```
#define max_size_str_out 10000
char * helper(char *str_in, int *idx_in)
char * decodeString(char *s)
{
  if(!s) return NULL;
  char str_out[MaxSize_str_out]
  char *str_new;
  int idx_out, idx = 0;
  while(s[idx] != '\0')
  {
    while((s[idx] >= 'a' && s[idx] <= 'z') ||
          (s[idx] >= 'A' && s[idx] <= 'Z'))
      str_out[idx_out++] = s[idx++];
    str_new = helper(s, &idx);
    strcpy(&str_out.out[idx_out], str_new);
    idx_out += strlen(str_new);
    free(str_new);
  }
str_out[idx_out++] = 0;
char * str_ret = (char *) malloc(idx_out)
char * str_new
int ind_out = 0, number = 0, new_number = 0

while(str_in[*idx_in] != '\0') {
    if(str_in[*idx_in] >= a && str_in
      [idx_in] <= Z)
```

```
else if (str_in[*idx_in] >= 0 && str_in[*idx_in] <= 9)
{
    new_num = 0;
    if (!number)
        while (str_in[(*idx_in)] >= 0 &&
               str_in[(*idx_in)] <= 9)
        {
            new_num = new_num * 10 +
            (str_in[(*idx_in)] - '0');
            (*idx_in)++;
        }
        Number = new_num;
}
else
{
    Str_new = helper(str_in, idx_in);
    strcpy(&str_out[idx_out], str_new);
    idx_out += strlen(str_new);
    (*idx_in)--;
    free(str_new);
}
else if (str_in[*idx_in] == ']')
{
    for (int i=0; i<number; memcpy
    (&str_out), str_out, idx_out);
    idx_out;
    str_out[idx_out] = 0;
    (*idx_in)++
    return str_out;
}
```

```
str_out[idx_out] = 0;
return str_out;
}
```

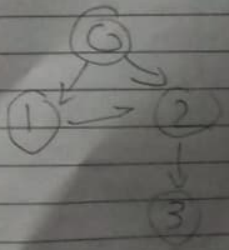
Write a program to obtain the ordering of DFS

```
#include <stdio.h>

void dfs(int n, int cost[10][10], int u, int s[7]) {
{
    int v;
    s[u] = 1;
    printf("%d->", u);
    for(v=0; v<n; v++)
        if (cost[u][v] == 1 && s[v] == 0)
            dfs(n, cost, v, s);
    }
}

void main()
{
    int n, i, j, cost[10][10], s[10], con=0, flag=0;
    printf("Enter the number of nodes \n");
    scanf("%d", &n);
    printf("Enter the adjacency matrix \n");
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            scanf("%d", &cost[i][j]);
```

```
    printf("DFS Path : ");
    dfs(n, cost, 0, s);
    printf("\n");

    for(i=0; i<n; i++)
        if(s[i] == 0){
            flag = 1;
            break;
        }

    if(flag == 0)
        printf("Graph is connected \n");
    else
        printf("Graph is not connected \n");
}
```

output

Enter the no of nodes
4
Enter the adjacency matrix
0
1
1
0
0
0
1
0
0
0



0
1
0
0
0
0

DFS Path : ~~3->2->1->0~~
0 -> 1 -> 2 -> 3

SURYA Gold
Date Page

Source Removal

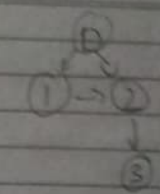
```
#include <stdio.h>
#define MAX_NODES 10

void topologicalSort(int n, int graph[MAX_NODES][MAX_NODES])
    int indegree[MAX_NODES] = {0};
    int i, j, k;
    for(i=0; i<n; i++) {
        for(j=0; j<n; j++)
            if (graph[j][i] == 1)
                indegree[i]++
    }

    for(i=0; i<n; i++) {
        for(j=0; j<n; j++) {
            if(indegree[j] == 0) {
                printf("%d", j);
                indegree[j] = -1;
                for(k=0; k<n; k++) {
                    if (graph[j][k] == 1)
                        indegree[k]--;
                }
                break;
            }
        }
    }
```

```
}
}

int main()
{
    int n, i, j;
    int graph[MAX_NODES][MAX_NODES];
    printf("Enter the no. of nodes: ");
    scanf("%d", &n);

    printf("Enter the adjacency matrix:\n");
    for(i=0; i<n; i++)
        for(j=0; j<n; j++){
            scanf("%d", &graph[i][j]);
        }
}

printf("Topological Sorting order: ");
topologicalSort(n, graph);
return 0;
}
```

Output

```
0
1
1
0
0
0
1
0
```

```
0
0
0
1
0
0
0
0
```



Topological Sorting order: 0 1 2 3

Merge Sort Array

```
#include <stdio.h>
#include <stdlib.h>
void merge(int b[], int p, int c[], int q, int a[]);
void mergesort(int a[], int n);
void mergesort(int a[], int n)
{
  if(n>1)
  {
    int b[n/2];
    int c[n-n/2];
    for(int i=0; i<n/2; i++)
    {
      b[i]=a[i];
    }
    for(int i=0; i<n-n/2; i++)
    {
      c[i]=a[n/2+i];
    }
    mergesort(b, n/2);
    mergesort(c, n-n/2);
    merge(b, n/2, c, n-n/2, a);
  }
}
```

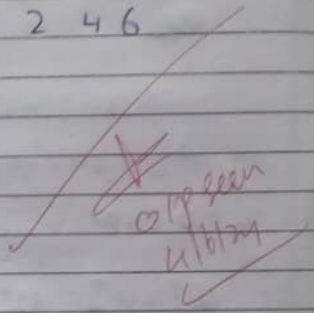
```
void merge(int b[], int p, int c[], int q, int a[])
{
  int i=0, j=0, k=0;
  while(i<p && j<q)
  {
    if(b[i]<c[j])
    {
      a[k++]=b[i++];
    }
    else
    {
      a[k++]=c[j++];
    }
  }
  while(i<p)
  {
    a[k++]=b[i++];
  }
  while(j<q)
  {
    a[k++]=c[j++];
  }
}

int main()
{
  int a[4]={1,4,2,6};
  int n=sizeof(a)/sizeof(a[0]);
  mergesort(a,n);
  printf("Sorted array:");
  for(int i=0; i<n; i++)
  {
    printf("%d ", a[i]);
  }
```

```
part6 ("\n")
return 0
}
```

Output

~~Sorted array : 1 4 2 6~~

Sorted array : 1 2 4 6



Quick sort

```
#include <stdio.h>
void swap(int *a, int *b){
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
int part(int a[], int l, int r)
    int p, a[l];
    int i = l,
    int j = r+1;
    while (i<j)
        do{
            i++;
        } while(a[i]<p);
        do{
            j--;
        } while(a[j]>p);
        if (i<j){
            swap(&a[i], &a[j]);
    } }
swap(&a[l], &a[j]);
};
```

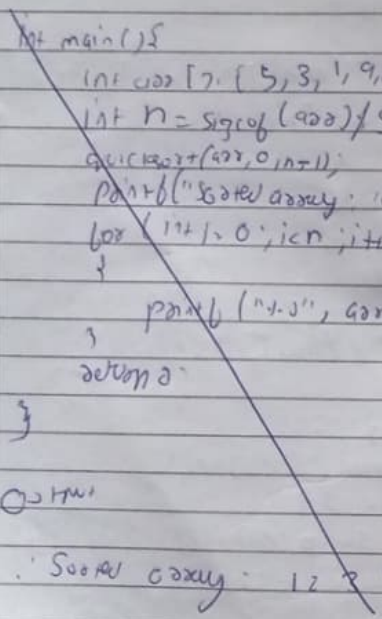
swap(&arr[i], &arr[j]);
return i;

void quicksort(int arr[], int l, int h){
if(l<h){
int s = part(arr, l, h);
quicksort(arr, l, s-1);
quicksort(arr, s+1, h);
}
}

int main(){
int arr[7] = {5,3,1,9,8,2,4,7};
int n = sizeof(arr)/sizeof(arr[0]);
quicksort(arr, 0, n-1);
printf("Sorted array: \n");
for(int i=0; i<n; i++)
{
printf("%d ", arr[i]);
}
return 0;
}

Output:

Sorted array: 1 2 3 4 5 7 8 9



int main(){
int n;
printf("Enter the size of the array: ");
scanf("%d", &n);
int *arr = (int*)malloc(n*sizeof(int));
for(int i=0; i<n; i++)
arr[i] = rand()%100;
quicksort(arr, 0, n-1);
printf("Sorted array: \n");
for(int i=0; i<n; i++){
printf("%d ", arr[i]);
}
free(arr);
return 0;
}

Output:

Enter size of array: 10
Sorted array
execution time: 2.1258 sec

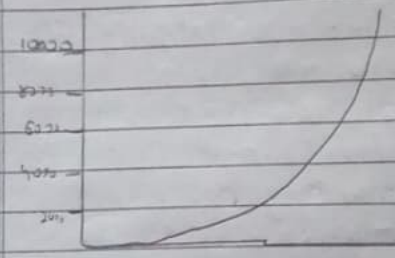
Enter size of array 100
Sorted array
execution time: 2.656 sec

Enter size of array: 1000
Sorted array
execution time: 3.2978

Enter size of array : 10000

Sorted array

Execution time : 3.7685



Heap Sort

```
#include <stdio.h>
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void heapify(int arr[], int n, int i)
{
    int max = i;
    int l = 2*i + 1;
    int r = 2*i + 2;
    if (l < n && arr[l] > arr[max])
        max = l;
    if (r < n && arr[r] > arr[max])
        max = r;
    if (max != i)
    {
        swap(&arr[i], &arr[max]);
        heapify(arr, n, i);
    }
}
```

SURYA Gold

Date ____ Page ____

```
void bh(int arr[], int n)
{
    for(int i = n/2 -1; i>=0; i--)
        heapify(arr, n, i);
}

void pa(int arr[], int n)
{
    for(int i=0; i<n; i++)
        print f("%d", arr[i]);
    print f("\n");
}

int main()
{
    int arr[] = {4,6,13,10,9,8,15,17};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Given array :");
    pa(arr,n);
    bh(arr,n);
    printf("Array opt of heap is: \n");
    pa(arr,n);
    return 0;
}
```

Answer

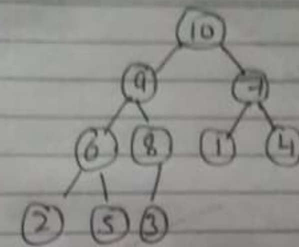
Given array: 10 2 4 5 3 1 7 6 9 8

Array representation of Heap is:

10 9 7 6 8 1 4 2 5 3

SURYA Gold

Date ____ Page ____



8/6/24

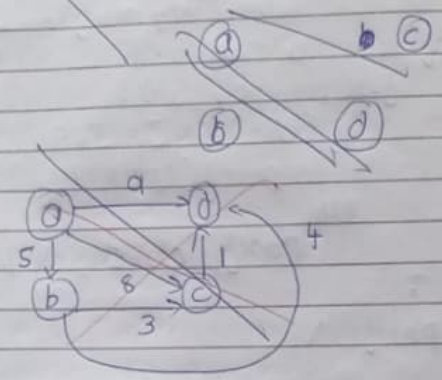
Floyd Warshall

```
#include <stdio.h>
#define V 4
#define INF 99999

void print(int dist[][V]);

void floydWarshall (int dist[][V])
{
    int i, j, k;
    for (k = 0; k < V; k++)
    {
        for (i = 0; i < V; i++)
        {
            for (j = 0; j < V; j++)
            {
                if (dist[i][j] > (dist[i][k] + dist[k][j])
                    && (dist[i][k] != INF)
                    && dist[k][j] != INF)
                dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
}
```

```
    print(dist);
}

void print(int dist[][V])
{
    printf("Following matrix shows shortest distances
            between every pair of vertices\n");
    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)
        {
            if (dist[i][j] == INF)
                printf("INF");
            else
                printf("%d\t", dist[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    int graph[V][V] = { {0, 5, INF, 10},
                        {INF, 0, 3, INF},
                        {INF, INF, 0, 1},
                        {INF, INF, INF, 0} };
    floydMarshall(graph);
    return 0;
}
```

Output

0 5 8 9
INF 0 3 4
INF INF 0 1
INF INF INF 0



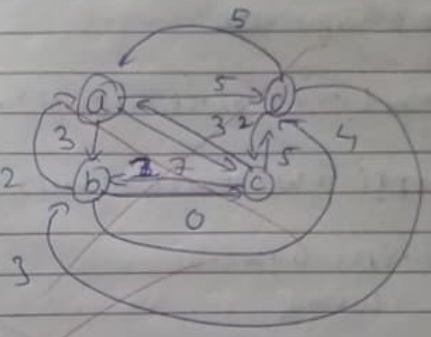
int graph[V][V] = { {0, 3, INF, 5},
{ 2, 0, INF, 4},
{ INF, 1, 0, INF},
{ INF, INF, 2, 0} };

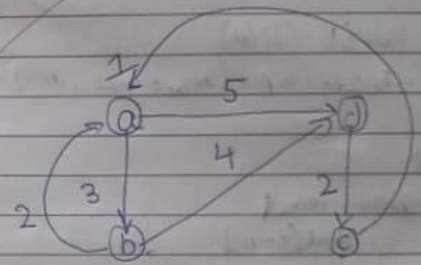
floydWarshall(graph);

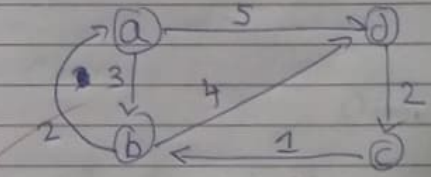
}

Output:

0 3 7 5
2 0 6 4
3 1 0 5
5 3 2 0







Knapsack

```
#include <stdio.h>
[redacted]

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

int kps(int W, int wt[], int val[], int n)
{
    int i, w;
    if (n == 0 || W == 0)
        return 0;

    if (wt[n-1] > W)
        return kps(W, wt, val, n-1);

    else
        return max(
            val[n-1]
            + kps(W, wt, val, n-1));
}

int main()
{
    int profit[] = {50, 70, 80, 100, 150};
    int weight[] = {10, 15, 25, 30, 20};
    int W = 50;
    int n = sizeof(profit) / sizeof(profit[0]);
    printf("%d", kps(W, weight, profit, n));
    return 0;
}

printf("Items:\n");
for(int i=0; i<n; i++)
{
    printf("Item %d - Profit: %d, weight: %d\n",
        i+1, profit[i], weight[i]);
}
printf("\nKnapsack capacity: %d\n", W);
int maxProfit = knapsack(W, weight, profit, n);
printf("Max profit that can be obtained: %d\n",
    maxProfit);
return 0;
}
```

Output

Items:
Item1 - Profit: 50, weight: 10
Item2 - Profit: 70, weight: 15
Item3 - Profit: 80, weight: 25
Item4 - Profit: 100, weight: 30
Item5 - Profit: 150, weight: 20

Knapsack capacity: 50
Max profit that can be obtained: 270

SURYA Gold

Date Page

	10	20	30	40	50	60	70	80	90	100
0	0	0	0	0	0	0	0	0	0	0
1	0	50	50	50	50	50	50	50	50	50
2	0	50	70	70	70	120	120	120	120	120
3	0	50	70							
4	0	50	70							
5	0	50	70							

	0	10	20	30	40	50	60	70	80	90	100
0	0	0	0	0	0	0	0	0	0	0	0
1	0	50	50	50	50	50	50	50	50	50	50
2	0	50	90	70	70	70	120	120	120	120	120
3	0	50	70	80	80	80	120	130	150	150	150
4	0	50	70	80	100	100	120	130	150	170	170
5	0	50	70	80	100	150	150	170	200	220	270

2/3/24

SURYA Gold

Date Page

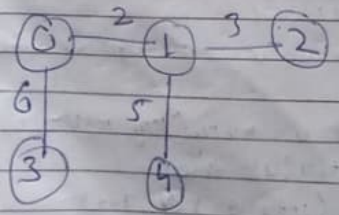
Prim's

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

int minKey(int key[], bool mstSet[])
{
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;
    return min_index;
}

void printMST(int graph[V][V])
{
    int parent[V];
    int key[V];
    bool mstSet[V];
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;
    key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < V - 1; count++)
    {
        int u = minKey(key, mstSet);
        mstSet[u] = true;
        for (int v = 0; v < V; v++)
```

```
if(graph[u][v] && mstSet[v] == false
    && graph[u][v] < key[v])
    parent[v] = u, key[v] = graph[u][v];
}

printMST(parent, graph)
}

int main()
{
    int graph[V][V] = {{0,2,0,6,0},
                       {2,0,3,8,5},
                       {0,3,0,0,7},
                       {6,8,0,0,9},
                       {0,5,7,9,0}};

    primMST(graph)
    return 0;
}
```

output

Edge	Weight
0-1	2
1-2	3
0-3	6
1-4	5



Kruskal's

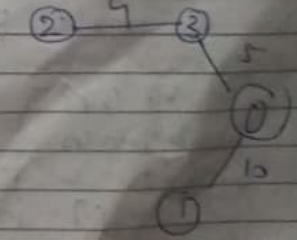
```
#include <stdio.h>
#include <stdlib.h>

int compr(const void* p1, const void* p2)
{
    const int(*x)[3] = p1;
    const int(*y)[3] = p2;
    return (*x)[2] - (*y)[2];
}

void makeset(int par[], int rank[], int n)
{
    for(int i = 0; i < n; i++){
        par[i] = i;
        rank[i] = 0;
    }
}

int findPar(int par[], int component)
{
    if(par[component] == component)
    {
        if(parent[component] == component)
            return component;
        return par[component]
            = findPar(par[par[component]]);
    }
```

```
void Unionset(int u, int v, int parent[], int rank[],
              int n)
{
    u = findParent(parent, u);
    v = findPar(parent, v);

    if (rank[u] < rank[v])
        parent[u] = v;
    else if (rank[u] > rank[v])
        parent[v] = u;
    else
    {
        parent[v] = u;
        rank[u]++;
    }
}

void KruskalAlgo(int n, int edge[n][3])
{
    qsort(edge, n, sizeof(edge[0]), comparator);
    int parent[n];
    int rank[n];
    makeset(parent, rank, n);
    int minCost = 0;
    printf("Following are edges in the
            MST\n");
    for (int i = 0; i < n; i++)
    {
        int v1 = findParent(parent, edge[i][0]);
        int v2 = findParent(parent, edge[i][1]);
        int wt = edge[i][2];
```

```
        if (v1 != v2)
        {
            unionset(v1, v2, parent, rank, n);
            minCost += wt;
            printf("%d -- %d == %d\n",
                   edge[i][0], edge[i][1],
                   wt);
        }
    }

    printf("MST : %d", minCost);
}

int main()
{
    int edge[5][3] = {{0,1,10}, {0,2,6},
                      {0,3,5}, {1,3,15},
                      {2,3,4}};
    kruskalAlgo(5, edge);
    return 0;
}
```

Op

2 -- 3 = 4
0 -- 3 = 5
0 -- 1 = = 10
MST : 19



Dijkstra

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

#define V 9

int minDist(int dist[], bool sptSet[])
{
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;
}

void printSolution(int dist[])
{
    printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}

void dijkstra(int graph[V][V], int src)
{
    int dist[V];
    bool sptSet[V];
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;
```

```
    dist[src] = 0;

    for (int v = 0; v < V; v++)
        if (!sptSet[v] && graph[u][v]
            dist[v] != INT_MAX
            dist[u] + graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
}

printSolution(dist, V);

int main()
{
    int graph[V][V] = { {0,50,0,0,0,5,0},
                        {4,0,0,0,0,0,11,2},
                        {0,6,7,14,0,0,3},
                        {0,0,9,0,14,0,0,0},
                        {0,0,3,6,10,0,0,0},
                        {6,0,0,0,0,2,0,0},
                        {8,11,0,0,0,1,0,7},
                        {0,2,0,0,0,6,7,0} };

    dijkstra(graph, 0);
    return 0;
}
```

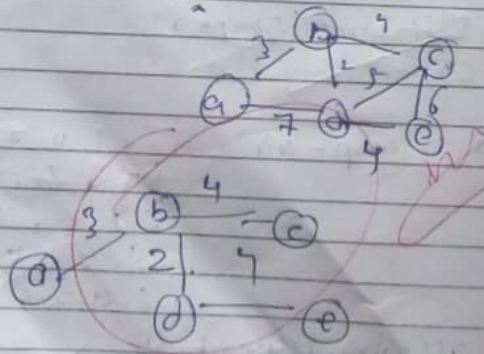
```
int graph[V][V] = { {0,3,0,7,0},
                    {3,0,4,2,0},
                    {0,4,0,5,0},
                    {7,2,5,0,3},
                    {0,0,6,4,0} };

dijkstra(graph, 0);
return 0;
}
```

Output

Vertex	Distance from Source
0	0
1	4
2	12
3	19
4	21
5	11
6	9
7	8
8	14

Output

Vertex	Distance
0	0
1	3
2	7
3	5
4	9



Fractional knapsack

```
Hi value [20] = {25, 5, 2, 1, 9};
int v[10] = {4, 2, 2, 1, 10};
int W = 15;

void simple fill()
{
    int cur_w;
    int tot_v;
    int i, maxi;
    int used[10];
    for(i=0; i<n; ++i)
        used[i] = 0;

    cur_w = W
    while(cur_w > 0) {
        maxi = -1;
        for(i=0; i<n; ++i)
            if((used[i] == 0) &&
               ((maxi == -1) || (p[i]/w[i] > p[maxi]/w[maxi])))
                maxi = i;
        used[maxi] = 1;
        cur_w -= w[maxi];
        tot_v += v[maxi];
```

```
if (cur_w >= 0)
    printf("item %d (%d$s, %d kg)
        Space left: %d.\n", maxi+1,
        v[maxi], c[maxi], cur_w);
else {
    printf("Added %d%% (%d$s, %d kg)
        of Object %d in the bag.\n",
        (int)((1+(float)cur_w/c[maxi])
        100),
    tot_v += v[maxi];
    tot_v += (1+(float)cur_w/c[maxi])
        v[maxi];
}
}
printf("Filled the bag with objects worth
    %.2f$s.\n", tot_v);
}

int main(int argc, char *argv[])
{
    simple_fill();
    return 0;
}
```

Output

```
item 5 (10$s, 4 kg) Space left : 1,
item 2 (20$s, 1 kg) Space left 10
item 3 (20$s, 2 kg) Space left : 8
item 4 (18$, 1 kg) Space left : 7
Added 58% (48$s, 12 kg) of object 1 in the bag

Total value : 17.33$s
```

SURYA Gold
Date Page

N-Queen

```
#define N 4
#include <stdio.h>
#include <stdbool.h>

void printSol(int board[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            if (board[i][j])
                printf("Q ");
            else
                printf(". ");
        printf("\n");
    }
}

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

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

    for (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
}

bool solveNQ()
{
    int board[N][N] = {{0, 0, 0, 0},
                       {0, 0, 0, 0},
                       {0, 0, 0, 0},
                       {0, 0, 0, 0}};
```

if (SolveUtil(board,0) == false)
{
cout << "Solution does not exist";
return false;
}

print();
return true

int main()
{
SolveQ();
return 0
}

output