1. Time Complexity: Constant vs Linear

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

// Constant Time O(1)
int constant_example(int arr[]) {
    return arr[0]; // Accessing the first element
}

// Linear Time O(n)
void linear_example(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Constant Time Example: %d\n", constant_example(arr));
    printf("Linear Time Example: ");
    linear_example(arr, n);
    return 0;
}
```

---

2. Divide and Conquer: Merge Sort

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

void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0; j = 0; k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
```

```
}

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

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array: ");
    printArray(arr, size);

    mergeSort(arr, 0, size - 1);

    printf("Sorted array: ");
    printArray(arr, size);
    return 0;
}
```

---

3. Greedy Algorithm: Fractional Knapsack Problem

c
Copy code

```
#include <stdio.h>

void fractionalKnapsack(int weight[], int profit[], int n, int capacity) {
    float ratio[n], totalProfit = 0;
    int i, j;

    for (i = 0; i < n; i++)
        ratio[i] = (float)profit[i] / weight[i];

    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (ratio[j] < ratio[j + 1]) {
                float temp = ratio[j];
                ratio[j] = ratio[j + 1];
                ratio[j + 1] = temp;

                int tempW = weight[j];
                weight[j] = weight[j + 1];
                weight[j + 1] = tempW;

                int tempP = profit[j];
                profit[j] = profit[j + 1];
                profit[j + 1] = tempP;
            }
        }
    }

    for (i = 0; i < n; i++) {
        if (weight[i] <= capacity) {
            capacity -= weight[i];
            totalProfit += profit[i];
        } else {
            totalProfit += ratio[i] * capacity;
            break;
```

```
        }
    }

    printf("Maximum Profit: %.2f\n", totalProfit);
}

int main() {
    int weight[] = {10, 20, 30};
    int profit[] = {60, 100, 120};
    int capacity = 50;
    int n = sizeof(weight) / sizeof(weight[0]);

    fractionalKnapsack(weight, profit, n, capacity);
    return 0;
}
```

---

4. Dynamic Programming: Fibonacci Sequence

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

int fibonacci(int n) {
    int dp[n + 1];
    dp[0] = 0;
    dp[1] = 1;

    for (int i = 2; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 2];
    }

    return dp[n];
}

int main() {
    int n = 10;
    printf("Fibonacci of %d: %d\n", n, fibonacci(n));
    return 0;
}
```

---

5. N-Queens Problem

c
Copy code
```
#include <stdbool.h>
#include <stdio.h>

#define N 4

void printSolution(int board[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf("%d ", board[i][j]);
        printf("\n");
    }
}

bool isSafe(int board[N][N], int row, int col) {
    for (int i = 0; i < col; i++)
        if (board[row][i])
            return false;

    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;

    for (int i = row, j = col; i < N && j >= 0; i++, j--)
        if (board[i][j])
```

```
            return false;

    return true;
}

bool solveNQUtil(int board[N][N], int col) {
    if (col >= N)
        return true;

    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col)) {
            board[i][col] = 1;

            if (solveNQUtil(board, col + 1))
                return true;

            board[i][col] = 0; // Backtrack
        }
    }

    return false;
}

void solveNQ() {
    int board[N][N] = {0};

    if (!solveNQUtil(board, 0)) {
        printf("Solution does not exist\n");
        return;
    }

    printSolution(board);
}

int main() {
    solveNQ();
    return 0;
}
```

6. Huffman Coding

Huffman coding implementation is a bit complex. Here's a simplified illustration to build a Huffman tree and calculate the codes.

c
Copy code

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

// Node structure
struct Node {
    char data;
    unsigned freq;
    struct Node *left, *right;
};

// Create a new node
struct Node* createNode(char data, unsigned freq) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->freq = freq;
    node->left = node->right = NULL;
    return node;
}

// In a real implementation, this would include a min-heap, priority queue,
// and the Huffman encoding logic. This is a simplified representation.

void printCodes(struct Node* root, int arr[], int top) {
    if (root->left) {
```

```
        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }

    if (root->right) {
        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }

    if (!root->left && !root->right) {
        printf("%c: ", root->data);
        for (int i = 0; i < top; i++)
            printf("%d", arr[i]);
        printf("\n");
    }
}

int main() {
    struct Node* root = createNode('$', 0);
    root->left = createNode('A', 5);
    root->right = createNode('B', 9);

    int arr[10], top = 0;
    printf("Huffman Codes:\n");
    printCodes(root, arr, top);
    return 0;
}
```

---

7. Longest Common Subsequence

c
Copy code

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

int max(int a, int b) {
    return (a > b) ? a : b;
}

void lcs(char* X, char* Y, int m, int n) {
    int 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] = max(L[i - 1][j], L[i][j - 1]);
        }
    }

    printf("Length of LCS: %d\n", L[m][n]);
}

int main() {
    char X[] = "ABCBDAB";
    char Y[] = "BDCAB";

    int m = strlen(X);
    int n = strlen(Y);

    lcs(X, Y, m, n);
    return 0;
}
```

---

8. Matrix Chain Multiplication

c
Copy code
```c
#include <stdio.h>
#include <limits.h>

void matrixChainOrder(int p[], int n) {
    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] = INT_MAX;
            for (int k = i; k <= j - 1; k++) {
                int q = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
                if (q < m[i][j])
                    m[i][j] = q;
            }
        }
    }

    printf("Minimum number of multiplications: %d\n", m[1][n - 1]);
}

int main() {
    int arr[] = {1, 2, 3, 4};
    int n = sizeof(arr) / sizeof(arr[0]);
    matrixChainOrder(arr, n);
    return 0;
}
```

---

9. Job Sequencing Problem

c
Copy code
```c
#include <stdio.h>
#include <stdbool.h>

struct Job {
    char id;
    int deadline;
    int profit;
};

void jobSequencing(struct Job jobs[], int n) {
    bool slot[n];
    char result[n];

    for (int i = 0; i < n; i++)
        slot[i] = false;

    for (int i = 0; i < n; i++) {
        for (int j = jobs[i].deadline - 1; j >= 0; j--) {
            if (!slot[j]) {
                result[j] = jobs[i].id;
                slot[j] = true;
                break;
            }
        }
    }

    printf("Job sequence: ");
    for (int i = 0; i < n; i++)
        if (slot[i])
            printf("%c ", result[i]);
    printf("\n");
}
```

```
int main() {
    struct Job jobs[] = {{'A', 2, 100}, {'B', 1, 19}, {'C', 2, 27}, {'D', 1, 25}, {'E', 3, 15}};
    int n = sizeof(jobs) / sizeof(jobs[0]);

    jobSequencing(jobs, n);
    return 0;
}
```

---

10. Strassen’s Matrix Multiplication

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

void strassenMultiply(int A[2][2], int B[2][2], int C[2][2]) {
    int P1 = A[0][0] * (B[0][1] - B[1][1]);
    int P2 = (A[0][0] + A[0][1]) * B[1][1];
    int P3 = (A[1][0] + A[1][1]) * B[0][0];
    int P4 = A[1][1] * (B[1][0] - B[0][0]);
    int P5 = (A[0][0] + A[1][1]) * (B[0][0] + B[1][1]);
    int P6 = (A[0][1] - A[1][1]) * (B[1][0] + B[1][1]);
    int P7 = (A[0][0] - A[1][0]) * (B[0][0] + B[0][1]);

    C[0][0] = P5 + P4 - P2 + P6;
    C[0][1] = P1 + P2;
    C[1][0] = P3 + P4;
    C[1][1] = P5 + P1 - P3 - P7;
}

int main() {
    int A[2][2] = {{1, 2}, {3, 4}};
    int B[2][2] = {{5, 6}, {7, 8}};
    int C[2][2];

    strassenMultiply(A, B, C);

    printf("Resultant Matrix:\n");
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

11. Recurrence Relation: Fibonacci Sequence

This program demonstrates the recurrence relation F(n)=F(n−1)+F(n−2)F(n) = F(n-1) + F(n-2)F(n)=F(n−1)+F(n−2).

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

int fibonacci(int n) {
    if (n <= 1)
        return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    int n = 10;
    printf("Fibonacci Sequence: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));
    }
    printf("\n");
```

```
    return 0;
}
```

12. Vertex Cover Problem

For simplicity, this program demonstrates a brute-force approach to find a vertex cover for a graph.

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

#define V 4 // Number of vertices

void printVertexCover(int graph[V][V]) {
    int visited[V] = {0};

    for (int u = 0; u < V; u++) {
        if (!visited[u]) {
            for (int v = 0; v < V; v++) {
                if (graph[u][v] && !visited[v]) {
                    visited[u] = 1;
                    visited[v] = 1;
                    printf("Edge (%d, %d) is covered\n", u, v);
                    break;
                }
            }
        }
    }

    printf("Vertex Cover: ");
    for (int i = 0; i < V; i++) {
        if (visited[i])
            printf("%d ", i);
    }
    printf("\n");
}

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

    printVertexCover(graph);
    return 0;
}
```

13. N-Queen Problem (Backtracking)

c
Copy code
```
#include <stdbool.h>
#include <stdio.h>

#define N 4

void printSolution(int board[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf("%d ", board[i][j]);
        printf("\n");
    }
}
```

```
bool isSafe(int board[N][N], int row, int col) {
    for (int i = 0; i < col; i++)
        if (board[row][i])
            return false;

    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;

    for (int i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}

bool solveNQUtil(int board[N][N], int col) {
    if (col >= N)
        return true;

    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col)) {
            board[i][col] = 1;

            if (solveNQUtil(board, col + 1))
                return true;

            board[i][col] = 0;
        }
    }

    return false;
}

void solveNQ() {
    int board[N][N] = {0};

    if (!solveNQUtil(board, 0)) {
        printf("Solution does not exist\n");
        return;
    }

    printSolution(board);
}

int main() {
    solveNQ();
    return 0;
}
```

---

14. Ant Colony Optimization Algorithm (Conceptual Example)

Due to the complexity of ACO, this is a simplified example to demonstrate the concept of optimization.

c
Copy code

```
#include <stdio.h>

#define N 4

void antColonyOptimization() {
    int distance[N][N] = {
        {0, 2, 2, 3},
        {2, 0, 3, 2},
        {2, 3, 0, 1},
        {3, 2, 1, 0}
    };
```

```
    int pheromone[N][N] = {0}; // Initial pheromone levels

    printf("Distance Matrix:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", distance[i][j]);
        }
        printf("\n");
    }

    printf("\nPheromone Matrix (Initially):\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            pheromone[i][j] = 1; // Initialize with uniform pheromone
            printf("%d ", pheromone[i][j]);
        }
        printf("\n");
    }
}

int main() {
    antColonyOptimization();
    return 0;
}
```