DSA Basics

Anchit Mulye

July 18, 2024

DSA Basics

Anchit Mulye

July 18, 2024

Contents

1 Numbers

1.1 Write a program to find the largest number among three numbers.

```
int findLargest(int a, int b, int c) {
    // Compare the three numbers and return the largest one
    if (a >= b && a >= c) {
        return a;
    } else if (b >= a && b >= c) {
        return b;
    } else {
        return c;
    }
}
```

1.2 Write a program to check if a number is even or odd.

```
bool isEven(int num) {
    // A number is even if it is divisible by 2
    return num % 2 == 0;
}
```

1.3 Write a program to check if a year is a leap year.

```
bool isLeapYear(int year) {
    // A year is a leap year if it is divisible by 4
    // but not divisible by 100 unless it is also divisible by 400
    if (year % 4 == 0) {
        if (year % 100 == 0) {
            if (year % 400 == 0) {
                return true;
            } else {
                return false;
            }
        } else {
            return true;
        }
    } else {
        return false;
    }
}
```

1.4 Write a program to swap two numbers.

```
void swapNumbers(int &a, int &b) {
    // Swap the values of a and b using a temporary variable
    int temp = a;
    a = b;
    b = temp;
}
```

1.5 Write a program to find the factorial of a number.

```
int factorial(int n) {
    // Initialize result to 1
    int result = 1;
    // Multiply result by each number from 1 to n
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

1.6 Write a program to check if a number is prime.

```
bool isPrime(int num) {
    // A number less than or equal to 1 is not prime
    if (num <= 1) return false;
    // Check divisibility from 2 to the square root of num
    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) return false;
    }
    return true;
}
```

1.7 Write a program to print the Fibonacci series up to a certain number of terms.

```
void printFibonacci(int n) {
    // First two terms
    int a = 0, b = 1;
    cout << a << " " << b << " ";
    for (int i = 2; i < n; i++) {
        int next = a + b;
        cout << next << " ";
        a = b;
        b = next;
    }
}
```

1.8 Write a program to reverse a number.

```
int reverseNumber(int num) {
    int reversed = 0;
    while (num > 0) {
        reversed = reversed * 10 + num % 10;
        num /= 10;
    }
    return reversed;
}
```

1.9 Write a program to check if a number is Armstrong number.

```
bool isArmstrong(int num) {
    int originalNum = num;
    int sum = 0;
    int n = to_string(num).length(); // Number of digits in num
    while (num > 0) {
        int digit = num % 10;
        sum += pow(digit, n);
        num /= 10;
    }
    return sum == originalNum;
}
```

1.10 Write a program to find the GCD (Greatest Common Divisor) of two numbers.

```
int gcd(int a, int b) {
    // Use the Euclidean algorithm
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
```

1.11 Write a program to find the LCM (Least Common Multiple) of two numbers.

```
int lcm(int a, int b) {
    return (a * b) / gcd(a, b); // Using LCM(a, b) * GCD(a, b) = a * b
}
```

1.12 Write a program to find the factorial of a number using recursion.

```
int factorialRecursive(int n) {
    if (n == 0) return 1; // Base case
    return n * factorialRecursive(n - 1); // Recursive case
}
```

1.13 Write a program to find the nth Fibonacci number using recursion.

```
int fibonacciRecursive(int n) {
    if (n <= 1) return n; // Base case
    return fibonacciRecursive(n - 1) + fibonacciRecursive(n - 2);
}
```

1.14 Write a program to check if a number is a perfect number.

```
bool isPerfectNumber(int num) {
    int sum = 1;
    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) {
            if (i * i != num) {
                sum += i + num / i;
            } else {
                sum += i;
            }
        }
    }
    return sum == num && num != 1;
}
```

1.15 Write a program to convert decimal to binary.

```
string decimalToBinary(int num) {
    string binary = "";
    while (num > 0) {
        binary = to_string(num % 2) + binary;
        num /= 2;
    }
    return binary;
}
```

1.16 Write a program to convert binary to decimal.

```
int binaryToDecimal(string binary) {
    int decimal = 0;
    int base = 1;
    for (int i = binary.length() - 1; i >= 0; i--) {
        if (binary[i] == '1') {
            decimal += base;
        }
        base *= 2;
    }
    return decimal;
}
```

1.17 Write a program to check if a number is a palindrome using recursion.

```
bool isPalindromeRecursive(string str, int start, int end) {
    if (start >= end) return true; // Base case
    if (str[start] != str[end]) return false;
    return isPalindromeRecursive(str, start + 1, end - 1); // Recursive case
}
```

1.18 Write a program to check if a number is a power of two.

```
bool isPowerOfTwo(int num) {
    // A number is a power of two if it has exactly one bit
    // set in its binary representation
    return (num > 0) && ((num & (num - 1)) == 0);
}
```

1.19 Write a program to find the factorial of a number using an iterative approach.

```
int factorialIterative(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

1.20 Write a program to check if a number is a perfect square.

```
bool isPerfectSquare(int num) {
    int sqrtNum = sqrt(num);
    return (sqrtNum * sqrtNum == num);
}
```

1.21 Write a program to convert a decimal number to its Roman numeral equivalent.

```
string decimalToRoman(int num) {
    // Arrays of roman numerals and their corresponding values
    string romans[] = {"M", "CM", "D", "CD", "C", "XC", "L", "XL",
                       "X", "IX", "V", "IV", "I"};
    int values[] = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
    string roman = "";

    for (int i = 0; i < 13; i++) {
        while (num >= values[i]) {
            num -= values[i];
            roman += romans[i];
        }
    }
    return roman;
}
```

1.22 Write a program to find the next prime number greater than a given number.

```
int nextPrime(int num) {
    num++;
    while (!isPrime(num)) {
        num++;
    }
    return num;
}
```

2 Arrays

2.1 Write a program to find the intersection of two arrays.

```
std::vector<int> findIntersection(const std::vector<int>& arr1,
                                  const std::vector<int>& arr2) {
    std::unordered_set<int> set(arr1.begin(), arr1.end());
    std::vector<int> intersection;

    for (int num : arr2) {
        if (set.count(num)) {
            intersection.push_back(num);
            set.erase(num); // to handle duplicates in arr2
        }
    }

    return intersection;
}
```

2.2 Write a program to find the union of two arrays.

```
std::vector<int> findUnion(const std::vector<int>& arr1,
                           const std::vector<int>& arr2) {
    std::unordered_set<int> set(arr1.begin(), arr1.end());

    for (int num : arr2) {
        set.insert(num);
    }

    std::vector<int> union_vec(set.begin(), set.end());
    return union_vec;
}
```

2.3 Write a program to remove duplicates from an array.

```
std::vector<int> removeDuplicates(const std::vector<int>& arr) {
    std::unordered_set<int> set;
    std::vector<int> unique;

    for (int num : arr) {
        if (set.insert(num).second) { // insert means num was not in the set
            unique.push_back(num);
        }
    }

    return unique;
}
```

2.4 Write a program to find the second largest number in an array.

```
int findSecondLargest(const std::vector<int>& arr) {
    int first = INT_MIN;
    int second = INT_MIN;

    for (int num : arr) {
        if (num > first) {
            second = first;
            first = num;
        } else if (num > second && num != first) {
            second = num;
        }
    }

    return second;
}
```

2.5 Write a program to find the median of an array of numbers.

```
double findMedian(std::vector<int>& arr) {
    size_t n = arr.size();
    std::sort(arr.begin(), arr.end());

    if (n % 2 == 0) {
        return (static_cast<double>(arr[n/2 - 1]) +
                static_cast<double>(arr[n/2])) / 2.0;
    } else {
        return static_cast<double>(arr[n/2]);
    }
}
```

2.6 Write a program to find the maximum and minimum elements in an array.

```
void findMinMax(const std::vector<int>& arr, int& min, int& max) {
    min = INT_MAX;
    max = INT_MIN;

    for (int num : arr) {
        if (num < min) min = num;
        if (num > max) max = num;
    }
}
```

3 Strings

3.1 Write a program to check if a string is a palindrome.

```
bool isPalindrome(string s) {
    int left = 0, right = s.length() - 1;
    while (left < right) {
        if (s[left] != s[right])
            return false;
        left++;
        right--;
    }
    return true;
}
```

3.2 Write a program to find the ASCII value of a character.

```
int getAsciiValue(char c) {
    return int(c);
}
```

3.3 Write a program to convert string to number.

```
int stringToNumber(const std::string& str) {
    std::stringstream ss(str);
    int num;
    ss >> num;
    return num;
}
```

3.4 Write a program to check if a string contains only digits.

```
bool containsOnlyDigits(const std::string& str) {
    for (char c : str) {
        if (!isdigit(c)) {
            return false;
        }
    }
    return true;
}
```

3.5 Write a program to check if two strings are anagrams of each other.

```
bool areAnagrams(const std::string& str1, const std::string& str2) {
    std::string s1 = str1;
    std::string s2 = str2;
    std::sort(s1.begin(), s1.end());
    std::sort(s2.begin(), s2.end());
    return (s1 == s2);
}
```

3.6 Write a program to reverse a string.

```
std::string reverseString(const std::string& str) {
    std::string reversed = str;
    std::reverse(reversed.begin(), reversed.end());
    return reversed;
}
```

3.7 Write a program to check if a string is a pangram (contains every letter of the alphabet at least once).

```
bool isPangram(const std::string& str) {
    bool letters[26] = {false};
    int index;

    for (char c : str) {
        if (isalpha(c)) {
            if (islower(c)) {
                index = c - 'a';
            } else {
                index = c - 'A';
            }
            letters[index] = true;
        }
    }

    for (int i = 0; i < 26; ++i) {
        if (!letters[i]) {
            return false;
        }
    }

    return true;
}
```

3.8 Write a program to check if a string is an anagram of a palindrome.

```
bool canFormPalindrome(const std::string& str) {
    std::unordered_map<char, int> freq;
    int oddCount = 0;

    for (char c : str) {
        freq[c]++;
    }

    for (auto& pair : freq) {
        if (pair.second % 2 != 0) {
            oddCount++;
        }
    }

    return (oddCount <= 1);
}
```

4 Data Structures

4.1 Write a program to implement a stack (using arrays or linked lists).

```
#define MAX_SIZE 100

class Stack {
private:
    int arr[MAX_SIZE];
    int top;

public:
    Stack() {
        top = -1;
    }

    void push(int value) {
        if (top >= MAX_SIZE - 1) {
            cout << "Stack Overflow\n";
            return;
        }
        arr[++top] = value;
    }

    int pop() {
        if (top < 0) {
            cout << "Stack Underflow\n";
            return -1; // or throw an exception
        }
        return arr[top--];
    }

    int peek() {
        if (top < 0) {
            cout << "Stack is empty\n";
            return -1; // or throw an exception
        }
        return arr[top];
    }

    bool isEmpty() {
        return (top == -1);
    }
};

int main() {
    Stack s;
    s.push(10);
    s.push(20);
    s.push(30);
    cout << s.pop() << " popped from stack\n";
    cout << "Top element is " << s.peek() << endl;
}
```

4.2 Write a program to implement a queue (using arrays or linked lists).

```
#define MAX_SIZE 100

class Queue {
private:
    int arr[MAX_SIZE];
    int front, rear;

public:
    Queue() {
        front = -1;
        rear = -1;
    }

    void enqueue(int value) {
        if (rear >= MAX_SIZE - 1) {
            cout << "Queue Overflow\n";
            return;
        }
        arr[++rear] = value;
        if (front == -1) {
            front = 0;
        }
    }

    int dequeue() {
        if (front == -1 || front > rear) {
            cout << "Queue Underflow\n";
            return -1; // or throw an exception
        }
        return arr[front++];
    }

    int peek() {
        if (front == -1 || front > rear) {
            cout << "Queue is empty\n";
            return -1; // or throw an exception
        }
        return arr[front];
    }

    bool isEmpty() {
        return (front == -1 || front > rear);
    }
};
```

4.3 Write a program to implement a binary search tree and perform insertions and deletions.

```
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int val) : data(val), left(nullptr), right(nullptr) {}
};

class BinarySearchTree {
private:
    TreeNode* root;
    TreeNode* insertRecursive(TreeNode* node, int key) {
        if (node == nullptr) {
            return new TreeNode(key);
        }
        if (key < node->data) {
            node->left = insertRecursive(node->left, key);
        } else if (key > node->data) {
            node->right = insertRecursive(node->right, key);
        }
        return node;
    }

    TreeNode* minValueNode(TreeNode* node) {
        TreeNode* current = node;
        while (current && current->left != nullptr) {
            current = current->left;
        }
        return current;
    }

    TreeNode* deleteRecursive(TreeNode* node, int key) {
        if (node == nullptr) {
            return node;
        }
        if (key < node->data) {
            node->left = deleteRecursive(node->left, key);
        } else if (key > node->data) {
            node->right = deleteRecursive(node->right, key);
        } else {
            if (node->left == nullptr) {
                TreeNode* temp = node->right;
                delete node;
                return temp;
            } else if (node->right == nullptr) {
                TreeNode* temp = node->left;
                delete node;
                return temp;
            }
            TreeNode* temp = minValueNode(node->right);
            node->data = temp->data;
            node->right = deleteRecursive(node->right, temp->data);
        }
        return node;
    }
```

```
    public:
        BinarySearchTree() : root(nullptr) {}

        void insert(int key) {
            root = insertRecursive(root, key);
        }

        void remove(int key) {
            root = deleteRecursive(root, key);
        }

        void inorderTraversal(TreeNode* node) {
            if (node == nullptr) {
                return;
            }
            inorderTraversal(node->left);
            cout << node->data << " ";
            inorderTraversal(node->right);
        }

        void inorder() {
            inorderTraversal(root);
            cout << endl;
        }
    };

    int main() {
    BinarySearchTree bst;
    bst.insert(50);
    bst.insert(30);
    bst.insert(20);
    bst.insert(40);
    bst.insert(70);
    bst.insert(60);
    bst.insert(80);

    cout << "Inorder traversal of BST: ";
    bst.inorder();

    bst.remove(20);
    cout << "Inorder traversal after deleting 20: ";
    bst.inorder();

    bst.remove(30);
    cout << "Inorder traversal after deleting 30: ";
    bst.inorder();

    bst.remove(50);
    cout << "Inorder traversal after deleting 50: ";
    bst.inorder();

    return 0;
}
```

4.4 Write a program to implement a linked list and perform insertions and deletions.

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

struct ListNode {
    int data;
    ListNode* next;
    ListNode(int val) : data(val), next(nullptr) {}
};

class LinkedList {
private:
    ListNode* head;

public:
    LinkedList() : head(nullptr) {}

    void insert(int val) {
        ListNode* newNode = new ListNode(val);
        newNode->next = head;
        head = newNode;
    }

    void remove(int val) {
        ListNode* current = head;
        ListNode* prev = nullptr;

        while (current != nullptr && current->data != val) {
            prev = current;
            current = current->next;
        }

        if (current == nullptr) {
            cout << "Element " << val << " not found in the list." << endl;
            return;
        }

        if (prev == nullptr) {
            head = current->next;
        } else {
            prev->next = current->next;
        }

        delete current;
    }

    void display() {
        ListNode* current = head;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};
```

4.5 Write a program to implement a basic queue using two stacks.

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

class Queue {
private:
    stack<int> s1;  // main stack for enqueue
    stack<int> s2;  // temporary stack for dequeue

public:
    void enqueue(int val) {
        s1.push(val);
    }

    int dequeue() {
        if (s1.empty() && s2.empty()) {
            cout << "Queue is empty." << endl;
            return -1;  // indicating queue is empty
        }

        if (s2.empty()) {
            // Transfer all elements from s1 to s2
            while (!s1.empty()) {
                s2.push(s1.top());
                s1.pop();
            }
        }

        int front = s2.top();
        s2.pop();
        return front;
    }

    bool isEmpty() {
        return s1.empty() && s2.empty();
    }
};

int main() {
    Queue q;
    q.enqueue(1);
    q.enqueue(2);
    q.enqueue(3);
    cout << "Dequeued element: " << q.dequeue() << endl;
    cout << "Dequeued element: " << q.dequeue() << endl;
    q.enqueue(4);
    q.enqueue(5);
    while (!q.isEmpty()) {
        cout << "Dequeued element: " << q.dequeue() << endl;
    }
    cout << "Dequeued element: " << q.dequeue() << endl; // Dequeue from empty
}
```

5 Algorithms

5.1 Write a program to multiply two matrices.

```
const int N = 3;

void multiplyMatrices(int mat1[][N], int mat2[][N], int result[][N]) {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            result[i][j] = 0;
            for (int k = 0; k < N; ++k) {
                result[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }
}
```

5.2 Write a program to implement a linear search algorithm.

```
int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; ++i) {
        if (arr[i] == key) {
            return i; // Return index of key if found
        }
    }
    return -1; // Return -1 if key is not found
}
```

5.3 Write a program to implement a binary search algorithm.

```
int binarySearch(int arr[], int l, int r, int key) {
    while (l <= r) {
        int mid = l + (r - l) / 2;
        if (arr[mid] == key) {
            return mid; // Return index of key if found
        }
        if (arr[mid] < key) {
            l = mid + 1; // Search in the right half
        } else {
            r = mid - 1; // Search in the left half
        }
    }
    return -1; // Return -1 if key is not found
}
```

5.4 Write a program to implement a bubble sort algorithm.

```
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; ++i) {
        for (int j = 0; j < n - i - 1; ++j) {
            if (arr[j] > arr[j + 1]) {
                // Swap arr[j] and arr[j + 1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

5.5 Write a program to implement a selection sort algorithm.

```
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; ++i) {
        int min_idx = i;
        for (int j = i + 1; j < n; ++j) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
        // Swap arr[i] and arr[min_idx]
        int temp = arr[i];
        arr[i] = arr[min_idx];
        arr[min_idx] = temp;
    }
}
```

5.6 Write a program to implement an insertion sort algorithm.

```
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; ++i) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
```

5.7 Write a program to implement an quick sort algorithm.

```
int partition(vector<int>& arr, int low, int high) {
    int pivot = arr[high];  // Choosing the last element as the pivot
    int i = low - 1;  // Index of the smaller element

    for (int j = low; j < high; ++j) {
        if (arr[j] < pivot) {
            ++i;
            swap(arr[i], arr[j]);
        }
    }

    swap(arr[i + 1], arr[high]);
    return (i + 1);
}

// Function to implement Quick Sort
void quickSort(vector<int>& arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Sort elements before partition
        quickSort(arr, pi + 1, high); // Sort elements after partition
    }
}
```

5.8 Write a program to implement an merge sort algorithm.

```
void merge(vector<int>& arr, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    // Create temporary arrays
    vector<int> L(n1), R(n2);

    // Copy data to temporary arrays L[] and R[]
    for (int i = 0; i < n1; ++i)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; ++j)
        R[j] = arr[mid + 1 + j];

    // Merge the temporary arrays back into arr[left..right]
    int i = 0;  // Initial index for left subarray
    int j = 0;  // Initial index for right subarray
    int k = left;  // Initial index for merged subarray

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            ++i;
        } else {
            arr[k] = R[j];
            ++j;
        }
        ++k;
    }

    // Copy the remaining elements of L[], if any
    while (i < n1) {
        arr[k] = L[i];
        ++i;
        ++k;
    }

    // Copy the remaining elements of R[], if any
    while (j < n2) {
        arr[k] = R[j];
        ++j;
        ++k;
    }
}

// Function to implement Merge Sort
void mergeSort(vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);  // Sort left half
        mergeSort(arr, mid + 1, right); // Sort right half
        merge(arr, left, mid, right); // Merge the sorted halves
    }
}
```

5.9 Write a program to find the largest sum contiguous subarray (Kadane's Algorithm).

```
int kadane(int arr[], int n) {
    int max_so_far = arr[0];
    int max_ending_here = arr[0];

    for (int i = 1; i < n; ++i) {
        max_ending_here = max(arr[i], max_ending_here + arr[i]);
        max_so_far = max(max_so_far, max_ending_here);
    }

    return max_so_far;
}
```