqueDigits(int n) {
    if (n == 0) return 1;
    int total = 10, uniqueDigits = 9, available = 9;
    for (int i = 2; i <= n && available > 0; i++) {
        uniqueDigits *= available;
        total += uniqueDigits;
        available--;
    }
    return total;
}
int main() {
    int n = 2;
    cout << "Count: " << countNumbersWithUniqueDigits(n);
    return 0;
}
```

21. 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.

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> countBits(int n) {
    vector<int> res(n + 1, 0);
    for (int i = 1; i <= n; i++)
```

```
        res[i] = res[i >> 1] + (i & 1);
    return res;
}
int main() {
    int n = 5;
    vector<int> result = countBits(n);
    for (int i : result)
        cout << i << " ";
    return 0;
}
```

22. 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.

```
#include <iostream>
using namespace std;
bool isPowerOfTwo(int n) {
    return n > 0 && (n & (n - 1)) == 0;
}
int main() {
    int n = 16;
    cout << (isPowerOfTwo(n) ? "Yes" : "No");
    return 0;
}
```

23. 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.

```
#include <iostream>
#include <vector>
#include <unordered_set>
using namespace std;
int findMaximumXOR(vector<int>& nums) {
    int maxXor = 0, mask = 0;
    for (int i = 31; i >= 0; i--) {
        mask |= (1 << i);
        unordered_set<int> s;
        for (int num : nums)
            s.insert(num & mask);
        int candidate = maxXor | (1 << i);
        for (int prefix : s) {
            if (s.count(candidate ^ prefix)) {
                maxXor = candidate;
                break;
            }
        }
    }
    return maxXor;
}
int main() {
    vector<int> nums = {3, 10, 5, 25, 2, 8};
    cout << "Max XOR: " << findMaximumXOR(nums);
    return 0;
}
```

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

```
#include <iostream>
using namespace std;
int main() {
    int x = 5;
    int y = 3;
    cout << "AND: " << (x & y) << endl;
    cout << "OR: " << (x | y) << endl;
    cout << "XOR: " << (x ^ y) << endl;
    cout << "LEFT SHIFT x<<1: " << (x << 1) << endl;
    cout << "RIGHT SHIFT x>>1: " << (x >> 1) << endl;
```

```
   return 0;
}
```

25. 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.

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;
vector<int> nextGreaterElements(vector<int>& nums) {
   int n = nums.size();
   vector<int> res(n, -1);
   stack<int> st;
   for (int i = 0; i < 2 * n; i++) {
      while (!st.empty() && nums[st.top()] < nums[i % n]) {
         res[st.top()] = nums[i % n];
         st.pop();
      }
      if (i < n)
         st.push(i);
   }
   return res;
}
int main() {
   vector<int> nums = {1, 2, 1};
   vector<int> res = nextGreaterElements(nums);
   for (int val : res)
      cout << val << " ";
   return 0;
}
```

26. 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.

```
#include <iostream>
using namespace std;
struct ListNode {
   int val;
   ListNode* next;
   ListNode(int x) : val(x), next(NULL) {}
};
ListNode* removeNthFromEnd(ListNode* head, int n) {
   ListNode* dummy = new ListNode(0);
   dummy->next = head;
   ListNode* first = dummy;
   ListNode* second = dummy;
   for (int i = 0; i <= n; i++)
      first = first->next;
   while (first != NULL) {
      first = first->next;
      second = second->next;
   }
   second->next = second->next->next;
   return dummy->next;
}
int main() {
   ListNode* head = new ListNode(1);
   head->next = new ListNode(2);
   head->next->next = new ListNode(3);
   head = removeNthFromEnd(head, 2);
   while (head) {
      cout << head->val << " ";
      head = head->next;
   }
   return 0;
}
```

27. Find the node where two singly linked lists intersect. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
using namespace std;
struct ListNode {
   int val;
   ListNode* next;
   ListNode(int x) : val(x), next(NULL) {}
};
ListNode* getIntersectionNode(ListNode* headA, ListNode*
headB) {
   ListNode* a = headA;
   ListNode* b = headB;
   while (a != b) {
      a = a ? a->next : headB;
      b = b ? b->next : headA;
   }
   return a;
}
int main() {
   ListNode* intersect = new ListNode(8);
   intersect->next = new ListNode(10);
   ListNode* headA = new ListNode(3);
   headA->next = new ListNode(7);
   headA->next->next = intersect;
   ListNode* headB = new ListNode(99);
   headB->next = intersect;
   ListNode* res = getIntersectionNode(headA, headB);
   cout << "Intersect at: " << (res ? res->val : -1);
   return 0;
}
```

28. Implement two stacks in a single array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
using namespace std;
class TwoStacks {
    int* arr;
    int size;
    int top1, top2;
public:
    TwoStacks(int n) {
        size = n;
        arr = new int[n];
        top1 = -1;
        top2 = size;
    }
    void push1(int x) {
        if (top1 + 1 < top2)
            arr[++top1] = x;
    }
    void push2(int x) {
        if (top1 + 1 < top2)
            arr[--top2] = x;
    }
    int pop1() {
        if (top1 >= 0) return arr[top1--];
        return -1;
    }
    int pop2() {
        if (top2 < size) return arr[top2++];
        return -1;
    }
};
```

```
int main() {
    TwoStacks ts(10);
    ts.push1(5);
    ts.push2(10);
    ts.push1(15);
    ts.push2(20);
    cout << ts.pop1() << " ";
    cout << ts.pop2() << " ";
    return 0;
}
```

29. Write a program to check if an integer is a palindrome without converting it to a string. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
using namespace std;
bool isPalindrome(int x) {
    if (x < 0 || (x % 10 == 0 && x != 0)) return false;
    int rev = 0;
    while (x > rev) {
        rev = rev * 10 + x % 10;
        x /= 10;
    }
    return x == rev || x == rev / 10;
}
int main() {
    int x = 121;
    cout << (isPalindrome(x) ? "Yes" : "No");
    return 0;
}
```

30. Explain the concept of linked lists and their applications in algorithm design.

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(NULL) {}
};
void insert(Node*& head, int data) {
    Node* newNode = new Node(data);
    newNode->next = head;
    head = newNode;
}
void display(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
}
int main() {
    Node* head = NULL;
    insert(head, 10);
    insert(head, 20);
    insert(head, 30);
    display(head);
    return 0;
}
```

31. Use a deque to find the maximum in every sliding window of size K. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
#include <vector>
#include <deque>
```

```
using namespace std;
vector<int> maxSlidingWindow(vector<int>& nums, int k) {
    deque<int> dq;
    vector<int> res;
    for (int i = 0; i < nums.size(); i++) {
        if (!dq.empty() && dq.front() == i - k)
            dq.pop_front();
        while (!dq.empty() && nums[dq.back()] < nums[i])
            dq.pop_back();
        dq.push_back(i);
        if (i >= k - 1)
            res.push_back(nums[dq.front()]);
    }
    return res;
}
int main() {
    vector<int> nums = {1, 3, -1, -3, 5, 3, 6, 7};
    int k = 3;
    vector<int> result = maxSlidingWindow(nums, k);
    for (int val : result)
        cout << val << " ";
    return 0;
}
```

32. How to find the largest rectangle that can be formed in a histogram. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;
int largestRectangleArea(vector<int>& heights) {
    stack<int> st;
    heights.push_back(0);
    int maxArea = 0;
    for (int i = 0; i < heights.size(); i++) {
        while (!st.empty() && heights[i] < heights[st.top()]) {
            int height = heights[st.top()];
            st.pop();
            int width = st.empty() ? i : i - st.top() - 1;
            maxArea = max(maxArea, height * width);
        }
        st.push(i);
    }
    return maxArea;
}
int main() {
    vector<int> heights = {2, 1, 5, 6, 2, 3};
    cout << "Largest Rectangle Area: " <<
largestRectangleArea(heights);
    return 0;
}
```

33. Explain the sliding window technique and its applications in array problems.

```
#include <iostream>
#include <vector>
using namespace std;
int slidingWindowSum(vector<int>& arr, int k) {
    int sum = 0;
    for (int i = 0; i < k; i++) sum += arr[i];
    int maxSum = sum;
    for (int i = k; i < arr.size(); i++) {
        sum += arr[i] - arr[i - k];
        maxSum = max(maxSum, sum);
    }
    return maxSum;
}
```

```
int main() {
   vector<int> arr = {1, 2, 3, 4, 5};
   int k = 3;
   cout << "Max Sliding Window Sum: " <<
slidingWindowSum(arr, k);
   return 0;
}
```

34. Solve the problem of finding the subarray sum equal to K using hashing. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
#include <unordered_map>
#include <vector>
using namespace std;
int subarraySum(vector<int>& nums, int k) {
   unordered_map<int, int> prefixSum;
   prefixSum[0] = 1;
   int sum = 0, count = 0;
   for (int n : nums) {
      sum += n;
      if (prefixSum.count(sum - k)) count += prefixSum[sum - k];
      prefixSum[sum]++;
   }
   return count;
}
int main() {
   vector<int> nums = {1, 1, 1};
   int k = 2;
   cout << "Subarrays with sum " << k << ": " <<
subarraySum(nums, k);
   return 0;
}
```

35. Find the k-most frequent elements in an array using a priority queue. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <queue>
using namespace std;
vector<int> topKFrequent(vector<int>& nums, int k) {
    unordered_map<int, int> freq;
    for (int n : nums) freq[n]++;
    priority_queue<pair<int, int>> pq;
    for (auto& it : freq)
        pq.push({it.second, it.first});
    vector<int> result;
    for (int i = 0; i < k; i++) {
        result.push_back(pq.top().second);
        pq.pop();
    }
    return result;
}
int main() {
    vector<int> nums = {1, 1, 1, 2, 2, 3};
    int k = 2;
    vector<int> res = topKFrequent(nums, k);
    for (int n : res)
        cout << n << " ";
    return 0;
}
```

36. Generate all subsets of a given array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
#include <vector>
using namespace std;
vector<vector<int>> subsets(vector<int>& nums) {
    vector<vector<int>> res = {{}};
    for (int n : nums) {
        int size = res.size();
        for (int i = 0; i < size; i++) {
            vector<int> subset = res[i];
            subset.push_back(n);
            res.push_back(subset);
        }
    }
    return res;
}
int main() {
    vector<int> nums = {1, 2, 3};
    vector<vector<int>> result = subsets(nums);
    for (auto& subset : result) {
        for (int n : subset)
            cout << n << " ";
        cout << endl;
    }
    return 0;
}
```

37. Find all unique combinations of numbers that sum to a target. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
#include <vector>
using namespace std;
void backtrack(vector<int>& candidates, int target,
vector<int>& curr, vector<vector<int>>& res, int start) {
    if (target == 0) {
        res.push_back(curr);
        return;
    }
    for (int i = start; i < candidates.size(); i++) {
        if (candidates[i] > target) continue;
        curr.push_back(candidates[i]);
        backtrack(candidates, target - candidates[i], curr, res, i);
        curr.pop_back();
    }
}
vector<vector<int>> combinationSum(vector<int>& candidates,
int target) {
    vector<vector<int>> res;
    vector<int> curr;
    backtrack(candidates, target, curr, res, 0);
    return res;
}
int main() {
    vector<int> candidates = {2, 3, 6, 7};
    int target = 7;
    vector<vector<int>> result = combinationSum(candidates,
target);
    for (auto& comb : result) {
        for (int n : comb)
            cout << n << " ";
        cout << endl;
    }
    return 0;
}
```

38. Generate all permutations of a given array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
#include <vector>
using namespace std;
void permute(vector<int>& nums, int l, int r,
vector<vector<int>>& res) {
    if (l == r) {
        res.push_back(nums);
        return;
    }
    for (int i = l; i <= r; i++) {
        swap(nums[l], nums[i]);
        permute(nums, l + 1, r, res);
        swap(nums[l], nums[i]);
    }
}
int main() {
    vector<int> nums = {1, 2, 3};
    vector<vector<int>> result;
    permute(nums, 0, nums.size() - 1, result);
    for (auto& p : result) {
        for (int n : p)
            cout << n << " ";
        cout << endl;
    }
    return 0;
}
```

39. Explain the difference between subsets and permutations with examples.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
    vector<int> nums = {1, 2, 3};
    cout << "Subsets:\n";
    int n = nums.size();
    for (int i = 0; i < (1 << n); i++) {
        for (int j = 0; j < n; j++)
            if (i & (1 << j)) cout << nums[j] << " ";
        cout << endl;
    }
    cout << "\nPermutations:\n";
    sort(nums.begin(), nums.end());
    do {
        for (int num : nums)
            cout << num << " ";
        cout << endl;
    } while (next_permutation(nums.begin(), nums.end()));
    return 0;
}
```

40. Solve the problem of finding the element with maximum frequency in an array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
#include <unordered_map>
#include <vector>
using namespace std;
int maxFrequencyElement(vector<int>& nums) {
    unordered_map<int, int> freq;
    int maxFreq = 0, element = -1;
    for (int n : nums) {
        freq[n]++;
        if (freq[n] > maxFreq) {
            maxFreq = freq[n];
            element = n;
        }
    }
    return element;
}
int main() {
    vector<int> nums = {1, 3, 2, 3, 4, 3, 5};
    cout << "Element with max frequency: " <<
maxFrequencyElement(nums);
    return 0;
}
```

41. Write a program to find the maximum subarray sum using Kadane’s algorithm.

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

int maxSubArraySum(vector<int>& nums) {
    int max_so_far = INT_MIN, max_ending_here = 0;

    for (int num : nums) {
        max_ending_here += num;
        max_so_far = max(max_so_far, max_ending_here);
        if (max_ending_here < 0)
            max_ending_here = 0;
    }
    return max_so_far;
}

int main() {
    vector<int> arr = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
    cout << "Maximum subarray sum is " <<
maxSubArraySum(arr);
    return 0;
}
```

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

42. Explain the concept of dynamic programming and its use in solving the maximum subarray problem.

Dynamic Programming (DP) is a method for solving complex problems by breaking them into simpler subproblems and storing the results to avoid redundant computations.

In Maximum Subarray Problem:
- Use a DP array `dp[i]` representing the maximum subarray sum ending at index `i`.
- `dp[i] = max(arr[i], arr[i] + dp[i-1])`
- The result is the maximum value in `dp`.

Optimized Version (Kadane's):
- It uses two variables instead of a DP array, maintaining O(1) space.

43. Solve the problem of finding the top K frequent elements in an array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <queue>
using namespace std;

vector<int> topKFrequent(vector<int>& nums, int k) {
   unordered_map<int, int> freq;
   for (int num : nums) freq[num]++;
```

```
    priority_queue<pair<int, int>, vector<pair<int, int>>,
greater<>> minHeap;

    for (auto& [num, count] : freq) {
        minHeap.push({count, num});
        if (minHeap.size() > k) minHeap.pop();
    }

    vector<int> result;
    while (!minHeap.empty()) {
        result.push_back(minHeap.top().second);
        minHeap.pop();
    }
    return result;
}

int main() {
    vector<int> nums = {1, 1, 1, 2, 2, 3};
    int k = 2;
    vector<int> res = topKFrequent(nums, k);
    for (int num : res) cout << num << " ";
    return 0;
}
```

Time Complexity: O(n log k)
Space Complexity: O(n)

44. How to find two numbers in an array that add up to a target using hashing. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
#include <unordered_map>
#include <vector>
```

```
using namespace std;

vector<int> twoSum(vector<int>& nums, int target) {
    unordered_map<int, int> m;
    for (int i = 0; i < nums.size(); i++) {
        int comp = target - nums[i];
        if (m.count(comp))
            return {m[comp], i};
        m[nums[i]] = i;
    }
    return {};
}

int main() {
    vector<int> nums = {2, 7, 11, 15};
    int target = 9;
    vector<int> res = twoSum(nums, target);
    cout << res[0] << " " << res[1];
    return 0;
}
```

Time Complexity: O(n)
Space Complexity: O(n)

45. Explain the concept of priority queues and their applications in algorithm design.

A priority queue is an abstract data type where each element has a "priority" assigned to it, and elements are served based on their priority.

Applications:
- Dijkstra's algorithm (shortest path)
- Huffman encoding
- Task scheduling

- Top-K problems (like Top-K frequent elements)

In C++, `priority_queue` uses a max-heap by default but can be converted to a min-heap using a custom comparator.

46. Write a program to find the longest palindromic substring in a given string. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
using namespace std;

string expand(string s, int left, int right) {
   while (left >= 0 && right < s.size() && s[left] == s[right]) {
      left--; right++;
   }
   return s.substr(left + 1, right - left - 1);
}

string longestPalindrome(string s) {
   string res = "";
   for (int i = 0; i < s.size(); i++) {
      string odd = expand(s, i, i);
      string even = expand(s, i, i + 1);
      if (odd.size() > res.size()) res = odd;
      if (even.size() > res.size()) res = even;
   }
   return res;
}

int main() {
   string s = "babad";
   cout << longestPalindrome(s);
   return 0;
}
```

Time Complexity: O(n²)
Space Complexity: O(1)

47. Explain the concept of histogram problems and their applications in algorithm design.

Histogram problems involve bars of varying heights, and many algorithms aim to compute the area, visibility, or stacking properties.

Applications:
- Finding largest rectangle in a histogram (stack-based)
- Rain water trapping
- Skyline problem

Histogram logic is widely used in graphics, image processing, and data analysis.

48. Solve the problem of finding the next permutation of a given array. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

void nextPermutation(vector<int>& nums) {
   int i = nums.size() - 2;
   while (i >= 0 && nums[i] >= nums[i + 1]) i--;

   if (i >= 0) {
      int j = nums.size() - 1;
      while (nums[j] <= nums[i]) j--;
```

```
        swap(nums[i], nums[j]);
    }
    reverse(nums.begin() + i + 1, nums.end());
}

int main() {
    vector<int> nums = {1, 2, 3};
    nextPermutation(nums);
    for (int num : nums) cout << num << " ";
    return 0;
}
```

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

49. How to find the intersection of two linked lists. Write its algorithm, program. Find its time and space complexities. Explain with suitable example.

```
#include <iostream>
using namespace std;

struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(NULL) {}
};

ListNode* getIntersectionNode(ListNode *a, ListNode *b) {
    ListNode *p1 = a, *p2 = b;
    while (p1 != p2) {
        p1 = p1 ? p1->next : b;
        p2 = p2 ? p2->next : a;
    }
    return p1;
}
```

Time Complexity: O(m + n)
Space Complexity: O(1)

50. Explain the concept of equilibrium index and its applications in array problems.

An equilibrium index in an array is an index such that the sum of elements to its left is equal to the sum of elements to its right.

Application:
- Used in balancing computations
- Efficient for analytics and decision making in stock/finance problems

Example:
For arr = {-7, 1, 5, 2, -4, 3, 0}, index 3 is an equilibrium index:
(-7 + 1 + 5) == (-4 + 3 + 0) = -1