DSA: 50 Programs — Question, C Code, and Output (Black & White)

1. Write a program to declare and initialize a 1D array.

```
#include <stdio.h>

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

Output: 1 2 3 4 5

2. Input and display elements of a 1D array.

```
#include <stdio.h>

int main() {
    int n, i;
    printf(\"Enter number of elements:\\n\");
    scanf(\"%d\", &n);
    int a[n];
    for (i = 0; i < n; i++) {
        scanf(\"%d\", &a[i]);
    }
    for (i = 0; i < n; i++) {
        printf(\"%d \", a[i]);
    }
    printf(\"\\n\");
    return 0;
}
```

Output: Example: 10 20 30 40 -> 10 20 30 40

3. Find the largest and smallest element in an array.

```
#include <stdio.h>

int main() {
    int n, i;
    scanf(\"%d\", &n);
    int a[n];
    for (i = 0; i < n; i++) {
        scanf(\"%d\", &a[i]);
    }
    int max = a[0];
    int min = a[0];
    for (i = 1; i < n; i++) {
        if (a[i] > max) {
            max = a[i];
        }
        if (a[i] < min) {
            min = a[i];
        }
    }
    printf(\"Max=%d\\n\", max);
    printf(\"Min=%d\\n\", min);
    return 0;
}
```

Output: Max=9 Min=1

4. Calculate sum and average of elements in an array.

```
#include <stdio.h>

int main() {
    int n, i;
    scanf(\"%d\", &n);
    int a[n];
    long sum = 0;
    for (i = 0; i < n; i++) {
        scanf(\"%d\", &a[i]);
        sum += a[i];
    }
    printf(\"Sum=%ld\\n\", sum);
    printf(\"Average=%.2f\\n\", (double)sum / n);
    return 0;
}
```

Output: Sum=30 Average=7.50

5. Insert an element at a given position in an array.

```
#include <stdio.h>

int main() {
    int n, i, pos, x;
    scanf(\"%d\", &n);
    int a[100];
    for (i = 0; i < n; i++) {
        scanf(\"%d\", &a[i]);
    }
    scanf(\"%d %d\", &pos, &x); /* pos: 0-based */
    for (i = n; i > pos; i--) {
        a[i] = a[i - 1];
    }
    a[pos] = x;
    n++;
    for (i = 0; i < n; i++) {
        printf(\"%d \", a[i]);
    }
    printf(\"\\n\");
    return 0;
}
```

Output: 1 2 3 4 -> insert 3 at pos 2 -> 1 2 3 3 4

6. Delete an element from an array.

```
#include <stdio.h>

int main() {
    int n, i, pos;
    scanf(\"%d\", &n);
    int a[100];
    for (i = 0; i < n; i++) {
        scanf(\"%d\", &a[i]);
    }
    scanf(\"%d\", &pos); /* index to delete (0-based) */
    for (i = pos; i < n - 1; i++) {
        a[i] = a[i + 1];
    }
    n--;
    for (i = 0; i < n; i++) {
        printf(\"%d \", a[i]);
    }
    printf(\"\\n\");
    return 0;
}
```

Output: 1 2 3 4 5 -> delete pos 1 -> 1 3 4 5

7. Merge two arrays into one.

```
#include <stdio.h>

int main() {
    int n1, n2, i;
    scanf(\"%d\", &n1);
    int a[n1];
    for (i = 0; i < n1; i++) {
        scanf(\"%d\", &a[i]);
    }
    scanf(\"%d\", &n2);
    int b[n2];
    for (i = 0; i < n2; i++) {
        scanf(\"%d\", &b[i]);
    }
    for (i = 0; i < n1; i++) {
        printf(\"%d \", a[i]);
    }
    for (i = 0; i < n2; i++) {
        printf(\"%d \", b[i]);
    }
    printf(\"\\n\");
    return 0;
}
```

Output: 1 3 5 + 2 4 -> 1 3 5 2 4

8. Add two matrices.

```
#include <stdio.h>

int main() {
    int r, c, i, j;
    scanf(\"%d %d\", &r, &c);
    int A[r][c], B[r][c];
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            scanf(\"%d\", &A[i][j]);
        }
    }
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            scanf(\"%d\", &B[i][j]);
        }
    }
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            printf(\"%d \", A[i][j] + B[i][j]);
        }
        printf(\"\\n\");
    }
    return 0;
}
```

Output: 6 8 10 12

9. Subtract two matrices.

```
#include <stdio.h>

int main() {
    int r, c, i, j;
    scanf(\"%d %d\", &r, &c);
    int A[r][c], B[r][c];
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            scanf(\"%d\", &A[i][j]);
```

```
        }
    }
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            scanf(\"%d\", &B[i][j]);
        }
    }
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            printf(\"%d \", A[i][j] - B[i][j]);
        }
        printf(\"\\n\");
    }
    return 0;
}
```

Output: 4 4 4 4

10. Multiply two matrices.

```
#include <stdio.h>

int main() {
    int r1, c1, r2, c2, i, j, k;
    scanf(\"%d %d\", &r1, &c1);
    int A[r1][c1];
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c1; j++) {
            scanf(\"%d\", &A[i][j]);
        }
    }
    scanf(\"%d %d\", &r2, &c2);
    int B[r2][c2];
    for (i = 0; i < r2; i++) {
        for (j = 0; j < c2; j++) {
            scanf(\"%d\", &B[i][j]);
        }
    }
    if (c1 != r2) {
        printf(\"Incompatible\\n\");
        return 0;
    }
    int C[r1][c2];
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c2; j++) {
            C[i][j] = 0;
            for (k = 0; k < c1; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c2; j++) {
            printf(\"%d \", C[i][j]);
        }
        printf(\"\\n\");
    }
    return 0;
}
```

Output: 19 22 43 50

11. Define stack structure and demonstrate push/pop operations.

```
#include <stdio.h>
#define MAX 100
typedef struct {
    int items[MAX];
    int top;
} Stack;

void push(Stack *s, int x) {
    if (s->top == MAX - 1) return;
    s->items[++s->top] = x;
}

int pop(Stack *s) {
    if (s->top == -1) return -1;
    return s->items[s->top--];
}

int main() {
    Stack s;
    s.top = -1;
    push(&s, 10);
```

```
    push(&s, 20);
    printf(\"Pop: %d\\n\", pop(&s));
    printf(\"Pop: %d\\n\", pop(&s));
    return 0;
}
```
Output: Pop: 20 Pop: 10

12. Implement stack using array.

```
#include <stdio.h>
#define MAX 50
int stack[MAX];
int top = -1;

void push(int x) {
    if (top == MAX - 1) {
        printf(\"Overflow\\n\");
        return;
    }
    stack[++top] = x;
}

int pop() {
    if (top == -1) {
        printf(\"Underflow\\n\");
        return -1;
    }
    return stack[top--];
}

int main() {
    push(5);
    push(6);
    printf(\"%d\\n\", pop());
    printf(\"%d\\n\", pop());
    return 0;
}
```
Output: 6 5

13. Implement stack using linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};

struct Node* top = NULL;

void push(int x) {
    struct Node* n = malloc(sizeof(*n));
    n->data = x;
    n->next = top;
    top = n;
}

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

int main() {
    push(1);
    push(2);
    printf(\"%d\\n\", pop());
    printf(\"%d\\n\", pop());
    return 0;
}
```

Output: 2 1

14. Check stack overflow and underflow conditions.

```
#include <stdio.h>
#define MAX 2
int stack[MAX];
int top = -1;

void push(int x) {
    if (top == MAX - 1) {
        printf(\"Overflow\\n\");
        return;
    }
    stack[++top] = x;
}

int pop() {
    if (top == -1) {
        printf(\"Underflow\\n\");
        return -1;
    }
    return stack[top--];
}

int main() {
    push(1);
    push(2);
    push(3); /* causes overflow */
    pop();
    pop();
    pop(); /* causes underflow */
```

```
    return 0;
}
```
Output: Overflow Underflow

15. Reverse a string using stack.

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

int main() {
    char s[100];
    fgets(s, sizeof(s), stdin);
    int n = strlen(s);
    if (n > 0 && s[n-1] == '\\n') s[n-1] = '\\0', n--;
    char st[100];
    int top = -1;
    int i;
    for (i = 0; i < n; i++) {
        st[++top] = s[i];
    }
    for (i = 0; i < n; i++) {
        s[i] = st[top--];
    }
    s[n] = '\\0';
    printf(\"%s\\n\", s);
    return 0;
}
```
Output: hello -> olleh

16. Define and demonstrate basic queue operations (enqueue/dequeue).

```
#include <stdio.h>
#define MAX 50
int q[MAX];
int front = 0, rear = 0;

void enqueue(int x) {
    if (rear == MAX) {
        printf(\"Overflow\\n\");
        return;
    }
    q[rear++] = x;
}

int dequeue() {
    if (front == rear) {
        printf(\"Underflow\\n\");
        return -1;
    }
    return q[front++];
}

int main() {
    enqueue(10);
    enqueue(20);
    printf(\"%d\\n\", dequeue());
    printf(\"%d\\n\", dequeue());
    return 0;
}
```

Output: 10 20

17. Implement queue using array.

```
#include <stdio.h>
#define MAX 100
int q[MAX];
int front = -1, rear = -1;

void enqueue(int x) {
    if (rear == MAX - 1) {
        printf(\"Overflow\\n\");
        return;
    }
    if (front == -1) front = 0;
    q[++rear] = x;
}

int dequeue() {
    if (front == -1 || front > rear) {
        printf(\"Underflow\\n\");
        return -1;
    }
    return q[front++];
}

int main() {
    enqueue(1);
    enqueue(2);
    printf(\"%d\\n\", dequeue());
    return 0;
}
```

Output: 1

18. Implement queue using linked list.

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

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

struct Node *front = NULL, *rear = NULL;

void enqueue(int x) {
    struct Node* n = malloc(sizeof(*n));
    n->data = x;
    n->next = NULL;
    if (!rear) front = rear = n;
    else { rear->next = n; rear = n; }
}

int dequeue() {
    if (!front) return -1;
    int v = front->data;
    struct Node* t = front;
    front = front->next;
    if (!front) rear = NULL;
    free(t);
    return v;
}

int main() {
    enqueue(5);
    enqueue(6);
    printf(\"%d\\n\", dequeue());
    return 0;
}
```
Output: 5

19. Implement circular queue using array.

```
#include <stdio.h>
#define MAX 5
int q[MAX];
int front = -1, rear = -1;

int isFull() {
    return (front == 0 && rear == MAX - 1) || (rear == (front - 1) % (MAX - 1));
}

int isEmpty() {
    return front == -1;
}

void enqueue(int x) {
    if (isFull()) {
        printf(\"Overflow\\n\");
        return;
    }
    if (front == -1) {
        front = rear = 0;
        q[rear] = x;
        return;
    }
    rear = (rear + 1) % MAX;
    q[rear] = x;
}

int dequeue() {
    if (isEmpty()) {
        printf(\"Underflow\\n\");
        return -1;
    }
    int val = q[front];
    if (front == rear) front = rear = -1;
    else front = (front + 1) % MAX;
    return val;
}

int main() {
    enqueue(1);
    enqueue(2);
    enqueue(3);
    printf(\"%d\\n\", dequeue());
    return 0;
}
```

Output: 1

20. Implement double-ended queue (Deque).

```
#include <stdio.h>
#define MAX 100
int deq[MAX];
int front = -1, rear = -1;

int isEmpty() {
    return front == -1;
}

void insertFront(int x) {
    if (front == 0) {
        printf(\"Overflow\\n\");
        return;
    }
    if (isEmpty()) {
```

```
        front = rear = 0;
        deq[front] = x;
    } else {
        deq[--front] = x;
    }
}

void insertRear(int x) {
    if (rear == MAX - 1) {
        printf(\"Overflow\\n\");
        return;
    }
    if (isEmpty()) {
        front = rear = 0;
        deq[rear] = x;
    } else {
        deq[++rear] = x;
    }
}

int deleteFront() {
    if (isEmpty()) {
        printf(\"Underflow\\n\");
        return -1;
    }
    return deq[front++];
}

int deleteRear() {
    if (isEmpty()) {
        printf(\"Underflow\\n\");
        return -1;
    }
    return deq[rear--];
}

int main() {
    insertRear(1);
    insertFront(2);
    printf(\"%d\\n\", deleteRear());
    return 0;
}
```

Output: 1

21. Implement priority queue using array.

```
#include <stdio.h>

int main() {
    int n = 5;
    int pq[5] = {3, 5, 1, 4, 2};
    int i, j;
    for (i = 0; i < n - 1; i++) {
        for (j = i + 1; j < n; j++) {
            if (pq[i] < pq[j]) {
                int t = pq[i];
                pq[i] = pq[j];
                pq[j] = t;
            }
        }
    }
    for (i = 0; i < n; i++) {
        printf(\"%d \", pq[i]);
    }
    printf(\"\\n\");
    return 0;
```

```
}
```
Output: 5 4 3 2 1

22. Simulate a queue using two stacks.

```
#include <stdio.h>
#define MAX 100
int s1[MAX], s2[MAX], t1 = -1, t2 = -1;

void push1(int x) { s1[++t1] = x; }
int pop1() { if (t1 == -1) return -1; return s1[t1--]; }
void push2(int x) { s2[++t2] = x; }
int pop2() { if (t2 == -1) return -1; return s2[t2--]; }

void enqueue(int x) { push1(x); }

int dequeue() {
    if (t2 == -1) {
        while (t1 != -1) push2(pop1());
    }
    return pop2();
}

int main() {
    enqueue(1);
    enqueue(2);
    printf(\"%d\\n\", dequeue());
    return 0;
}
```

Output: 1

23. Demonstrate queue overflow and underflow.

```
#include <stdio.h>
#define MAX 2
int q[MAX], front = 0, rear = 0;

void enqueue(int x) {
    if (rear == MAX) {
        printf(\"Overflow\\n\");
        return;
    }
    q[rear++] = x;
}

int dequeue() {
    if (front == rear) {
        printf(\"Underflow\\n\");
        return -1;
    }
    return q[front++];
}

int main() {
    enqueue(1);
    enqueue(2);
    enqueue(3);
    dequeue();
    dequeue();
    dequeue();
    return 0;
}
```

Output: Overflow Underflow

24. Define node structure of linked list.

```
#include <stdio.h>

struct Node {
```

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

int main() {
    struct Node n;
    n.data = 5;
    n.next = NULL;
    printf(\"%d\\n\", n.data);
    return 0;
}
```
Output: 5

25. Create and display a singly linked list.

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

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

int main() {
    struct Node *head = NULL, *temp;
    for (int i = 1; i <= 3; i++) {
        temp = malloc(sizeof(*temp));
        temp->data = i;
        temp->next = head;
        head = temp;
    }
    while (head) {
        printf(\"%d \", head->data);
        head = head->next;
    }
    return 0;
}
```

Output: 3 2 1

26. Insert node at the beginning of linked list.

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

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

int main() {
    struct Node *head = NULL, *n = malloc(sizeof(*n));
    n->data = 10;
    n->next = head;
    head = n;
    printf(\"%d\\n\", head->data);
    return 0;
}
```

Output: 10

27. Insert node at the end of linked list.

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

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

int main() {
    struct Node *head = NULL, *temp, *last;
    temp = malloc(sizeof(*temp));
    temp->data = 5;
    temp->next = NULL;
    head = temp;
    last = temp;
    temp = malloc(sizeof(*temp));
    temp->data = 10;
    temp->next = NULL;
```

```
    last->next = temp;
    last = temp;
    while (head) {
        printf(\"%d \", head->data);
        head = head->next;
    }
    return 0;
}
```

Output: 5 10

28. Insert node at any given position in linked list.

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

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

int main() {
    struct Node *head = NULL, *temp;
    temp = malloc(sizeof(*temp));
    temp->data = 1;
    temp->next = NULL;
    head = temp;
    struct Node *n = malloc(sizeof(*n));
    n->data = 2;
    n->next = head->next;
    head->next = n;
    while (head) {
        printf(\"%d \", head->data);
        head = head->next;
    }
    return 0;
}
```

Output: 1 2

29. Delete node from beginning.

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

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

int main() {
    struct Node *head = malloc(sizeof(*head));
    head->data = 1;
    struct Node *n = malloc(sizeof(*n));
    n->data = 2;
    head->next = n;
    n->next = NULL;
    struct Node *t = head;
    head = head->next;
    free(t);
    printf(\"%d\\n\", head->data);
    return 0;
}
```

Output: 2

30. Delete node from end.

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

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

int main() {
    struct Node *head = malloc(sizeof(*head)), *second;
    head->data = 1;
```

```
    second = malloc(sizeof(*second));
    second->data = 2;
    head->next = second;
    second->next = NULL;
    struct Node *cur = head;
    while (cur->next->next) cur = cur->next;
    free(cur->next);
    cur->next = NULL;
    cur = head;
    while (cur) {
        printf(\"%d \", cur->data);
        cur = cur->next;
    }
    return 0;
}
```

Output: 1

31. Delete node from any given position.

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

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

int main() {
    struct Node *head = NULL, *t;
    for (int i = 1; i <= 3; i++) {
        t = malloc(sizeof(*t));
        t->data = i;
        t->next = head;
        head = t;
    }
    struct Node *p = head;
    p = p->next; /* delete second node */
    head->next = p->next;
    free(p);
    while (head) {
        printf(\"%d \", head->data);
        head = head->next;
    }
    return 0;
}
```

Output: 3 1

32. Search an element in a linked list.

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

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

int main() {
    struct Node *head = NULL, *t;
    for (int i = 1; i <= 3; i++) {
        t = malloc(sizeof(*t));
        t->data = i;
        t->next = head;
        head = t;
    }
    int key = 2;
    int found = 0;
    while (head) {
        if (head->data == key) {
            found = 1;
            break;
        }
        head = head->next;
    }
    if (found) printf(\"Found\\n\"); else printf(\"Not Found\\n\");
    return 0;
}
```

Output: Found

33. Count total nodes in a linked list.

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

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

int main() {
    struct Node *head = NULL, *t;
    for (int i = 1; i <= 4; i++) {
        t = malloc(sizeof(*t));
        t->data = i;
        t->next = head;
        head = t;
    }
    int cnt = 0;
    while (head) {
        cnt++;
        head = head->next;
    }
    printf(\"Count=%d\\n\", cnt);
    return 0;
}
```

Output: Count=4

34. Create a simple binary tree (root, left, right).

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

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

int main() {
    struct Node *root = malloc(sizeof(*root));
    root->data = 1;
    root->left = malloc(sizeof(*root));
    root->right = malloc(sizeof(*root));
    root->left->data = 2;
    root->right->data = 3;
    printf(\"Root=%d Left=%d Right=%d\\n\", root->data, root->left->data, root->right->data);
    return 0;
}
```

Output: Root=1 Left=2 Right=3

35. Implement inorder traversal.

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

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

void inorder(struct Node* r) {
    if (!r) return;
    inorder(r->left);
    printf(\"%d \", r->data);
    inorder(r->right);
}

int main() {
    struct Node *root = malloc(sizeof(*root));
    root->data = 1;
    root->left = malloc(sizeof(*root));
    root->right = malloc(sizeof(*root));
    root->left->data = 2;
    root->right->data = 3;
    root->left->left = root->left->right = NULL;
    root->right->left = root->right->right = NULL;
    inorder(root);
    return 0;
}
```

Output: 2 1 3

36. Implement preorder traversal.

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};
void preorder(struct Node* r) {
```

```
    if (!r) return;
    printf(\"%d \", r->data);
    preorder(r->left);
    preorder(r->right);
}

int main() {
    struct Node *root = malloc(sizeof(*root));
    root->data = 1;
    root->left = malloc(sizeof(*root));
    root->right = malloc(sizeof(*root));
    root->left->data = 2;
    root->right->data = 3;
    root->left->left = root->left->right = NULL;
    root->right->left = root->right->right = NULL;
    preorder(root);
    return 0;
}
```

Output: 1 2 3

37. Implement postorder traversal.

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

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

void postorder(struct Node* r) {
    if (!r) return;
    postorder(r->left);
    postorder(r->right);
    printf(\"%d \", r->data);
}

int main() {
    struct Node *root = malloc(sizeof(*root));
    root->data = 1;
    root->left = malloc(sizeof(*root));
    root->right = malloc(sizeof(*root));
    root->left->data = 2;
    root->right->data = 3;
    root->left->left = root->left->right = NULL;
    root->right->left = root->right->right = NULL;
    postorder(root);
    return 0;
}
```

Output: 2 3 1

38. Insert a node in Binary Search Tree (BST).

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

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

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

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

int main() {
    struct Node* root = NULL;
    root = insert(root, 5);
    insert(root, 3);
    insert(root, 7);
    printf(\"Root=%d\\n\", root->data);
    return 0;
}
```

Output: Root=5

39. Search an element in BST.

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

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

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

int search(struct Node* r, int x) {
    if (!r) return 0;
    if (r->data == x) return 1;
    if (x < r->data) return search(r->left, x);
    return search(r->right, x);
}

int main() {
    struct Node* root = newNode(5);
    root->left = newNode(3);
    root->right = newNode(7);
    printf(search(root, 3) ? \"Found\\n\" : \"Not Found\\n\");
    return 0;
}
```

Output: Found

40. Find the height of a binary tree.

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

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

int height(struct Node* r) {
    if (!r) return 0;
    int l = height(r->left);
    int rr = height(r->right);
    return 1 + (l > rr ? l : rr);
}

int main() {
    struct Node* root = malloc(sizeof(*root));
    root->left = malloc(sizeof(*root));
    root->right = NULL;
    root->left->left = NULL;
    root->left->right = NULL;
    printf(\"%d\\n\", height(root));
    return 0;
}
```

Output: 2

41. Count total number of nodes in binary tree.

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

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

int countNodes(struct Node* r) {
    if (!r) return 0;
    return 1 + countNodes(r->left) + countNodes(r->right);
}

int main() {
    struct Node* root = malloc(sizeof(*root));
    root->left = malloc(sizeof(*root));
    root->right = NULL;
    root->left->left = NULL;
    root->left->right = NULL;
    printf(\"%d\\n\", countNodes(root));
    return 0;
}
```

Output: 2

42. Implement Linear (Sequential) Search.

```
#include <stdio.h>

int main() {
    int n;
    scanf(\"%d\", &n);
    int a[n];
    for (int i = 0; i < n; i++) scanf(\"%d\", &a[i]);
    int key;
```

```
    scanf(\"%d\", &key);
    for (int i = 0; i < n; i++) {
        if (a[i] == key) {
            printf(\"Found at %d\\n\", i);
            return 0;
        }
    }
    printf(\"Not Found\\n\");
    return 0;
}
```

Output: Found at 2

43. Implement Binary Search (Iterative).

```
#include <stdio.h>

int main() {
    int n;
    scanf(\"%d\", &n);
    int a[n];
    for (int i = 0; i < n; i++) scanf(\"%d\", &a[i]);
    int key;
    scanf(\"%d\", &key);
    int l = 0, r = n - 1;
    while (l <= r) {
        int m = (l + r) / 2;
        if (a[m] == key) {
            printf(\"Found at %d\\n\", m);
            return 0;
        }
        if (a[m] < key) l = m + 1;
        else r = m - 1;
    }
    printf(\"Not Found\\n\");
    return 0;
}
```

Output: Found at 3

44. Implement Hashing using Modulus Method.

```
#include <stdio.h>
#define SIZE 10

int main() {
    int table[SIZE];
    for (int i = 0; i < SIZE; i++) table[i] = -1;
    int keys[5] = {12, 22, 42, 32, 52};
    for (int i = 0; i < 5; i++) {
        int h = keys[i] % SIZE;
        while (table[h] != -1) h = (h + 1) % SIZE;
        table[h] = keys[i];
    }
    for (int i = 0; i < SIZE; i++) printf(\"%d \", table[i]);
    printf(\"\\n\");
    return 0;
}
```

Output: -1 -1 12 -1 -1 52 -1 22 42 32

45. Implement Bubble Sort.

```
#include <stdio.h>

int main() {
    int n;
    scanf(\"%d\", &n);
    int a[n];
    for (int i = 0; i < n; i++) scanf(\"%d\", &a[i]);
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - 1 - i; 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]);
    printf(\"\\n\");
```

```
    return 0;
}
```
Output: 1 2 4 5 8

46. Implement Selection Sort.

```
#include <stdio.h>

int main() {
    int n;
    scanf(\"%d\", &n);
    int a[n];
    for (int i = 0; i < n; i++) scanf(\"%d\", &a[i]);
    for (int i = 0; i < n - 1; i++) {
        int min = i;
        for (int j = i + 1; j < n; j++) {
            if (a[j] < a[min]) min = j;
        }
        int t = a[i];
        a[i] = a[min];
        a[min] = t;
    }
    for (int i = 0; i < n; i++) printf(\"%d \", a[i]);
    printf(\"\\n\");
    return 0;
}
```

Output: 11 12 22 25 64

47. Implement Insertion Sort.

```
#include <stdio.h>

int main() {
    int n;
    scanf(\"%d\", &n);
    int a[n];
    for (int i = 0; i < n; i++) scanf(\"%d\", &a[i]);
    for (int i = 1; i < n; i++) {
        int key = a[i];
        int 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]);
    printf(\"\\n\");
    return 0;
}
```

Output: 5 6 11 12 13

48. Implement Quick Sort.

```
#include <stdio.h>

void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int partition(int a[], int l, int r) {
    int pivot = a[r];
    int i = l - 1;
    for (int j = l; j < r; j++) {
        if (a[j] < pivot) {
            i++;
            swap(&a[i], &a[j]);
        }
```

```
    }
    swap(&a[i + 1], &a[r]);
    return i + 1;
}

void quick(int a[], int l, int r) {
    if (l < r) {
        int p = partition(a, l, r);
        quick(a, l, p - 1);
        quick(a, p + 1, r);
    }
}

int main() {
    int n;
    scanf(\"%d\", &n);
    int a[n];
    for (int i = 0; i < n; i++) scanf(\"%d\", &a[i]);
    quick(a, 0, n - 1);
    for (int i = 0; i < n; i++) printf(\"%d \", a[i]);
    printf(\"\\n\");
    return 0;
}
```

Output: 1 7 8 9 10

49. Implement Merge Sort.

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

void merge(int a[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    int i, j, k;
    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; k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) a[k++] = L[i++];
        else a[k++] = R[j++];
    }
    while (i < n1) a[k++] = L[i++];
    while (j < n2) a[k++] = R[j++];
}

void mergeSort(int a[], int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        mergeSort(a, l, m);
        mergeSort(a, m + 1, r);
        merge(a, l, m, r);
    }
}

int main() {
    int n;
    scanf(\"%d\", &n);
    int a[n];
    for (int i = 0; i < n; i++) scanf(\"%d\", &a[i]);
    mergeSort(a, 0, n - 1);
    for (int i = 0; i < n; i++) printf(\"%d \", a[i]);
    printf(\"\\n\");
    return 0;
}
```

Output: 3 9 27 38 43

50. Implement Heap Sort.

```
#include <stdio.h>

void heapify(int a[], int n, int i) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < n && a[l] > a[largest]) largest = l;
    if (r < n && a[r] > a[largest]) largest = r;
    if (largest != i) {
        int t = a[i];
        a[i] = a[largest];
        a[largest] = t;
        heapify(a, n, largest);
    }
}

void heapSort(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 n;
    scanf(\"%d\", &n);
    int a[n];
    for (int i = 0; i < n; i++) scanf(\"%d\", &a[i]);
    heapSort(a, n);
    for (int i = 0; i < n; i++) printf(\"%d \", a[i]);
    printf(\"\\n\");
    return 0;
}
```

Output: 5 6 7 11 12 13