1 . Write a C program to Graph traversal using Depth First Search.

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
int adj[MAX][MAX], visited[MAX], n;
void DFS(int v) {
    printf("%d ", v);
    visited[v] = 1;
    for (int i = 0; i < n; i++) {
        if (adj[v][i] && !visited[i]) {
            DFS(i);
        }}}
int main() {
    int edges, i, j, v1, v2, start;
 printf("Enter number of vertices: ");
    scanf("%d", &n);
printf("Enter number of edges: ");
    scanf("%d", &edges);
    for (i = 0; i < n; i++) {
        visited[i] = 0;
        for (j = 0; j < n; j++)
            adj[i][j] = 0;
```

```
Enter number of vertices: 3
Enter number of edges: 3
Enter edges (v1 v2):
1 2
2 3
3 6
Enter starting vertex: 1
DFS traversal: 1 2

=== Code Execution Successful ===
```

```c
    }
 printf("Enter edges (v1 v2):\n");
    for (i = 0; i < edges; i++) {
        scanf("%d%d", &v1, &v2);
        adj[v1][v2] = 1;
        adj[v2][v1] = 1;
    }
  printf("Enter starting vertex: ");
    scanf("%d", &start);
printf("DFS traversal: ");
    DFS(start);
return 0;
}
```

RESULT : Code executed successfully.

2. Implementation of Shortest Path Algorithms using Dijkstra's Algorithm.

```c
#include <stdio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX], int n, int startnode) {
    int cost[MAX][MAX], distance[MAX], pred[MAX];
    int visited[MAX], count, mindistance, nextnode, i, j;
 for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if (G[i][j] == 0)
                cost[i][j] = INFINITY;
            else
                cost[i][j] = G[i][j];
for (i = 0; i < n; i++) {
        distance[i] = cost[startnode][i];
        pred[i] = startnode;
        visited[i] = 0;
    }
distance[startnode] = 0;
    visited[startnode] = 1;
    count = 1;
while (count < n - 1) {
```

```
Enter number of vertices: 3
Enter the adjacency matrix:
3

1 2
3 4
5 6
6 8
Enter starting node: 4

Distance from node 4 to node 0: -566927842
Path: 0 <- 2 <- 4

Distance from node 4 to node 1: -566927842
Path: 1 <- 2 <- 4

Distance from node 4 to node 2: -566927848
Path: 2 <- 4


=== Code Execution Successful ===
```

```c
        mindistance = INFINITY;
  for (i = 0; i < n; i++)
            if (distance[i] < mindistance && !visited[i]) {
                mindistance = distance[i];
                nextnode = i;
            }
  visited[nextnode] = 1;
 for (i = 0; i < n; i++)
            if (!visited[i])
                if (mindistance + cost[nextnode][i] < distance[i]) {
                    distance[i] = mindistance + cost[nextnode][i];
                    pred[i] = nextnode;
                }
        count++;
    }
    for (i = 0; i < n; i++)
        if (i != startnode) {
            printf("\nDistance from node %d to node %d: %d",
                startnode, i, distance[i]);
            printf("\nPath: %d", i);
```

```
41  j = i;
42              do {
43                  j = pred[j];
44                  printf(" <- %d", j);
45              } while (j != startnode);
46              printf("\n");
47          }}
48  int main() {
49      int G[MAX][MAX], i, j, n, u;
50   printf("Enter number of vertices: ");
51      scanf("%d", &n);
52    printf("Enter the adjacency matrix:\n");
53      for (i = 0; i < n; i++)
54          for (j = 0; j < n; j++)
55              scanf("%d", &G[i][j]);
56    printf("Enter starting node: ");
57      scanf("%d", &u);
58    dijkstra(G, n, u);
59    return 0;
```

3 . Implementation of Minimum Spanning Tree using Prim'sAlgorithm.

main.c

```
#include <stdio.h>
#define INF 9999
#define MAX 10
int main() {
    int G[MAX][MAX], visited[MAX], n, i, j, ne = 1;
    int min, a, b, u, v;
printf("Enter number of vertices: ");
    scanf("%d", &n);
printf("Enter adjacency matrix:\n");
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++) {
            scanf("%d", &G[i][j]);
            if (G[i][j] == 0)
                G[i][j] = INF;
        }
for (i = 1; i <= n; i++)
        visited[i] = 0;
visited[1] = 1;
 printf("Edges of Minimum Spanning Tree:\n");
    while (ne < n) {
```

```
Enter number of vertices: 4
Enter adjacency matrix:
4
1 2 3
5 6 7
3 4 6
5 8 7
3 5 8
Edges of Minimum Spanning Tree:
1 edge (1, 2) = 1
2 edge (1, 3) = 2
3 edge (1, 4) = 3


=== Code Execution Successful ===
```

```
        min = INF;
        for (i = 1; i <= n; i++) {
            if (visited[i]) {
                for (j = 1; j <= n; j++) {
                    if (!visited[j] && G[i][j] < min) {
                        min = G[i][j];
                        u = i;
                        v = j;
                    } } }}
  printf("%d edge (%d, %d) = %d\n", ne++, u, v, min);
        visited[v] = 1;
    }
  return 0;
```

Result: code executed successfully.

4. Implementation of Minimum Spanning Tree using Kruskal Algorithm.

```c
#include <stdio.h>
#define MAX 30
int parent[MAX];
int find(int i) {
    while (parent[i])
        i = parent[i];
    return i;
}
int uni(int i, int j) {
    if (i != j) {
        parent[j] = i;
        return 1;
    }
    return 0;
}
int main() {
    int G[MAX][MAX], i, j, k, n, ne = 1;
    int min, a, b, u, v, cost = 0;
printf("Enter number of vertices: ");
    scanf("%d", &n);
```

```
Enter number of vertices: 2
Enter adjacency matrix:
2
1 2
6 7
Edges of Minimum Spanning Tree:
1 edge (1, 2) = 1
Minimum cost = 1


=== Code Execution Successful ===
```

```c
printf("Enter adjacency matrix:\n");
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++) {
            scanf("%d", &G[i][j]);
            if (G[i][j] == 0)
                G[i][j] = 9999;
        }
printf("Edges of Minimum Spanning Tree:\n");
    while (ne < n) {
        min = 9999;
        for (i = 1; i <= n; i++)
            for (j = 1; j <= n; j++)
                if (G[i][j] < min) {
                    min = G[i][j];
                    a = u = i;
                    b = v = j;
                }
 u = find(u);
        v = find(v);
```

```c
if (uni(u, v)) {
                printf("%d edge (%d, %d) = %d\n", ne++, a, b, min);
                cost += min;
        }
   G[a][b] = G[b][a] = 9999;
    }
printf("Minimum cost = %d\n", cost);
    return 0;
```

Result: code executed successfully.

5. Reversing a 32bit signed integers

main.c

```c
#include <stdio.h>

int main() {
    int num, reversed = 0, rem;

    printf("Enter a 32-bit signed integer: ");
    scanf("%d", &num);

    while (num != 0) {
        rem = num % 10;
        if (reversed > 214748364 || reversed < -214748364) {
            printf("Overflow! Cannot reverse safely.\n");
            return 0;
        }
        reversed = reversed * 10 + rem;
        num /= 10;
    }

    printf("Reversed integer: %d\n", reversed);
    return 0;
```

```
Enter a 32-bit signed integer: 12345678
Reversed integer: 87654321


=== Code Execution Successful ===
```

6. Check for a valid String

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

int main() {
    char str[100];
    int i, valid = 1;
    printf("Enter a string: ");
    scanf("%s", str);
    for (i = 0; str[i] != '\0'; i++) {
        if (!isalpha(str[i])) {
            valid = 0;
            break;
        }
    }
    if (valid)
        printf("Valid string (only letters).\n");
    else
        printf("Invalid string (contains non-letters).\n");
    return 0;
}
```

```
Enter a string: haritha
Valid string (only letters).


=== Code Execution Successful ===
```

7. Merging two Arrays

```c
#include <stdio.h>
main() {
    int a[50], b[50], c[100], m, n, i, k = 0;
printf("Enter number of elements in first array: ");
    scanf("%d", &m);
    printf("Enter elements:\n");
    for (i = 0; i < m; i++)
        scanf("%d", &a[i]);
 printf("Enter number of elements in second array: ");
    scanf("%d", &n);
    printf("Enter elements:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &b[i]);
  for (i = 0; i < m; i++)
        c[k++] = a[i];
    for (i = 0; i < n; i++)
        c[k++] = b[i];
 printf("Merged array:\n");
    for (i = 0; i < m + n; i++)
        printf("%d ", c[i]);
    printf("\n");
```

```
Enter number of elements in first array: 5
Enter elements:
5
6
6
8
o
Enter number of elements in second array: Enter elements:
Merged array:
5 6 6 8 524288


=== Code Execution Successful ===
```

8. Given an array finding duplication values

```c
#include <stdio.h>

int main() {
    int arr[50], n, i, j;

    printf("Enter number of elements: ");
    scanf("%d", &n);

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

    printf("Duplicate elements are:\n");
    for (i = 0; i < n; i++)
        for (j = i + 1; j < n; j++)
            if (arr[i] == arr[j])
                printf("%d\n", arr[i]);

    return 0;
}
```

```
Enter number of elements: 3
Enter elements:
1
56
45
Duplicate elements are:


=== Code Execution Successful ===
```

9. Merging of list

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
void insert(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
 if (*head == NULL)
        *head = newNode;
    else {
        struct Node* temp = *head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
    }
}
void display(struct Node* head) {
```

```
Enter number of elements for first list: 5
Enter elements:
23
56
45
34
78
Enter number of elements for second list: 3
Enter elements:
23 87 56
Merged Linked List:
23 -> 56 -> 45 -> 34 -> 78 -> 23 -> 87 -> 56 -> NULL


=== Code Execution Successful ===
```

main.c

```c
    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}
struct Node* merge(struct Node* head1, struct Node* head2) {
    if (head1 == NULL) return head2;
    if (head2 == NULL) return head1;
struct Node* temp = head1;
    while (temp->next != NULL)
        temp = temp->next;
temp->next = head2;
    return head1;
}
int main() {
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;
    struct Node* merged = NULL;
    int n, data, i;
```

```c
  printf("Enter number of elements for first list: ");
    scanf("%d", &n);
    printf("Enter elements:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &data);
        insert(&head1, data);
    }
  printf("Enter number of elements for second list: ");
    scanf("%d", &n);
    printf("Enter elements:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &data);
        insert(&head2, data);
    }
  merged = merge(head1, head2);
  printf("Merged Linked List:\n");
    display(merged);
  return 0;
```

10. Given array of regno sneed to search for particular regno

```c
#include <stdio.h>
int main() {
    int reg[50], n, i, key, found = 0;
 printf("Enter number of reg numbers: ");
    scanf("%d", &n);
 printf("Enter reg numbers:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &reg[i]);
printf("Enter reg number to search: ");
    scanf("%d", &key);
for (i = 0; i < n; i++) {
        if (reg[i] == key) {
            found = 1;
            printf("Reg number found at position %d\n", i + 1);
            break;
        }
    }
if (!found)
        printf("Reg number not found.\n");
return 0;
```

```
Enter number of reg numbers: 4
Enter reg numbers:
123
564
574
64
Enter reg number to search: 564
Reg number found at position 2


=== Code Execution Successful ===
```

7. Identify location of element in given array

```c
#include <stdio.h>
int main() {
    int arr[50], n, key, i, found = 0;
 printf("Enter number of elements: ");
    scanf("%d", &n);
 printf("Enter array 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 %d found at position %d\n", key, i + 1);
            found = 1;
        }
    }
if (!found)
        printf("Element not found.\n");
 return 0;
}
```

```
Enter number of elements: 4
Enter array elements:
4
5
6
8
Enter element to search: 5
Element 5 found at position 2


=== Code Execution Successful ===
```

8. Given array print odd and even values

```c
#include <stdio.h>
int main() {
    int arr[50], n, i;
 printf("Enter number of elements: ");
    scanf("%d", &n);
 printf("Enter array elements:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);
printf("Odd numbers: ");
    for (i = 0; i < n; i++)
        if (arr[i] % 2 != 0)
            printf("%d ", arr[i]);
printf("\nEven numbers: ");
    for (i = 0; i < n; i++)
        if (arr[i] % 2 == 0)
            printf("%d ", arr[i]);
 printf("\n");
return 0;
}
```

```
Enter number of elements: 5
Enter array elements:
34 56 34 23 56
Odd numbers: 23
Even numbers: 34 56 34 56


=== Code Execution Successful ===
```

9. sum of Fibonacci Series

```c
#include <stdio.h>
int main() {
    int n, i, first = 0, second = 1, next, sum = 0;
 printf("Enter number of terms: ");
    scanf("%d", &n);
 printf("Fibonacci Series: ");
    for (i = 0; i < n; i++) {
        printf("%d ", first);
        sum += first;
        next = first + second;
        first = second;
        second = next;
    }
 printf("\nSum of Fibonacci Series = %d\n", sum);
    return 0;
}
```

```
Enter number of terms: 4
Fibonacci Series: 0 1 1 2
Sum of Fibonacci Series = 4


=== Code Execution Successful ===
```

10. Finding factorial of a number

```c
#include <stdio.h>

int main() {
    int n, i;
    unsigned long long fact = 1;

    printf("Enter a number: ");
    scanf("%d", &n);

    for (i = 1; i <= n; i++)
        fact *= i;

    printf("Factorial of %d = %llu\n", n, fact);

return 0;
}
```

```
Enter a number: 45
Factorial of 45 = 9649395409222631424


=== Code Execution Successful ===
```

11. AVL tree

main.c

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int key;
    struct Node *left, *right;
    int height;
};
int max(int a, int b) {
    return (a > b) ? a : b;
}int height(struct Node *N) {
    if (N == NULL)
        return 0;
    return N->height;
}struct Node* newNode(int key) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->key = key;
    node->left = node->right = NULL;
    node->height = 1;
    return node;
}struct Node* rightRotate(struct Node *y) {
```

```
1.Insert 2.Search 3.Display 4.Exit
1
Enter key to insert: 4

1.Insert 2.Search 3.Display 4.Exit
2
Enter key to search: 6
Not found

1.Insert 2.Search 3.Display 4.Exit
```

```c
    struct Node *x = y->left;
    struct Node *T2 = x->right;
 x->right = y;
    y->left = T2;
y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;
return x;
}
struct Node* leftRotate(struct Node *x) {
    struct Node *y = x->right;
    struct Node *T2 = y->left;
 y->left = x;
    x->right = T2;
 x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;
 return y;
}int getBalance(struct Node *N) {
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
```

```
}
struct Node* insert(struct Node* node, int key) {
    if (node == NULL)
        return newNode(key);
 if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    else
        return node;
 node->height = 1 + max(height(node->left), height(node->right));
    int balance = getBalance(node);
 if (balance > 1 && key < node->left->key)
        return rightRotate(node);
if (balance < -1 && key > node->right->key)
        return leftRotate(node);
if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    } if (balance < -1 && key < node->right->key) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }return node;
}
int search(struct Node* root, int key) {
    if (root == NULL)
        return 0;
    if (root->key == key)
        return 1;
    else if (key < root->key)
        return search(root->left, key);
    else
        return search(root->right, key);
}void preOrder(struct Node *root) {
    if (root != NULL) {
        printf("%d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }}int main() {
    struct Node *root = NULL;
```

```
    int ch, key;
    do {
        printf("\n1.Insert 2.Search 3.Display 4.Exit\n");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf("Enter key to insert: ");
                scanf("%d", &key);
                root = insert(root, key);
                break;
            case 2:
                printf("Enter key to search: ");
                scanf("%d", &key);
                if (search(root, key))
                    printf("Found\n");
                else
                    printf("Not found\n");
                break;
            case 3:
                printf("Preorder: ");
```

```
                preOrder(root);
                printf("\n");
                break;
        }
    } while (ch != 4);
    return 0;
```

12. Valid stack

```
#include <stdio.h>
#define MAX 10
int stack[MAX], top = -1;
void push(int val) {
    if (top == MAX - 1)
        printf("Stack Overflow\n");
    else
        stack[++top] = val;
}
void pop() {
    if (top == -1)
        printf("Stack Underflow\n");
    else
        printf("Popped: %d\n", stack[top--]);
}void peek() {
    if (top == -1)
        printf("Stack is Empty\n");
    else
        printf("Top Element: %d\n", stack[top]);
}void display() {
```

```
1.Push 2.Pop 3.Peek 4.Display 5.Exit
3
Stack is Empty
1.Push 2.Pop 3.Peek 4.Display 5.Exit
2
Stack Underflow
1.Push 2.Pop 3.Peek 4.Display 5.Exit
4
Stack is Empty
1.Push 2.Pop 3.Peek 4.Display 5.Exit
3
Stack is Empty
1.Push 2.Pop 3.Peek 4.Display 5.Exit
1
Enter value: 2
1.Push 2.Pop 3.Peek 4.Display 5.Exit
```

```
        printf("Stack is Empty\n");
    else {
        printf("Stack: ");
        for (int i = 0; i <= top; i++)
            printf("%d ", stack[i]);
        printf("\n");
    }
}int main() {
    int ch, val;
    do {
        printf("1.Push 2.Pop 3.Peek 4.Display 5.Exit\n");
        scanf("%d", &ch);
        switch (ch) {
            case 1: printf("Enter value: "); scanf("%d", &val); push
                (val); break;
            case 2: pop(); break;
            case 3: peek(); break;
            case 4: display(); break;
        }
    } while (ch != 5);
    return 0;
```

13. Graph-shortest path

```
main.c
clude <stdio.h>
#define MAX 10
#define INF 9999
int main() {
    int n, i, j, u, v, min, start;
    int cost[MAX][MAX], dist[MAX], visited[MAX] = {0};
printf("Enter number of vertices: ");
    scanf("%d", &n);
 printf("Enter adjacency matrix (0 if no edge):\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if (i == j)
                cost[i][j] = 0;
            else {
                scanf("%d", &cost[i][j]);
                if (cost[i][j] == 0)
                    cost[i][j] = INF;
            } printf("Enter source vertex: ");
    scanf("%d", &start);
for (i = 0; i < n; i++) {
```

```
Enter number of vertices: 3
Enter adjacency matrix (0 if no edge):
3
5 7
5 8
7 9
Enter source vertex: Shortest distances:
To 0 = 0
To 1 = 0
To 2 = 0


=== Code Execution Successful ===
```

```
        dist[i] = cost[start][i];
    }
    visited[start] = 1;
for (i = 0; i < n - 1; i++) {
        min = INF;
        for (j = 0; j < n; j++)
            if (!visited[j] && dist[j] < min) {
                min = dist[j];
                u = j;
            }
  visited[u] = 1;
        for (v = 0; v < n; v++)
            if (!visited[v] && dist[u] + cost[u][v] < dist[v])
                dist[v] = dist[u] + cost[u][v];
    } printf("Shortest distances:\n");
    for (i = 0; i < n; i++)
        printf("To %d = %d\n", i, dist[i]);
  return 0;
```

14. Traveling Salesman Problem

main.c

```c
#include <stdio.h>
#define MAX 10
#define INF 999

int n, cost[MAX][MAX], visited[MAX];
int min_cost = INF;

void tsp(int city, int count, int cost_sum, int start) {
    if (count == n && cost[city][start]) {
        if (cost_sum + cost[city][start] < min_cost)
            min_cost = cost_sum + cost[city][start];
        return;
    }

    for (int i = 0; i < n; i++) {
        if (!visited[i] && cost[city][i]) {
            visited[i] = 1;
            tsp(i, count + 1, cost_sum + cost[city][i], start);
            visited[i] = 0;
        }
    }
}

int main() {
    printf("Enter number of cities: ");
    scanf("%d", &n);

    printf("Enter cost matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &cost[i][j]);

    visited[0] = 1;
    tsp(0, 1, 0, 0);

    printf("Minimum Cost: %d\n", min_cost);
    return 0;
}
```

```
Enter number of cities: 3
Enter cost matrix:
4
6
3
5
7
6
8
yi
Minimum Cost: 999


=== Code Execution Successful ===
```

15. Binary search tree-search for a element, min element and Max element

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

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

struct Node* insert(struct Node* node, int data) {
    if (node == NULL) {
        struct Node* temp = (struct Node*)malloc(sizeof(struct Node
            ));
        temp->data = data;
        temp->left = temp->right = NULL;
        return temp;
    }
    if (data < node->data)
        node->left = insert(node->left, data);
    else if (data > node->data)
        node->right = insert(node->right, data);
```

```
1.Insert 2.Search 3.Min 4.Max 5.Display 6.Exit
1
Enter value: 2

1.Insert 2.Search 3.Min 4.Max 5.Display 6.Exit
2
Enter value to search: 3
Not Found

1.Insert 2.Search 3.Min 4.Max 5.Display 6.Exit
```

```c
struct Node* search(struct Node* root, int key) {
    if (root == NULL || root->data == key)
        return root;
    if (key < root->data)
        return search(root->left, key);
    return search(root->right, key);
}

struct Node* findMin(struct Node* root) {
    while (root->left != NULL)
        root = root->left;
    return root;
}

struct Node* findMax(struct Node* root) {
    while (root->right != NULL)
        root = root->right;
    return root;
}
```

```c
int main() {
    struct Node* root = NULL;
    int ch, val;
    do {
        printf("\n1.Insert 2.Search 3.Min 4.Max 5.Display 6.Exit\n");
        scanf("%d", &ch);
        switch (ch) {
            case 1: printf("Enter value: "); scanf("%d", &val); root
                = insert(root, val); break;
            case 2: printf("Enter value to search: "); scanf("%d",
                &val);
                    if (search(root, val)) printf("Found\n"); else
                        printf("Not Found\n"); break;
            case 3: printf("Min: %d\n", findMin(root)->data); break;
            case 4: printf("Max: %d\n", findMax(root)->data); break;
            case 5: inorder(root); printf("\n"); break;
        }
    } while (ch != 6);
    return 0;
```

16. Array sort-ascending and descending

```c
#include <stdio.h>
int main() {
    int arr[50], n, i, j, temp;
 printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter array elements:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    for (i = 0; i < n - 1; i++)
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
  printf("Ascending Order: ");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
        for (i = 0; i < n - 1; i++)
        for (j = 0; j < n - i - 1; j++)
```

```
Enter number of elements: 3
Enter array elements:
2
1
4
Ascending Order: 1 2 4
Descending Order: 4 2 1


=== Code Execution Successful ===
```

```c
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] < arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
   printf("\nDescending Order: ");
      for (i = 0; i < n; i++)
          printf("%d ", arr[i]);
   printf("\n");
      return 0;
```

17. Array search-linear and binary

```c
#include <stdio.h>
int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1, mid;
    while (low <= high) {
        mid = (low + high) / 2;
        if (arr[mid] == key)
            return mid;
        else if (arr[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}int main() {
    int arr[50], n, i, key, pos;
printf("Enter number of elements: ");
    scanf("%d", &n);
printf("Enter sorted array elements for Binary Search:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);
```

```
Enter number of elements: 3
Enter sorted array elements for Binary Search:
34
56
7
Enter element to search: 7
Linear Search: Found at position 3
Binary Search: Not found


=== Code Execution Successful ===
```

```c
    printf("Enter element to search: ");
    scanf("%d", &key);
    int found = 0;
    for (i = 0; i < n; i++)
        if (arr[i] == key) {
            printf("Linear Search: Found at position %d\n", i + 1);
            found = 1;
            break;
        }
    if (!found)
        printf("Linear Search: Not found\n");
    pos = binarySearch(arr, n, key);
    if (pos != -1)
        printf("Binary Search: Found at position %d\n", pos + 1);
    else
        printf("Binary Search: Not found\n");
return 0;
```

18. given set of Array elements-display 5th iterated element

```c
#include <stdio.h>

int main() {
    int arr[50], n, i;

    printf("Enter number of elements: ");
    scanf("%d", &n);

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

    if (n >= 5)
        printf("5th Iterated Element: %d\n", arr[4]);
    else
        printf("Array has less than 5 elements.\n");

    return 0;
}
```

```
Enter number of elements: 6
Enter array elements:
23 45 67 34 23 23
5th Iterated Element: 23


=== Code Execution Successful ===
```

19. Given unsorted array- Display missing element

```c
#include <stdio.h>

int main() {
    int arr[50], n, i, total = 0, sum = 0;

    printf("Enter number of elements (missing one): ");
    scanf("%d", &n);

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

    total = (n + 1) * (n + 2) / 2;
    printf("Missing Element: %d\n", total - sum);

    return 0;
}
```

```
Enter number of elements (missing one): 4
Enter elements:
34
23
67
45
Missing Element: -154


=== Code Execution Successful ===
```

20. Array concatenation

main.c

```c
ude <stdio.h>
int main() {
    int a[50], b[50], c[100];
    int m, n, i, k = 0;
printf("Enter size of first array: ");
    scanf("%d", &m);
    printf("Enter elements of first array:\n");
    for (i = 0; i < m; i++)
        scanf("%d", &a[i]);
printf("Enter size of second array: ");
    scanf("%d", &n);
    printf("Enter elements of second array:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &b[i]);
 for (i = 0; i < m; i++)
        c[k++] = a[i];
    for (i = 0; i < n; i++)
        c[k++] = b[i];
 printf("Concatenated Array:\n");
    for (i = 0; i < m + n; i++)
        printf("%d ", c[i]);
    printf("\n");
return 0;
```

```
Enter size of first array: 3
Enter elements of first array:
23
34
67
Enter size of second array: 3
Enter elements of second array:
23
45
78
Concatenated Array:
23 34 67 23 45 78


=== Code Execution Successful ===
```

21. Haystack

main.c

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

int main() {
    char haystack[100], needle[50];
    printf("Enter main string (haystack): ");
    scanf("%s", haystack);
    printf("Enter substring to search (needle): ");
    scanf("%s", needle);

    if (strstr(haystack, needle) != NULL)
        printf("Substring found.\n");
    else
        printf("Substring not found.\n");

    return 0;
}
```

```
Enter main string (haystack): 100
Enter substring to search (needle): 50
Substring not found.


=== Code Execution Successful ===
```

22. Given Graph convert to array and print minimum edges

```c
#include <stdio.h>
int main() {
    int adj[10][10], edges[20][2];
    int n, i, j, k = 0;
printf("Enter number of vertices: ");
    scanf("%d", &n);
printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &adj[i][j]);
for (i = 0; i < n; i++)
        for (j = i + 1; j < n; j++)
            if (adj[i][j] == 1) {
                edges[k][0] = i;
                edges[k][1] = j;
                k++;
            } printf("Edges in graph:\n");
    for (i = 0; i < k; i++)
        printf("(%d, %d)\n", edges[i][0], edges[i][1]);
printf("Minimum edges to connect all nodes (MST): %d\n", n - 1);
    return 0;
```

```
Enter number of vertices: 4
Enter adjacency matrix:
4
1 3 4
2 6 4
3 8 6
5 8 4
5 8 3
Edges in graph:
(0, 1)
Minimum edges to connect all nodes (MST): 3


=== Code Execution Successful ===
```

23. Given Graph-Print valid path

```c
#include <stdio.h>
main.c
adj[10][10], visited[10], path[10], n, found = 0;
void dfs(int src, int dest, int idx) {
    visited[src] = 1;
    path[idx] = src;
 if (src == dest) {
        printf("Valid path: ");
        for (int i = 0; i <= idx; i++)
            printf("%d ", path[i]);
        printf("\n");
        found = 1;
        return;
    }
 for (int i = 0; i < n; i++) {
        if (adj[src][i] && !visited[i])
            dfs(i, dest, idx + 1);
    }
 visited[src] = 0;  // backtrack
}int main() {
    int src, dest;
```

```
Enter number of vertices: 3
Enter adjacency matrix:
3
2 7 4
5 9 7
5 9 4
Enter source and destination: 2
No valid path found.


=== Code Execution Successful ===
```

```c
    int src, dest;
 printf("Enter number of vertices: ");
    scanf("%d", &n);
 printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &adj[i][j]);
            printf("Enter source and destination: ");
    scanf("%d %d", &src, &dest);
 dfs(src, dest, 0);
    if (!found)
        printf("No valid path found.\n");
 return 0;
```

24. heap, merge, insertion and quick sort

```c
#include <stdio.h>
void heapify(int arr[], int n, int i) {
    int largest = i, l = 2*i+1, r = 2*i+2, temp;
 if (l < n && arr[l] > arr[largest])
        largest = l;
    if (r < n && arr[r] > arr[largest])
        largest = r;
 if (largest != i) {
        temp = arr[i]; arr[i] = arr[largest]; arr[largest] = temp;
        heapify(arr, n, largest);
    }
}void heapSort(int arr[], int n) {
    for (int i = n/2-1; i >= 0; i--)
        heapify(arr, n, i);
 for (int i = n-1; i >= 0; i--) {
        int temp = arr[0]; arr[0] = arr[i]; arr[i] = temp;
        heapify(arr, i, 0);
    }
}int main() {
    int arr[50], n;
```

```
Enter size: 3
Enter elements:
2 5 7
Heap Sorted: 2 5 7


=== Code Execution Successful ===
```

```c
    int arr[50], n;
    printf("Enter size: ");
    scanf("%d", &n);
    printf("Enter elements:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
heapSort(arr, n);
printf("Heap Sorted: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
    return 0;
```

25. Print no of nodes in the given linked list

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

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

int count(struct Node* head) {
    int c = 0;
    while (head != NULL) {
        c++;
        head = head->next;
    }
    return c;
}

void insert(struct Node** head, int val) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = *head;
    *head = newNode;
}

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

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

    printf("Enter elements:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &val);
        insert(&head, val);
    }
    printf("Number of nodes: %d\n", count(head));
    return 0;
}
```

```
Enter number of nodes: 3
Enter elements:
3 4 7
Number of nodes: 3


=== Code Execution Successful ===
```

26. Given 2D matrix print largest element

```c
#include <stdio.h>
int main() {
    int mat[10][10], r, c, i, j, max;
printf("Enter rows and columns: ");
    scanf("%d %d", &r, &c);
 printf("Enter elements:\n");
    for (i = 0; i < r; i++)
        for (j = 0; j < c; j++)
            scanf("%d", &mat[i][j]);

    max = mat[0][0];
    for (i = 0; i < r; i++)
        for (j = 0; j < c; j++)
            if (mat[i][j] > max)
                max = mat[i][j];

    printf("Largest element: %d\n", max);
    return 0;
}
```

```
Enter rows and columns: 2 2
Enter elements:
2 4  56 6
Largest element: 56


=== Code Execution Successful ===
```

27. Given a string-sort in alphabetical order

```c
#include <stdio.h>
#include <string.h>
int main() {
    char str[100], temp;
    int i, j, len;
printf("Enter string: ");
    scanf("%s", str);
    len = strlen(str);
    for (i = 0; i < len - 1; i++)
        for (j = i + 1; j < len; j++)
            if (str[i] > str[j]) {
                temp = str[i];
                str[i] = str[j];
                str[j] = temp;
            } printf("Sorted string: %s\n", str);
    return 0;
}
```

```
Enter string: 34 56
Sorted string: 34


=== Code Execution Successful ===
```

28. Print the index of repeated characters given in an array

```c
#include <stdio.h>
int main() {
    int arr[50], n, i, j;

    printf("Enter size: ");
    scanf("%d", &n);

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

    printf("Indexes of repeated elements:\n");
    for (i = 0; i < n; i++)
        for (j = i + 1; j < n; j++)
            if (arr[i] == arr[j])
                printf("Element %d at index %d and %d\n", arr[i], i,
                    j);

    return 0;
```

```
Enter size: 3
Enter elements:
45 45 67
Indexes of repeated elements:
Element 45 at index 0 and 1


=== Code Execution Successful ===
```

29. Print the frequently repeated numbers count from an array

```c
#include <stdio.h>
int main() {
    int arr[50], freq[50] = {0}, n, i, j;
 printf("Enter size: ");
    scanf("%d", &n);
printf("Enter elements:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);
 printf("Repeated numbers and their counts:\n");
    for (i = 0; i < n; i++) {
        if (freq[i] == 0) {
            int count = 1;
            for (j = i + 1; j < n; j++) {
                if (arr[i] == arr[j]) {
                    count++;
                    freq[j] = 1;
                }
            }
            if (count > 1)
                printf("%d occurs %d times\n", arr[i], count);
        } } return 0;
```

```
Enter size: 3
Enter elements:
34 56 34
Repeated numbers and their counts:
34 occurs 2 times


=== Code Execution Successful ===
```

30. Palindrome using SL

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    char data;
    struct Node* next;
};
void push(struct Node** head, char data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}int isPalindrome(struct Node* head) {
    char str[100];
    int len = 0;
while (head != NULL) {
        str[len++] = head->data;
        head = head->next;
    }
for (int i = 0; i < len / 2; i++)
        if (str[i] != str[len - i - 1])
```

```
Enter size: 3
Enter elements:
34 56 34
Repeated numbers and their counts:
34 occurs 2 times


=== Code Execution Successful ===
```

```c
            return 0;
 return 1;
}int main() {
    struct Node* head = NULL;
    char word[100];
  printf("Enter string: ");
    scanf("%s", word);
 for (int i = 0; word[i] != '\0'; i++)
        push(&head, word[i]);
  if (isPalindrome(head))
        printf("Palindrome\n");
    else
        printf("Not Palindrome\n");
 return 0;
```