SAARTHILMS

C++

SAARTHILMS

1. Two Sum

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

SAARTHILMS

2. Add Two Numbers

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode dummy(0);
        ListNode* curr = &dummy;
        int carry = 0;

        while(l1 || l2 || carry) {
            int sum = carry;
            if(l1) sum += l1->val, l1 = l1->next;
            if(l2) sum += l2->val, l2 = l2->next;
            carry = sum / 10;
            curr->next = new ListNode(sum % 10);
            curr = curr->next;
        }
        return dummy.next;
    }
};
```

SAARTHILMS

7. Reverse Integer

```
class Solution {
public:
    int reverse(int x) {
        long rev = 0;
        while(x) {
            rev = rev * 10 + x % 10;
            x /= 10;
        }
        return (rev > INT_MAX || rev < INT_MIN) ? 0 : rev;
    }
};
```

SAARTHILMS

9. Palindrome Number

```
class Solution {
public:
    bool isPalindrome(int x) {
        if(x < 0) return false;
        long rev = 0, temp = x;
        while(temp) {
            rev = rev * 10 + temp % 10;
            temp /= 10;
        }
        return rev == x;
    }
};
```

SAARTHILMS

14. Longest Common Prefix

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        if(strs.empty()) return "";
        string pref = strs[0];
        for(int i = 1; i < strs.size(); i++) {
            while(strs[i].find(pref) != 0)
                pref.pop_back();
        }
        return pref;
    }
};
```

SAARTHILMS

20. Valid Parentheses

```
class Solution {
public:
    bool isValid(string s) {
        stack<char> st;
        for(char c : s) {
            if(c=='('||c=='{'||c=='[') st.push(c);
            else {
                if(st.empty()) return false;
                if((c==')'&&st.top()!='(') ||
                   (c=='}'&&st.top()!='{') ||
                   (c==']'&&st.top()!='['))
                    return false;
                st.pop();
            }
        }
        return st.empty();
    }
};
```

SAARTHILMS

21. Merge Two Sorted Lists

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        ListNode dummy(0);
        ListNode* curr = &dummy;
        while(l1 && l2) {
            if(l1->val < l2->val) curr->next = l1, l1 = l1->next;
            else curr->next = l2, l2 = l2->next;
            curr = curr->next;
        }
        curr->next = l1 ? l1 : l2;
        return dummy.next;
    }
};
```

SAARTHILMS

26. Remove Duplicates from Sorted Array

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

SAARTHILMS

27. Remove Element

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

SAARTHILMS

28. Find the Index of the First Occurrence

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

SAARTHILMS

32. Longest Valid Parentheses

```
class Solution {
public:
    int longestValidParentheses(string s) {
        stack<int> st;
        st.push(-1);
        int ans = 0;
        for(int i = 0; i < s.size(); i++) {
            if(s[i] == '(') st.push(i);
            else {
                st.pop();
                if(st.empty()) st.push(i);
                else ans = max(ans, i - st.top());
            }
        }
        return ans;
    }
};
```

SAARTHILMS

35. Search Insert Position

```
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        int l = 0, r = nums.size() - 1;
        while(l <= r) {
            int mid = l + (r - l) / 2;
            if(nums[mid] == target) return mid;
            if(nums[mid] < target) l = mid + 1;
            else r = mid - 1;
        }
        return l;
    }
};
```

SAARTHILMS

58. Length of Last Word

```
class Solution {
public:
    int lengthOfLastWord(string s) {
        int len = 0;
        for(int i = s.size() - 1; i >= 0; i--) {
            if(s[i] != ' ') len++;
            else if(len > 0) break;
        }
        return len;
    }
};
```

SAARTHILMS

66. Plus One

```
class Solution {
public:
    vector<int> plusOne(vector<int>& digits) {
        for(int i = digits.size() - 1; i >= 0; i--) {
            if(digits[i] < 9) {
                digits[i]++;
                return digits;
            }
            digits[i] = 0;
        }
        digits.insert(digits.begin(), 1);
        return digits;
    }
};
```

SAARTHILMS

67. Add Binary

```
class Solution {
public:
    string addBinary(string a, string b) {
        string res = "";
        int i = a.size()-1, j = b.size()-1, carry = 0;
        while(i>=0 || j>=0 || carry) {
            int sum = carry;
            if(i>=0) sum += a[i--]-'0';
            if(j>=0) sum += b[j--]-'0';
            res = char(sum%2 + '0') + res;
            carry = sum/2;
        }
        return res;
    }
};
```

SAARTHILMS

69. Sqrt(x)

```
class Solution {
public:
    int mySqrt(int x) {
        long l = 0, r = x;
        while(l <= r) {
            long mid = (l + r) / 2;
            if(mid * mid <= x && (mid + 1) * (mid + 1) > x)
                return mid;
            if(mid * mid > x) r = mid - 1;
            else l = mid + 1;
        }
        return 0;
    }
};
```

SAARTHILMS

70. Climbing Stairs

```
class Solution {
public:
    int climbStairs(int n) {
        if(n <= 2) return n;
        int a = 1, b = 2;
        for(int i = 3; i <= n; i++) {
            int c = a + b;
            a = b;
            b = c;
        }
        return b;
    }
};
```

SAARTHILMS

75. Sort Colors

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int low = 0, mid = 0, high = nums.size() - 1;
        while(mid <= high) {
            if(nums[mid] == 0) swap(nums[low++], nums[mid++]);
            else if(nums[mid] == 1) mid++;
            else swap(nums[mid], nums[high--]);
        }
    }
};
```

SAARTHILMS

83. Remove Duplicates from Sorted List

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* curr = head;
        while(curr && curr->next) {
            if(curr->val == curr->next->val)
                curr->next = curr->next->next;
            else
                curr = curr->next;
        }
        return head;
    }
};
```

SAARTHILMS

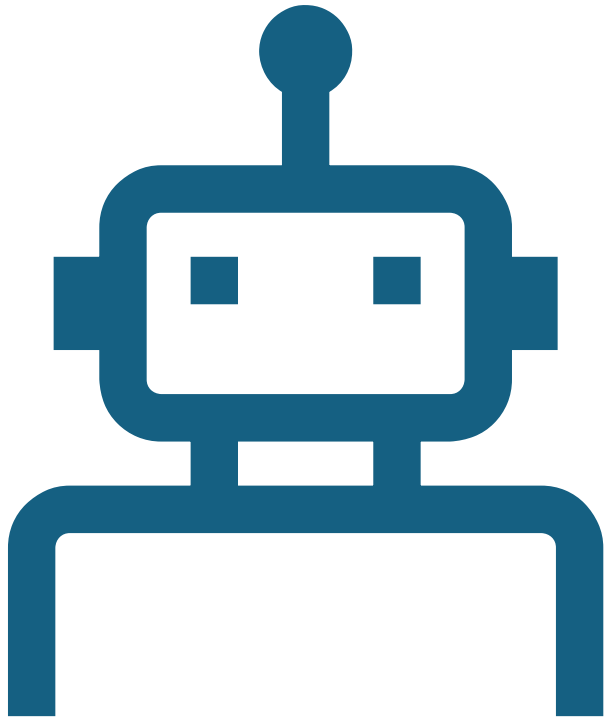
88. Merge Sorted Array

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

SAARTHILMS

94. Binary Tree Inorder Traversal

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> res;
        stack<TreeNode*> st;
        while(root || !st.empty()) {
            while(root) {
                st.push(root);
                root = root->left;
            }
            root = st.top(); st.pop();
            res.push_back(root->val);
            root = root->right;
        }
        return res;
    }
};
```



SAARTHILMS

100. Same Tree

```
class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if(!p && !q) return true;
        if(!p || !q || p->val != q->val) return false;
        return isSameTree(p->left, q->left) &&
               isSameTree(p->right, q->right);
    }
};
```

SAARTHILMS

101. Symmetric Tree

```
class Solution {
public:
    bool mirror(TreeNode* a, TreeNode* b) {
        if(!a && !b) return true;
        if(!a || !b || a->val != b->val) return false;
        return mirror(a->left, b->right) &&
               mirror(a->right, b->left);
    }

    bool isSymmetric(TreeNode* root) {
        return mirror(root, root);
    }
};
```

SAARTHILMS

104. Maximum Depth of Binary Tree

```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(!root) return 0;
        return 1 + max(maxDepth(root->left),
                       maxDepth(root->right));
    }
};
```

SAARTHILMS

108. Convert Sorted Array to BST

```
class Solution {
public:
    TreeNode* build(vector<int>& nums, int l, int r) {
        if(l > r) return nullptr;
        int mid = (l + r) / 2;
        TreeNode* root = new TreeNode(nums[mid]);
        root->left = build(nums, l, mid - 1);
        root->right = build(nums, mid + 1, r);
        return root;
    }

    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return build(nums, 0, nums.size() - 1);
    }
};
```

110. Balanced Binary Tree

```
class Solution {
public:
    int height(TreeNode* root) {
        if(!root) return 0;
        int l = height(root->left);
        int r = height(root->right);
        if(l == -1 || r == -1 || abs(l - r) > 1) return -1;
        return 1 + max(l, r);
    }

    bool isBalanced(TreeNode* root) {
        return height(root) != -1;
    }
};
```

SAARTHILMS

111. Minimum Depth of Binary Tree

```
class Solution {
public:
    int minDepth(TreeNode* root) {
        if(!root) return 0;
        if(!root->left) return 1 + minDepth(root->right);
        if(!root->right) return 1 + minDepth(root->left);
        return 1 + min(minDepth(root->left),
                       minDepth(root->right));
    }
};
```

SAARTHILMS

112. Path Sum

```
class Solution {
public:
    bool hasPathSum(TreeNode* root, int targetSum) {
        if(!root) return false;
        if(!root->left && !root->right)
            return targetSum == root->val;
        return hasPathSum(root->left, targetSum - root->val) ||
               hasPathSum(root->right, targetSum - root->val);
    }
};
```

118.
Pascal’s Triangle

```
class Solution {
public:
    vector<vector<int>> generate(int n) {
        vector<vector<int>> res(n);
        for(int i = 0; i < n; i++) {
            res[i].resize(i + 1, 1);
            for(int j = 1; j < i; j++)
                res[i][j] = res[i-1][j-1] + res[i-1][j];
        }
        return res;
    }
};
```

SAARTHILMS

119. Pascal’s Triangle II

```
class Solution {
public:
    vector<int> getRow(int row) {
        vector<int> res(row + 1, 1);
        for(int i = 1; i < row; i++)
            for(int j = i; j > 0; j--)
                res[j] += res[j - 1];
        return res;
    }
};
```

SAARTHILMS

121. Best Time to Buy and Sell Stock

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int minP = INT_MAX, profit = 0;
        for(int p : prices) {
            minP = min(minP, p);
            profit = max(profit, p - minP);
        }
        return profit;
    }
};
```

SAARTHILMS

125. Valid Palindrome

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

SAARTHILMS

136. Single Number

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int x = 0;
        for(int n : nums) x ^= n;
        return x;
    }
};
```

SAARTHILMS

141. Linked List Cycle

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode *slow = head, *fast = head;
        while(fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if(slow == fast) return true;
        }
        return false;
    }
};
```

SAARTHILMS

151. Reverse Words in a String

```
class Solution {
public:
    string reverseWords(string s) {
        string res, word;
        stringstream ss(s);
        while(ss >> word)
            res = word + " " + res;
        return res.substr(0, res.size() - 1);
    }
};
```

SAARTHILMS

160. Intersection of Two Linked Lists

```
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *a, ListNode *b) {
        ListNode *p = a, *q = b;
        while(p != q) {
            p = p ? p->next : b;
            q = q ? q->next : a;
        }
        return p;
    }
};
```

SAARTHILMS

168. Excel Sheet Column Title

```
class Solution {
public:
    string convertToTitle(int n) {
        string res;
        while(n) {
            n--;
            res = char('A' + n % 26) + res;
            n /= 26;
        }
        return res;
    }
};
```

SAARTHILMS

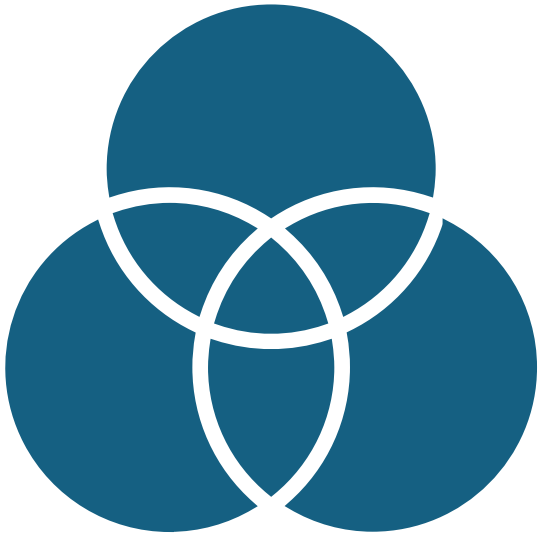
169. Majority Element

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int cnt = 0, cand = 0;
        for(int n : nums) {
            if(cnt == 0) cand = n;
            cnt += (n == cand) ? 1 : -1;
        }
        return cand;
    }
};
```

SAARTHILMS

171. Excel Sheet Column Number

```
class Solution {
public:
    int titleToNumber(string s) {
        int res = 0;
        for(char c : s)
            res = res * 26 + (c - 'A' + 1);
        return res;
    }
};
```



SAARTHILMS