

QUESTION:

To find the given number is check 2^n or not.

INPUT:

32

OUTPUT:

Yes

Ex No:4	POWER OF 2
Date:	

AIM:

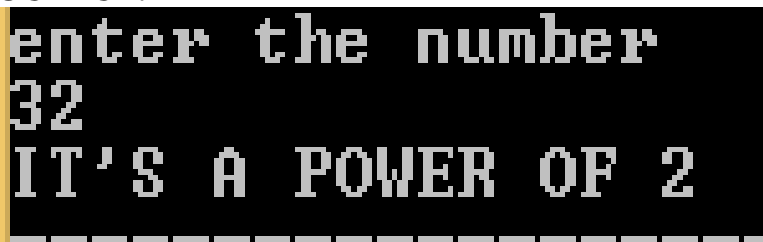
To find the given number is check 2^n or not.

PSEUDOCODE:

```
Void power(int a)
{
if(num&(num-1))
    printf("IT'S NOT A POWER OF 2");
else
    printf("IT'S A POWER OF 2");
}
```

SOURCECODE:

```
#include<stdio.h>
#include<string.h>
int main()
{
    int num;
    printf("enter the number");
    scanf("%d",&num);
    if(num&(num-1))
        printf("IT'S NOT A POWER OF 2");
    else
        printf("IT'S A POWER OF 2");
}
```

OUTPUT:A screenshot of a terminal window with a black background and white text. The text shows the program's execution: it prompts 'enter the number', the user enters '32', and the program outputs 'IT'S A POWER OF 2'.**RESULT:**

Thus the program to check power of 2 number is successfully executed and output was verified.

QUESTION :

To find the closest pair of points using closest pair algorithm

INPUT :

(1, 1) , (2, 8) , (5, 2) , (2, 9) , (8, 9)

OUTPUT:

The closest two points are (2,8) and (2,9)

Ex No:7

CLOSEST PAIR ALGORITHM

Date:

AIM :

To find the closest points using closest pair algorithm

PSEUDOCODE :

```
double shortestDistance = getDistance( points[p1][0], points[p1][1], points[p2][0], points[p2][1]);
for ( i=0;i<NUMBER_OF_POINTS;i++){
    for ( j=i+1;j<NUMBER_OF_POINTS;j++){
        double distance=getDistance(points[i][0],points[i][1],points[j][0],points[j][1]);

        if(shortestDistance > distance){
            p1=i;
            p2=j;
            shortestDistance = distance;
        }
    }
}
```

SOURCE CODE:

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

double getDistance (double x1,double y1, double x2, double y2)
{
    return (sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1)));
}

int main(){
    int i,j=0;
    int NUMBER_OF_POINTS;
    printf("enter number of points :");
    scanf("%d",&NUMBER_OF_POINTS);
    double points[NUMBER_OF_POINTS][2];
    printf ("Enter %d points: \n" , NUMBER_OF_POINTS );
    for(i=0; i< NUMBER_OF_POINTS;i++)
    {
        scanf ("%lf", &points[i][0]);
        scanf ("%lf", &points[i][1]);
    }

    int p1=0,p2=1;
    double shortestDistance = getDistance( points[p1][0], points[p1][1], points[p2][0], points[p2][1]);
    for ( i=0;i<NUMBER_OF_POINTS;i++){
```

```

    for ( j=i+1;j<NUMBER_OF_POINTS;j++){
        double distance=getDistance(points[i][0],points[i][1],points[j][0],points[j][1]);

        if(shortestDistance > distance){
            p1=i;
            p2=j;
            shortestDistance = distance;
        }
    }
}

printf ("The closest two points are (%.0lf,%.0lf) and (%.0lf,%.0lf) \n",
points[p1][0],points[p1][1],points[p2][0],points[p2][1]);
return 0;
}

```

OUTPUT :

```

enter number of points :5
Enter 5 points:
1 1
2 8
5 2
2 9
8 9
The closest two points are (2,8) and (2,9)

```

RESULT:

Thus the closest pair Algorithm is implemented and the output is verified

QUESTION :

Given a set of points .Your task is to find the set of points which includes all the other points inside i.e should compute the boundary in which all other points lies inside the boundary

INPUT :

{0, 3}, {2, 2}, {1, 1}, {2, 1}, {3, 0}, {0, 0}, {3, 3}

OUTPUT :

(0,3) , (0,0) , (3,0) , (3,3)

Ex No:8	CONVEX HULL ALGORITHM
Date:	

AIM :

To construct convex hull points using Convex Hull Algorithm

PSEUDOCODE :

```
void convexHull(struct Point points[], int n)
{
    if (n < 3) return;
    struct Point hull[100];
    int k=0;
    int l = 0,i;
    for (i = 1; i < n; i++)
        if (points[i].x < points[l].x)
            l = i;
    int p = l, q;
    do
    {
        hull[k]=points[p];
        k++;
        q = (p+1)%n;
        for ( i = 0; i < n; i++)
        {
            if (orientation(points[p], points[i], points[q]) == 2)
                q = i;
        }
        p = q;
    } while (p != l);
    for (i = 0; i < k; i++)
        printf("(%d,%d)\n",hull[i].x ,hull[i].y );
}
```

SOURCE CODE:

```
#include<stdio.h>

struct Point
{
    int x, y;
};

int orientation(struct Point p, struct Point q, struct Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0;
```

```

    return (val > 0)? 1: 2;
}

void convexHull(struct Point points[], int n)
{
    if (n < 3) return;
    struct Point hull[100];
    int k=0;
    int l = 0,i;
    for (i = 1; i < n; i++)
        if (points[i].x < points[l].x)
            l = i;

    int p = l, q;
    do
    {
        hull[k]=points[p];
        k++;
        q = (p+1)%n;
        for ( i = 0; i < n; i++)
        {
            if (orientation(points[p], points[i], points[q]) == 2)
                q = i;
        }
        p = q;
    } while (p != l);
    for (i = 0; i < k; i++)
        printf("(%d,%d)\n",hull[i].x ,hull[i].y );
}

int main()
{
    struct Point points[100];
    int n,i;
    printf("Enter number of points :");
    scanf("%d",&n);
    printf("Enter the points : \n");
    for(i=0;i<n;i++)
    {
        scanf("%d %d",&points[i].x,&points[i].y);
    }
    convexHull(points, n);
    return 0;
}

```


OUTPUT :

```
Enter number of points :7
Enter the points :
0 3
2 2
1 1
2 1
3 0
0 0
3 3
(0,3)
(0,0)
(3,0)
(3,3)
```

RESULT:

Thus the Convex Hull Algorithm is implemented and the output is verified

QUESTION :

Given a problem consists of n jobs each associated with a deadline and profit and our objective is to earn maximum profit. We will earn profit only when job is completed on or before deadline. We assume that each job will take unit time to complete.

INPUT:

Consider the following 5 jobs and their associated deadline and profit.

index	1	2	3	4	5
JOB	j1	j2	j3	j4	j5
DEADLINE	2	1	3	2	1
PROFIT	60	100	20	40	20

OUTPUT :

Required Jobs : j2 => j1 => j3

Max Profit : 180

Ex No:19	JOB SEQUENCING ALGORITHM
Date:	

AIM :

To find the maximum profit of the scheduled jobs within the deadlines

PSEUDOCODE:

```
for i = 1 to n do
  Set k = min(dmax, DEADLINE(i))
  while k >= 1 do
    if timeslot[k] is EMPTY then
      timeslot[k] = job(i)
      break
    endif
    Set k = k - 1
  endwhile
endfor
```

PSEUDOCODE:

```
#include <stdio.h>
#define MAX 100
int dmax = 0;

typedef struct Job {
    char id[10];
    int deadline;
    int profit;
} Job;

void jobSequencingWithDeadline(Job jobs[], int n);

int minValue(int x, int y) {
    if(x < y) return x;
    return y;
}

int main(void) {
    int i, j;
    Job jobs[100];
    int n;
    printf("Enter Number of Jobs :");
    scanf("%d",&n);
    printf("Enter the Job name with deadline and profit : \n");
    for(i=0;i<n;i++)
    {
        scanf("%s %d %d",jobs[i].id,&jobs[i].deadline,&jobs[i].profit);
        if(jobs[i].deadline > dmax) {
            dmax = jobs[i].deadline;
        }
    }
}
```

```

    }
}
Job temp;
for(i = 1; i < n; i++) {
    for(j = 0; j < n - i; j++) {
        if(jobs[j+1].profit > jobs[j].profit) {
            temp = jobs[j+1];
            jobs[j+1] = jobs[j];
            jobs[j] = temp;
        }
    }
}
jobSequencingWithDeadline(jobs, n);
return 0;
}

```

```

void jobSequencingWithDeadline(Job jobs[], int n) {
    int i, j, k, maxprofit;
    int timeslot[MAX];
    int filledTimeSlot = 0;

    printf("\ndeadline: %d\n", dmax);

    for(i = 1; i <= dmax; i++) {
        timeslot[i] = -1;
    }

    for(i = 1; i <= n; i++) {
        k = minValue(dmax, jobs[i - 1].deadline);
        while(k >= 1) {
            if(timeslot[k] == -1) {
                timeslot[k] = i-1;
                filledTimeSlot++;
                break;
            }
            k--;
        }

        if(filledTimeSlot == dmax) {
            break;
        }
    }

    printf("\nRequired Jobs: ");
    for(i = 1; i <= dmax; i++) {
        printf("%s", jobs[timeslot[i]].id);

        if(i < dmax) {
            printf(" --> ");
        }
    }

    maxprofit = 0;
    for(i = 1; i <= dmax; i++) {

```

```
        maxprofit += jobs[timeslot[i]].profit;
    }
    printf("\nMax Profit: %d\n", maxprofit);
}
```

OUTPUT :

```
Enter Number of Jobs :5
Enter the Job name with deadline and profit :
a 2 60
b 1 100
c 3 20
d 2 40
e 1 20

deadline: 3

Required Jobs: b --> a --> c
Max Profit: 180
```

RESULT:

Thus the maximum profit for a given deadline is implemented and the output is verified

QUESTION :

To find the minimum spanning tree (minimum cost) using Prim's Algorithm

INPUT :

```
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
```

OUTPUT :

```
1 -> 1 : cost=2
2 -> 2 : cost=3
3 -> 2 : cost=5
4 -> 1 : cost=4
```

Total cost of spanning tree=16

Ex No:20

MST USING PRIM'S ALGORITHM

Date:

AIM :

To find the minimum spanning tree using prim's algorithm

PSEUDOCODE :

```
ne=1
while(ne < n)
{
    for(i=1,min=999;i<=n;i++)
    for(j=1;j<=n;j++)
    if(cost[i][j]< min)
    if(visited[i]!=0)
    {
        min=cost[i][j];
        a=u=i;
        b=v=j;
    }
    if(visited[u]==0 || visited[v]==0)
    {
        printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
        mincost+=min;
        visited[b]=1;
    }
    cost[a][b]=cost[b][a]=999;
}
return mincost
```

PSUDOCODE:

```
#include<stdio.h>

int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
{
    printf("\nEnter the number of nodes:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");
    for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
    {
        scanf("%d",&cost[i][j]);
        if(cost[i][j]==0)
            cost[i][j]=999;
    }
    visited[1]=1;
    printf("\n");
    while(ne < n)
```

```

    {
        for(i=1,min=999;i<=n;i++)
        for(j=1;j<=n;j++)
        if(cost[i][j]< min)
        if(visited[i]!=0)
        {
            min=cost[i][j];
            a=u=i;
            b=v=j;
        }
        if(visited[u]==0 || visited[v]==0)
        {
            printf("\n %d -> %d : cost:%d",ne++,a,b,min);
            mincost+=min;
            visited[b]=1;
        }
        cost[a][b]=cost[b][a]=999;
    }

    printf("\n Minimun cost=%d",mincost);
}

```

OUTPUT :

```

Enter the number of nodes:5
Enter the adjacency matrix:
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

1 -> 1 : cost:2
2 -> 2 : cost:3
3 -> 2 : cost:5
4 -> 1 : cost:4
Minimun cost=16

```

RESULT:

Thus the MST using prim's algorithm is implemented and the output is verified

QUESTION :

To find the minimum spanning tree using Kruskal's Algorithm

INPUT :

```
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0
```

OUTPUT:

```
1 -> 1 : cost=3
2 -> 4 : cost=6
3 -> 1 : cost=2
4 -> 2 : cost=5
5 -> 3 : cost=6
Minimum cost = 13
```

Ex No:21

KRUSKAL'S ALGORITHM

Date:

AIM :

To find the minimum spanning tree using Kruskal's Algorithm

PSEUDOCODE:

MST-KRUSKAL(G, w)

$A \leftarrow \emptyset$

for each vertex $v \in V[G]$

do MAKE-SET(v)

sort the edges of E into non-decreasing order by weight w

for each edge $(u, v) \in E$, taken in non-decreasing order by weight

do if FIND-SET(u) \neq FIND-SET(v)

then $A \leftarrow A \cup \{(u, v)\}$

UNION(u, v)

return A

PSEUDOCODE:

```
#include<stdio.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
    printf("\nEnter the no. of vertices:");
    scanf("%d",&n);
    printf("\nEnter the cost adjacency matrix:\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
                cost[i][j]=999;
        }
    }
    printf("The edges of Minimum Cost Spanning Tree are\n");
    while(ne < n)
```

```

    {
        for(i=1,min=999;i<=n;i++)
        {
            for(j=1;j <= n;j++)
            {
                if(cost[i][j] < min)
                {
                    min=cost[i][j];
                    a=u=i;
                    b=v=j;
                }
            }
            u=find(u);
            v=find(v);
            if(uni(u,v))
            {
                printf("\n %d -> %d : cost=%d",ne++,a,b,min);
                mincost +=min;
            }
            cost[a][b]=cost[b][a]=999;
        }
        printf("\n\tMinimum cost = %d\n",mincost);
    }
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;
}

```

OUTPUT :

```
Enter the no. of vertices:6
Enter the cost adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0
The edges of Minimum Cost Spanning Tree are
1 -> 1 : cost=3
2 -> 4 : cost=6
3 -> 1 : cost=2
4 -> 2 : cost=5
5 -> 3 : cost=6
      Minimum cost = 13
```

RESULT:

Thus the MST using Kruskal's Algorithm is implemented and the output is verified

QUESTION :

Finding the shortest path using Dijkstra's Algorithm

INPUT :

```
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0
```

OUTPUT:

Enter the starting node:: 1

Distance of 0 = 10

Path = 0 <=1

Distance of 2 = 50

Path = 2 <=1

Distance of 3 = 40

Path = 3 <= 0 <=1

Distance of 4 = 60

Path = 4 <= 2 <=1

Ex No:22

Date:

DIJKSTRA'S ALGORITHM

AIM :

To find single source shortest path using Dijkstra's Algorithm

PSEUDOCODE :

DIJKSTRA(G,s)

INITIALIZE-SINGLE-SOURCE(G, S)

$S \leftarrow \emptyset$

$Q \leftarrow V[G]$

while $Q \neq \emptyset$

do $u \leftarrow \text{EXTRACT-MIN}(Q)$

$S \leftarrow S \cup \{u\}$

for each vertex $v \in \text{Adj}[u]$

do if $\text{dist}[v] > \text{dist}[u] + w(u,v)$

then $d[v] \leftarrow d[u] + w(u,v)$

PSEUDOCODE:

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

```
#define MAX 10
```

```
void dijkstra(int G[MAX][MAX], int n, int startnode);
```

```
void main(){
```

```
    int G[MAX][MAX], i, j, n, u;
```

```
    printf("\nEnter the no. of vertices:: ");
```

```
    scanf("%d", &n);
```

```
    printf("\nEnter the adjacency matrix::\n");
```

```
    for(i=0;i < n;i++)
```

```
        for(j=0;j < n;j++)
```

```
            scanf("%d", &G[i][j]);
```

```
    printf("\nEnter the starting node:: ");
```

```
    scanf("%d", &u);
```

```
    dijkstra(G,n,u);
```

```
    getch();
```

```
}
```

```
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]=9999;
            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){
        mindistance=9999;
        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 of %d = %d", i, distance[i]);
            printf("\nPath = %d", i);
            j=i;
            do
            {
                j=pred[j];
                printf(" <-%d", j);
            }
            while(j!=startnode);
        }
}

```

OUTPUT :

```
Enter the no. of vertices:: 5
```

```
Enter the adjacency matrix::
```

```
0  10  0  30  100
10  0  50  0  0
0  50  0  20  10
30  0  20  0  60
100 0  10  60  0
```

```
Enter the starting node:: 1
```

```
Distance of 0 = 10
```

```
Path = 0 <-1
```

```
Distance of 2 = 50
```

```
Path = 2 <-1
```

```
Distance of 3 = 40
```

```
Path = 3 <-0 <-1
```

```
Distance of 4 = 60
```

```
Path = 4 <-2 <-1
```

RESULT:

Thus the single source shortest path is implemented using Dijkstra's Algorithm and the output is verified

QUESTION:

Find the optimum Binary Search(minimum cost) Tree for a given vertices

INPUT : -

4
12 15 20 25
4 3 6 2

OUTPUT : -

27

Ex No:27

OPTIMAL BINARY SEARCH TREE

Date:

AIM:-

To find the optimal binary Search Tree (Minimum cost tree)

PSEUDOCODE :-

```
int optCost(int freq[], int i, int j)
{
    if (j < i)
        return 0;
    if (j == i)
        return freq[i];

    int fsum = sum(freq, i, j);
    int min = INT_MAX;
    for (int r = i; r <= j; ++r)
    {
        int cost = optCost(freq, i, r-1) +
                    optCost(freq, r+1, j);
        if (cost < min)
            min = cost;
    }
    return min + fsum;
}
```

```
int optimalSearchTree(int vertex[], int freq[], int n)
{
    return optCost(freq, 0, n-1);
}
```

SOURCE CODE :

```
#include <stdio.h>
#include <limits.h>
int sum(int freq[], int i, int j);
int optimalBinarySearchTree(int keys[], int freq[], int n)
{
    int cost[n][n], i, L;
    for (i = 0; i < n; i++)
        cost[i][i] = freq[i];
    for (L=2; L<=n; L++)
    {
        for (i=0; i<=n-L+1; i++)
        {
            int j = i+L-1, r;
            cost[i][j] = INT_MAX;
            for (r=i; r<=j; r++)
            {
```

```

        int c = 0;
        if (r > i)
        {
            c = c + cost[i][r-1];
        }
        else
        {
            c = c + 0;
        }
        if (r < j)
        {
            c = c + cost[r+1][j];
        }
        else {
            c = c + 0;
        }
        c = c + sum(freq, i, j);
        if (c < cost[i][j])
            cost[i][j] = c;
    }
}
return cost[0][n-1];
}

int sum(int freq[], int i, int j)
{
    int s = 0, k;
    for (k = i; k <= j; k++)
        s += freq[k];
    return s;
}

int main()
{
    int n, i;
    printf("enter number of nodes \n");
    scanf("%d", &n);
    int vertex[n];
    int freq[n];
    printf("Enter the vertices : \n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &vertex[i]);
    }
    printf("Enter the Respective frequencies \n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &freq[i]);
    }
    printf("Cost of Optimal BST is %d ", optimalBinarySearchTree(vertex, freq, n));
    return 0;
}

```

OUTPUT:-

```
enter number of nodes
4
Enter the vertices :
12 15 20 25
Enter the Respective frequencies
4 3 6 2
Cost of Optimal BST is 27
Process returned 0 (0x0)   execution time : 28.311 s
Press any key to continue.
```

RESULT: -

Thus the program to find optimal binary search tree is executed and the output is verified

QUESTION:-

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

Input :-

3 4 5 6 2 5

Output :-

154

Ex No:28

Date:

MATRIX CHAIN MULTIPLICATION

AIM:-

To find the minimum number of multiplication of the multi matrix

PSEUDOCODE :-

```
int MatrixChain(int p[], int i, int j)
{
    if(i == j)
        return 0;
    int k;
    int min = INT_MAX;
    int count;
    for (k = i; k < j; k++)
    {
        count = MatrixChainOrder(p, i, k) +
                MatrixChainOrder(p, k+1, j) +
                p[i-1]*p[k]*p[j];

        if (count < min)
            min = count;
    }
    return min;
}
```

SOURCECODE :-

```
#include<stdio.h>
#include<limits.h>
int MatrixChain(int p[], int n)
{
    int m[n][n];
    int i, j, k, L, min;
    for (i=1; i<n; i++)
        m[i][i] = 0;
    for (L=2; L<n; L++)
    {
        for (i=1; i<n-L+1; i++)
        {
            j = i+L-1;
            m[i][j] = INT_MAX;
            for (k=i; k<=j-1; k++)
            {
                min = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
                if (min < m[i][j])

```

```

        m[i][j] = min;
    }
}

return m[1][n-1];
}

int main()
{
    int arr[100],n,i;
    printf("enter the number of matrix :");
    scanf("%d",&n);
    printf("Enter the sequence of Matrix \n");
    for(i=0;i<=n;i++){
        scanf("%d",&arr[i]);
    }

    printf("Minimum number of multiplications is %d ",MatrixChain(arr, n+1));
    return 0;
}

```

OUTPUT :-

```

enter the number of matrix : 5
Enter the sequence of Matrix
3 4 5 6 2 5
Minimum number of multiplications is 154
Process returned 0 (0x0)   execution time : 15.960 s
Press any key to continue.

```

RESULT :-

Thus the minimum multiplication cost of the given set of matrix is compiled and the output is verified

QUESTION:

You are Given a graph G . Your task is to find the minimum number of colors used to color the vertices so that no two colors are adjacent to each others

INPUT :

Enter the number of nodes & edges

4 5

edge : 1

(0 1)

edge : 2

(1 3)

edge : 3

(3 2)

edge : 4

(2 1)

edge : 5

(1 0)

OUTPUT :

The Vertices To be Coloured As...

V1:=1

V2:=2

V3:=3

V4:=1

The Chromatic Number is 3

Ex No:29	GRAPH COLORING PROBLEM
Date:	

AIM :

To find the chromatic number of the given graph

PSEUDOCODE:

```
void Gen_Col_Value(int k,int n)
{
    while(1)
    {
        a=color_tab[k]+1;
        b=m+1;
        color_tab[k] = a%b;
        if(color_tab[k]==0) return;
        for(j=1;j<=n;j++)
        {
            if(G[k][j] && color_tab[k]==color_tab[j])
                break;
        }
        if(j==n+1) return;
    }
}
void Gr_coloring(int k,int n)
{
    Gen_Col_Value(k,n);
    if(color_tab[k]==0) return;
    if(k==n) return;
    else Gr_coloring(k+1,n);
}
```

ALGORITHM:

```
#include<stdio.h>
int G[10][10],m,edges,color_tab[10],v1,v2,i,j,n,a,b;
void Gen_Col_Value(int,int);
void Gr_coloring(int,int);
int main()
{
    printf("\nEnter the number of nodes & edges\n");
    scanf("%d%d",&n,&edges);
    m=n-1;
    printf("\nEnter the edges of the graph\n\n");
    for (i=1;i<=edges; i++)
    {
        printf("Enter starting and ending of edge : %d\n",i);
        scanf("%d%d",&v1,&v2);
        G[v1][v2] = G[v2][v1] = 1;
        printf("\n");
    }
}
```

```

}
Gr_coloring(1,n);
printf("\n The Vertices To be Coloured As...\n");
int max=0;
for(i=1;i<=n;i++){
    printf("\n V%d:=%d",i,color_tab[i]);
    if(color_tab[i]>max)
        max=color_tab[i];
}
printf("\nThe chromatic color is %d",max);
return 0;
}
void Gen_Col_Value(int k,int n)
{
    while(1)
    {
        a=color_tab[k]+1;
        b=m+1;
        color_tab[k] = a%b;
        if(color_tab[k]==0) return;
        for(j=1;j<=n;j++)
        {
            if(G[k][j] && color_tab[k]==color_tab[j])
                break;
        }
        if(j==n+1) return;
    }
}
void Gr_coloring(int k,int n)
{
    Gen_Col_Value(k,n);
    if(color_tab[k]==0) return;
    if(k==n) return;
    else Gr_coloring(k+1,n);
}

```

OUTPUT :

```
Enter the number of nodes & edges
4 5
Enter the edges of the graph
Enter starting and ending of edge : 1
0 1
Enter starting and ending of edge : 2
1 3
Enter starting and ending of edge : 3
3 2
Enter starting and ending of edge : 4
2 1
Enter starting and ending of edge : 5
1 0

The Vertices To be Coloured As...
U1:=1
U2:=2
U3:=3
U4:=1
The chromatic color is 3
```

RESULT:

Thus the Chromatic number of the graph is found using graph coloring algorithm

N-QUEEN:

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other.

Input:

16

Output:

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

Ex No:30

N-queen problem

Date:

AIM:

Implement the N-queen problem using Backtracking.

PSEUDOCODE:

int isSafe(int board[N][N], int row, int col)

```
{
    int i, j;
    for (i = 0; i < col; i++)
        if (board[row][i])
            return 0;
    for (i=row, j=col; i>=0 && j>=0; i--, j--)
        if (board[i][j])
            return 0;
    for (i=row, j=col; j>=0 && i<N; i++, j--)
        if (board[i][j])
            return 0;
    return 1;
}
```

int solveNQUtil(int board[N][N], int col)

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

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

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

    return 0;
}
```

SOURCECODE:

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

int isSafe(int board[N][N], int row, int col)
{
    int i, j;

    for (i = 0; i < col; i++)
        if (board[row][i])
            return 0;

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

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

    return 1;
}

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

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

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

```
    }

    return 0;
}

int solveNQ()
{
    int i,j;
    int board[N][N];
    for(i=0;i<N;i++)
    {
        for(j=0;j<N;j++)
        {
            board[i][j]=0;
        }
    }

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

    printSolution(board);
    return 1;
}

int main()
{
    printf("enter the number of queens");
    scanf("%d",&N);
    solveNQ();
    return 0;
}
```

OUTPUT:

```
enter the number of queens16
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
```

Process returned 0 (0x0) execution time : 1.866 s
Press any key to continue.

RESULT:

Thus the program to implement the N-queen problem was implemented and the output was verified.

QUESTION:

Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian Path such that there is an edge (in graph) from the last vertex to the first vertex of the Hamiltonian Path.

INPUT

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

OUTPUT

```
0->1->2->4->3->0
```

Ex No:31

HAMILTONIAN PATH PROBLEM

Date:

AIM

To implement algorithm for Hamiltonian Cycle

PSEUDOCODE

```
function check_all_permutations(adj[][], n)
    for i = 0 to n
        p[i]=i

    while next permutation is possible
        valid = true
        for i = 0 to n-1
            if adj[p[i]][p[i+1]] == false
                valid = false
                break
        if valid == true
            return true
        p = get_next_permutation(p)
    return false
```

PSEUDOCODE:

```
#include<stdio.h>
int V;

void printSolution(int path[]);
int isSafe(int v, int graph[V][V], int path[], int pos)
{
    int i;
    if (graph [ path[pos-1] ][ v ] == 0)
        return 0;
    for (i = 0; i < pos; i++)
        if (path[i] == v)
            return 0;
    return 1;
}
int hamCycleUtil(int graph[V][V], int path[], int pos)
{
    int v;
    if (pos == V)
    {
        if ( graph[ path[pos-1] ][ path[0] ] == 1 )
            return 1;
        else
            return 0;
    }
}
```

```

for (v = 1; v < V; v++)
{
    if (isSafe(v, graph, path, pos))
    {
        path[pos] = v;
        if (hamCycleUtil (graph, path, pos+1) == 1)
            return 1;
        path[pos] = -1;
    }
}
return 0;
}

int hamCycle(int graph[V][V])
{
    int i;
    int path[5];
    for (i = 0; i < V; i++)
        path[i] = -1;
    path[0] = 0;
    if ( hamCycleUtil(graph, path, 1) == 0 )
    {
        printf("\nSolution does not exist");
        return 0;
    }

    print(path);
    return 1;
}

void print(int path[])
{
    int i;
    printf ("Hamiltonian Cycle is\n");
    for (i = 0; i < V; i++)
        printf("%d->", path[i]);
    printf(" %d ", path[0]);
    printf("\n");
}

int main()
{
    int i,j;
    printf("Enter the number of vertex : ");
    scanf("%d",&V);
    int graph[V][V];
    printf("Enter the adjacency matrix for hamiltonian cycle\n");
    for(i=0;i<V;i++){
        for(j=0;j<V;j++){
            scanf("%d",&graph[i][j]);
        }
    }
    hamCycle(graph);
    return 0;
}

```

OUTPUT :

```
Enter the number of vertex : 5
Enter the adjacency matrix for hamiltonian cycle
0 1 0 1 0
1 0 1 1 1
0 1 0 0 1
1 1 0 0 1
0 1 1 1 0
Hamiltonian Cycle is
0->1->2->4->3-> 0
```

RESULT:

Thus the hamiltonian path is implemented and the output is verified

QUESTION:

Given a set of non-negative integers, and a value sum, determine if there is a subset of the given set with sum equal to given sum.

INPUT:

Array size:3

Array value:1 , 2 , 3

Expected sum:6

OUTPUT:

Subset found

Ex No:32

SUBSET SUM

Date:

AIM:

To find if there is any sum of subset for a given sum

PSEUDOCODE:

```
bool isSubsetSum(int set[], int n, int sum)
{
    if (sum == 0)
        return true;
    if (n == 0 && sum != 0)
        return false;
    if (set[n-1] > sum)
        return isSubsetSum(set, n-1, sum);

    return isSubsetSum(set, n-1, sum) || isSubsetSum(set, n-1, sum-set[n-1]);
}
```

SOURCECODE:

```
#include <stdio.h>
int isSum(int set[], int n, int sum)
{
    int subset[n+1][sum+1],i,j;

    for (i = 0; i <= n; i++)
        subset[i][0] = 1;

    for (i = 1; i <= sum; i++)
        subset[0][i] = 0;
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= sum; j++)
        {
            if(j<set[i-1])
                subset[i][j] = subset[i-1][j];
            if (j >= set[i-1])
                subset[i][j] = subset[i-1][j] ||
                    subset[i - 1][j-set[i-1]];
        }
    }

    return subset[n][sum];
}

int main()
{
    int n,i,sum;
    printf("enter the size of array :");
```

```
scanf("%d",&n);
int arr[n];
printf("Enter the elements in array : ");
for(i=0;i<n;i++){
    scanf("%d",&arr[i]);
}
printf("Enter the expected sum : ");
scanf("%d",&sum);
if (isSum(arr, n, sum) == 1)
    printf("Found");
else
    printf("Not Found");
return 0;
}
```

OUTPUT:-

```
enter the size of array :3
Enter the elements in array : 1 2 3
Enter the expected sum : 6
Found
Process returned 0 (0x0)   execution time : 4.457 s
Press any key to continue.
```

Result:

Thus the program to print whether the sum available in subset or not is compiled and the output is verified

QUESTION :

The problem consist of a sorted array with positive numbers and a key X .your task is to find whether the element is present in the array using randomised search. If found return the index of the element present

INPUT :

ARR[]={25 , 30 , 33 , 87 , 121}

K=87

OUTPUT :

Element is present at index 3

Ex No:33	RANDOMISED SEARCH
Date:	

AIM :

To search the element in the sorted array using randomised search algorithm

PSEUDOCODE :

```
int getRandom(int x, int y)
{
    srand(time(NULL));
    return (x + rand() % (y-x+1));
}
int randomizedBinarySearch(int arr[], int l,
                           int r, int x)
{
    if (r >= l)
    {
        int mid = getRandom(l, r);
        if (arr[mid] == x)
            return mid;
        if (arr[mid] > x)
            return randomizedBinarySearch(arr, l,
                                           mid-1, x);
        return randomizedBinarySearch(arr, mid+1,
                                       r, x);
    }
    return -1;
}
```

SOURCECODE:

```
#include<stdio.h>
int getRandom(int x, int y)
{
    srand(time(NULL));
    return (x + rand() % (y-x+1));
}
int randomizedBinarySearch(int arr[], int l,int r, int x)
{
    if (r >= l)
    {
        int mid = getRandom(l, r);
        if (arr[mid] == x)
            return mid;
        if (arr[mid] > x)
            return randomizedBinarySearch(arr, l, mid-1, x);
        return randomizedBinarySearch(arr, mid+1, r, x);
    }
    return -1;
}
```

```

int main(void)
{
    int arr[100],n,i,x;
    printf("Enter Number of elements : ");
    scanf("%d",&n);
    printf("Enter the elements : \n");
    for(i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    printf("Enter the key to be searched : ");
    scanf("%d",&x);
    int result = randomizedBinarySearch(arr, 0, n-1, x);
    (result == -1)? printf("Element is not present in array")
    : printf("Element is present at index %d", result);
    return 0;
}

```

OUTPUT :

```

Enter Number of elements : 5
Enter the elements :
25 30 33 87 121
Enter the key to be searched : 87
Element is present at index 3
Process returned 0 (0x0)   execution time : 17.593 s
Press any key to continue.

```

RESULT:

Thus the element searching is performed using randomised search algorithm

QUESTION:

Given an array of unsorted elements . your task is to sort the array using randomised quick sort

INPUT :

3 4 5 78 62 1 9 6 8 3

OUTPUT :

1 3 3 4 5 6 8 9 62 78

Ex No:34	RANDOMISED QUICK SORT
Date:	

AIM :

To sort the given array using randomised quick sort

PSEUDOCODE :

```
int partition_r(int arr[], int low, int high)
{
    int temp;
    srand(time(NULL));
    int random = low + rand() % (high - low);

    temp=arr[random];
    arr[random]=arr[high];
    arr[high]=temp;

    return partition(arr, low, high);
}

int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1);
}
```

SOURCECODE:

```
#include<stdio.h>
int partition(int arr[], int low, int high)
{
    int i,j,temp;
    int pivot = arr[high];
    i = (low - 1);

    for (j = low; j <= high - 1; j++) {
        if (arr[j] <= pivot) {

            i++;
```

```

        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);
}

int partition_r(int arr[], int low, int high)
{
    int temp;
    srand(time(NULL));
    int random = low + rand() % (high - low);

    temp=arr[random];
    arr[random]=arr[high];
    arr[high]=temp;

    return partition(arr, low, high);
}

void quickSort(int arr[], int low, int high)
{
    if (low < high) {
        int pivot = partition_r(arr, low, high);
        quickSort(arr, low, pivot - 1);
        quickSort(arr, pivot + 1, high);
    }
}

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

int main()
{
    int arr[100],n,i;
    printf("Enter Number of elements to be sorted : ");
    scanf("%d",&n);
    printf("Enter the elements : \n");
    for(i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    quickSort(arr, 0, n - 1);
    printf("Sorted array after random quick sort: \n");
    printArray(arr, n);
    return 0;
}

```

}

OUTPUT :

```
Enter Number of elements to be sorted : 10
Enter the elements :
3 4 5 78 62 1 9 6 8 3
Sorted array after random quick sort:
1 3 3 4 5 6 8 9 62 78
```

RESULT:

Thus the array is sorted and the output is verified using randomised quick sort

QUESTION: -

Given a set of cities and distance between every pair of cities, the problem is to find the minimum cost of route that visits every city exactly once and returns to the starting point.

INPUT:-

Enter the Adjacency Matrix

0	10	15	20
5	0	9	10
6	13	0	12
8	8	9	0

OUTPUT : -

The Travelling sales man distance is : 35

Ex No:35

TRAVELLING SALESMAN PROBLEM

Date:

AIM : -

To find the minimum cost of travelling from the starting vertex and return back to the same vertex by visiting all other vertex exactly ones

PSEUDOCODE :-

```
int tsp(int mask,int pos){  
    if(mask==VISITED_ALL){  
        return dist[pos][0];  
    }  
  
    int ans = INT_MAX , city;  
    for(city=0;city<n;city++){  
        if((mask&(1<<city))==0){  
            int newAns = dist[pos][city] + tsp( mask|(1<<city), city);  
            ans = min(ans, newAns);  
        }  
    }  
  
    return ans;  
}
```

SOURCE CODE: -

```
#include<stdio.h>  
#define INT_MAX 999999  
  
int n=4,i,j;  
int dist[10][10];  
int VISITED_ALL;  
int dp[16][4];  
  
int min(int x,int y){  
    if(x<y){  
        return x;  
    }  
    else if (x>y){  
        return y;  
    }  
    else{  
        return x;  
    }  
}
```



```

}
}

int tsp(int mask,int pos){

    if(mask==VISITED_ALL){
        return dist[pos][0];
    }
    if(dp[mask][pos]!=-1){
        return dp[mask][pos];
    }
    int ans = INT_MAX,city;
    for(city=0;city<n;city++){

        if((mask&(1<<city))==0){

            int newAns = dist[pos][city] + tsp( mask|(1<<city), city);
            ans = min(ans, newAns);

        }

    }

    return dp[mask][pos] = ans;
}

int main(){

    VISITED_ALL = (1<<n) -1;
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            scanf("%d",&dist[i][j]);
        }
        for(i=0;i<(1<<n);i++){
            for(j=0;j<n;j++){
                dp[i][j] = -1;
            }
        }
        printf("Travelling Saleman Distance is %d",tsp(1,0));

    }

return 0;
}

```

OUTPUT :-

```
enter the number of nodes : 4
Enter the Adjacency Matrix
0 10 15 20
5 0 9 10
6 13 0 12
8 8 9 0
Travelling Saleman Distance is 35
Process returned 0 (0x0)   execution time : 44.880 s
Press any key to continue.
```

RESULT :-

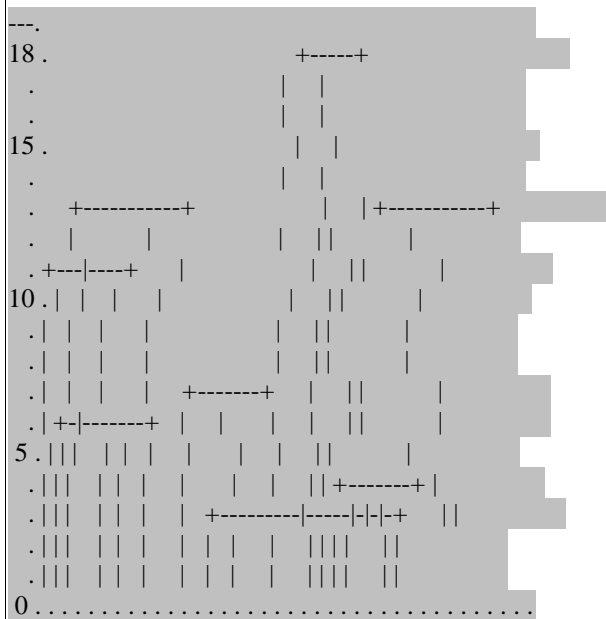
Thus the minimum cost of traveling from the starting vertex and comes back to the same vertex with visiting all nodes exactly ones is executed and the output is verified

QUESTION : -

Given n rectangular buildings in a 2-dimensional city, computes the skyline of these buildings, eliminating hidden lines. The main task is to view buildings from a side and remove all sections that are not visible. All buildings share common bottom and every **building** is represented by triplet (left, right,height)

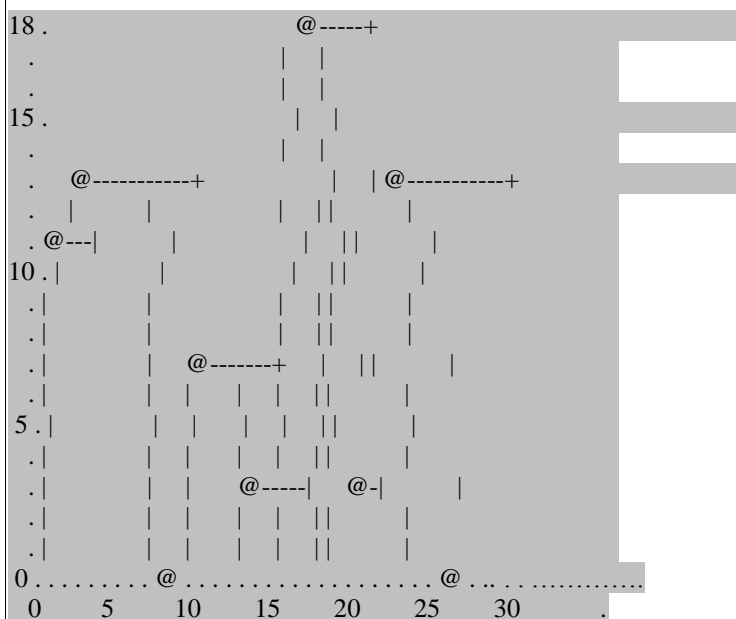
INPUT : -

[[1, 5, 11], [2, 7, 6], [3, 9, 13], [12, 16, 7], [14, 25, 3],
[19, 22, 18], [23, 29, 13], [24, 28, 4]]



OUTPUT:-

[[1, 11], [3, 13], [9, 0], [12, 7], [16, 3], [19, 18], [22, 3], [23, 13], [29, 0]]



Ex No:	SKYLINE-PROBLEM
Date:	

AIM :-

To find the outer view co-ordinates of the building

PSEUDOCODE:-

```
for(i=0;i<n*n;i++)
{
    if(arr[i].start==1){
        insert(arr[i].height);
        max=pri_que[front];
        if(tempmax!=max){
            printf("[%d , %d]\n",arr[i].x,max);
            tempmax=max;
        }
    }
    if(arr[i].start==0){
        delete(arr[i].height);
        max=pri_que[front];
        if(tempmax!=max){
            printf("[%d , %d]\n",arr[i].x,max);
            tempmax=max;
        }
    }
}
```

SOURCE CODE:-

```
#include<stdio.h>

void insert(int);
void delete(int);
void create();
void check(int);
void display_pqueue();

int pri_que[100];
int front=-1, rear=-1;

struct build{
int x;
int height;
int start;
}arr[100];

void create()
{
    front = rear = -1;
}
```

```

void insert(int data)
{
    if ((front == -1) && (rear == -1))
    {
        front++;
        rear++;
        pri_que[rear] = data;
        return;
    }
    else
        check(data);
    rear++;
}

void check(int data)
{
    int i,j;

    for (i = 0; i <= rear; i++)
    {
        if (data >= pri_que[i])
        {
            for (j = rear + 1; j > i; j--)
            {
                pri_que[j] = pri_que[j - 1];
            }
            pri_que[i] = data;
            return;
        }
    }
    pri_que[i] = data;
}

void delete(int data)
{
    int i;
    for (i = 0; i <= rear; i++)
    {
        if (data == pri_que[i])
        {
            for (; i < rear; i++)
            {
                pri_que[i] = pri_que[i + 1];
            }

            pri_que[i] = -99;
            rear--;

            if (rear == -1)
                front = -1;
            return;
        }
    }
}

```

```

main(){
int i,n,left,right,height,j;
printf("enter no of building : ");
scanf("%d",&n);

int k=0;
for(i=0;i<n;i++){
    scanf("%d %d %d",&left,&right,&height);
    arr[k].x=left;
    arr[k].height=height;
    arr[k].start=1;

    k++;

    arr[k].x=right;
    arr[k].height=height;
    arr[k].start=0;

    k++;
}

int temp;

for(i=0;i<k;i++)
{
    for(j=i+1;j<k;j++){
        //sort based on x
        //dissimilar x
        if(arr[i].x!=arr[j].x){
            if(arr[i].x>arr[j].x){

                temp=arr[i].x;
                arr[i].x=arr[j].x;
                arr[j].x=temp;

                temp=arr[i].height;
                arr[i].height=arr[j].height;
                arr[j].height=temp;

                temp=arr[i].start;
                arr[i].start=arr[j].start;
                arr[j].start=temp;

            }
        }
    }
    else if(arr[i].x!=arr[j].x){
        if(arr[i].start==1&&arr[j].start==1){
            if(arr[i].height<arr[j].height)
            {
                temp=arr[i].x;
                arr[i].x=arr[j].x;
                arr[j].x=temp;
            }
        }
    }
}

```

```

        temp=arr[i].height;
        arr[i].height=arr[j].height;
        arr[j].height=temp;

        temp=arr[i].start;
        arr[i].start=arr[j].start;
        arr[j].start=temp;
    }
}
if(arr[i].start==0&&arr[j].start==0){
    if(arr[i].height>arr[j].height)
    {
        temp=arr[i].x;
        arr[i].x=arr[j].x;
        arr[j].x=temp;

        temp=arr[i].height;
        arr[i].height=arr[j].height;
        arr[j].height=temp;

        temp=arr[i].start;
        arr[i].start=arr[j].start;
        arr[j].start=temp;
    }
}
if(arr[i].start==0&&arr[j].start==0){
if(arr[i].height<arr[j].height)
{
    temp=arr[i].x;
    arr[i].x=arr[j].x;
    arr[j].x=temp;

    temp=arr[i].height;
    arr[i].height=arr[j].height;
    arr[j].height=temp;

    temp=arr[i].start;
    arr[i].start=arr[j].start;
    arr[j].start=temp;
}
}
}
}

int tempmax=0,max=0;

printf("\nThe co-ordinates of skyline are : \n");
for(i=0;i<k;i++)
{
    if(arr[i].start==1){

```

```

insert(arr[i].height);
max=pri_que[front];
if(tempmax!=max){
    printf("[%d , %d]\n",arr[i].x,max);
    tempmax=max;
}
}
if(arr[i].start==0){
    delete(arr[i].height);
    max=pri_que[front];
    if(tempmax!=max){
        printf("[%d , %d]\n",arr[i].x,max);
        tempmax=max;
    }
}
}
}

```

OUTPUT :-

```

enter no of building : 8
1 5 11
2 7 6
3 9 13
12 16 7
14 25 3
19 22 18
23 29 13
24 28 4

The co-ordinates of skyline are :
[1 , 11]
[3 , 13]
[9 , 0]
[12 , 7]
[16 , 3]
[19 , 18]
[22 , 3]
[23 , 13]
[29 , 0]

```

RESULT :-

Thus the skyline program is executed and the output is verified