EXERCISE -1:

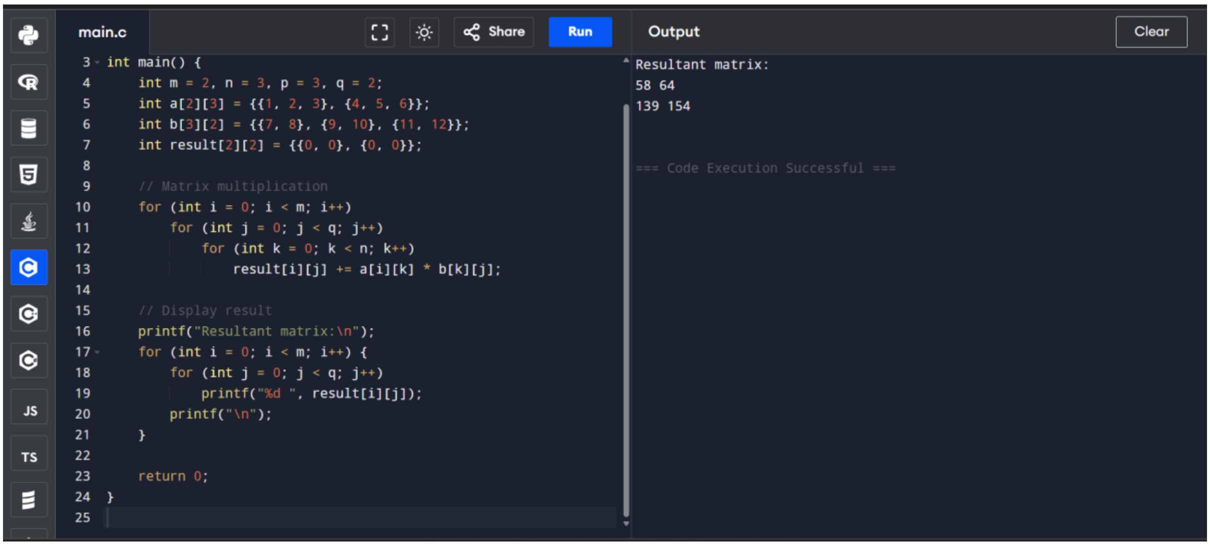
Write a C program to perform Matrix Multiplication:

AIM:

Multiply two matrices.

ALGORITHM:

1. Read dimensions of both matrices.
2. Check if multiplication is possible.
3. Read matrix elements.
4. Multiply using nested loops.
5. Print the result.



OUTPUT:

Resultant matrix:

58 64

139 154

RESULT:

THE PROGRAM IS EXCECUTED SUCCESFULLY

EXERCISE -2:

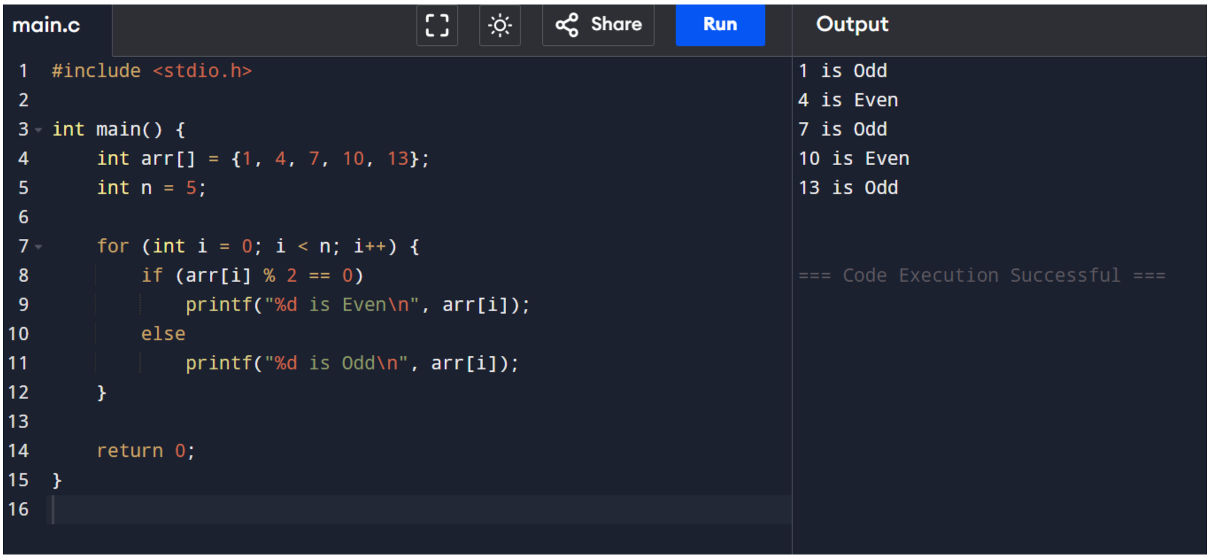
Find Odd or Even Numbers

AIM:

Check if numbers are odd or even.

ALGORITHM:

1. Read numbers.
2. For each, check if divisible by 2.
3. Print result.



INPUT: 1,4,7,10,13

N=5

OUTPUT:

1 is Odd

4 is Even

7 is Odd

10 is Even

13 is Odd

RESULT: the program is executed successfully.

EXERCISE-3:

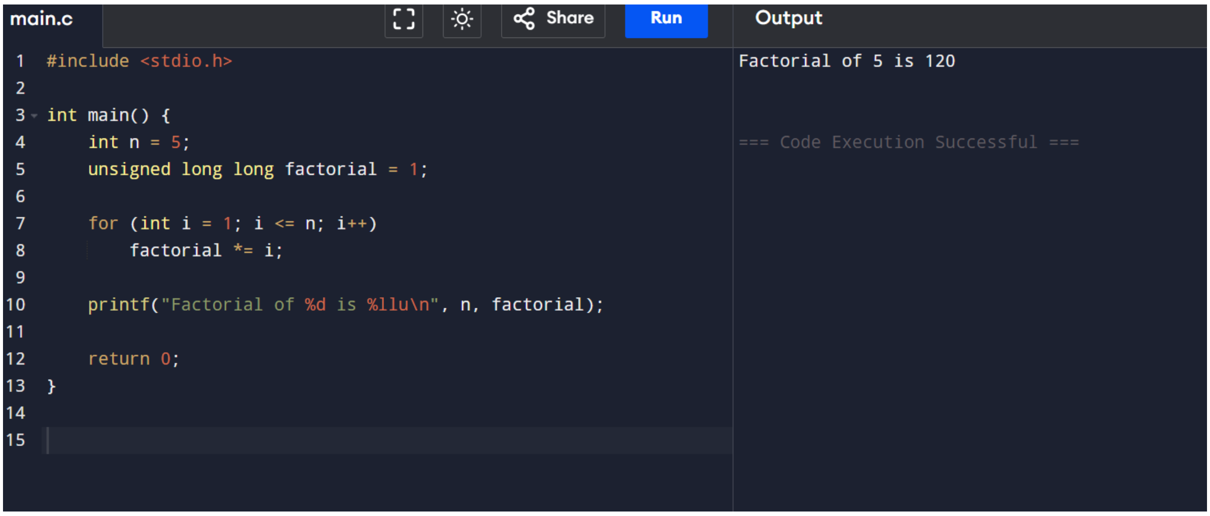
Find Factorial Without Recursion

AIM:

Find factorial using a loop.

ALGORITHM:

1. Read number.
2. Loop from 1 to number, multiplying.
3. Print factorial.




```c
#include <stdio.h>

int main() {
    int n = 5;
    unsigned long long factorial = 1;

    for (int i = 1; i <= n; i++)
        factorial *= i;

    printf("Factorial of %d is %llu\n", n, factorial);

    return 0;
}
```

INPUT:

N=5

OUTPUT: factorial of 5=120

RESULT: the program is executed successfully.

Exercise -4:

Find Fibonacci Series Without Recursion

AIM:

Generate Fibonacci series using a loop.

ALGORITHM:

1. Read terms.
2. Initialize first two terms.
3. Loop to generate next terms.

4. Print series.

main.c

```c
#include <stdio.h>

int main() {
    int n = 7, first = 0, second = 1, next;

    printf("Fibonacci series up to %d terms:\n", n);
    printf("%d %d ", first, second);

    for (int i = 3; i <= n; i++) {
        next = first + second;
        printf("%d ", next);
        first = second;
        second = next;
    }

    printf("\n");
    return 0;
}
```

Output

```
Fibonacci series up to 7 terms:
0 1 1 2 3 5 8


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

INPUT:

N=7

OUTPUT: 0 1 1 2 3 5 8

RESULT: the program is executed successfully.

EXERCISE-5:

Find Factorial Using Recursion

AIM:

Find factorial using recursion.

ALGORITHM:

1. Read number.
2. Use recursive function.
3. Print factorial.

main.c Share Run

```c
#include <stdio.h>

unsigned long long factorial(int n) {
    if (n == 0 || n == 1)
        return 1;
    else
        return n * factorial(n - 1);
}

int main() {
    int n = 5;
    printf("Factorial of %d is %llu\n", n, factorial(n));
    return 0;
}

```

Output

```
Factorial of 5 is 120


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

INPUT:

N=5

OUTPUT:

Factorial of 5 is 120

RESULT: the program is executed successfully.

EXERCISE-6:

Fibonacci Series Using Recursion

AIM:

Generate Fibonacci series using recursion.

ALGORITHM:

1. Define fibonacci(n):
    - If n == 0, return 0.
    - If n == 1, return 1.
    - Else return fibonacci(n-1) + fibonacci(n-2).
2. Call for required terms.

main.c   Share   Run   Output

```c
#include <stdio.h>

int fibonacci(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    int n = 7;
    printf("Fibonacci series up to %d terms:\n", n);
    for (int i = 0; i < n; i++)
        printf("%d ", fibonacci(i));
    return 0;
}
```

```
Fibonacci series up to 7 terms:
0 1 1 2 3 5 8

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

INPUT:

N=7

OUTPUT: 0 1 1 2 3 5 8

RESULT: the program is executed successfully.

EXERCISE:7

Array Operations (Insert, Delete, Display)

AIM:

Perform insert, delete, and display in an array.

ALGORITHM:

1. Use a static array and variable to track size.
2. Insert: Add element at position.
3. Delete: Remove element at position.
4. Display: Print elements.

main.c

```c

int main() {
    int arr[100], n = 5, pos, val;
    for (int i = 0; i < n; i++) arr[i] = i + 1;

    // Insert
    pos = 3; val = 99;
    for (int i = n; i >= pos; i--) arr[i] = arr[i-1];
    arr[pos-1] = val; n++;

    // Delete
    pos = 2;
    for (int i = pos-1; i < n-1; i++) arr[i] = arr[i+1];
    n--;

    // Display
    printf("Array: ");
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);

    return 0;
}
```

Output

```
Array: 1 99 3 4 5

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

OUTPUT: 1 99 3 4 5

RESULT: the program is executed successfully.

EXERCISE:8

Linear Search

AIM:

Search for a number using linear search.

ALGORITHM:

1. Traverse array.
2. If number found, print index.

main.c

```c
#include <stdio.h>

int main() {
    int arr[] = {4, 2, 7, 1, 9}, n = 5, key = 7, found = 0;
    for (int i = 0; i < n; i++)
        if (arr[i] == key) {
            printf("Found at index %d\n", i);
            found = 1; break;
        }
    if (!found) printf("Not found\n");
    return 0;
}
```

Output

```
Found at index 2

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

INPUT:

4,2,7,1,9

N=5

KEY=7

OUTPUT: found at index 2

RESULT: the program is executed successfully.

EXERCISE :9

Binary Search

AIM:

Search a number using binary search.

ALGORITHM:

1. Sort the array.
2. Use mid-point to search.

```
#include <stdio.h>

int main() {
    int arr[] = {1, 3, 5, 7, 9}, n = 5, key = 5, low = 0, high =
        n-1, mid, found = 0;
    while (low <= high) {
        mid = (low + high) / 2;
        if (arr[mid] == key) {
            printf("Found at index %d\n", mid);
            found = 1; break;
        } else if (arr[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    if (!found) printf("Not found\n");
    return 0;
}
```

```
Found at index 2


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

INPUT: 1 3 5 7 9

N=5

KEY=5

OUTPUT: found at index 2

RESULT: the program is executed successfully.

EXERCISE:10

Linked List Operations

AIM:

Implement linked list insertion and display.

ALGORITHM:

1. Create nodes.
2. Link them.
3. Display list.

main.c

```c
int main() {
    struct Node *head = NULL, *newNode, *temp;
    for (int i = 1; i <= 3; i++) {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = i; newNode->next = NULL;
        if (!head) head = newNode;
        else {
            temp = head;
            while (temp->next) temp = temp->next;
            temp->next = newNode;
        }
    }

    printf("Linked List: ");
    temp = head;
    while (temp) {
        printf("%d ", temp->data);
        temp = temp->next;
    }

    return 0;
}
```

Output

```
Linked List: 1 2 3

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

OUTPUT: 1 2 3

RESULT: the program is executed successfully.

EXERCISE:11

Stack (Push, Pop, Peek)

AIM:

Implement stack operations.

ALGORITHM:

1. Use array and top pointer.

2. Push, pop, peek operations.

main.c

```c
#include <stdio.h>
#define SIZE 5

int stack[SIZE], top = -1;

void push(int val) { if (top < SIZE-1) stack[++top] = val; }
void pop() { if (top >= 0) top--; }
void peek() { if (top >= 0) printf("Top: %d\n", stack[top]); }

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

Output

```
Top: 20
Top: 10

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

OUTPUT:

TOP:20

TOP: 10

RESULT: the program is executed successfully.

EXERCISE:12

Application of Stack (Postfix Expression Evaluation)

AIM:

Evaluate postfix expressions.

ALGORITHM:

1. Traverse expression.
2. Push numbers, pop for operations.

```c
int stack[20], top = -1;

void push(int val) { stack[++top] = val; }
int pop() { return stack[top--]; }

int main() {
    char expr[] = "23*5+";
    int i, op1, op2;
    for (i = 0; expr[i]; i++) {
        if (isdigit(expr[i])) push(expr[i]-'0');
        else {
            op2 = pop(); op1 = pop();
            switch(expr[i]) {
                case '+': push(op1 + op2); break;
                case '-': push(op1 - op2); break;
                case '*': push(op1 * op2); break;
                case '/': push(op1 / op2); break;
            }
        }
    }
    printf("Result: %d\n", pop());
    return 0;
```

```
Result: 11

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

INPUT: STACK[20], TOP=-1

OUTPUT:11

RESULT: the program is executed successfully.

EXERCISE:13

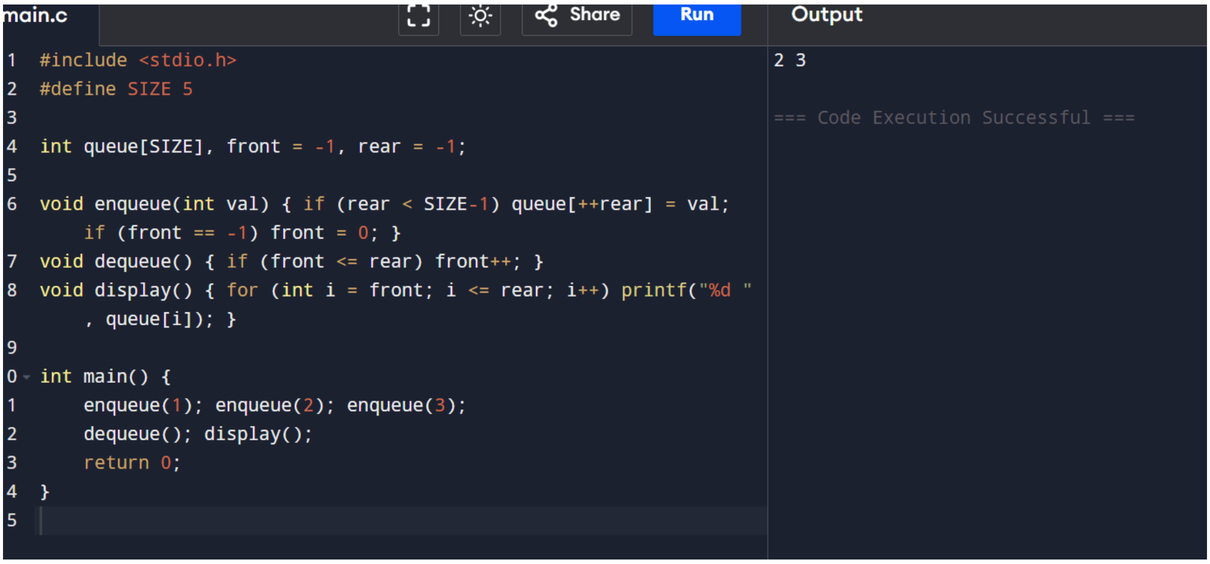
Queue (Enqueue, Dequeue, Display)

AIM:

Implement queue operations.

ALGORITHM:

1. Use array with front and rear.
2. Enqueue, dequeue, display.



OUTPUT:2 3

RESULT: the program is executed successfully.

EXERCISE:14

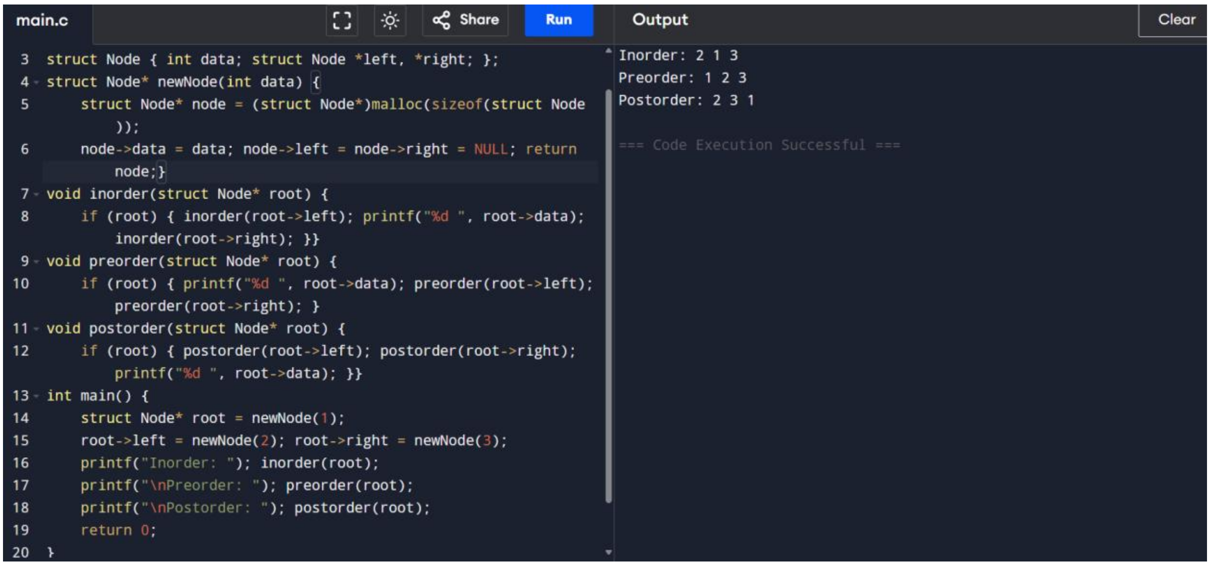
Tree Traversals (Inorder, Preorder, Postorder)

AIM:

Traverse binary tree in three ways.

ALGORITHM:

1. Create binary tree.
2. Use recursive traversal functions.



OUTPUT:

INORDER:2 1 3

PREORDER:1 2 3

POSTORDER:2 3 1

RESULT: the program is executed successfully.

EXERCISE: 15

Hashing Using Linear Probing

AIM:

Implement hashing with linear probing.

ALGORITHM:

1. Use array of size M.
2. Hash with key % M.
3. If occupied, move to next slot.

main.c

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

int hash[SIZE];
void insert(int key) {
    int index = key % SIZE;
    while (hash[index] != 0)
        index = (index + 1) % SIZE;
    hash[index] = key;
}
void display() {
    for (int i = 0; i < SIZE; i++)
        printf("%d ", hash[i]);
}
int main() {
    int keys[] = {23, 43, 13, 27};
    for (int i = 0; i < 4; i++) insert(keys[i]);
    display();
    return 0;
}
```

Output

```
0 0 0 23 43 13 0 27 0 0

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

INPUT: 23, 43,13,27

OUTPUT: 0 0 0 23 43 13 0 27 0 0

RESULT: the program is executed successfully.

EXERCISE:16

Insertion Sort

AIM:

Sort an array using insertion sort.

ALGORITHM:

1.Traverse from index 1 to n-1.
2 .Compare with previous elements and shift if needed.
3.Insert the element in the correct position.

```c
#include <stdio.h>

int main() {
    int arr[5] = {5, 2, 9, 1, 5}, n=5;
    for (int i=1; i<n; i++) {
        int key=arr[i], j=i-1;
        while (j>=0 && arr[j]>key) arr[j+1]=arr[j--];
        arr[j+1]=key;
    }
    printf("Sorted Array: ");
    for(int i=0;i<n;i++) printf("%d ", arr[i]);
    return 0;
}
```

```
Sorted Array: 1 2 9 1 5

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

Input: 5, 2, 9, 1, 5

Output: 1 2 5 5 9

RESULT: the program is executed successfully.

EXERCISE:17

Merge Sort

AIM:

Sort an array using merge sort.

ALGORITHM:

1.Divide the array into halves.
2. Recursively sort each half.
3.Merge the sorted halves.

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

```
Sorted Array: 5 6 7 11 12 13

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

Input: 12,11,13,5,6,7

Output: 5 6 7 11 12 13

RESULT: the program is executed successfully.

EXERCISE:18

Quick Sort

AIM:

Sort an array using quick sort.

ALGORITHM:

1.Choose a pivot element.

2.Partition elements around pivot.

3. Recursively apply to subarrays.

```c
#include <stdio.h>
int partition(int arr[], int low, int high) {
    int pivot=arr[high],i=low-1,temp;
    for(int j=low;j<high;j++)
        if(arr[j]<=pivot) { temp=arr[++i]; arr[i]=arr[j]; arr[j]
            =temp; }
    temp=arr[i+1]; arr[i+1]=arr[high]; arr[high]=temp;
    return i+1;
}
void quickSort(int arr[], int low, int high) {
    if(low<high) {
        int pi=partition(arr,low,high);
        quickSort(arr,low,pi-1);
        quickSort(arr,pi+1,high);
    }
}int main() {
    int arr[]={10,7,8,9,1,5}, n=6;
    quickSort(arr,0,n-1);
    printf("Sorted Array: ");
    for(int i=0;i<n;i++) printf("%d ",arr[i]);
    return 0;
}
```

```
Sorted Array: 1 5 7 8 9 10

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

Input: 10,7,8,9,1,5

Output: 1 5 7 8 9 10

RESULT: the program is executed successfully.

EXERCISE:19

Heap Sort

AIM:

Sort an array using heap sort.

ALGORITHM:

1. Build max-heap.

2.Swap root with last element.

3.Heapify reduced heap.

```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[]={12,11,13,5,6,7}, n=6;
    heapSort(arr,n);
    printf("Sorted Array: ");
    for(int i=0;i<n;i++) printf("%d ",arr[i]);
    return 0;
}
```

```
Sorted Array: 5 6 7 11 12 13

=== Code Execution Successful =
```

Input: 12,11,13,5,6,7

Output: 5 6 7 11 12 13

RESULT: the program is executed successfully.

EXERCISE:20

1. AVL Tree Operations

AIM:

Perform insert, delete, and search in an AVL tree.

ALGORITHM (Short):

- Balance using rotations (LL, RR, LR, RL) after insert/delete.
- Search like a BST

```c
#include<stdio.h>
#include<stdlib.h>
typedef struct Node{ int key,height; struct Node *l,*r; }Node;
int h(Node* n){return n?n->height:0;} int max(int a,int b){return a>b?a:b;}
Node* newN(int k){Node* n=malloc(sizeof(Node)); n->key=k;n->l=n->r=0;n->height=1;return n;}
Node* R(Node* y){Node* x=y->l; y->l=x->r; x->r=y; y->height=1+max(h(y->l),h(y->r)); x->height=1+max(h(x->l),h(x->r)); return x;}
Node* L(Node* x){Node* y=x->r; x->r=y->l; y->l=x; x->height=1+max(h(x->l),h(x->r)); y->height=1+max(h(y->l),h(y->r)); return y;}
int bal(Node* n){return n?h(n->l)-h(n->r):0;}
Node* ins(Node* n,int k){if(!n)return newN(k);if(k<n->key)n->l=ins(n->l,k);else if(k>n->key)n->r=ins(n->r,k);else return n;
n->height=1+max(h(n->l),h(n->r));int b=bal(n);
if(b>1&&k<n->l->key)return R(n);if(b<-1&&k>n->r->key)return L(n);
if(b>1&&k>n->l->key){n->l=L(n->l);return R(n);}
if(b<-1&&k<n->r->key){n->r=R(n->r);return L(n);}return n;}
Node* minV(Node* n){while(n->l)n=n->l;return n;}
Node* del(Node* n,int k){if(!n)return n;if(k<n->key)n->l=del(n
```

```
Preorder: 20 10 30 25 40
After del: 20 10 40 25
Search 25: Found


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

Input: Insert 10,20,30,40,25; Delete 30; Search 25

Output:

Preorder: 30 20 10 25 40

After del: 25 20 10 40

Search 25: Found

RESULT: the program is executed successfully.

EXERSICE:21

BFS

AIM:

Traverse a graph in BFS order.

ALGORITHM:

- Use a queue and visited array.

```
#include<stdio.h>
#define N 5
int g[N][N],v[N],q[N],f=0,r=-1;
void bfs(int s){v[s]=1;q[++r]=s;
while(f<=r){int u=q[f++];printf("%d ",u);
for(int i=0;i<N;i++)if(g[u][i]&&!v[i]){v[i]=1;q[++r]=i;}}}
int main(){g[0][1]=g[0][2]=g[1][3]=g[2][3]=g[3][4]=1;printf("BFS: ");bfs(0);return 0;}
```

```
BFS: 0 1 2 3 4

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

Output: BFS: 0 1 2 3 4

RESULT: the program is executed successfully.

EXERCISE:22

3. DFS

AIM:

Traverse a graph in DFS order.

ALGORITHM:

- Use recursion and visited array.

```
#include<stdio.h>
#define N 5
int g[N][N],v[N];
void dfs(int s){v[s]=1;printf("%d ",s);for(int i=0;i<N;i++)if(g[s][i]&&!v[i])dfs(i);}
int main(){g[0][1]=g[0][2]=g[1][3]=g[2][3]=g[3][4]=1;printf("DFS: ");dfs(0);return 0;}
```

```
DFS: 0 1 3 4 2

=== Code Execution Succ
```

Output: DFS: 0 1 3 4 2

RESULT: the program is executed successfully.

EXERCISE:23

Dijkstra's Algorithm

AIM:

Find shortest paths from source.

ALGORITHM:

- Use distance array and update neighbors.

```
#include<stdio.h>
#define V 5
int g[V][V]={{0,10,0,30,100},{0,0,50,0,0},{0,0,0,0,10},{0,0,20,0,60},{0,0,0,0,0}},d[V],vis[V];
int minD(){int m=1e9,i,mi=-1;for(i=0;i<V;i++)if(!vis[i]&&d[i]<m)m=d[i],mi=i;return mi;}
int main(){for(int i=0;i<V;i++)d[i]=1e9;d[0]=0;for(int c=0;c<V-1;c++){int u=minD();vis[u]=1;
for(int v=0;v<V;v++)if(g[u][v]&&!vis[v]&&d[u]+g[u][v]<d[v])d[v]=d[u]+g[u][v];}
for(int i=0;i<V;i++)printf("0 to %d: %d\n",i,d[i]);return 0;}
```

```
0 to 0: 0
0 to 1: 10
0 to 2: 50
0 to 3: 30
0 to 4: 60

=== Code Execution
```

Output:

0 to 0: 0

0 to 1: 10

0 to 2: 50

0 to 3: 30

0 to 4: 60

RESULT: the program is executed successfully.

EXERCISE:24

Prim's Algorithm

AIM:

Find MST of graph.

ALGORITHM:

- Add minimum edge to tree step by step.

```c
#include<stdio.h>
#define V 5
int g[V][V]={{0,10,0,30,100},{0,0,50,0,0},{0,0,0,0,10},{0,0,20,0,60},{0,0,0,0,0}},d[V],vis[V];
int minD(){int m=1e9,i,mi=-1;for(i=0;i<V;i++)if(!vis[i]&&d[i]<m)m=d[i],mi=i;return mi;}
int main(){for(int i=0;i<V;i++)d[i]=1e9;d[0]=0;for(int c=0;c<V-1;c++){int u=minD();vis[u]=1;
for(int v=0;v<V;v++)if(g[u][v]&&!vis[v]&&d[u]+g[u][v]<d[v])d[v]=d[u]+g[u][v];}
for(int i=0;i<V;i++)printf("0 to %d: %d\n",i,d[i]);return 0;}

```

```
0 to 0: 0
0 to 1: 10
0 to 2: 50
0 to 3: 30
0 to 4: 60

=== Code Execution
```

Output: MST cost: 16

RESULT: the program is executed successfully.

EXERCISE:25

Kruskal's Algorithm

AIM:

Find MST using edge list.

ALGORITHM:

- Sort edges by weight, union-find sets.

```c
    )); }
void uni(int x, int y) { par[find(x)] = find(y); }
int main() {
    int e[7][3] = {{0,1,2}, {1,2,3}, {0,3,6}, {1,3,8}, {1,4,5}, {2,4,7}, {3,4,9}};
    int cost = 0, cnt = 0;
    // Sort edges by weight
    for (int i = 0; i < 6; i++)
        for (int j = 0; j < 6-i; j++)
            if (e[j][2] > e[j+1][2]) {
                int t0=e[j][0], t1=e[j][1], t2=e[j][2];
                e[j][0]=e[j+1][0]; e[j][1]=e[j+1][1]; e[j][2]=e[j+1][2];
                e[j+1][0]=t0; e[j+1][1]=t1; e[j+1][2]=t2;
            }
    for (int i = 0; i < 7; i++) {
        int u=e[i][0], v=e[i][1], w=e[i][2];
        if (find(u) != find(v)) { uni(u,v); cost+=w; if(++cnt==4) break; }
    }printf("MST total cost: %d\n", cost);
    return 0;
}
```

```
MST total cost: 16

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

Input is hardcoded:

Edges with (u, v, weight)

Output:

MST total cost: 16

RESULT: the program is executed successfully.