GALGOTIAS UNIVERSITY NAAC GRADE A+ Accredited University

Advanced Algorithmic Problem Solving (R1UC601B)

Assignment for MTE

NAME: - ANMOL SHARMA

SEM: - 6

YEAR: - 3

ADDMISION NO.: - 22SCSE1012171

ENROLLMENT NO.: - 22131012052

SUBMITTED TO: - MR. ADITYA TRIVEDI

Advanced Algorithmic Problem Solving (R1UC601B)

Assignment for MTE

1. Explain the concept of a prefix sum array and its applications.
A prefix sum array stores cumulative sums of elements from the start. It allows constant-time range sum queries.
Applications: Range sum queries, subarray problems, frequency count.

2. Write a program to find the sum of elements in a given range [L, R] using a prefix sum array.

Algorithm:

- Build prefix sum array.
- Range sum = prefix[R] - prefix[L-1]

Program:

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

int main() {
    int arr[] = {2, 4, 1, 3, 6}, n = 5;
    int prefix[n];
    prefix[0] = arr[0];
    for (int i = 1; i < n; i++) prefix[i] = prefix[i-1] + arr[i];
    int L = 1, R = 3;
    int rangeSum = prefix[R] - (L > 0 ? prefix[L-1] : 0);
    cout << "Sum = " << rangeSum;
}
```

Time: O(n) pre-processing, O(1) query
Space: O(n)
Example: Range [1,3] → 4+1+3 = 8

3. Solve the problem of finding the equilibrium index in an array.

Algorithm:

- Total sum of array
- Iterate and check if leftSum == total - leftSum - arr[i]

Program:

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

```
int main() {
    int arr[] = {1, 3, 5, 2, 2}, n = 5;
    int total = 0, leftSum = 0;
    for (int i = 0; i < n; i++) total += arr[i];
    for (int i = 0; i < n; i++) {
        if (leftSum == total - leftSum - arr[i]) {
            cout << "Equilibrium Index: " << i;
            break;
        }
        leftSum += arr[i];
    }
}
```

Time: O(n)
Space: O(1)
Example: Index 2 → 1+3 == 2+2

4. Check if an array can be split into two parts such that the sum of prefix equals suffix.

Algorithm:

- Calculate total sum
- Traverse and track prefix sum
- Check if prefix sum == total - prefix

Program:

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

int main() {
    int arr[] = {1, 2, 3, 3}, n = 4;
    int total = 0, preSum = 0;
    for (int i = 0; i < n; i++) total += arr[i];
    for (int i = 0; i < n; i++) {
        preSum += arr[i];
        if (preSum == total - preSum) {
            cout << "Can be split at index " << i;
            break;
        }
    }
}
```

Time: O(n)
Space: O(1)
Example: Split at index 2 → 1+2 = 3

---

5. Find the maximum sum of any subarray of size K in a given array.

Algorithm:

- Use sliding window of size k
- Update window sum by adding next and removing previous

Program:

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

int main() {
    int arr[] = {1, 4, 2, 10, 2, 3}, n = 6, k = 3;
    int sum = 0, maxSum = 0;
    for (int i = 0; i < k; i++) sum += arr[i];
    maxSum = sum;
    for (int i = k; i < n; i++) {
        sum += arr[i] - arr[i-k];
        maxSum = max(maxSum, sum);
    }
    cout << "Max sum = " << maxSum;
}
```

Time: O(n)
Space: O(1)
Example: Max sum of size 3 → 10+2+3 = 15

6. Find the length of the longest substring without repeating characters.

Algorithm:

- Use sliding window with a set to track characters

Program:

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

int main() {
    string s = "abcabcbb";
    unordered_set<char> seen;
    int left = 0, maxLen = 0;
    for (int right = 0; right < s.size(); right++) {
        while (seen.count(s[right])) seen.erase(s[left++]);
        seen.insert(s[right]);
        maxLen = max(maxLen, right - left + 1);
    }
    cout << "Max Length = " << maxLen;
}
```

Time: O(n)
Space: O(n)
Example: "abc" → length 3

7. Explain the sliding window technique and its use in string problems.
Sliding window is a method for reducing time complexity by maintaining a subrange (window) that slides over data.
Use: Optimal for substring, subarray problems like longest unique substring, max sum of size K, etc.

8. Find the longest palindromic substring in a given string.

Algorithm:

- Expand around center

Program:

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

string expand(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);
}

int main() {
   string s = "babad", 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;
   }
   cout << "Longest Palindrome = " << res;
}
```

Time: O(n^2)
Space: O(1)
Example: "aba" from "babad"

9. Find the longest common prefix among a list of strings.

Algorithm:

- Compare characters of all strings

Program:

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

int main() {
   vector<string> strs = {"flower", "flow", "flight"};
```

```
    string res = "";
    for (int i = 0; i < strs[0].size(); i++) {
        char c = strs[0][i];
        for (auto s : strs)
            if (i >= s.size() || s[i] != c) {
                cout << "LCP = " << res;
                return 0;
            }
        res += c;
    }
    cout << "LCP = " << res;
}
```

Time: O(n * m)
Space: O(1)
Example: "fl"

10. Generate all permutations of a given string.

Algorithm:

- Backtracking with swap

Program:

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

void permute(string s, int l, int r) {
    if (l == r) cout << s << " ";
    else {
        for (int i = l; i <= r; i++) {
            swap(s[l], s[i]);
            permute(s, l+1, r);
            swap(s[l], s[i]);
        }
    }
}

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

Time: O(n * n!)
Space: O(n)
Example: "abc" → abc, acb, bac, bca, cab, cba

11. Find two numbers in a sorted array that add up to a target.

Algorithm:
Use two pointers from start and end.

Program:

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

int main() {
    int arr[] = {1, 2, 4, 6, 10}, n = 5, target = 8;
    int l = 0, r = n - 1;
    while (l < r) {
        int sum = arr[l] + arr[r];
        if (sum == target) {
            cout << arr[l] << " " << arr[r];
            break;
        } else if (sum < target) l++;
        else r--;
    }
}
```

Time: O(n)
Space: O(1)
Example: 2 + 6 = 8

12. Rearrange numbers into the lexicographically next greater permutation.

Algorithm:
Find pivot, swap with successor, reverse suffix.

Program:

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

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

Time: O(n)
Space: O(1)
Example: 1 2 3 → 1 3 2

13. How to merge two sorted linked lists into one sorted list.

Algorithm:
Use dummy node and compare nodes one by one.

Program:

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

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

Node* merge(Node* l1, Node* l2) {
    Node dummy(0), *tail = &dummy;
    while (l1 && l2) {
        if (l1->val < l2->val) tail->next = l1, l1 = l1->next;
        else tail->next = l2, l2 = l2->next;
        tail = tail->next;
    }
    tail->next = l1 ? l1 : l2;
    return dummy.next;
}
```

Time: O(n + m)
Space: O(1)
Example: [1,3] + [2,4] → [1,2,3,4]

14. Find the median of two sorted arrays using binary search.

Algorithm:
Binary search on smaller array to find partition.

Program:

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

double findMedian(vector<int>& A, vector<int>& B) {
    if (A.size() > B.size()) return findMedian(B, A);
    int m = A.size(), n = B.size();
    int l = 0, r = m;
    while (l <= r) {
        int i = (l + r) / 2;
        int j = (m + n + 1) / 2 - i;
        int Aleft = (i == 0) ? INT_MIN : A[i - 1];
        int Aright = (i == m) ? INT_MAX : A[i];
        int Bleft = (j == 0) ? INT_MIN : B[j - 1];
        int Bright = (j == n) ? INT_MAX : B[j];
        if (Aleft <= Bright && Bleft <= Aright) {
            if ((m + n) % 2 == 0) return (max(Aleft, Bleft) + min(Aright, Bright)) / 2.0;
            else return max(Aleft, Bleft);
        } else if (Aleft > Bright) r = i - 1;
        else l = i + 1;
    }
    return 0.0;
}

int main() {
    vector<int> a = {1, 3}, b = {2};
    cout << findMedian(a, b);
}
```

Time: O(log(min(m,n)))
Space: O(1)
Example: [1,3] + [2] → median = 2

15. Find the k-th smallest element in a sorted matrix.

Algorithm:
Use min-heap or binary search on value range.

Program:

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

int countLE(vector<vector<int>>& mat, int mid) {
    int count = 0, n = mat.size(), j = n - 1;
    for (int i = 0; i < n; i++) {
        while (j >= 0 && mat[i][j] > mid) j--;
        count += (j + 1);
    }
    return count;
}

int kthSmallest(vector<vector<int>>& mat, int k) {
    int n = mat.size(), l = mat[0][0], r = mat[n-1][n-1];
    while (l < r) {
        int mid = (l + r) / 2;
        if (countLE(mat, mid) < k) l = mid + 1;
        else r = mid;
    }
    return l;
}

int main() {
    vector<vector<int>> mat = {{1, 5, 9}, {10, 11, 13}, {12, 13, 15}};
    cout << kthSmallest(mat, 8);
}
```

Time: O(n log(max-min))
Space: O(1)
Example: 8th smallest → 13

16. Find the majority element in an array that appears more than n/2 times.

Algorithm:
Use Boyer-Moore Voting Algorithm

Program:

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

int main() {
    int arr[] = {2, 2, 1, 1, 2}, n = 5, count = 0, candidate;
    for (int i = 0; i < n; i++) {
        if (count == 0) candidate = arr[i];
        count += (arr[i] == candidate) ? 1 : -1;
    }
    cout << "Majority = " << candidate;
}
```

Time: O(n)
Space: O(1)
Example: 2 appears 3 times

17. Calculate how much water can be trapped between the bars of a histogram.

Algorithm:
Use two-pointer approach tracking leftMax and rightMax.

Program:

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

int main() {
    int arr[] = {0, 1, 0, 2, 1, 0, 1, 3}, n = 8;
    int l = 0, r = n - 1, leftMax = 0, rightMax = 0, water = 0;
    while (l < r) {
        if (arr[l] < arr[r]) {
            leftMax = max(leftMax, arr[l]);
            water += leftMax - arr[l++];
        } else {
            rightMax = max(rightMax, arr[r]);
            water += rightMax - arr[r--];
        }
    }
    cout << "Water = " << water;
}
```

Time: O(n)
Space: O(1)
Example: Water trapped = 6

18. Find the maximum XOR of two numbers in an array.

Algorithm:
Use trie or greedy method with set.

Program:

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

int main() {
    vector<int> nums = {3, 10, 5, 25, 2, 8}, 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;
            }
    }
    cout << "Max XOR = " << maxXor;
}
```

Time: O(n * 32)
Space: O(n)
Example: Max XOR = 28

19. How to find the maximum product subarray.

Algorithm:
Track maxProd and minProd due to sign flip.

Program:

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

int main() {
    int arr[] = {2, 3, -2, 4}, n = 4;
    int maxP = arr[0], minP = arr[0], res = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] < 0) swap(maxP, minP);
        maxP = max(arr[i], maxP * arr[i]);
        minP = min(arr[i], minP * arr[i]);
        res = max(res, maxP);
    }
    cout << "Max Product = " << res;
}
```

Time: O(n)
Space: O(1)
Example: 2×3 = 6

20. Count all numbers with unique digits for a given number of digits.

Algorithm:
Use combinatorics: 9 × 9 × 8 × ...

Program:

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

int countNumbers(int n) {
    if (n == 0) return 1;
    int res = 10, prod = 9, available = 9;
    for (int i = 2; i <= n && available; i++) {
        prod *= available--;
        res += prod;
    }
    return res;
}

int main() {
    cout << countNumbers(2);
}
```

Time: O(n)
Space: O(1)
Example: n = 2 → 91

21. How to count the number of 1s in the binary representation of numbers from 0 to n.

Algorithm:

- Use dynamic programming where bits[i] = bits[i >> 1] + (i & 1).

Program:

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

vector<int> countBits(int n) {
    vector<int> bits(n + 1);
    for (int i = 1; i <= n; ++i)
        bits[i] = bits[i >> 1] + (i & 1);
    return bits;
}

int main() {
    int n = 5;
    vector<int> result = countBits(n);
    for (int i = 0; i <= n; ++i)
        cout << "Number of 1s in " << i << " = " << result[i] << endl;
}
```

□

Time Complexity: O(n)
Space Complexity: O(n)
Example: For n = 5, output is [0, 1, 1, 2, 1, 2].□

22. How to check if a number is a power of two using bit manipulation.

Algorithm:

- A number is a power of two if n > 0 and n & (n - 1) == 0.□

Program:

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

bool isPowerOfTwo(int n) {
    return n > 0 && (n & (n - 1)) == 0;
}

int main() {
    int num = 16;
    cout << num << " is power of two: " << boolalpha << isPowerOfTwo(num) << endl;
}
```

□

Time Complexity: O(1)
Space Complexity: O(1)
Example: 16 is a power of two.□

23. How to find the maximum XOR of two numbers in an array.

Algorithm:

- Iterate over each pair and calculate XOR to find the maximum.□

Program:

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

int findMaximumXOR(vector<int>& nums) {
    int maxXOR = 0;
    for (int i = 0; i < nums.size(); ++i)
        for (int j = i + 1; j < nums.size(); ++j)
            maxXOR = max(maxXOR, nums[i] ^ nums[j]);
    return maxXOR;
}

int main() {
    vector<int> nums = {3, 10, 5, 25, 2, 8};
    cout << "Maximum XOR: " << findMaximumXOR(nums) << endl;
}
```

□

Time Complexity: O(n^2)
Space Complexity: O(1)
Example: Maximum XOR is 28.□

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

Explanation:

- Bit manipulation involves direct operations on bits using operators like AND, OR, XOR, NOT, shifts, etc.□
- **Advantages:**
  - Efficient computation.□
  - Memory optimization.□
  - Useful in low-level programming.□

25. Solve the problem of finding the next greater element for each element in an array.

Algorithm:

- Use a stack to keep track of elements for which the next greater element hasn't been found.□

Program:

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

vector<int> nextGreaterElements(vector<int>& nums) {
    vector<int> nge(nums.size(), -1);
    stack<int> s;
    for (int i = 0; i < nums.size(); ++i) {
        while (!s.empty() && nums[s.top()] < nums[i]) {
            nge[s.top()] = nums[i];
            s.pop();
        }
        s.push(i);
    }
    return nge;
}

int main() {
    vector<int> nums = {2, 1, 2, 4, 3};
    vector<int> result = nextGreaterElements(nums);
    for (int i = 0; i < nums.size(); ++i)
        cout << "Next greater for " << nums[i] << " is " << result[i] << endl;
}
```

□

Time Complexity: O(n)
Space Complexity: O(n)
Example: For [2, 1, 2, 4, 3], output is [4, 2, 4, -1, -1].□

26. Remove the n-th node from the end of a singly linked list.

Algorithm:

- Use two pointers; move one ahead by n steps, then move both until the first reaches the end.□

Program:

```
#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(0);
    dummy.next = head;
    ListNode* first = &dummy;
    ListNode* second = &dummy;
    for (int i = 0; i <= n; ++i)
        first = first->next;
    while (first) {
        first = first->next;
        second = second->next;
    }
    second->next = second->next->next;
    return dummy.next;
}
```

□

Time Complexity: O(L)
Space Complexity: O(1)
Example: Removing 2nd from end in [1,2,3,4,5] results in [1,2,3,5].□

27. Find the node where two singly linked lists intersect.

Algorithm:

- Traverse both lists to determine their lengths.□
- Align the start of the longer list with the shorter by advancing the pointer of the longer list by the difference in lengths.□
- Traverse both lists simultaneously until a common node is found or the end is reached.□

Program:

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

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

int getLength(ListNode* head) {
    int length = 0;
    while (head) {
        ++length;
        head = head->next;
    }
    return length;
}

ListNode* getIntersectionNode(ListNode* headA, ListNode* headB) {
    int lenA = getLength(headA);
    int lenB = getLength(headB);
    while (lenA > lenB) {
        headA = headA->next;
        --lenA;
    }
    while (lenB > lenA) {
        headB = headB->next;
        --lenB;
    }
    while (headA && headB) {
        if (headA == headB)
            return headA;
        headA = headA->next;
        headB = headB->next;
    }
    return NULL;
}
```

□

Time Complexity: O(m + n)□ **Space Complexity:** O(1)□ **Example:** For lists A: [1, 2, 3, 4, 5] and B: [9, 4, 5], the intersection is at node with value 4.□

28. Implement two stacks in a single array.

Algorithm:

- Use a single array with two pointers: one starting from the beginning (top1) for the first stack and one from the end (top2) for the second stack.□
- Ensure that top1 and top2 do not cross each other to prevent overflow.□

Program:

```
#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 < top2 - 1) {
            arr[++top1] = x;
        } else {
            cout << "Stack Overflow\n";
        }
    }
    void push2(int x) {
        if (top1 < top2 - 1) {
            arr[--top2] = x;
        } else {
            cout << "Stack Overflow\n";
        }
    }
    int pop1() {
        if (top1 >= 0) {
            return arr[top1--];
        } else {
            cout << "Stack Underflow\n";
            return -1;
        }
    }
    int pop2() {
        if (top2 < size) {
            return arr[top2++];
        } else {
            cout << "Stack Underflow\n";
            return -1;
        }
    }
};
```

□

Time Complexity: O(1) for push and pop operations.□ **Space Complexity:** O(n)□ **Example:** Pushing elements 1, 2, 3 to stack1 and 9, 8, 7 to stack2 in an array of size 6.□

29. Write a program to check if an integer is a palindrome without converting it to a string.

Algorithm:

- Handle negative numbers by returning false.□
- Reverse the second half of the number and compare it with the first half.□

Program:

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

bool 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;
}

int main() {
    int num = 121;
    cout << num << " is palindrome: " << boolalpha << isPalindrome(num) << endl;
}
```

□

Time Complexity: $O(\log_{10}(n))$□ **Space Complexity:** O(1)□ **Example:** 121 is a palindrome.□

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

Explanation:

- A linked list is a linear data structure where each element (node) contains a data part and a reference (or link) to the next node in the sequence.□
- **Applications:**
  - Dynamic memory allocation.□
  - Implementation of stacks, queues, and other abstract data types.□
  - Efficient insertions and deletions.□

31. Find the longest consecutive sequence in an unsorted array.

```
#include <unordered_set>

int longestConsecutive(vector<int>& nums) {
    unordered_set<int> s(nums.begin(), nums.end());
    int longest = 0;
    for (int num : nums) {
        if (!s.count(num - 1)) {
            int length = 0;
            while (s.count(num++)) length++;
            longest = max(longest, length);
        }
    }
    return longest;
}
```

Time: O(n), Space: O(n)

Example: nums = [100, 4, 200, 1, 3, 2] => Output: 4 (sequence: 1,2,3,4)

32. Count the number of islands in a 2D grid.

```
void dfs(vector<vector<char>>& grid, int i, int j) {
    if (i < 0 || i >= grid.size() || j < 0 || j >= grid[0].size() || grid[i][j] == '0') return;
    grid[i][j] = '0';
    dfs(grid, i+1, j); dfs(grid, i-1, j); dfs(grid, i, j+1); dfs(grid, i, j-1);
}
```

```
int numIslands(vector<vector<char>>& grid) {
    int count = 0;
    for (int i = 0; i < grid.size(); i++) {
        for (int j = 0; j < grid[0].size(); j++) {
            if (grid[i][j] == '1') {
                dfs(grid, i, j);
                count++;
            }
        }
    }
    return count;
}
```

Time: O(m*n), Space: O(m*n)

Example: grid = {{'1','1','0','0','0'},{'1','1','0','0','0'},{'0','0','1','0','0'},{'0','0','0','1','1'}} => Output: 3

33. Rotate an n x n matrix 90 degrees clockwise in-place.

```
void rotate(vector<vector<int>>& matrix) {
    int n = matrix.size();
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            swap(matrix[i][j], matrix[j][i]);
        }
    }
    for (int i = 0; i < n; i++) {
        reverse(matrix[i].begin(), matrix[i].end());
    }
}
```

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

Example: Input = [[1,2,3],[4,5,6],[7,8,9]] => Output = [[7,4,1],[8,5,2],[9,6,3]]

34. Implement a LRU (Least Recently Used) Cache.

```
#include <list>
#include <unordered_map>

class LRUCache {
    int cap;
    list<pair<int, int>> lru;
    unordered_map<int, list<pair<int, int>>::iterator> m;

public:
    LRUCache(int capacity) { cap = capacity; }

    int get(int key) {
        if (m.find(key) == m.end()) return -1;
        lru.splice(lru.begin(), lru, m[key]);
        return m[key]->second;
    }

    void put(int key, int value) {
        if (m.find(key) != m.end()) {
            lru.splice(lru.begin(), lru, m[key]);
            m[key]->second = value;
        } else {
            if (lru.size() == cap) {
                m.erase(lru.back().first);
                lru.pop_back();
            }
            lru.emplace_front(key, value);
            m[key] = lru.begin();
        }
    }
};
```

Time: O(1), Space: O(capacity)

Example: put(1,1), put(2,2), get(1), put(3,3), get(2) => Output: 1, -1

35. Find the minimum window substring containing all characters of another string.

```
#include <climits>

string minWindow(string s, string t) {
    vector<int> freq(128, 0);
    for (char c : t) freq[c]++;
    int left = 0, minLen = INT_MAX, minStart = 0, count = t.size();
    for (int right = 0; right < s.size(); right++) {
        if (freq[s[right]]-- > 0) count--;
        while (count == 0) {
            if (right - left + 1 < minLen) {
                minLen = right - left + 1;
                minStart = left;
            }
            if (++freq[s[left++]] > 0) count++;
        }
    }
    return minLen == INT_MAX ? "" : s.substr(minStart, minLen);
}
```

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

Example: s = "ADOBECODEBANC", t = "ABC" => Output: "BANC"

36. Merge two sorted arrays without using extra space.

```
void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
    int i = m - 1, j = n - 1, k = m + n - 1;
    while (i >= 0 && j >= 0) {
        nums1[k--] = nums1[i] > nums2[j] ? nums1[i--] : nums2[j--];
    }
    while (j >= 0) nums1[k--] = nums2[j--];
}
```

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

Example: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3 => Output: [1,2,2,3,5,6]

37. Implement binary search recursively.

```
int binarySearch(vector<int>& arr, int low, int high, int target) {
    if (low > high) return -1;
    int mid = low + (high - low) / 2;
    if (arr[mid] == target) return mid;
    else if (arr[mid] > target) return binarySearch(arr, low, mid - 1, target);
    else return binarySearch(arr, mid + 1, high, target);
}
```

Time: O(log n), Space: O(log n) (due to recursion)

Example: arr = [1,2,3,4,5], target = 3 => Output: 2

38. Implement BFS traversal for a graph.

```
#include <queue>

vector<int> bfsOfGraph(int V, vector<int> adj[]) {
    vector<int> visited(V, 0), res;
    queue<int> q;
    q.push(0);
    visited[0] = 1;
    while (!q.empty()) {
        int node = q.front(); q.pop();
        res.push_back(node);
        for (int neighbor : adj[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = 1;
                q.push(neighbor);
            }
        }
    }
    return res;
}
```

Time: O(V + E), Space: O(V)

Example: V = 5, edges = {0-1, 0-2, 1-3, 1-4} => Output: [0,1,2,3,4]

39. Implement DFS traversal for a graph.

```
void dfs(int node, vector<int> adj[], vector<int>& visited, vector<int>& res) {
    visited[node] = 1;
    res.push_back(node);
    for (int neighbor : adj[node]) {
        if (!visited[neighbor]) dfs(neighbor, adj, visited, res);
    }
}

vector<int> dfsOfGraph(int V, vector<int> adj[]) {
    vector<int> visited(V, 0), res;
    dfs(0, adj, visited, res);
    return res;
}
```

Time: O(V + E), Space: O(V)

Example: V = 5, edges = {0-1, 0-2, 1-3, 1-4} => Output: [0,1,3,4,2]

40. Find the shortest path in an unweighted graph using BFS.

```
vector<int> shortestPath(int V, vector<int> adj[], int src) {
    vector<int> dist(V, -1);
    queue<int> q;
    q.push(src);
    dist[src] = 0;
    while (!q.empty()) {
        int node = q.front(); q.pop();
        for (int neighbor : adj[node]) {
            if (dist[neighbor] == -1) {
                dist[neighbor] = dist[node] + 1;
                q.push(neighbor);
            }
        }
    }
    return dist;
}
```

Time: O(V + E), Space: O(V)

Example: V = 5, edges = {0-1, 0-2, 1-3, 1-4}, src = 0 => Output: [0,1,1,2,2]

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

Algorithm:

1. Initialize max_so_far = arr[0] and current_max = arr[0].
2. Traverse the array from index 1:
   - current_max = max(arr[i], current_max + arr[i])
   - max_so_far = max(max_so_far, current_max)
3. Return max_so_far.

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

```
int maxSubArraySum(vector<int>& nums) {
    int current_max = nums[0], max_so_far = nums[0];
    for (int i = 1; i < nums.size(); i++) {
        current_max = max(nums[i], current_max + nums[i]);
        max_so_far = max(max_so_far, current_max);
    }
    return max_so_far;
}
```

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

Example:

Input: [−2,1,−3,4,−1,2,1,−5,4]
Output: 6 → Subarray [4,−1,2,1]

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

Answer:

Dynamic Programming (DP) is a method to solve problems by breaking them into smaller overlapping subproblems and storing their results.

In **Kadane's Algorithm**, DP helps track:

- The **maximum sum ending at index i**.
- Using the recurrence: dp[i] = max(arr[i], dp[i-1] + arr[i])

This avoids recalculating subarray sums, making it efficient (O(n) time).

43. Solve the problem of finding the top K frequent elements in an array.

Algorithm:

1. Count frequency using a hash map.
2. Use a min-heap (priority queue) to store top K elements.
3. Pop and return the top K frequent.

```
#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;
}
```

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

Example:

Input: [1,1,1,2,2,3], k = 2
Output: [1,2]

44. How to find two numbers in an array that add up to a target using hashing.

Algorithm:

1. Initialize an empty hash map.
2. For each element, check if target - current exists in map.
3. If yes, return the pair. Else insert current element.

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

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

Time: O(n) | Space: O(n)

Example:

Input: [2, 7, 11, 15], target = 9
Output: [0,1] → 2 + 7 = 9

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

Answer:

Priority Queue is an abstract data structure where each element has a priority. Elements with higher priority are served before others.

C++ STL: priority_queue<int> pq; (max-heap by default)

Applications:

- Dijkstra's algorithm
- Huffman coding
- A* search
- Task scheduling

46. Write a program to find the longest palindromic substring in a given string.

Algorithm (Expand Around Center):

- For each character, expand around center (odd and even lengths).
- Track the longest palindrome.

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

string longestPalindrome(string s) {
    int start = 0, maxLen = 0;
    for (int i = 0; i < s.length(); i++) {
        int l = i, r = i;
        while (l >= 0 && r < s.length() && s[l] == s[r]) {
            if (r - l + 1 > maxLen) {
                maxLen = r - l + 1;
                start = l;
            }
            l--; r++;
        }
        l = i; r = i+1;
        while (l >= 0 && r < s.length() && s[l] == s[r]) {
            if (r - l + 1 > maxLen) {
                maxLen = r - l + 1;
                start = l;
            }
            l--; r++;
        }
    }
    return s.substr(start, maxLen);
}
```

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

Example:

Input: "babad" → Output: "bab" or "aba"

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

Answer:

Histogram problem: Given heights of bars, find the largest rectangle that fits under the histogram.

Use a **monotonic stack** to track increasing bar indices.

Applications:

- Max area in binary matrix
- Skyline silhouette
- Text editor layout

48. Solve the problem of finding the next permutation of a given array.

Algorithm:

1. Find first i where nums[i] < nums[i+1] from right.
2. Find j > i such that nums[j] > nums[i].
3. Swap nums[i] and nums[j].
4. Reverse nums[i+1:].

```
#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());
}
```

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

Example:

Input: [1,2,3] → Output: [1,3,2]

49. How to find the intersection of two linked lists.

Algorithm:

1. Use two pointers, a and b on lists A and B.
2. When a pointer reaches the end, move to the start of the other list.
3. They meet at the intersection node or nullptr.

```
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;
}
```

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

Example:

Lists intersect at node with value 8.

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

Answer:

An **equilibrium index** is a position i such that:

sum(arr[0..i-1]) == sum(arr[i+1..n-1])

Applications:

- Balance point problems
- Load distribution
- Prefix-suffix analysis

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

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

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