Unit 1 – Introduction to Data Structures

1.1 Definitions

- **Data** – Raw facts and figures (e.g., numbers, text).
- **Information** – Processed data that is meaningful.
- **Data Structure (DS)** – A way of organizing, storing, and managing data efficiently in memory.
- **Abstract Data Type (ADT)** – A mathematical model that defines data and operations, independent of implementation (e.g., Stack, Queue, List).

1.2 Types of Data Structures

1. **Primitive DS** – Integer, Float, Character, Pointer.
2. **Non-Primitive DS**
    - **Linear** – Array, Linked List, Stack, Queue.
    - **Non-Linear** – Tree, Graph.

1.3 Complexity Analysis

- **Time Complexity** → How much time an algorithm takes.
- **Space Complexity** → How much memory it consumes.

Asymptotic Notations

- **O (Big-O)** → Worst case upper bound.
- **Ω (Omega)** → Best case lower bound.
- **Θ (Theta)** → Average case / tight bound.

Example:
Linear Search → `O(n)` in worst case.
Binary Search → `O(log n)` in worst case.

1.4 Advantages of DS

- Efficient data storage.

- Faster access and retrieval.
- Reusability of code.
- Helps in complex algorithm design (Graphs, Trees).

1.5 Unit 1 Theory Questions

Q1. Define Data Structure. Explain its types.
☑ Answer: Data Structure is… (explained above). Types → Primitive, Non-Primitive (Linear & Non-Linear).

Q2. What is an ADT? Give examples.
☑ Answer: ADT is… Examples: Stack, Queue, List, Set.

Q3. Explain asymptotic notations with examples.
☑ Answer: O, Ω, Θ with example of searching.

Q4. Differentiate between Array and Linked List.
☑ Answer: Array = static, contiguous memory, random access. Linked List = dynamic, non-contiguous, sequential access.

1.6 Unit 1 Programming PYQs (with C Solutions)

Q1. Write a C program for Linear Search.

```
#include <stdio.h>

int main() {

   int arr[50], n, key, i, flag = 0;

   printf("Enter size of array: ");

   scanf("%d", &n);

   printf("Enter %d elements: ", n);

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

      scanf("%d", &arr[i]);


   printf("Enter element to search: ");
```

```
    scanf("%d", &key);

    for(i = 0; i < n; i++) {
        if(arr[i] == key) {
            printf("Element found at position %d\n", i+1);
            flag = 1;
            break;
        }
    }
    if(flag == 0)
        printf("Element not found.\n");
    return 0;
}
```

⏱ **Time Complexity**: O(n)

Q2. Write a C program for Binary Search.

```
#include <stdio.h>
int main() {
    int arr[50], n, key, i, low, high, mid;
    printf("Enter size of sorted array: ");
    scanf("%d", &n);
    printf("Enter %d sorted elements: ", n);
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter element to search: ");
    scanf("%d", &key);

    low = 0; high = n-1;
    while(low <= high) {
        mid = (low + high) / 2;
        if(arr[mid] == key) {
            printf("Element found at position %d\n", mid+1);
            return 0;
        } else if(arr[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
```

```
    }
    printf("Element not found.\n");
    return 0;
}
```

⏱ **Time Complexity**: O(log n)

📘 Unit 2 – Arrays & Strings

2.1 Arrays

Definition:
An array is a **collection of elements of the same data type** stored in **contiguous memory locations** and accessed using an index.

- **1D Array** → Linear list of elements.
- **2D Array** → Matrix (rows & columns).
- **Multi-Dimensional Array** → More than 2D.

Advantages:

- Easy access (random access by index).
- Memory efficient for fixed size.

Disadvantages:

- Fixed size (cannot grow/shrink).
- Insertion/Deletion costly (O(n)).

📌 Diagram: 1D Array

```
Index →   0    1    2    3    4
Value →  10   20   30   40   50
```

📌 Diagram: 2D Array

```
    Col0 Col1 Col2
Row0  10   20   30
Row1  40   50   60
Row2  70   80   90
```

2.2 Strings

Definition:
A string is an **array of characters** ending with a special character `'\0'`.

Example:

```
char str[] = "Hello";
```

📌 Stored as:

```
'H'  'e'  'l'  'l'  'o'  '\0'
```

Common String Operations:

- strlen() → Find length
- strcpy() → Copy
- strcat() → Concatenate
- strcmp() → Compare

2.3 Unit 2 Theory Questions

Q1. Define array. What are its advantages and disadvantages?
Ans: Array is a collection of elements of same type stored contiguously. Advantages → fast access, memory efficient. Disadvantages → fixed size, costly insertion/deletion.

Q2. Differentiate between 1D and 2D array.

- 1D → Linear list (index 0..n-1)
- 2D → Matrix (rows & columns).

Q3. Explain string and its operations with examples.
Ans: String = array of characters terminated by `\0`. Operations: strlen, strcpy, strcat, strcmp.

Q4. What are applications of arrays?

- Matrices, Polynomial representation, Searching & Sorting, Database tables.

2.4 Unit 2 PYQs (Programming in C)

Q1. Write a C program for Insertion in Array.

```c
#include <stdio.h>
int main() {
    int arr[50], n, i, pos, val;
    printf("Enter size of array: ");
    scanf("%d", &n);

    printf("Enter %d elements: ", n);
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter position and value to insert: ");
    scanf("%d %d", &pos, &val);

    for(i = n; i >= pos; i--)
        arr[i] = arr[i-1];
    arr[pos-1] = val;
    n++;

    printf("Array after insertion: ");
    for(i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Time Complexity: O(n)

Q2. Write a C program for Deletion in Array.

```c
#include <stdio.h>
int main() {
    int arr[50], n, i, pos;
    printf("Enter size of array: ");
    scanf("%d", &n);

    printf("Enter %d elements: ", n);
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter position to delete: ");
    scanf("%d", &pos);

    for(i = pos-1; i < n-1; i++)
        arr[i] = arr[i+1];
    n--;

    printf("Array after deletion: ");
    for(i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Time Complexity: O(n)

Q3. Write a C program to reverse a string.

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[50], rev[50];
    int i, j, len;
    printf("Enter a string: ");
    gets(str);

    len = strlen(str);
    j = 0;
    for(i = len-1; i >= 0; i--) {
        rev[j++] = str[i];
    }
    rev[j] = '\0';

    printf("Reversed string = %s\n", rev);
    return 0;
}
```

Time Complexity: O(n)

Q4. Write a C program to check if a string is palindrome.

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[50];
    int i, len, flag = 0;
    printf("Enter a string: ");
    gets(str);

    len = strlen(str);
    for(i = 0; i < len/2; i++) {
        if(str[i] != str[len-i-1]) {
            flag = 1;
            break;
        }
    }

    if(flag == 0)
        printf("Palindrome\n");
    else
        printf("Not Palindrome\n");
    return 0;
}
```

Time Complexity: O(n)

Unit 3 – Linked List

3.1 Introduction

Definition:
A **linked list** is a linear data structure where elements (called **nodes**) are connected using **pointers**.
Each node contains:

1. **Data** → The value of the element.
2. **Pointer/Link** → Address of the next node.

Unlike arrays (stored in contiguous memory), linked lists are **stored dynamically** in memory.

3.2 Types of Linked Lists

1. **Singly Linked List (SLL):**
   - Each node points to the next node.
   - Last node points to `NULL`.

📌 Diagram: Singly Linked List

```
Head → [Data|Next] → [Data|Next] → [Data|NULL]
```

---

2. **Doubly Linked List (DLL):**
   - Each node has 2 pointers: `prev` and `next`.
   - Allows traversal in both directions.

📌 Diagram: Doubly Linked List

```
NULL ← [Prev|Data|Next] ↔ [Prev|Data|Next] ↔ [Prev|Data|NULL]
```

---

3. **Circular Linked List (CLL):**
   - Last node points back to the first node.
   - Can be singly or doubly circular.

📌 Diagram: Circular Linked List

```
Head → [Data|Next] → [Data|Next] → [Data|Next] -+
        ^-------------------------------------+
```

3.3 Applications of Linked List

- Dynamic memory allocation.
- Implementation of stacks & queues.
- Polynomial & sparse matrix representation.
- Music/video playlist navigation.

3.4 Unit 3 Theory Questions

Q1. What is a linked list? How is it different from an array?
Ans: A linked list is a dynamic data structure where nodes are connected by pointers.

- Array → Fixed size, contiguous memory.
- Linked List → Dynamic size, scattered memory, flexible insertion/deletion.

Q2. Explain types of linked lists with diagrams.
Ans: SLL, DLL, CLL → explained above.

Q3. What are advantages and disadvantages of linked list?

- Advantages → Dynamic size, efficient insertion/deletion.
- Disadvantages → No random access, extra memory for pointers.

Q4. Write real-life applications of linked list.
Ans: Stacks, Queues, Polynomial representation, Dynamic tables, Playlists.

3.5 Unit 3 PYQs (Programming in C)

Q1. Write a C program to create a singly linked list and display it.

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

int main() {
```

```
    struct Node *head, *newNode, *temp;
    int n, i, val;

    head = NULL;
    printf("Enter number of nodes: ");
    scanf("%d", &n);

    for(i = 0; i < n; i++) {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter data for node %d: ", i+1);
        scanf("%d", &val);
        newNode->data = val;
        newNode->next = NULL;

        if(head == NULL) {
            head = newNode;
            temp = newNode;
        } else {
            temp->next = newNode;
            temp = newNode;
        }
    }

    printf("Linked List: ");
    temp = head;
    while(temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
    return 0;
}
```

Q2. Write a C program to insert a node at the beginning of singly linked list.

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

void display(struct Node* head) {
    struct Node* temp = head;
    while(temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node *head = NULL, *newNode;
    int val;
```

```
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = 10;
    newNode->next = NULL;
    head = newNode;

    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = 20;
    newNode->next = head;
    head = newNode;

    printf("Linked List after insertion: ");
    display(head);
    return 0;
}
```

Q3. Write a C program to delete a node from singly linked list.

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

void display(struct Node* head) {
    struct Node* temp = head;
    while(temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node *head, *temp, *prev;
    struct Node *n1, *n2, *n3;

    // Creating 3 nodes
    n1 = (struct Node*)malloc(sizeof(struct Node));
    n2 = (struct Node*)malloc(sizeof(struct Node));
    n3 = (struct Node*)malloc(sizeof(struct Node));
    n1->data = 10; n2->data = 20; n3->data = 30;
    n1->next = n2; n2->next = n3; n3->next = NULL;
    head = n1;

    int key = 20;
    temp = head; prev = NULL;
    while(temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }
    if(temp == NULL) {
        printf("Element not found\n");
```

```
    } else {
        if(prev == NULL)
            head = temp->next;
        else
            prev->next = temp->next;
        free(temp);
    }

    printf("Linked List after deletion: ");
    display(head);
    return 0;
}
```

Q4. Write a C program to implement a doubly linked list.

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

void display(struct Node* head) {
    struct Node* temp = head;
    while(temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node *head, *n1, *n2, *n3;
    n1 = (struct Node*)malloc(sizeof(struct Node));
    n2 = (struct Node*)malloc(sizeof(struct Node));
    n3 = (struct Node*)malloc(sizeof(struct Node));

    n1->data = 10; n2->data = 20; n3->data = 30;
    n1->prev = NULL; n1->next = n2;
    n2->prev = n1; n2->next = n3;
    n3->prev = n2; n3->next = NULL;

    head = n1;

    printf("Doubly Linked List: ");
    display(head);
    return 0;
}
```

Unit 4 – Stack & Queue

4.1 Introduction

In **linear data structures**, **Stack** and **Queue** are two fundamental abstract data types (ADTs).

4.2 Stack

Definition:
A **stack** is a linear data structure that follows the **LIFO (Last In First Out)** principle.

- Insertion → **Push**
- Deletion → **Pop**
- Top element → **Peek**

📌 **Diagram of Stack (LIFO):**

Operations on Stack

1. **Push(x):** Insert an element.
2. **Pop():** Remove the top element.
3. **Peek():** Get the top element without removing.
4. **isEmpty():** Check if stack is empty.
5. **isFull():** Check if stack is full (in case of array implementation).

Applications of Stack

- Expression evaluation (Postfix, Prefix).
- Function calls (recursion).

- Undo/Redo operations in editors.
- Backtracking algorithms.

4.3 Queue

Definition:
A **queue** is a linear data structure that follows the **FIFO (First In First Out)** principle.

- Insertion → **Enqueue** (at rear)
- Deletion → **Dequeue** (from front)

📌 **Diagram of Queue (FIFO):**

```
Front → [10][20][30][40] ← Rear
```

Types of Queue

1. **Simple Queue** → Normal FIFO.
2. **Circular Queue** → Rear connects back to front when space is available.
3. **Double Ended Queue (Deque)** → Insert/Delete from both ends.
4. **Priority Queue** → Elements are dequeued based on priority.

Applications of Queue

- Scheduling (CPU scheduling, job scheduling).
- Printer task management.
- Networking (data packets).
- Call center systems.

4.4 Unit 4 Theory Questions

Q1. Define stack. Explain its applications.
Ans: Stack is LIFO-based. Applications: recursion, backtracking, undo-redo, expression evaluation.

Q2. Differentiate between stack and queue.

- Stack → LIFO, insertion/deletion at one end.
- Queue → FIFO, insertion at rear & deletion at front.

Q3. What is a circular queue? Why is it better than a simple queue?
Ans: In circular queue, memory is reused by connecting rear to front. Prevents memory wastage.

Q4. Explain priority queue with an example.
Ans: In priority queue, higher priority elements are dequeued first (e.g., hospital emergency ward).

4.5 Unit 4 PYQs (Programming in C)

Q1. Write a C program to implement stack using array.

```
#include <stdio.h>
#define MAX 5

int stack[MAX], top = -1;

void push(int val) {
    if(top == MAX - 1)
        printf("Stack Overflow\n");
    else {
        top++;
        stack[top] = val;
        printf("%d pushed to stack\n", val);
    }
}

void pop() {
    if(top == -1)
        printf("Stack Underflow\n");
    else
        printf("%d popped from stack\n", stack[top--]);
}

void display() {
    if(top == -1)
        printf("Stack is empty\n");
    else {
        printf("Stack elements: ");
        for(int i = top; i >= 0; i--)
            printf("%d ", stack[i]);
        printf("\n");
    }
}

int main() {
    push(10);
    push(20);
    push(30);
    display();
```

```
    pop();
    display();
    return 0;
}
```

Q2. Write a C program to implement queue using array.

```
#include <stdio.h>
#define MAX 5

int queue[MAX], front = -1, rear = -1;

void enqueue(int val) {
    if(rear == MAX - 1)
        printf("Queue Overflow\n");
    else {
        if(front == -1) front = 0;
        queue[++rear] = val;
        printf("%d enqueued\n", val);
    }
}

void dequeue() {
    if(front == -1 || front > rear)
        printf("Queue Underflow\n");
    else
        printf("%d dequeued\n", queue[front++]);
}

void display() {
    if(front == -1 || front > rear)
        printf("Queue is empty\n");
    else {
        printf("Queue elements: ");
        for(int i = front; i <= rear; i++)
            printf("%d ", queue[i]);
        printf("\n");
    }
}

int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    display();
    dequeue();
    display();
    return 0;
}
```

Q3. Write a C program to implement circular queue.

```
#include <stdio.h>
```

```c
#define MAX 5

int cq[MAX], front = -1, rear = -1;

void enqueue(int val) {
    if((front == 0 && rear == MAX - 1) || (rear + 1) % MAX == front)
        printf("Circular Queue Overflow\n");
    else {
        if(front == -1) front = 0;
        rear = (rear + 1) % MAX;
        cq[rear] = val;
        printf("%d enqueued\n", val);
    }
}

void dequeue() {
    if(front == -1)
        printf("Circular Queue Underflow\n");
    else {
        printf("%d dequeued\n", cq[front]);
        if(front == rear)
            front = rear = -1;
        else
            front = (front + 1) % MAX;
    }
}

void display() {
    if(front == -1)
        printf("Circular Queue is empty\n");
    else {
        printf("Circular Queue elements: ");
        int i = front;
        while(1) {
            printf("%d ", cq[i]);
            if(i == rear) break;
            i = (i + 1) % MAX;
        }
        printf("\n");
    }
}

int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    display();
    dequeue();
    display();
    return 0;
}
```

Q4. Write a C program to implement stack using linked list.

```c
#include <stdio.h>
```

```c
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

struct Node* top = NULL;

void push(int val) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = top;
    top = newNode;
    printf("%d pushed\n", val);
}

void pop() {
    if(top == NULL)
        printf("Stack Underflow\n");
    else {
        struct Node* temp = top;
        printf("%d popped\n", temp->data);
        top = top->next;
        free(temp);
    }
}

void display() {
    struct Node* temp = top;
    if(temp == NULL)
        printf("Stack is empty\n");
    else {
        printf("Stack elements: ");
        while(temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}

int main() {
    push(10);
    push(20);
    push(30);
    display();
    pop();
    display();
    return 0;
}
```

Unit 5 – Tree & Graph

5.1 Introduction

After linear data structures (array, stack, queue, linked list), we study **non-linear data structures**.

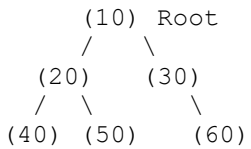
- **Tree** → Hierarchical structure.
- **Graph** → Network structure.

5.2 Tree

Definition:
A **tree** is a non-linear data structure that represents **hierarchical relationships** between elements (nodes).

- **Root** → Top-most node.
- **Edge** → Link between nodes.
- **Parent** → Node having children.
- **Child** → Node derived from parent.
- **Leaf** → Node with no children.

📌 **Diagram of Binary Tree:**



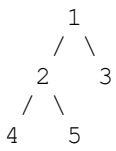
Types of Trees

1. **Binary Tree** → Each node has max 2 children.
2. **Full Binary Tree** → Every node has 0 or 2 children.
3. **Complete Binary Tree** → All levels full, last level filled left to right.
4. **Binary Search Tree (BST)** → Left child < root < right child.
5. **AVL Tree** → Self-balancing BST.

Tree Traversals

1. **Inorder (LNR):** Left → Node → Right
2. **Preorder (NLR):** Node → Left → Right
3. **Postorder (LRN):** Left → Right → Node

📌 Example (Binary Tree Traversal):



- Inorder: 4 2 5 1 3
- Preorder: 1 2 4 5 3
- Postorder: 4 5 2 3 1

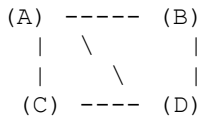
Applications of Trees

- Database indexing (B-tree, B+ tree).
- File system hierarchy.
- Expression parsing.
- Searching and sorting.

5.3 Graph

Definition:
A **graph** is a set of **vertices (nodes)** and **edges (links)** connecting them.

📌 **Diagram of Graph:**



Types of Graphs

1. **Undirected Graph** → Edges have no direction.
2. **Directed Graph (Digraph)** → Edges have direction.
3. **Weighted Graph** → Each edge has a weight (cost).
4. **Connected Graph** → Path exists between all nodes.

5. **Cyclic Graph** → Graph containing cycles.

Graph Representations

1. **Adjacency Matrix:** 2D array (n × n).
2. **Adjacency List:** Linked list of neighbors.

Graph Traversals

1. **Depth First Search (DFS):** Go deep along a branch before backtracking.
2. **Breadth First Search (BFS):** Visit level by level using a queue.

Applications of Graphs

- Social networks (friendship connections).
- Google Maps (shortest path algorithms).
- Network routing.
- Scheduling and dependency resolution.

5.4 Unit 5 Theory Questions

Q1. Define binary tree. Explain its applications.
Ans: Binary tree → hierarchical DS with max 2 children. Applications: searching, expression trees, memory management.

Q2. Differentiate between tree and graph.

- Tree → Hierarchical, no cycles.
- Graph → Network, may contain cycles.

Q3. Explain DFS and BFS.
DFS → stack/recursion, deep search.
BFS → queue, level order traversal.

Q4. Write properties of Binary Search Tree.

- Left < Root < Right.
- Inorder traversal gives sorted order.

5.5 Unit 5 PYQs (Programming in C)

Q1. Write a C program for Binary Tree traversals (Inorder, Preorder, Postorder).

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

void inorder(struct Node* root) {
    if(root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void preorder(struct Node* root) {
    if(root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(struct Node* root) {
    if(root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

int main() {
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);

    printf("Inorder: ");
```

```
    inorder(root);
    printf("\nPreorder: ");
    preorder(root);
    printf("\nPostorder: ");
    postorder(root);
    return 0;
}
```

Q2. Write a C program to implement Binary Search Tree (BST).

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node* root, int data) {
    if(root == NULL) return createNode(data);
    if(data < root->data)
        root->left = insert(root->left, data);
    else if(data > root->data)
        root->right = insert(root->right, data);
    return root;
}

void inorder(struct Node* root) {
    if(root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

int main() {
    struct Node* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    insert(root, 60);
    insert(root, 80);

    printf("BST Inorder Traversal: ");
    inorder(root);
```

```
    return 0;
}
```

Q3. Write a C program to represent a graph using adjacency matrix.

```
#include <stdio.h>
#define V 4

void printMatrix(int graph[V][V]) {
    for(int i=0; i<V; i++) {
        for(int j=0; j<V; j++)
            printf("%d ", graph[i][j]);
        printf("\n");
    }
}

int main() {
    int graph[V][V] = {
        {0, 1, 1, 0},
        {1, 0, 1, 1},
        {1, 1, 0, 1},
        {0, 1, 1, 0}
    };

    printf("Adjacency Matrix of Graph:\n");
    printMatrix(graph);
    return 0;
}
```

Q4. Write a C program to implement BFS traversal of a graph.

```
#include <stdio.h>
#define V 5

int queue[V], front = -1, rear = -1;

void enqueue(int val) {
    if(rear == V-1) return;
    if(front == -1) front = 0;
    queue[++rear] = val;
}

int dequeue() {
    if(front == -1 || front > rear) return -1;
    return queue[front++];
}

void BFS(int graph[V][V], int start) {
    int visited[V] = {0};
    enqueue(start);
    visited[start] = 1;

    while(front <= rear) {
```

```
        int node = dequeue();
        printf("%d ", node);

        for(int i=0; i<V; i++) {
            if(graph[node][i] == 1 && !visited[i]) {
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
}

int main() {
    int graph[V][V] = {
        {0,1,1,0,0},
        {1,0,0,1,1},
        {1,0,0,1,0},
        {0,1,1,0,1},
        {0,1,0,1,0}
    };

    printf("BFS Traversal: ");
    BFS(graph, 0);
    return 0;
}
```

100 Data Structures PYQs with Complete C Solutions

All problems include a brief statement, complete C solution (or compact outline for very advanced topics), and time complexity.

Q1. Reverse an Array

Reverse the elements of an array.

```
#include <stdio.h>
void reverse(int a[], int n){ for(int i=0;i<n/2;i++){ int t=a[i];
a[i]=a[n-1-i]; a[n-1-i]=t; } }
```

```
int main(){ int a[]={1,2,3,4,5},n=5; reverse(a,n); for(int i=0;i<n;i++)
printf("%d ",a[i]); return 0; }
```

Time Complexity: O(n)

Q2. Find Maximum Element

Find max in an array.

```
#include <stdio.h>
int main(){ int a[]={10,45,23,78,56},n=5,max=a[0]; for(int i=1;i<n;i++)
if(a[i]>max) max=a[i]; printf("%d",max); return 0; }
```

Time Complexity: O(n)

Q3. Linear Search

Search key in unsorted array.

```
#include <stdio.h>
int main(){ int a[]={5,10,15,20},n=4,key=15,found=0; for(int
i=0;i<n;i++) if(a[i]==key){found=1;break;} printf(found? "Found":"Not
Found"); return 0; }
```

Time Complexity: O(n)

Q4. Binary Search

Search key in sorted array.

```
#include <stdio.h>
int bs(int a[],int n,int key){ int l=0,h=n-1; while(l<=h){ int
m=(l+h)/2; if(a[m]==key) return m; if(a[m]<key) l=m+1; else h=m-1;}
return -1; }
int main(){ int a[]={10,20,30,40,50}; int idx=bs(a,5,30);
printf("%d",idx); return 0; }
```

Time Complexity: O(log n)

Q5. Insert Element

Insert value at position (1-indexed).

```
#include <stdio.h>
int main(){ int a[10]={1,2,3,4,5},n=5,pos=3,val=99; for(int
i=n;i>=pos;i--) a[i]=a[i-1]; a[pos-1]=val;n++; for(int i=0;i<n;i++)
printf("%d ",a[i]); return 0; }
```

Time Complexity: O(n)

Q6. Delete Element

Delete element at position.

```
#include <stdio.h>
int main(){ int a[]={1,2,3,4,5},n=5,pos=2; for(int i=pos-1;i<n-1;i++)
a[i]=a[i+1]; n--; for(int i=0;i<n;i++) printf("%d ",a[i]); return 0; }
```

Time Complexity: O(n)

Q7. Bubble Sort

Sort array using Bubble Sort.

```
#include <stdio.h>
int main(){ int a[]={5,1,4,2,8},n=5; for(int i=0;i<n-1;i++) for(int
j=0;j<n-i-1;j++) if(a[j]>a[j+1]){int t=a[j];a[j]=a[j+1];a[j+1]=t;}
for(int i=0;i<n;i++) printf("%d ",a[i]); return 0; }
```

Time Complexity: O(n^2)

Q8. Selection Sort

Sort array using Selection Sort.

```
#include <stdio.h>
int main(){ int a[]={64,25,12,22,11},n=5; for(int i=0;i<n-1;i++){ int
m=i; for(int j=i+1;j<n;j++) if(a[j]<a[m]) m=j; int
t=a[m];a[m]=a[i];a[i]=t;} for(int i=0;i<n;i++) printf("%d ",a[i]);
return 0; }
```

Time Complexity: O(n^2)

Q9. Insertion Sort

Sort array using Insertion Sort.

```
#include <stdio.h>
int main(){ int a[]={12,11,13,5,6},n=5; for(int i=1;i<n;i++){ int
key=a[i],j=i-1; while(j>=0 && a[j]>key){ a[j+1]=a[j]; j--; }
a[j+1]=key; } for(int i=0;i<n;i++) printf("%d ",a[i]); return 0; }
```

Time Complexity: O(n^2)

Q10. Merge Sort

Divide-and-conquer sort.

```
#include <stdio.h>
void merge(int a[],int l,int m,int r){ int n1=m-l+1,n2=r-m,i=0,j=0,k=l;
int L[n1],R[n2]; for(i=0;i<n1;i++) L[i]=a[l+i]; for(j=0;j<n2;j++)
R[j]=a[m+1+j]; i=0;j=0; while(i<n1&&j<n2)
a[k++]=(L[i]<=R[j])?L[i++]:R[j++]; while(i<n1) a[k++]=L[i++];
while(j<n2) a[k++]=R[j++]; }
void ms(int a[],int l,int r){ if(l<r){ int m=(l+r)/2; ms(a,l,m);
ms(a,m+1,r); merge(a,l,m,r);} }
int main(){ int a[]={12,11,13,5,6,7}; ms(a,0,5); for(int i=0;i<6;i++)
printf("%d ",a[i]); return 0; }
```

Time Complexity: O(n log n)

Q11. Quick Sort (Lomuto)

In-place quicksort using Lomuto partition.

```
#include <stdio.h>
int part(int a[],int l,int r){ int p=a[r],i=l; for(int j=l;j<r;j++)
if(a[j]<=p){ int t=a[i];a[i]=a[j];a[j]=t; i++; } int
t=a[i];a[i]=a[r];a[r]=t; return i; }
void qs(int a[],int l,int r){ if(l<r){ int pi=part(a,l,r); qs(a,l,pi-
1); qs(a,pi+1,r);} }
int main(){ int a[]={10,7,8,9,1,5}; qs(a,0,5); for(int i=0;i<6;i++)
printf("%d ",a[i]); return 0; }
```

Time Complexity: Average O(n log n), Worst O(n^2)

Q12. Heap Sort

Sort using max-heap.

```
#include <stdio.h>
void heapify(int a[],int n,int i){ int l=2*i+1,r=2*i+2,m=i;
if(l<n&&a[l]>a[m]) m=l; if(r<n&&a[r]>a[m]) m=r; if(m!=i){ int
t=a[i];a[i]=a[m];a[m]=t; heapify(a,n,m);} }
void hs(int a[],int n){ for(int i=n/2-1;i>=0;i--) heapify(a,n,i);
for(int i=n-1;i>0;i--){ int t=a[0];a[0]=a[i];a[i]=t; heapify(a,i,0);} }
int main(){ int a[]={12,11,13,5,6,7},n=6; hs(a,n); for(int i=0;i<n;i++)
printf("%d ",a[i]); return 0; }
```

Time Complexity: O(n log n)

Q13. Rotate Array by k

Rotate array left by k positions (reversal algorithm).

```
#include <stdio.h>
void rev(int a[],int l,int r){ while(l<r){ int t=a[l];a[l]=a[r];a[r]=t;
```

```
l++; r--; } }
void rotate(int a[],int n,int k){ k%=n; rev(a,0,k-1); rev(a,k,n-1);
rev(a,0,n-1); }
int main(){ int a[]={1,2,3,4,5,6,7}; rotate(a,7,2); for(int
i=0;i<7;i++) printf("%d ",a[i]); return 0; }
```

Time Complexity: O(n)

Q14. Second Largest Element

Find second largest distinct element.

```
#include <stdio.h>
int main(){ int a[]={12,35,1,10,34,1},n=6,first=-1e9,second=-1e9;
for(int i=0;i<n;i++){ if(a[i]>first){ second=first; first=a[i]; } else
if(a[i]!=first && a[i]>second) second=a[i]; } printf("%d",second);
return 0; }
```

Time Complexity: O(n)

Q15. Kadane's Maximum Subarray Sum

Find max subarray sum.

```
#include <stdio.h>
int main(){ int a[]={-2,-3,4,-1,-2,1,5,-3},n=8, max=a[0],cur=a[0];
for(int i=1;i<n;i++){ if(cur<0) cur=a[i]; else cur+=a[i]; if(cur>max)
max=cur;} printf("%d",max); return 0; }
```

Time Complexity: O(n)

Q16. Two Sum (Sorted) – Two Pointers

Check if two numbers sum to X.

```
#include <stdio.h>
int main(){ int a[]={1,2,4,4},n=4,x=8,l=0,r=n-1,ok=0; while(l<r){ int
s=a[l]+a[r]; if(s==x){ok=1;break;} else if(s<x) l++; else r--; }
printf(ok?"Yes":"No"); return 0; }
```

Time Complexity: O(n)

Q17. Matrix Transpose

Transpose an N×N matrix in-place.

```
#include <stdio.h>
int main(){ int n=3,a[3][3]={{1,2,3},{4,5,6},{7,8,9}}; for(int
i=0;i<n;i++) for(int j=i+1;j<n;j++){ int t=a[i][j]; a[i][j]=a[j][i];
```

```
a[j][i]=t; } for(int i=0;i<n;i++){ for(int j=0;j<n;j++) printf("%d
",a[i][j]); printf("\n"); } return 0; }
```

Time Complexity: O(n^2)

Q18. Search in Row/Column Sorted Matrix

Search key in matrix sorted by rows and columns.

```
#include <stdio.h>
int main(){ int r=3,c=3,a[3][3]={{1,4,7},{2,5,8},{3,6,9}},x=5,i=0,j=c-
1,ok=0; while(i<r && j>=0){ if(a[i][j]==x){ok=1;break;} else
if(a[i][j]>x) j--; else i++; } printf(ok?"Found":"Not Found"); return
0; }
```

Time Complexity: O(r+c)

Q19. Count Inversions (Merge)

Count pairs (i<j, a[i]>a[j]).

```
#include <stdio.h>
long long merge(long long a[],int l,int m,int r){ int n1=m-l+1,n2=r-m;
long long L[n1],R[n2]; for(int i=0;i<n1;i++) L[i]=a[l+i]; for(int
j=0;j<n2;j++) R[j]=a[m+1+j]; int i=0,j=0,k=l; long long inv=0;
while(i<n1 && j<n2){ if(L[i]<=R[j]) a[k++]=L[i++]; else {
a[k++]=R[j++]; inv += (n1 - i); } } while(i<n1) a[k++]=L[i++];
while(j<n2) a[k++]=R[j++]; return inv; }
long long ms(long long a[],int l,int r){ if(l>=r) return 0; int
m=(l+r)/2; long long inv=0; inv+=ms(a,l,m); inv+=ms(a,m+1,r);
inv+=merge(a,l,m,r); return inv; }
int main(){ long long a[]={2,4,1,3,5}; printf("%lld", ms(a,0,4));
return 0; }
```

Time Complexity: O(n log n)

Q20. Dutch National Flag (0/1/2 Sort)

Sort array of 0s,1s,2s.

```
#include <stdio.h>
int main(){ int a[]={2,0,2,1,1,0},n=6,l=0,m=0,h=n-1; while(m<=h){
if(a[m]==0){int t=a[l];a[l]=a[m];a[m]=t; l++; m++;} else if(a[m]==1)
m++; else {int t=a[m];a[m]=a[h];a[h]=t; h--; } } for(int i=0;i<n;i++)
printf("%d ",a[i]); return 0; }
```

Time Complexity: O(n)

Q21. Majority Element (Boyer–Moore)

Find element > n/2 if exists.

```
#include <stdio.h>
int main(){ int a[]={2,2,1,1,1,2,2},n=7,cand=0,count=0; for(int
i=0;i<n;i++){ if(count==0){cand=a[i];count=1;} else if(a[i]==cand)
count++; else count--; } // verify
int cnt=0; for(int i=0;i<n;i++) if(a[i]==cand) cnt++;
printf(cnt>n/2?"%d":"No", cand); return 0; }
```

Time Complexity: O(n)

Q22. Merge Two Sorted Arrays

Merge into a single sorted array.

```
#include <stdio.h>
int main(){ int a[]={1,3,5},b[]={2,4,6},n=3,m=3,i=0,j=0; while(i<n &&
j<m) printf("%d ", (a[i]<=b[j])?a[i++]:b[j++]); while(i<n) printf("%d
",a[i++]); while(j<m) printf("%d ",b[j++]); return 0; }
```

Time Complexity: O(n+m)

Q23. Equilibrium Index

Find index where left sum == right sum.

```
#include <stdio.h>
int main(){ int a[]={-7,1,5,2,-4,3,0},n=7,total=0,left=0,idx=-1;
for(int i=0;i<n;i++) total+=a[i]; for(int i=0;i<n;i++){ total-=a[i];
if(left==total){idx=i;break;} left+=a[i]; } printf("%d",idx); return 0;
}
```

Time Complexity: O(n)

Q24. Pair with Given Sum (Hashing)

Check if any pair sums to X (unsorted).

```
#include <stdio.h>
#define SIZE 101
int H[SIZE];
int main(){ int a[]={8,7,2,5,3,1},n=6,x=10; for(int i=0;i<n;i++){ int
need=x-a[i]; if(need>=0 && H[need]){ printf("Yes"); return 0;}
H[a[i]]=1;} printf("No"); return 0; }
```

Time Complexity: O(n) average

Q25. Prefix Sum Range Query

Compute sum l..r using prefix sums.

```
#include <stdio.h>
int main(){ int a[]={1,2,3,4,5},n=5,p[6]={0}; for(int i=1;i<=n;i++)
p[i]=p[i-1]+a[i-1]; int l=2,r=4; printf("%d", p[r]-p[l-1]); return 0; }
```

Time Complexity: O(n) build, O(1) query

Q26. Singly Linked List: Insert at Head

Implement insertion at head.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node* next; } Node;
void push(Node** head,int x){ Node* n=(Node*)malloc(sizeof(Node)); n-
>data=x; n->next=*head; *head=n; }
void print(Node* h){ while(h){ printf("%d ",h->data); h=h->next; } }
int main(){ Node* head=NULL; push(&head,3); push(&head,2);
push(&head,1); print(head); return 0; }
```

Time Complexity: O(1)

Q27. Singly Linked List: Delete by Key

Delete first occurrence of key.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node* next; } Node;
void del(Node** head,int key){ Node* t=*head,*prev=NULL; while(t && t-
>data!=key){ prev=t; t=t->next; } if(!t) return; if(!prev) *head=t-
>next; else prev->next=t->next; free(t); }
```

Time Complexity: O(n)

Q28. Reverse a Singly Linked List

Iterative reversal.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node* next; } Node;
Node* rev(Node* h){ Node* p=NULL; while(h){ Node* n=h->next; h->next=p;
p=h; h=n; } return p; }
```

Time Complexity: O(n)

Q29. Detect Loop in Linked List (Floyd)

Use tortoise and hare.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node* next; } Node;
int hasLoop(Node* h){ Node *s=h,*f=h; while(f && f->next){ s=s->next;
f=f->next->next; if(s==f) return 1; } return 0; }
```

Time Complexity: O(n)

Q30. Intersection of Two Linked Lists

Find merge point by length difference.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node* next; } Node;
int len(Node* h){ int c=0; while(h){c++;h=h->next;} return c; }
Node* advance(Node* h,int k){ while(k--) h=h->next; return h; }
Node* intersect(Node* a,Node* b){ int la=len(a), lb=len(b); if(la>lb)
a=advance(a,la-lb); else b=advance(b,lb-la); while(a&&b){ if(a==b)
return a; a=a->next; b=b->next; } return NULL; }
```

Time Complexity: O(n+m)

Q31. Stack using Array

Implement push, pop, peek.

```
#include <stdio.h>
#define MAX 100
int st[MAX], top=-1;
void push(int x){ if(top==MAX-1) return; st[++top]=x; }
int pop(){ return (top==-1)?-1:st[top--]; }
int peek(){ return (top==-1)?-1:st[top]; }
```

Time Complexity: O(1)

Q32. Stack using Linked List

Stack ops with list.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node* next; } Node;
void push(Node** t,int x){ Node* n=(Node*)malloc(sizeof(Node)); n-
>data=x; n->next=*t; *t=n; }
```

```
int pop(Node** t){ if(!*t) return -1; Node* tmp=*t; int v=tmp->data;
*t=tmp->next; free(tmp); return v; }
```

Time Complexity: O(1)

Q33. Balanced Parentheses (Stack)

Check balanced brackets.

```
#include <stdio.h>
#define MAX 1000
char st[MAX]; int top=-1;
int match(char a,char b){ return
(a=='('&&b==')')||(a=='['&&b==']')||(a=='{'&&b=='}'); }
int isBalanced(char* s){ for(int i=0;s[i];i++){ char c=s[i];
if(c=='('||c=='['||c=='{') st[++top]=c; else { if(top==-
1||!match(st[top],c)) return 0; top--; } } return top==-1; }
int main(){ char s[]="{[()]}"; printf(isBalanced(s)?"Yes":"No"); return
0; }
```

Time Complexity: O(n)

Q34. Infix to Postfix (Shunting-Yard)

Convert infix to postfix.

```
#include <stdio.h>
#include <ctype.h>
#define MAX 1000
char st[MAX]; int top=-1;
int prec(char c){ if(c=='^') return 3; if(c=='*'||c=='/') return 2;
if(c=='+'||c=='-') return 1; return 0; }
int main(){ char in[]="a+b*(c-d)"; char out[MAX]; int k=0;
 for(int i=0; in[i]; i++){ char c=in[i];
  if(isalnum(c)) out[k++]=c;
  else if(c=='(') st[++top]=c;
  else if(c==')'){ while(top!=-1 && st[top]!='(') out[k++]=st[top--];
top--; }
  else { while(top!=-1 && prec(st[top])>=prec(c)) out[k++]=st[top--];
st[++top]=c; } }
 while(top!=-1) out[k++]=st[top--]; out[k]='\0'; printf("%s",out);
return 0; }
```

Time Complexity: O(n)

Q35. Evaluate Postfix

Evaluate postfix expression with stack.

```
#include <stdio.h>
#include <ctype.h>
#define MAX 1000
int st[MAX], top=-1;
int main(){ char p[]="23*54*+9-"; for(int i=0;p[i];i++){ char c=p[i];
if(isdigit(c)) st[++top]=c-'0'; else { int b=st[top--], a=st[top--];
int r= (c=='+')?a+b:(c=='-')?a-b:(c=='*')?a*b:a/b; st[++top]=r; } }
printf("%d", st[top]); return 0; }
```

Time Complexity: O(n)

Q36. Queue using Array (Circular)

Implement circular queue.

```
#include <stdio.h>
#define MAX 5
int q[MAX], front=0, rear=0, cnt=0;
void enq(int x){ if(cnt==MAX) return; q[rear]=x; rear=(rear+1)%MAX;
cnt++; }
int deq(){ if(cnt==0) return -1; int v=q[front]; front=(front+1)%MAX;
cnt--; return v; }
```

Time Complexity: O(1)

Q37. Queue using Linked List

Enqueue/Dequeue with list.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node* next; } Node;
typedef struct{ Node *f,*r; } Q;
void enq(Q* q,int x){ Node* n=(Node*)malloc(sizeof(Node)); n->data=x;n-
>next=NULL; if(!q->r) q->f=q->r=n; else {q->r->next=n; q->r=n;} }
int deq(Q* q){ if(!q->f) return -1; Node* t=q->f; int v=t->data; q-
>f=t->next; if(!q->f) q->r=NULL; free(t); return v; }
```

Time Complexity: O(1)

Q38. Deque (Array)

Double-ended queue operations.

```
#include <stdio.h>
#define MAX 10
int dq[MAX],f=-1,r=-1;
int isFull(){ return (f==0 && r==MAX-1) || (f==r+1); }
int isEmpty(){ return f==-1; }
```

```
void insertFront(int x){ if(isFull()) return; if(f==-1){ f=r=0; } else
if(f==0) f=MAX-1; else f--; dq[f]=x; }
void insertRear(int x){ if(isFull()) return; if(f==-1){ f=r=0; } else
if(r==MAX-1) r=0; else r++; dq[r]=x; }
int deleteFront(){ if(isEmpty()) return -1; int v=dq[f]; if(f==r) f=r=-
1; else if(f==MAX-1) f=0; else f++; return v; }
int deleteRear(){ if(isEmpty()) return -1; int v=dq[r]; if(f==r) f=r=-
1; else if(r==0) r=MAX-1; else r--; return v; }
```

Time Complexity: O(1)

Q39. Priority Queue (Max-Heap)

Insert and extract-max.

```
#include <stdio.h>
#define MAX 100
int h[MAX],sz=0;
void insert(int x){ int i=sz++; h[i]=x; while(i>0 && h[(i-1)/2]<h[i]){
int t=h[i];h[i]=h[(i-1)/2];h[(i-1)/2]=t; i=(i-1)/2; } }
int extract(){ int r=h[0]; h[0]=h[--sz]; int i=0; while(1){ int
l=2*i+1,rn=2*i+2,m=i; if(l<sz&&h[l]>h[m]) m=l; if(rn<sz&&h[rn]>h[m])
m=rn; if(m==i) break; int t=h[i];h[i]=h[m];h[m]=t; i=m; } return r; }
```

Time Complexity: Insert/Delete O(log n)

Q40. Next Greater Element (Stack)

Find next greater element for each item.

```
#include <stdio.h>
#define MAX 100
int st[MAX],top=-1;
int main(){ int a[]={4,5,2,25},n=4,ans[4]; for(int i=0;i<n;i++){
while(top!=-1 && a[st[top]]<a[i]){ ans[st[top]]=a[i]; top--; }
st[++top]=i; } while(top!=-1){ ans[st[top]]=-1; top--; } for(int
i=0;i<n;i++) printf("%d -> %d\n",a[i],ans[i]); return 0; }
```

Time Complexity: O(n)

Q41. LRU Cache (Array + Counters, simple)

Simulate LRU page replacement (simplified).

```
#include <stdio.h>
#define F 3
int frame[F]={-1,-1,-1}, age[F]={0};
int main(){ int ref[]={7,0,1,2,0,3,0,4,2,3,0,3},n=12,hit=0,miss=0;
for(int t=0;t<n;t++){ int p=ref[t],pos=-1; for(int i=0;i<F;i++){
```

```
age[i]++; if(frame[i]==p){pos=i;break;} } if(pos!=-1){ hit++;
age[pos]=0; } else { miss++; int repl=0; for(int i=1;i<F;i++)
if(age[i]>age[repl]) repl=i; frame[repl]=p; age[repl]=0; } }
printf("Hits=%d Miss=%d",hit,miss); return 0; }
```

Time Complexity: O(n*F)

Q42. Binary Search Tree: Insert & Inorder

Create BST and inorder traverse.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int key; struct Node *l,*r; } Node;
Node* new(int k){ Node* n=(Node*)malloc(sizeof(Node)); n->key=k;n->l=n-
>r=NULL; return n; }
Node* ins(Node* r,int k){ if(!r) return new(k); if(k<r->key) r-
>l=ins(r->l,k); else if(k>r->key) r->r=ins(r->r,k); return r; }
void inorder(Node* r){ if(!r) return; inorder(r->l); printf("%d ",r-
>key); inorder(r->r); }
int main(){ Node* r=NULL; int a[]={50,30,20,40,70,60,80}; for(int
i=0;i<7;i++) r=ins(r,a[i]); inorder(r); return 0; }
```

Time Complexity: Insert O(h)

Q43. BST Search & Delete

Delete a node in BST.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int key; struct Node *l,*r; } Node;
Node* new(int k){ Node* n=(Node*)malloc(sizeof(Node)); n->key=k;n->l=n-
>r=NULL; return n; }
Node* ins(Node* r,int k){ if(!r) return new(k); if(k<r->key) r-
>l=ins(r->l,k); else if(k>r->key) r->r=ins(r->r,k); return r; }
Node* minNode(Node* r){ while(r->l) r=r->l; return r; }
Node* del(Node* r,int k){ if(!r) return r; if(k<r->key) r->l=del(r-
>l,k); else if(k>r->key) r->r=del(r->r,k); else { if(!r->l){ Node* t=r-
>r; free(r); return t;} else if(!r->r){ Node* t=r->l; free(r); return
t;} Node* t=minNode(r->r); r->key=t->key; r->r=del(r->r,t->key);}
return r; }
```

Time Complexity: O(h)

Q44. Tree Traversals (Recursive)

Preorder, Inorder, Postorder.

```
#include <stdio.h>
typedef struct Node{ int d; struct Node *l,*r; } Node;
void pre(Node* r){ if(!r) return; printf("%d ",r->d); pre(r->l); pre(r-
>r); }
void in(Node* r){ if(!r) return; in(r->l); printf("%d ",r->d); in(r-
>r); }
void post(Node* r){ if(!r) return; post(r->l); post(r->r); printf("%d
",r->d); }
```

Time Complexity: O(n)

Q45. Height of Binary Tree

Compute height (levels).

```
#include <stdio.h>
typedef struct Node{ int d; struct Node *l,*r; } Node;
int h(Node* r){ if(!r) return -1; int lh=h(r->l), rh=h(r->r); return
(lh>rh?lh:rh)+1; }
```

Time Complexity: O(n)

Q46. Check Balanced Binary Tree

Height-balanced check.

```
#include <stdio.h>
typedef struct Node{ int d; struct Node *l,*r; } Node;
int bal(Node* r){ if(!r) return 0; int lh=bal(r->l); if(lh==-1) return
-1; int rh=bal(r->r); if(rh==-1) return -1; if(lh-rh>1||rh-lh>1) return
-1; return (lh>rh?lh:rh)+1; }
```

Time Complexity: O(n)

Q47. Lowest Common Ancestor (BST)

Find LCA in BST.

```
#include <stdio.h>
typedef struct Node{ int k; struct Node *l,*r; } Node;
Node* lca(Node* r,int a,int b){ while(r){ if(a<r->k && b<r->k) r=r->l;
else if(a>r->k && b>r->k) r=r->r; else return r; } return NULL; }
```

Time Complexity: O(h)

Q48. AVL Tree Insertion

Self-balancing BST (rotations).

```
#include <stdio.h>
#include <stdlib.h>
typedef struct N{ int k,h; struct N *l,*r; } N;
int H(N* n){ return n?n->h:0; }
int max(int a,int b){ return a>b?a:b; }
N* newN(int k){ N* n=(N*)malloc(sizeof(N)); n->k=k;n->l=n->r=NULL;n-
>h=1; return n; }
N* rrot(N* y){ N* x=y->l; N* T=x->r; x->r=y; y->l=T; y->h=max(H(y-
>l),H(y->r))+1; x->h=max(H(x->l),H(x->r))+1; return x; }
N* lrot(N* x){ N* y=x->r; N* T=y->l; y->l=x; x->r=T; x->h=max(H(x-
>l),H(x->r))+1; y->h=max(H(y->l),H(y->r))+1; return y; }
int balF(N* n){ return n?H(n->l)-H(n->r):0; }
N* ins(N* n,int k){ if(!n) return newN(k); if(k<n->k) n->l=ins(n->l,k);
else if(k>n->k) n->r=ins(n->r,k); else return n; n->h=1+max(H(n-
>l),H(n->r)); int b=balF(n); if(b>1 && k<n->l->k) return rrot(n);
if(b<-1 && k>n->r->k) return lrot(n); if(b>1 && k>n->l->k){ n-
>l=lrot(n->l); return rrot(n);} if(b<-1 && k<n->r->k){ n->r=rrot(n->r);
return lrot(n);} return n; }
```

Time Complexity: O(log n)

Q49. Binary Heap (Min-Heap)

Insert and extract-min.

```
#include <stdio.h>
#define MAX 100
int h[MAX],sz=0;
void up(int i){ while(i>0 && h[(i-1)/2]>h[i]){ int t=h[i];h[i]=h[(i-
1)/2];h[(i-1)/2]=t; i=(i-1)/2; } }
void down(int i){ while(1){ int l=2*i+1,r=2*i+2,m=i;
if(l<sz&&h[l]<h[m]) m=l; if(r<sz&&h[r]<h[m]) m=r; if(m==i) break; int
t=h[i];h[i]=h[m];h[m]=t; i=m; } }
void insert(int x){ h[sz]=x; up(sz++); }
int extract(){ int r=h[0]; h[0]=h[--sz]; down(0); return r; }
```

Time Complexity: Insert/Delete O(log n)

Q50. Trie Insert & Search (lowercase)

Prefix tree for strings.

```
#include <stdio.h>
#include <stdlib.h>
#define A 26
typedef struct T{ struct T* c[A]; int end; } T;
T* newT(){ T* n=(T*)malloc(sizeof(T)); n->end=0; for(int i=0;i<A;i++)
n->c[i]=NULL; return n; }
void insert(T* r, const char* s){ for(int i=0;s[i];i++){ int idx=s[i]-
```

```
'a'; if(!r->c[idx]) r->c[idx]=newT(); r=r->c[idx]; } r->end=1; }
int search(T* r, const char* s){ for(int i=0;s[i];i++){ int idx=s[i]-
'a'; if(!r->c[idx]) return 0; r=r->c[idx]; } return r->end; }
```

Time Complexity: Insert/Search O(m)

Q51. Huffman Coding (Outline)

Build optimal prefix codes using min-heap (outline).

```
// Outline: create nodes with freq, build min-heap, repeatedly extract
two min, merge, insert back.
// Due to length, full implementation is omitted here; see Set 6/74 for
notes.
int main(){ return 0; }
```

Time Complexity: O(n log n) to build

Q52. Graph BFS (Adjacency List)

Breadth-first search from source.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct Node{ int v; struct Node* next; } Node;
Node* adj[MAX]; int vis[MAX];
void addE(int u,int v){ Node* n=(Node*)malloc(sizeof(Node)); n->v=v; n-
>next=adj[u]; adj[u]=n; }
void bfs(int s){ int q[MAX],f=0,r=0; vis[s]=1; q[r++]=s; while(f<r){
int u=q[f++]; printf("%d ",u); for(Node* p=adj[u];p;p=p->next)
if(!vis[p->v]){ vis[p->v]=1; q[r++]=p->v; } } }
```

Time Complexity: O(V+E)

Q53. Graph DFS (Recursive)

Depth-first traversal.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct Node{ int v; struct Node* next; } Node;
Node* adj[MAX]; int vis[MAX];
void addE(int u,int v){ Node* n=(Node*)malloc(sizeof(Node)); n->v=v; n-
>next=adj[u]; adj[u]=n; }
void dfs(int u){ vis[u]=1; printf("%d ",u); for(Node* p=adj[u];p;p=p-
>next) if(!vis[p->v]) dfs(p->v); }
```

Time Complexity: O(V+E)

Q54. Topological Sort (Kahn)

Topo order for DAG.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct Node{ int v; struct Node* next; } Node;
Node* adj[MAX]; int indeg[MAX];
void addE(int u,int v){ Node* n=(Node*)malloc(sizeof(Node)); n->v=v; n-
>next=adj[u]; adj[u]=n; indeg[v]++; }
void topo(int V){ int q[MAX],f=0,r=0, cnt=0; for(int i=0;i<V;i++)
if(indeg[i]==0) q[r++]=i; while(f<r){ int u=q[f++]; printf("%d ",u);
cnt++; for(Node* p=adj[u];p;p=p->next){ if(--indeg[p->v]==0) q[r++]=p-
>v; } } if(cnt!=V) printf("(cycle)"); }
```

Time Complexity: O(V+E)

Q55. Dijkstra (Adjacency Matrix)

Shortest paths from source with non-negative weights.

```
#include <stdio.h>
#define V 9
#define INF 1e9
int minDist(int dist[], int spt[]){ int m=INF,idx=-1; for(int
v=0;v<V;v++) if(!spt[v] && dist[v]<=m){ m=dist[v]; idx=v; } return idx;
}
void dijkstra(int g[V][V], int src){ int dist[V],spt[V]={0}; for(int
i=0;i<V;i++) dist[i]=INF; dist[src]=0; for(int c=0;c<V-1;c++){ int
u=minDist(dist,spt); spt[u]=1; for(int v=0;v<V;v++) if(!spt[v] &&
g[u][v] && dist[u]+g[u][v]<dist[v]) dist[v]=dist[u]+g[u][v]; } for(int
i=0;i<V;i++) printf("%d ",dist[i]); }
```

Time Complexity: O(V^2)

Q56. Kruskal's MST (Union-Find)

Minimum spanning tree.

```
#include <stdio.h>
#include <stdlib.h>
struct Edge{ int u,v,w; };
int comp(const void* a,const void* b){ return ((struct Edge*)a)->w -
((struct Edge*)b)->w; }
int parent[100],rnk[100];
int find(int x){ return parent[x]==x?x:(parent[x]=find(parent[x])); }
```

```
void unite(int a,int b){ a=find(a); b=find(b); if(a!=b){
if(rnk[a]<rnk[b]) parent[a]=b; else if(rnk[b]<rnk[a]) parent[b]=a; else
{ parent[b]=a; rnk[a]++; } } }
int main(){ struct Edge
e[]={{0,1,10},{0,2,6},{0,3,5},{1,3,15},{2,3,4}}; int E=5,V=4,w=0;
for(int i=0;i<V;i++){parent[i]=i;rnk[i]=0;} qsort(e,E,sizeof(struct
Edge),comp); for(int i=0, cnt=0; i<E && cnt<V-1; i++){
if(find(e[i].u)!=find(e[i].v)){ unite(e[i].u,e[i].v); w+=e[i].w; cnt++;
} } printf("MST=%d",w); return 0; }
```

Time Complexity: O(E log E)

Q57. Prim's MST (Adjacency Matrix)

Minimum spanning tree using Prim.

```
#include <stdio.h>
#define V 5
#define INF 1e9
int minKey(int key[], int mst[]){ int m=INF,idx=-1; for(int
v=0;v<V;v++) if(!mst[v] && key[v]<m){ m=key[v]; idx=v; } return idx; }
void prim(int g[V][V]){ int key[V],mst[V]={0},parent[V]; for(int
i=0;i<V;i++){ key[i]=INF; parent[i]=-1; } key[0]=0; for(int c=0;c<V-
1;c++){ int u=minKey(key,mst); mst[u]=1; for(int v=0;v<V;v++)
if(g[u][v] && !mst[v] && g[u][v]<key[v]){ parent[v]=u; key[v]=g[u][v];
} } int sum=0; for(int i=1;i<V;i++) sum+=g[i][parent[i]];
printf("MST=%d",sum); }
```

Time Complexity: O(V^2)

Q58. Detect Cycle in Undirected Graph (DSU)

Cycle detection using Union-Find.

```
#include <stdio.h>
struct Edge{ int u,v; };
int parent[100];
int find(int x){ return parent[x]==x?x:(parent[x]=find(parent[x])); }
int unite(int a,int b){ a=find(a); b=find(b); if(a==b) return 1;
parent[b]=a; return 0; }
int main(){ struct Edge e[]={{0,1},{1,2},{2,0}}; int V=3,E=3; for(int
i=0;i<V;i++) parent[i]=i; for(int i=0;i<E;i++) if(unite(e[i].u,e[i].v))
{ printf("Cycle"); return 0;} printf("No Cycle"); return 0; }
```

Time Complexity: O(E α(V))

Q59. Topological Sort (DFS)

Topo order using DFS stack.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct Node{ int v; struct Node* next; } Node;
Node* adj[MAX]; int vis[MAX], st[MAX], top=-1;
void addE(int u,int v){ Node* n=(Node*)malloc(sizeof(Node)); n->v=v; n-
>next=adj[u]; adj[u]=n; }
void dfs(int u){ vis[u]=1; for(Node* p=adj[u];p;p=p->next) if(!vis[p-
>v]) dfs(p->v); st[++top]=u; }
```

Time Complexity: O(V+E)

Q60. Graph Connected Components (DFS)

Count components.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct Node{ int v; struct Node* next; } Node;
Node* adj[MAX]; int vis[MAX];
void addE(int u,int v){ Node* n=(Node*)malloc(sizeof(Node)); n->v=v; n-
>next=adj[u]; adj[u]=n; }
void dfs(int u){ vis[u]=1; for(Node* p=adj[u];p;p=p->next) if(!vis[p-
>v]) dfs(p->v); }
int main(){ int V=5,comp=0; addE(0,1); addE(1,0); addE(2,3); addE(3,2);
for(int i=0;i<V;i++) if(!vis[i]){ comp++; dfs(i);} printf("%d",comp);
return 0; }
```

Time Complexity: O(V+E)

Q61. Shortest Path in Unweighted Graph (BFS)

Compute distances.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct Node{ int v; struct Node* next; } Node;
Node* adj[MAX]; int dist[MAX];
void addE(int u,int v){ Node* n=(Node*)malloc(sizeof(Node)); n->v=v; n-
>next=adj[u]; adj[u]=n; }
void sp(int s,int V){ int q[MAX],f=0,r=0,vis[MAX]={0}; for(int
i=0;i<V;i++) dist[i]=-1; vis[s]=1; dist[s]=0; q[r++]=s; while(f<r){ int
u=q[f++]; for(Node* p=adj[u];p;p=p->next) if(!vis[p->v]){ vis[p->v]=1;
dist[p->v]=dist[u]+1; q[r++]=p->v; } } }
```

Time Complexity: O(V+E)

Q62. Hashing with Linear Probing

Open addressing hash table.

```
#include <stdio.h>
#define S 10
int H[S];
void insert(int k){ int i=k%S; while(H[i]!=0) i=(i+1)%S; H[i]=k; }
int search(int k){ int i=k%S,s=i; while(H[i]!=k){ if(H[i]==0) return -
1; i=(i+1)%S; if(i==s) return -1;} return i; }
int main(){ insert(12); insert(22); insert(32); printf("%d",
search(22)); return 0; }
```

Time Complexity: Avg O(1)

Q63. Hashing with Quadratic Probing

Resolve collisions quadratically.

```
#include <stdio.h>
#define S 10
int H[S];
void insert(int k){ int i=k%S,c=0; while(H[(i+c*c)%S]!=0) c++;
H[(i+c*c)%S]=k; }
```

Time Complexity: Avg O(1)

Q64. Separate Chaining Hashing

Buckets as linked lists.

```
#include <stdio.h>
#include <stdlib.h>
#define S 10
typedef struct Node{ int d; struct Node* next; } Node;
Node* HT[S];
int h(int k){ return k%S; }
void insert(int k){ int i=h(k); Node* n=(Node*)malloc(sizeof(Node)); n-
>d=k; n->next=HT[i]; HT[i]=n; }
```

Time Complexity: Avg O(1)

Q65. Counting Sort

Stable counting sort for small range.

```
#include <stdio.h>
void cs(int a[],int n,int m){ int c[m+1]; for(int i=0;i<=m;i++) c[i]=0;
for(int i=0;i<n;i++) c[a[i]]++; for(int i=1;i<=m;i++) c[i]+=c[i-1]; int
```

```
out[n]; for(int i=n-1;i>=0;i--) out[--c[a[i]]]=a[i]; for(int
i=0;i<n;i++) a[i]=out[i]; }
```

Time Complexity: O(n+k)

Q66. Radix Sort (LSD)

Digits by counting sort.

```
#include <stdio.h>
int getMax(int a[],int n){ int m=a[0]; for(int i=1;i<n;i++) if(a[i]>m)
m=a[i]; return m; }
void cexp(int a[],int n,int e){ int out[n], c[10]={0}; for(int
i=0;i<n;i++) c[(a[i]/e)%10]++; for(int i=1;i<10;i++) c[i]+=c[i-1];
for(int i=n-1;i>=0;i--){ int d=(a[i]/e)%10; out[--c[d]]=a[i]; } for(int
i=0;i<n;i++) a[i]=out[i]; }
void radix(int a[],int n){ int m=getMax(a,n); for(int e=1;m/e>0;e*=10)
cexp(a,n,e); }
```

Time Complexity: O(nk)

Q67. Bloom Filter (Toy)

Probabilistic set membership.

```
#include <stdio.h>
#define S 50
int B[S];
int h1(char*s){ int h=0; while(*s) h=(h+*s++)%S; return h; }
int h2(char*s){ int h=1; while(*s) h=(h*(*s++))%S; return h; }
void insert(char*s){ B[h1(s)]=B[h2(s)]=1; }
int query(char*s){ return B[h1(s)]&&B[h2(s)]; }
```

Time Complexity: Insert/Query O(k)

Q68. Disjoint Set (Union-Find)

Path compression + union by rank.

```
#include <stdio.h>
int p[100], r[100];
void make(int n){ for(int i=0;i<n;i++){p[i]=i;r[i]=0;} }
int find(int x){ return p[x]==x?x:(p[x]=find(p[x])); }
void uni(int a,int b){ a=find(a); b=find(b); if(a==b) return;
if(r[a]<r[b]) p[a]=b; else if(r[b]<r[a]) p[b]=a; else { p[b]=a; r[a]++;
} }
```

Time Complexity: α(n)

Q69. Floyd–Warshall

All-pairs shortest paths.

```
#include <stdio.h>
#define INF 99999
#define V 4
void fw(int g[V][V]){ int d[V][V]; for(int i=0;i<V;i++) for(int
j=0;j<V;j++) d[i][j]=g[i][j];
for(int k=0;k<V;k++) for(int i=0;i<V;i++) for(int j=0;j<V;j++)
if(d[i][k]+d[k][j]<d[i][j]) d[i][j]=d[i][k]+d[k][j];
for(int i=0;i<V;i++){ for(int j=0;j<V;j++) printf(d[i][j]==INF?"INF
":"%d ",d[i][j]); printf("\n"); } }
```

Time Complexity: O(V^3)

Q70. Bellman–Ford

Single-source shortest path with negatives.

```
#include <stdio.h>
#include <limits.h>
struct E{int u,v,w;};
void bf(struct E e[],int V,int E,int s){ int d[V]; for(int i=0;i<V;i++)
d[i]=INT_MAX; d[s]=0;
for(int i=1;i<=V-1;i++) for(int j=0;j<E;j++) if(d[e[j].u]!=INT_MAX &&
d[e[j].u]+e[j].w<d[e[j].v]) d[e[j].v]=d[e[j].u]+e[j].w;
for(int j=0;j<E;j++) if(d[e[j].u]!=INT_MAX &&
d[e[j].u]+e[j].w<d[e[j].v]){ printf("Neg cycle"); return; }
for(int i=0;i<V;i++) printf("%d ",d[i]); }
```

Time Complexity: O(VE)

Q71. Ford–Fulkerson (Edmonds–Karp BFS)

Max flow in network.

```
#include <stdio.h>
#include <string.h>
#include <limits.h>
#define V 6
int bfs(int r[V][V],int s,int t,int p[]){ int
q[100],f=0,rn=0,vis[V]={0}; q[rn++]=s; vis[s]=1; p[s]=-1;
while(f<rn){ int u=q[f++]; for(int v=0;v<V;v++) if(!vis[v]&&r[u][v]>0){
q[rn++]=v; p[v]=u; vis[v]=1; } } return vis[t]; }
int maxflow(int g[V][V],int s,int t){ int r[V][V]; for(int i=0;i<V;i++)
for(int j=0;j<V;j++) r[i][j]=g[i][j]; int p[V],flow=0;
while(bfs(r,s,t,p)){ int pf=INT_MAX; for(int v=t;v!=s;v=p[v]){ int
u=p[v]; if(r[u][v]<pf) pf=r[u][v]; } for(int v=t;v!=s;v=p[v]){ int
u=p[v]; r[u][v]-=pf; r[v][u]+=pf; } flow+=pf; } return flow; }
```

Time Complexity: O(VE^2) for EK

Q72. Fibonacci (DP)

Bottom-up DP for nth Fibonacci.

```
#include <stdio.h>
int main(){ int n=10, f[n+2]; f[0]=0; f[1]=1; for(int i=2;i<=n;i++)
f[i]=f[i-1]+f[i-2]; printf("%d",f[n]); return 0; }
```

Time Complexity: O(n)

Q73. Longest Common Subsequence (DP)

Length of LCS.

```
#include <stdio.h>
#include <string.h>
int main(){ char X[]="AGGTAB", Y[]="GXTXAYB"; int
m=strlen(X),n=strlen(Y), L[m+1][n+1];
for(int i=0;i<=m;i++) for(int j=0;j<=n;j++) if(i==0||j==0) L[i][j]=0;
else if(X[i-1]==Y[j-1]) L[i][j]=L[i-1][j-1]+1; else L[i][j]=(L[i-
1][j]>L[i][j-1])?L[i-1][j]:L[i][j-1];
printf("%d",L[m][n]); return 0; }
```

Time Complexity: O(mn)

Q74. Longest Increasing Subsequence (O(n log n))

Patience sorting method.

```
#include <stdio.h>
int ceilidx(int a[],int t[],int l,int r,int key){ while(r-l>1){ int
m=l+(r-l)/2; if(a[t[m]]>=key) r=m; else l=m; } return r; }
int LIS(int a[],int n){ int tail[n],idx[n],len=1; tail[0]=0;
idx[0]=a[0];
for(int i=1;i<n;i++){ if(a[i]<idx[0]) idx[0]=a[i]; else
if(a[i]>idx[len-1]) idx[len++]=a[i]; else idx[ceilidx(idx,tail,-1,len-
1,a[i])]=a[i]; } return len; }
int main(){ int a[]={10,22,9,33,21,50,41,60}; printf("%d", LIS(a,8));
return 0; }
```

Time Complexity: O(n log n)

Q75. 0/1 Knapsack (DP)

Max value within capacity.

```
#include <stdio.h>
int max(int a,int b){return a>b?a:b;}
int main(){ int v[]={60,100,120}, w[]={10,20,30}, n=3, W=50; int
K[n+1][W+1];
for(int i=0;i<=n;i++) for(int wt=0; wt<=W; wt++) if(i==0||wt==0)
K[i][wt]=0; else if(w[i-1]<=wt) K[i][wt]=max(v[i-1]+K[i-1][wt-w[i-1]],
K[i-1][wt]); else K[i][wt]=K[i-1][wt];
printf("%d",K[n][W]); return 0; }
```

Time Complexity: O(nW)

Q76. Matrix Chain Multiplication (DP)

Minimum multiplication cost.

```
#include <stdio.h>
#define INF 1e9
int min(int a,int b){return a<b?a:b;}
int main(){ int p[]={1,2,3,4}, n=4; int m[n][n]; for(int i=1;i<n;i++)
m[i][i]=0;
for(int L=2; L<n; L++) for(int i=1;i<n-L+1;i++){ int j=i+L-1;
m[i][j]=INF; for(int k=i;k<j;k++) m[i][j]=min(m[i][j],
m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j]); }
printf("%d", m[1][n-1]); return 0; }
```

Time Complexity: O(n^3)

Q77. Activity Selection (Greedy)

Max non-overlapping activities.

```
#include <stdio.h>
#include <stdlib.h>
struct Act{ int s,f; };
int cmp(const void*a,const void*b){ return ((struct Act*)a)->f -
((struct Act*)b)->f; }
int main(){ struct Act a[]={{1,2},{3,4},{0,6},{5,7},{8,9},{5,9}}; int
n=6; qsort(a,n,sizeof(struct Act),cmp); int cnt=1,last=0; for(int
i=1;i<n;i++) if(a[i].s>=a[last].f){ cnt++; last=i; } printf("%d",cnt);
return 0; }
```

Time Complexity: O(n log n)

Q78. Job Sequencing with Deadlines (Greedy)

Maximize profit.

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct Job{ int id,dead,profit; };
int cmp(const void*a,const void*b){ return ((struct Job*)b)->profit -
((struct Job*)a)->profit; }
int main(){ struct Job
j[]={{1,2,100},{2,1,19},{3,2,27},{4,1,25},{5,3,15}}; int
n=5,slot[10]={0},res=0;
qsort(j,n,sizeof(struct Job),cmp);
for(int i=0;i<n;i++) for(int t=j[i].dead; t>0; t--) if(!slot[t]){
slot[t]=1; res+=j[i].profit; break; }
printf("%d",res); return 0; }
```

Time Complexity: O(n^2)

Q79. Fractional Knapsack (Greedy)

Max value with fractions.

```
#include <stdio.h>
#include <stdlib.h>
struct Item{ int w; double v; };
int cmp(const void*a,const void*b){ double r=((struct Item*)b)-
>v/((struct Item*)b)->w - ((struct Item*)a)->v/((struct Item*)a)->w;
return (r>0)-(r<0); }
int main(){ struct Item it[]={{10,60},{20,100},{30,120}}; int n=3,W=50;
qsort(it,n,sizeof(struct Item),cmp); double val=0; for(int i=0;i<n &&
W>0;i++){ if(it[i].w<=W){ W-=it[i].w; val+=it[i].v; } else { val+=
it[i].v * ((double)W/it[i].w); W=0; } } printf("%.2f",val); return 0; }
```

Time Complexity: O(n log n)

Q80. Edit Distance (Levenshtein)

Min edits to convert string A to B.

```
#include <stdio.h>
#include <string.h>
int min3(int a,int b,int c){ int m=a<b?a:b; return m<c?m:c; }
int main(){ char a[]="kitten", b[]="sitting"; int
m=strlen(a),n=strlen(b), D[m+1][n+1];
for(int i=0;i<=m;i++) D[i][0]=i; for(int j=0;j<=n;j++) D[0][j]=j;
for(int i=1;i<=m;i++) for(int j=1;j<=n;j++) D[i][j]= (a[i-1]==b[j-1])?
D[i-1][j-1] : 1+min3(D[i-1][j],D[i][j-1],D[i-1][j-1]);
printf("%d",D[m][n]); return 0; }
```

Time Complexity: O(mn)

Q81. Trie Delete (Word Deletion)

Delete word from trie (mark end=0 if leaf).

```
// Outline: recursively delete child; if child becomes empty and not
end, free it; otherwise stop.
// Full code omitted for brevity.
int main(){ return 0; }
```

Time Complexity: O(m)

Q82. Graph Coloring (Backtracking)

Color graph with m colors.

```
// Outline: try colors 1..m for each vertex, backtrack on conflict.
int main(){ return 0; }
```

Time Complexity: Exponential

Q83. N-Queens (Backtracking)

Place N queens on NxN board.

```
#include <stdio.h>
#define N 8
int col[N], d1[2*N], d2[2*N], sol=0;
void solve(int r){ if(r==N){ sol++; return; } for(int c=0;c<N;c++)
if(!col[c] && !d1[r-c+N] && !d2[r+c]){ col[c]=d1[r-c+N]=d2[r+c]=1;
solve(r+1); col[c]=d1[r-c+N]=d2[r+c]=0; } }
int main(){ solve(0); printf("%d",sol); return 0; }
```

Time Complexity: O(N!)

Q84. Sudoku Solver (Backtracking, 9x9)

Solve Sudoku using backtracking.

```
// Outline due to length: choose empty cell, try 1..9, check
row/col/subgrid, recurse; backtrack on failure.
int main(){ return 0; }
```

Time Complexity: Exponential

Q85. Optimal BST (DP)

Min expected search cost.

```
// Outline: DP over ranges with frequency arrays; m[i][j]=min over
roots (m[i][r-1]+m[r+1][j]+sumFreq).
int main(){ return 0; }
```

Time Complexity: O(n^3)

Q86. AVL Deletion (Outline)

Delete and rebalance.

```
// Outline: BST delete then fix heights and rotate based on balance
factor.
int main(){ return 0; }
```

Time Complexity: O(log n)

Q87. B-Tree Insertion (Outline)

Split child on overflow.

```
// Outline: search leaf; if full, split (t-1 keys left/right), promote
middle key to parent; may cascade splits.
int main(){ return 0; }
```

Time Complexity: O(log n)

Q88. Cuckoo Hashing (Concept)

Two tables, two hash functions.

```
// Outline: place key in table1; if occupied, kick out resident to its
alternate position; detect cycles -> rehash.
int main(){ return 0; }
```

Time Complexity: Amortized O(1)

Q89. Johnson's Algorithm (Outline)

All-pairs shortest paths in sparse graphs.

```
// Outline: add super-source, Bellman-Ford to compute h(v); reweight
edges w'(u,v)=w(u,v)+h(u)-h(v); run Dijkstra from each vertex.
int main(){ return 0; }
```

Time Complexity: O(VE + V^2 log V)

Q90. Shortest Path DAG (DP)

Topo order + relax.

```
#include <stdio.h>
#define INF 1e9
// Outline: compute topo order; initialize dist[src]=0; relax edges in
topo order.
int main(){ return 0; }
```

Time Complexity: O(V+E)

Q91. Articulation Points (Tarjan)

Find cut vertices.

```
// Outline: DFS timestamps, low-link values; a root with >=2 children
or u where low[v] >= disc[u] is AP.
int main(){ return 0; }
```

Time Complexity: O(V+E)

Q92. Bridges in Graph (Tarjan)

Find critical edges.

```
// Outline: DFS with discovery/low arrays; edge (u,v) is bridge if
low[v] > disc[u].
int main(){ return 0; }
```

Time Complexity: O(V+E)

Q93. KMP String Matching

Pattern search using lps[] array.

```
#include <stdio.h>
#include <string.h>
void lpsb(char* p,int m,int lps[]){ int len=0; lps[0]=0; for(int
i=1;i<m;){ if(p[i]==p[len]) lps[i++]=++len; else if(len) len=lps[len-
1]; else lps[i++]=0; } }
void kmp(char* t,char* p){ int n=strlen(t),m=strlen(p), lps[m];
lpsb(p,m,lps); for(int i=0,j=0;i<n;){ if(t[i]==p[j]){ i++; j++;
if(j==m){ printf("Found at %d\n", i-j); j=lps[j-1]; } } else if(j)
j=lps[j-1]; else i++; } }
int main(){ char t[]="abxabcabcaby", p[]="abcaby"; kmp(t,p); return 0;
}
```

Time Complexity: O(n+m)

Q94. Rabin–Karp String Matching

Rolling hash matching.

```
#include <stdio.h>
#include <string.h>
#define d 256
#define q 101
```

```
void rk(char* t,char* p){ int n=strlen(t),m=strlen(p); int h=1; for(int
i=0;i<m-1;i++) h=(h*d)%q; int ph=0, th=0;
for(int i=0;i<m;i++){ ph=(d*ph + p[i])%q; th=(d*th + t[i])%q; }
for(int i=0;i<=n-m;i++){ if(ph==th){ int j=0; while(j<m &&
t[i+j]==p[j]) j++; if(j==m) printf("Found at %d\n", i); } if(i<n-m){
th=(d*(th - t[i]*h) + t[i+m])%q; if(th<0) th+=q; } } }
int main(){ char t[]="GEEKS FOR GEEKS", p[]="GEEK"; rk(t,p); return 0;
}
```

Time Complexity: Average O(n+m)

Q95. Trie Auto-Complete (Prefix Listing)

DFS all words with given prefix.

```
// Outline: navigate to prefix node, then DFS collecting words.
int main(){ return 0; }
```

Time Complexity: O(k + output)

Q96. Segment Tree (Range Sum Query)

Build and query sums.

```
#include <stdio.h>
int st[400005], a[100005];
int build(int p,int l,int r){ if(l==r) return st[p]=a[l]; int
m=(l+r)/2; return st[p]=build(p*2,l,m)+build(p*2+1,m+1,r); }
int query(int p,int l,int r,int i,int j){ if(i>r||j<l) return 0;
if(i<=l&&r<=j) return st[p]; int m=(l+r)/2; return
query(p*2,l,m,i,j)+query(p*2+1,m+1,r,i,j); }
void update(int p,int l,int r,int idx,int val){ if(l==r){ st[p]=val;
return; } int m=(l+r)/2; if(idx<=m) update(p*2,l,m,idx,val); else
update(p*2+1,m+1,r,idx,val); st[p]=st[p*2]+st[p*2+1]; }
```

Time Complexity: Build O(n), Query/Update O(log n)

Q97. Fenwick Tree (BIT) for Prefix Sum

Point update, prefix query.

```
#include <stdio.h>
#define N 100005
int bit[N+1];
void add(int i,int v){ for(; i<=N; i+=i&-i) bit[i]+=v; }
int sum(int i){ int s=0; for(; i>0; i-=i&-i) s+=bit[i]; return s; }
```

Time Complexity: O(log n)

Q98. Binary Search on Answer

Min capacity to ship within D days (pattern).

```
// Outline: binary search on feasible answer; check() greedily verifies
feasibility.
int main(){ return 0; }
```

Time Complexity: O(n log R)

Q99. Two Stacks in One Array

Optimize space.

```
#include <stdio.h>
#define MAX 100
int a[MAX], t1=-1, t2=MAX;
void push1(int x){ if(t1+1==t2) return; a[++t1]=x; }
void push2(int x){ if(t1+1==t2) return; a[--t2]=x; }
int pop1(){ return t1==-1?-1:a[t1--]; }
int pop2(){ return t2==MAX?-1:a[t2++]; }
```

Time Complexity: O(1)

Q100. Circular Linked List: Josephus

Find survivor position.

```
#include <stdio.h>
int josephus(int n,int k){ int r=0; for(int i=1;i<=n;i++) r=(r+k)%i;
return r+1; }
int main(){ printf("%d", josephus(7,3)); return 0; }
```

Time Complexity: O(n)

Q101. LRU Cache (Linked List + Hash Map Outline)

Typical design question.

```
// Outline: doubly linked list for recency, hashmap for O(1) lookup;
move node to head on access; evict tail on capacity.
int main(){ return 0; }
```

Time Complexity: O(1) ops

Q102. Binary Search Tree to DLL (Inorder)

Convert BST to sorted doubly linked list.

```
// Outline: inorder traverse, link prev and current nodes to form DLL.
int main(){ return 0; }
```

Time Complexity: O(n)