VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

SAMRITH SANJOO.S (1BM21CS185)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
June-2023 to September-2023

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Analysis and Design of Algorithms**" carried out by **SAMRITH SANJOO.S(1BM21CS185),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to September-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms (22CS4PCADA)** work prescribed for the said degree.

Name of the Lab-In charge:	Dr. Jyothi S Nayak
Designation	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

Lab Program No.	Program Details	Page No.
1	Write program to do the following: a. Print all the nodes reachable from a given starting node in a digraph using BFS method. b. Check whether a given graph is connected or not using DFS method.	5
2	Write program to obtain the Topological ordering of vertices in a given digraph.	9
3	Implement Johnson Trotter algorithm to generate permutations.	11
4	Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.	15
5	Sort a given set of N integer elements using Quick Sort technique and compute its time taken.	21
6	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	28
7	Implement 0/1 Knapsack problem using dynamic programming.	30
8	Implement All Pair Shortest paths problem using Floyd's algorithm.	34
9	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's and Kruskal's algorithm.	36
10	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	42
11	Implement "N-Queens Problem" using Backtracking.	45

	Leetcode problems 1)Problem 1 2)Problem 2 3)Problem 3 4)Problem 4	50

Course Outcome

CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms and conduct practical experiments to solve problems.

1)

Write program to do the following:

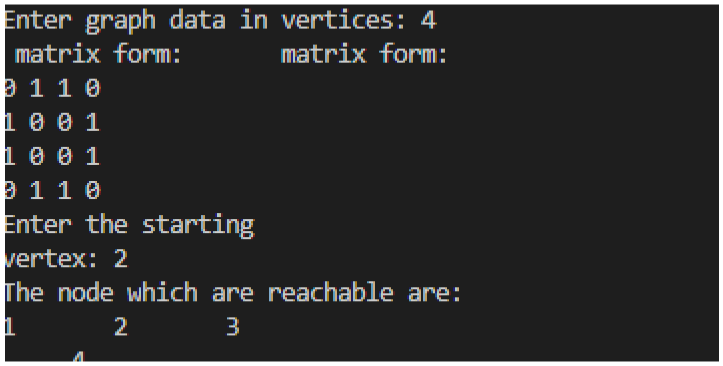
a. Print all the nodes reachable from a given starting node in a digraph using BFS method.

b. Check whether a given graph is connected or not using DFS method.

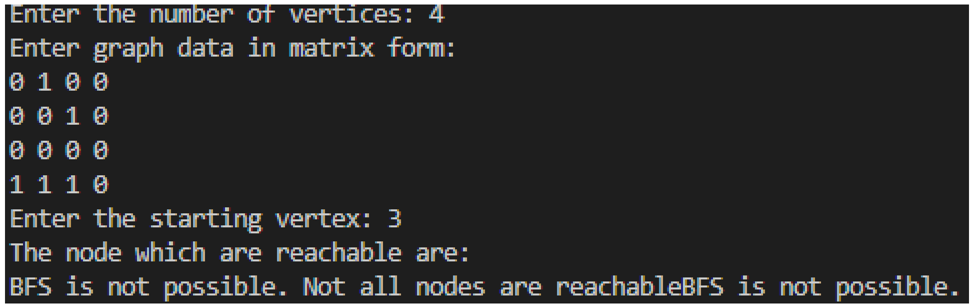
```
a)#include<stdio.h>
#include<conio.h>
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v)
{
for(i=1;i<=n;i++)
if(a[v][i] && !visited[i])
q[++r]=i;
if(f<=r)
{
visited[q[f]]=1;
bfs(q[f++]);
}
}
void main()
{
int v;

printf("\n Enter the number of vertices:");
scanf("%d",&n);
for(i=1;i<=n;i++)
```

```
{
q[i]=0;
visited[i]=0;
}
printf("\n Enter graph data in matrix form:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
printf("\n Enter the starting vertex:");
scanf("%d",&v);
bfs(v);
printf("\n The node which are reachable are:\n");
for(i=1;i<=n;i++)
if(visited[i])
printf("%d\t",i);
getch();
}
```

OUTPUT:



```
Enter graph data in vertices: 4
 matrix form:       matrix form:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
Enter the starting
vertex: 2
The node which are reachable are:
1       2       3
     4
```

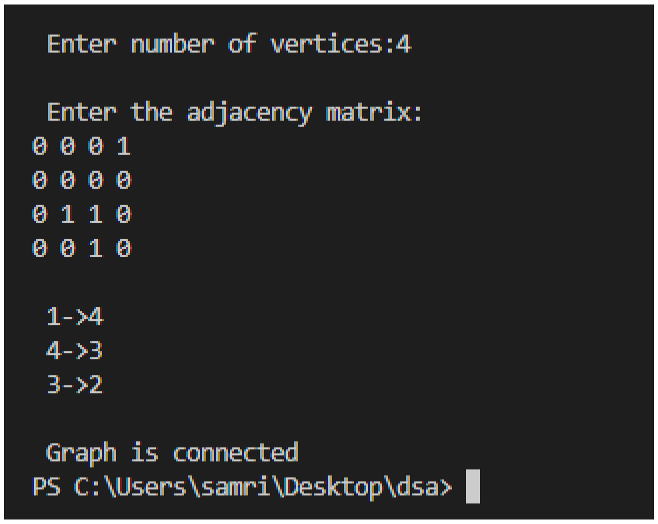


```
Enter the number of vertices: 4
Enter graph data in matrix form:
0 1 0 0
0 0 1 0
0 0 0 0
1 1 1 0
Enter the starting vertex: 3
The node which are reachable are:
BFS is not possible. Not all nodes are reachableBFS is not possible.
```

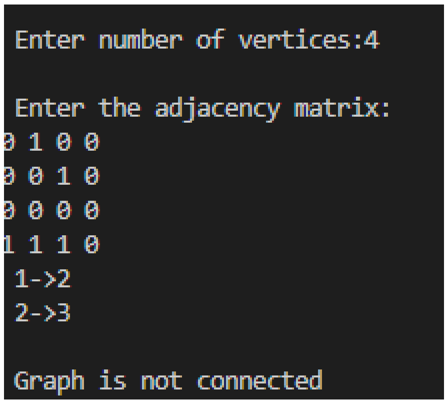
B)

```
#include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(int v) {
        int i;
        reach[v]=1;
        for (i=1;i<=n;i++)
         if(a[v][i] && !reach[i]) {
                printf("\n %d->%d",v,i);
                dfs(i);
        }
}
void main() {
        int i,j,count=0;
        clrscr();
        printf("\n Enter number of vertices:");
        scanf("%d",&n);
        for (i=1;i<=n;i++) {
                reach[i]=0;
                for (j=1;j<=n;j++)
                 a[i][j]=0;
        }
        printf("\n Enter the adjacency matrix:\n");
        for (i=1;i<=n;i++)
         for (j=1;j<=n;j++)
         scanf("%d",&a[i][j]);
        dfs(1);
        printf("\n");
        for (i=1;i<=n;i++) {
                if(reach[i])
                 count++;
        }
        if(count==n)
         printf("\n Graph is connected"); else
         printf("\n Graph is not connected");
        getch();
```

OUTPUT:



```
 Enter number of vertices:4

 Enter the adjacency matrix:
0 0 0 1
0 0 0 0
0 1 1 0
0 0 1 0

 1->4
 4->3
 3->2

 Graph is connected
PS C:\Users\samri\Desktop\dsa> 
```



```
Enter number of vertices:4

Enter the adjacency matrix:
0 1 0 0
0 0 1 0
0 0 0 0
1 1 1 0
1->2
2->3

Graph is not connected
```

2)

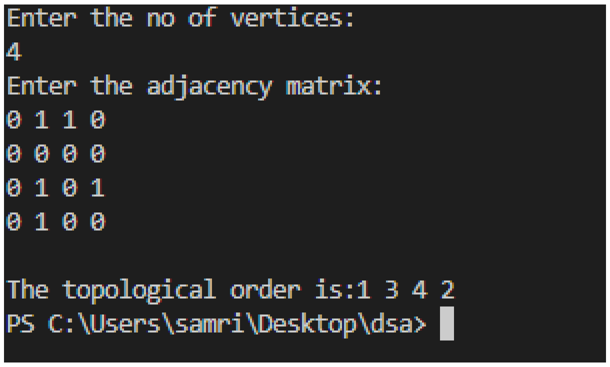
Write program to obtain the Topological ordering of vertices in a given digraph.

sol:

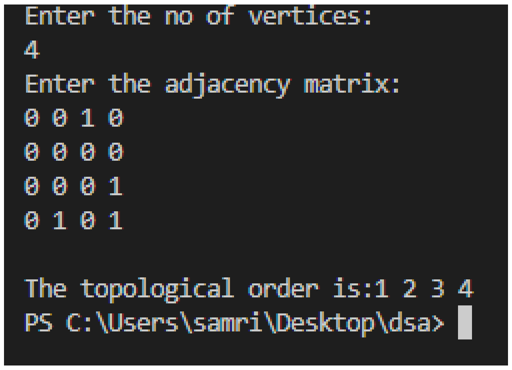
```
#include <stdio.h>

int main()
{
   int i, j, k, n, a[10][10], indeg[10], flag[10], count = 0;

   printf("Enter the no of vertices:\n");
   scanf("%d", &n);

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

   for (i = 0; i < n; i++)
   {
      indeg[i] = 0;
      flag[i] = 0;
   }

   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         indeg[i] = indeg[i] + a[j][i];
```

```
    printf("\nThe topological order is:");

    while (count < n)

    {
        for (k = 0; k < n; k++)
        {
            if ((indeg[k] == 0) && (flag[k] == 0))
            {
                printf("%d ", (k + 1));
                flag[k] = 1;
            }
             for (i = 0; i < n; i++)

            {
                if (a[i][k] == 1&& flag[i]==0)
                   a[k][i]=0;
                    indeg[k]--;
                    break;
            }



        }

        count++;
    }

    return 0;
}
```

OUTPUT:



```
Enter the no of vertices:
4
Enter the adjacency matrix:
0 1 1 0
0 0 0 0
0 1 0 1
0 1 0 0

The topological order is:1 3 4 2
PS C:\Users\samri\Desktop\dsa>
```



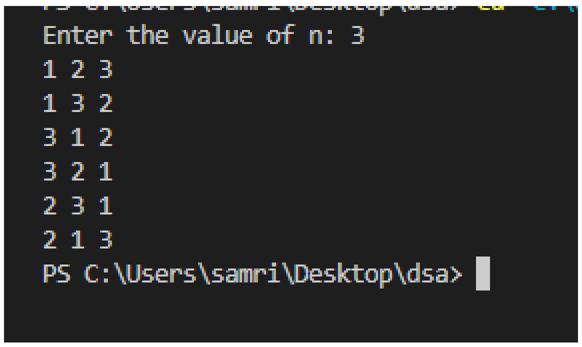
```
Enter the no of vertices:
4
Enter the adjacency matrix:
0 0 1 0
0 0 0 0
0 0 0 1
0 1 0 1

The topological order is:1 2 3 4
PS C:\Users\samri\Desktop\dsa>
```

3)
Implement Johnson Trotter algorithm to generate permutations.

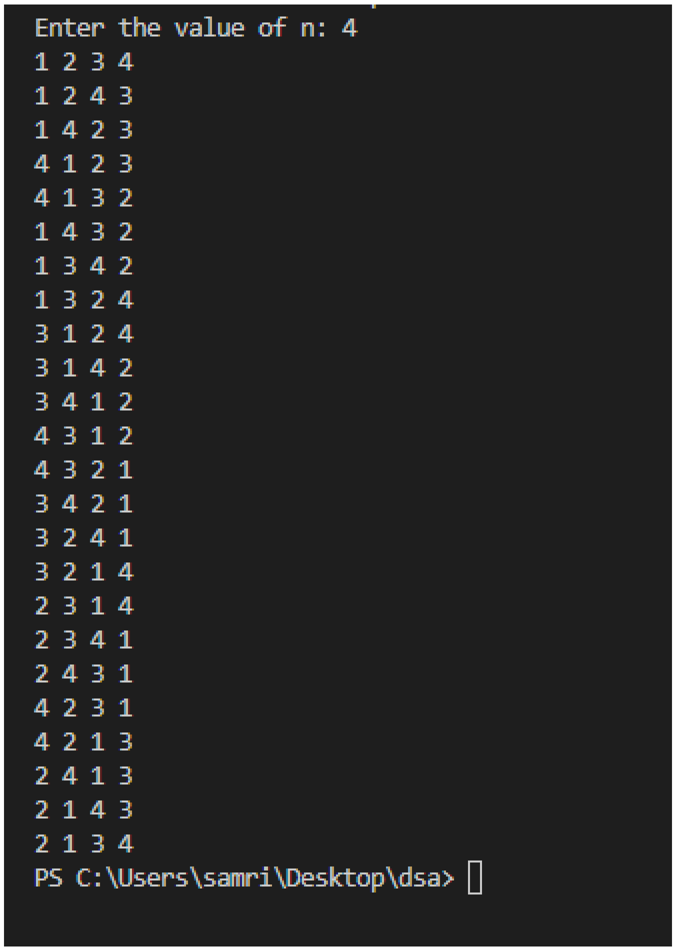
SOL:

```
#include <stdio.h>
#define RIGHT_TO_LEFT 0
#define LEFT_TO_RIGHT 1
void swap(int *a, int *b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}
int searchArr(int a[], int n, int mobile) {
   for (int i = 0; i < n; i++) {
       if (a[i] == mobile) {
           return i + 1;
       }
   }
   return -1;  // Mobile not found
}
int getMobile(int a[], int dir[], int n) {
   int mobile_prev = 0, mobile = 0;
   for (int i = 0; i < n; i++) {
       // Direction 0 represents RIGHT TO LEFT.
       if (dir[a[i] - 1] == RIGHT_TO_LEFT && i != 0) {
           if (a[i] > a[i - 1] && a[i] > mobile_prev) {
               mobile = a[i];
               mobile_prev = mobile;
           }
       }

       if (dir[a[i] - 1] == LEFT_TO_RIGHT && i != n - 1) {
           if (a[i] > a[i + 1] && a[i] > mobile_prev) {
               mobile = a[i];
               mobile_prev = mobile;
           }
       }
   }

   if (mobile == 0 && mobile_prev == 0) {
       return 0;  // No mobile element found
   } else {
       return mobile;
   }
}
void printOnePerm(int a[], int dir[], int n) {
   int mobile = getMobile(a, dir, n);
   int pos = searchArr(a, n, mobile);
   if (dir[a[pos - 1] - 1] == RIGHT_TO_LEFT) {
       swap(&a[pos - 1], &a[pos - 2]);
   } else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT) {
       swap(&a[pos], &a[pos - 1]);
   }

   for (int i = 0; i < n; i++) {
       if (a[i] > mobile) {
           if (dir[a[i] - 1] == LEFT_TO_RIGHT) {
               dir[a[i] - 1] = RIGHT_TO_LEFT;
           } else if (dir[a[i] - 1] == RIGHT_TO_LEFT) {
               dir[a[i] - 1] = LEFT_TO_RIGHT;
           }
       }
   }

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

int factorial(int n) {
   int res = 1;
   for (int i = 1; i <= n; i++) {
       res = res * i;
   }
   return res;
}
void printPermutation(int n) {
   int a[n];
   int dir[n];

   for (int i = 0; i < n; i++) {
       a[i] = i + 1;
       printf("%d ", a[i]);
   }
   printf("\n");

   for (int i = 0; i < n; i++) {
       dir[i] = RIGHT_TO_LEFT;
   }

   for (int i = 1; i < factorial(n); i++) {
       printOnePerm(a, dir, n);
   }
}
int main() {
   int n;
   printf("Enter the value of n: ");
   scanf("%d", &n);

   printPermutation(n);

   return 0;
}
```

OUTPUT:



```
Enter the value of n: 3
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3
PS C:\Users\samri\Desktop\dsa>
```



```
Enter the value of n: 4
1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
4 1 3 2
1 4 3 2
1 3 4 2
1 3 2 4
3 1 2 4
3 1 4 2
3 4 1 2
4 3 1 2
4 3 2 1
3 4 2 1
3 2 4 1
3 2 1 4
2 3 1 4
2 3 4 1
2 4 3 1
4 2 3 1
4 2 1 3
2 4 1 3
2 1 4 3
2 1 3 4
PS C:\Users\samri\Desktop\dsa>
```

4)

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

SOL:

```
#include<stdio.h>

void merge(int arr[],int low,int high){
   int k,s,i, mid ,leng1 ,leng2,  main array index, index1, index2;
   mid=(low+high)/2;
```

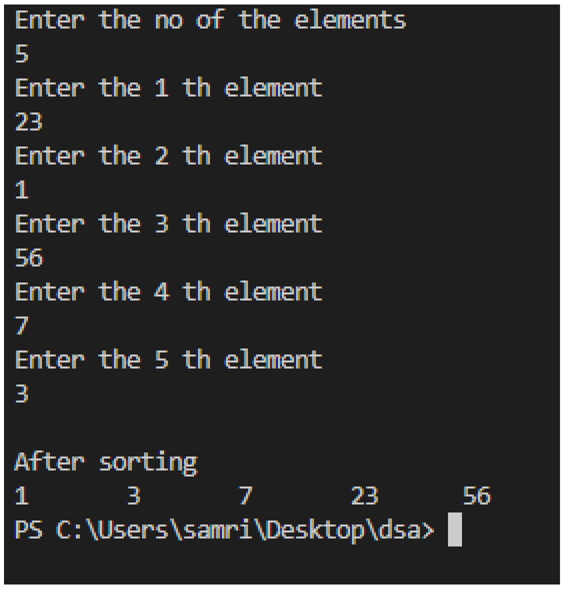
```
leng1=mid-low+1;
leng2=high-mid;

int  first[leng1];
int second[leng2];

 mainarrayindex=low;
for( i=0;i<leng1;i++){
   first[i]=arr[mainarrayindex++];
}

 mainarrayindex=mid+1;
 for(i=0;i<leng2;i++){
   second[i]=arr[mainarrayindex++];
}

//merge sorted arrays

 index1=0;
 index2=0;
 mainarrayindex=low;


while(index1<leng1 && index2<leng2){
   if(first[index1]< second[index2]){
      arr[mainarrayindex++]=first[index1++];
   }
   else{
       arr[mainarrayindex++]=second[index2++];
   }
```

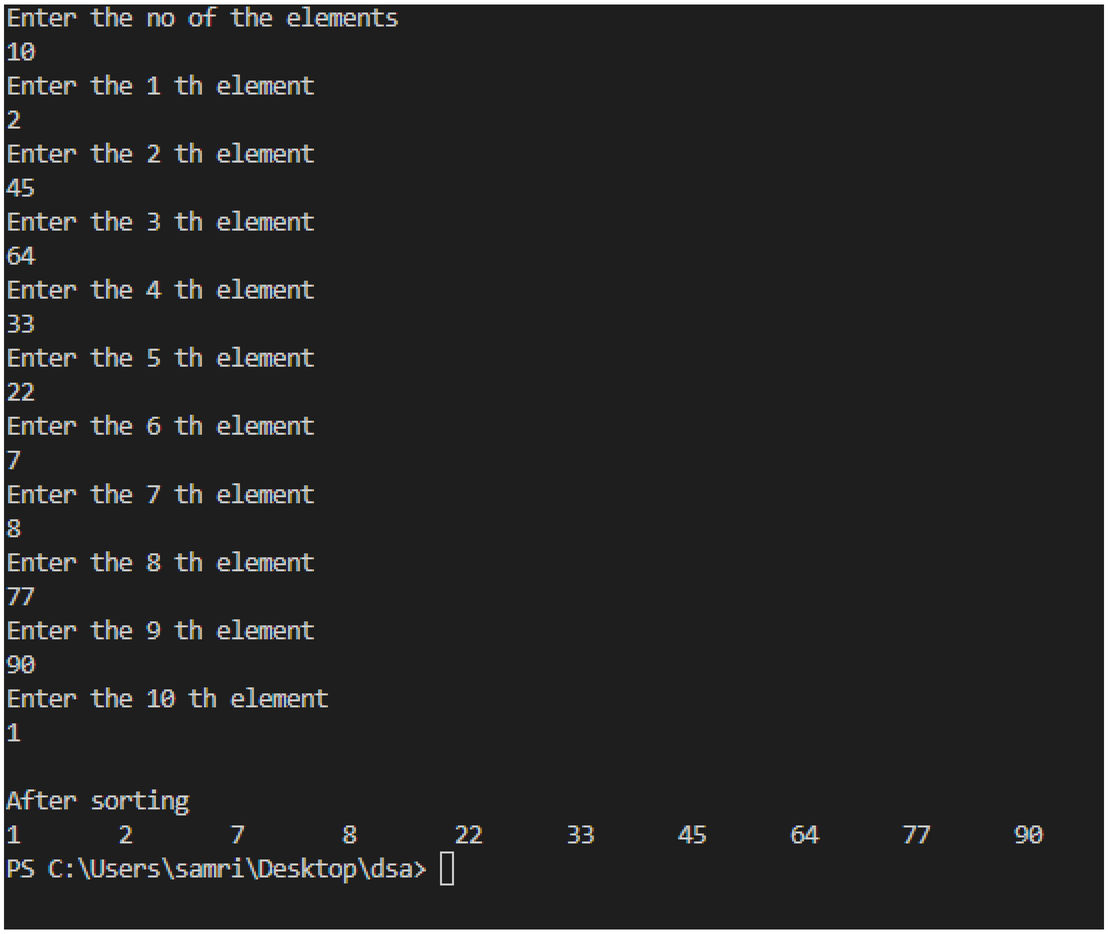
```
    }

   while(index1<leng1){
    arr[mainarrayindex++]=first[index1++];
   }

  while(index2<leng2){
    arr[mainarrayindex++]=second[index2++];
   }

}
```

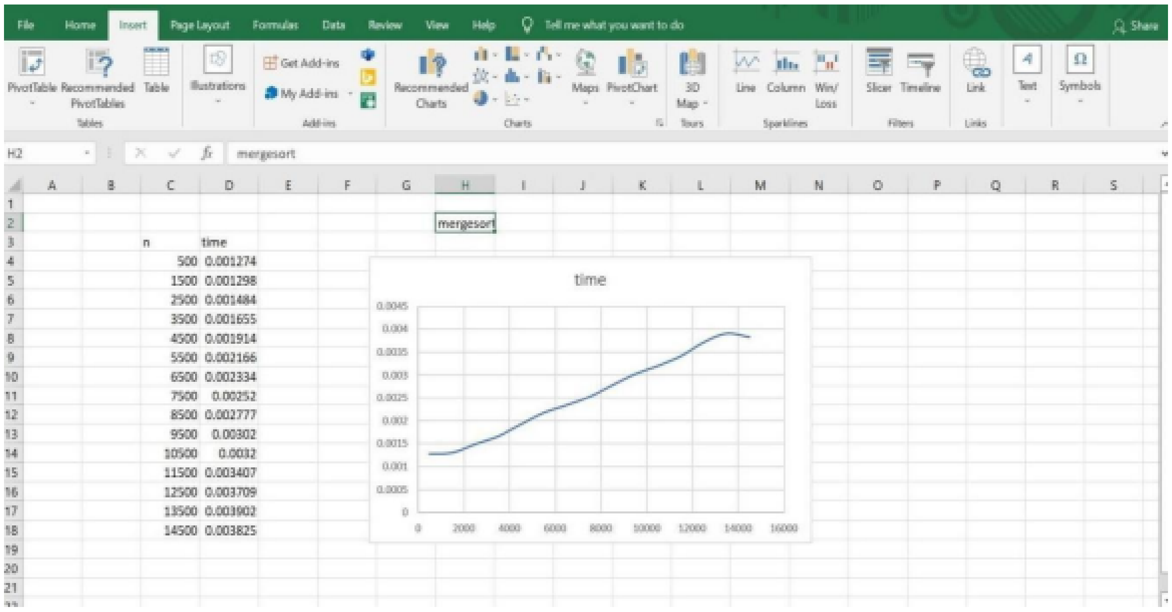
```
void mergeSort(int arr[],int low,int high){
     int mid;
    if(low>=high){
        return;
    }
    mid=(low+high)/2;
    mergeSort(arr,low,mid);
    mergeSort(arr,mid+1,high);

    merge(arr,low,high);

}
```

```
int main(){
   int n; int i=0;
   printf("Enter the no of the elements\n");
   scanf("%d",&n);
   int arr[n];
   for(i=0;i<n;i++){
      printf("Enter the %d th element\n",i+1);
      scanf("%d",&arr[i]);
   }


   mergeSort(arr,0,n-1);
 printf("\nAfter sorting\n");
 for(i=0;i<n;i++){
   printf("%d\t",arr[i]);
 }

}
```

OUTPUT:


```
Enter the no of the elements
5
Enter the 1 th element
23
Enter the 2 th element
1
Enter the 3 th element
56
Enter the 4 th element
7
Enter the 5 th element
3

After sorting
1       3       7       23      56
PS C:\Users\samri\Desktop\dsa>
```


```
Enter the no of the elements
10
Enter the 1 th element
2
Enter the 2 th element
45
Enter the 3 th element
64
Enter the 4 th element
33
Enter the 5 th element
22
Enter the 6 th element
7
Enter the 7 th element
8
Enter the 8 th element
77
Enter the 9 th element
90
Enter the 10 th element
1

After sorting
1       2       7       8       22      33      45      64      77      90
PS C:\Users\samri\Desktop\dsa>
```



n	time
500	0.001274
1500	0.001298
2500	0.001484
3500	0.001655
4500	0.001914
5500	0.002166
6500	0.002334
7500	0.00252
8500	0.002777
9500	0.00302
10500	0.0032
11500	0.003407
12500	0.003709
13500	0.003902
14500	0.003825

5)

Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

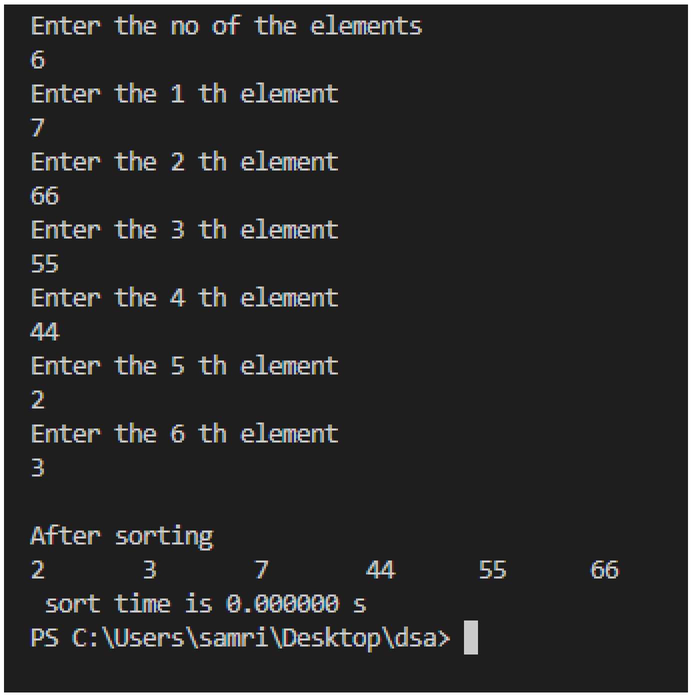
SOL:

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

void swap(int *xp, int *yp)
{
  int temp;
  temp = *xp;
  *xp = *yp;
  *yp = temp;
}


int partition(int arr[],int s,int e){
    int pivot;
    int k=0;
    pivot=arr[s];

    int cnt=0;
    for(k=s+1;k<=e;k++){
        if(pivot>=arr[k]){
            cnt++;
        }
    }

     int pivotIndex;
     pivotIndex=s+cnt;
     swap(&arr[pivotIndex],&arr[s]);
```

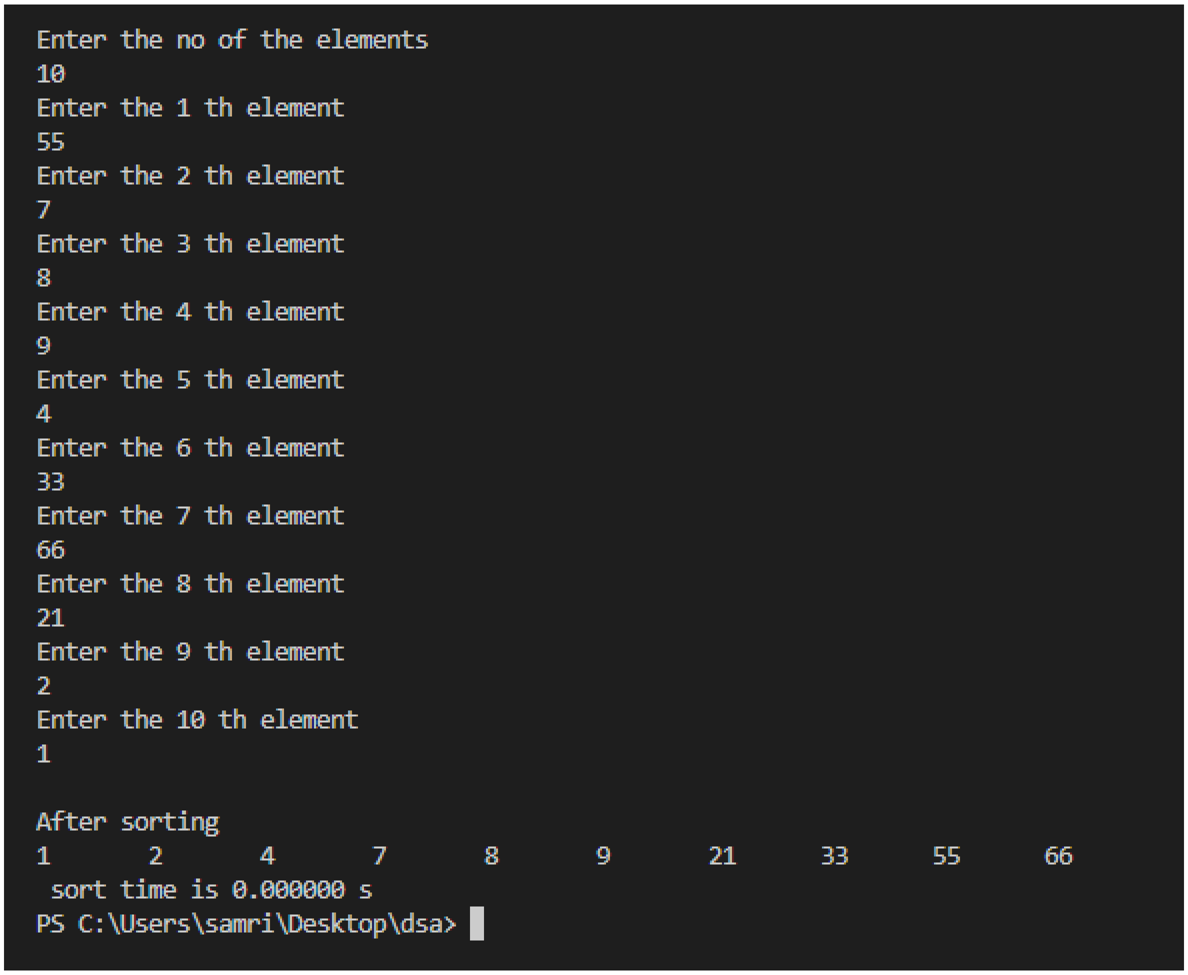
```
 int i=s;
 int j=e;

 while(i<pivotIndex && j>pivotIndex){
   while(arr[i]<pivot){
       i++;
   }
   while(arr[j]>pivot){
       j--;
   }

   if(i<pivotIndex && j>pivotIndex){
       swap(&arr[i++],&arr[j--]);
   }
 }
return pivotIndex;

}







void quicksort(int arr[],int s,int e){

//base case
if(s>=e){
   return;
}
//partition
int p;
p=partition(arr,s,e);
```

```
//recursive cases
quicksort(arr,s,p-1);
quicksort(arr,p+1,e);

}



int main(){
     int n; int i=0;
    printf("Enter the no of the elements\n");
    scanf("%d",&n);
    int arr[n];
    for(i=0;i<n;i++){
        printf("Enter the %d th element\n",i+1);
        scanf("%d",&arr[i]);
    }

  int start=clock();
 quicksort(arr,0,n-1);
   int end=clock();
  printf("\nAfter sorting\n");

 for(i=0;i<n;i++){
    printf("%d\t",arr[i]);
 }
double sort_time=(double)(end-start)/CLOCKS_PER_SEC;
 printf("\n sort time is %f s",sort_time);

}
```

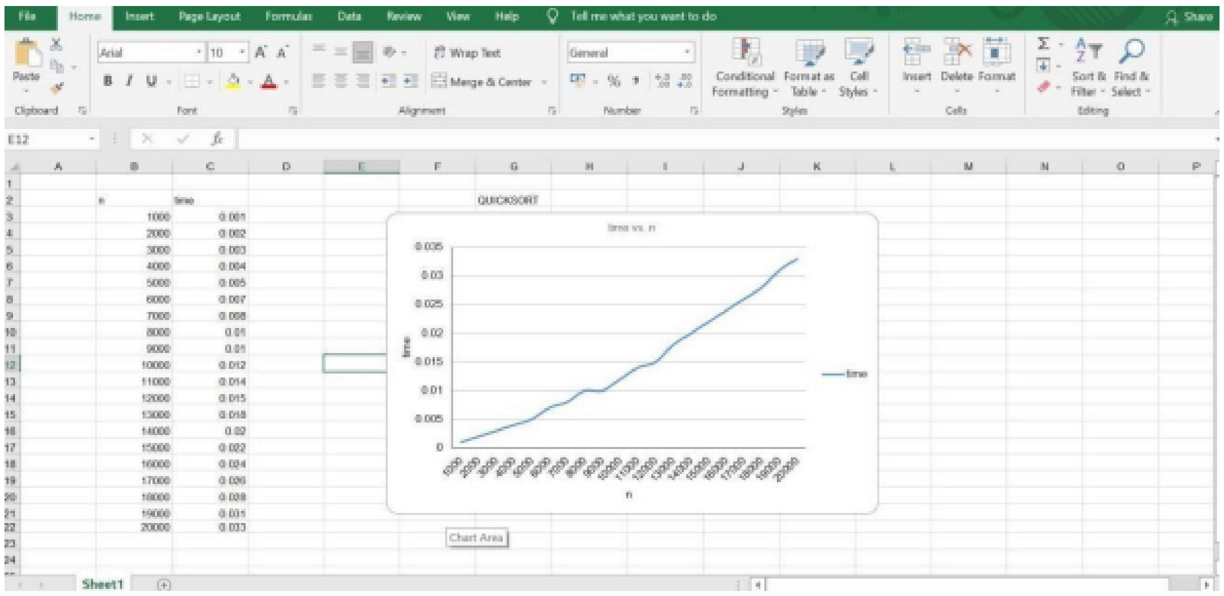
OUTPUT:



```
Enter the no of the elements
6
Enter the 1 th element
7
Enter the 2 th element
66
Enter the 3 th element
55
Enter the 4 th element
44
Enter the 5 th element
2
Enter the 6 th element
3

After sorting
2       3       7       44      55      66
 sort time is 0.000000 s
PS C:\Users\samri\Desktop\dsa>
```



```
Enter the no of the elements
10
Enter the 1 th element
55
Enter the 2 th element
7
Enter the 3 th element
8
Enter the 4 th element
9
Enter the 5 th element
4
Enter the 6 th element
33
Enter the 7 th element
66
Enter the 8 th element
21
Enter the 9 th element
2
Enter the 10 th element
1

After sorting
1       2       4       7       8       9       21      33      55      66
 sort time is 0.000000 s
PS C:\Users\samri\Desktop\dsa>
```



n	time
1000	0.001
2000	0.002
3000	0.003
4000	0.004
5000	0.005
6000	0.007
7000	0.008
8000	0.01
9000	0.01
10000	0.012
11000	0.014
12000	0.015
13000	0.018
14000	0.02
15000	0.022
16000	0.024
17000	0.026
18000	0.028
19000	0.031
20000	0.033

6)

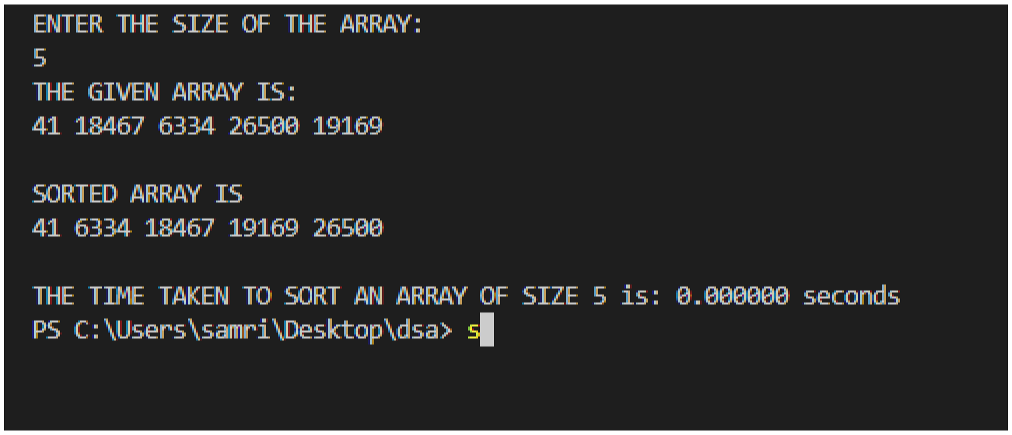
Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

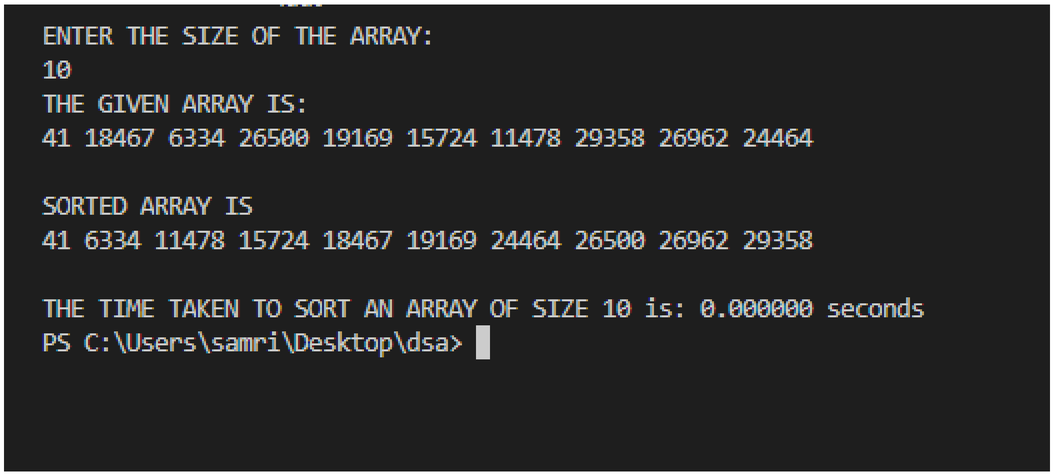
sol:

```
#include <stdio.h>
#include <time.h>
#define SIZE 20

void swap(int* a, int* b)
{

int temp = *a;
*a = *b;
*b = temp;
}
```

```
void heapify(int arr[], int N, int i)
{
int largest = i;
int left = 2 * i + 1;
int right = 2 * i + 2;
if (left < N && arr[left] > arr[largest])

largest = left;
if (right < N && arr[right] > arr[largest])

largest = right;
if (largest != i) {

swap(&arr[i], &arr[largest]);
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--)
   {
swap(&arr[0], &arr[i]);
heapify(arr, i, 0);
}
}


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


int main()
{
int arr[SIZE];
int n;
clock_t start,end;
double time_taken;
   printf("ENTER THE SIZE OF THE ARRAY: \n");
   scanf("%d",&n);
   for(int i=0;i<n;i++)
   {
      arr[i]=rand();
   }
   printf("THE GIVEN ARRAY IS:\n");
   printArray(arr,n);
   start=clock();
heapSort(arr, n);
end=clock();
printf("SORTED ARRAY IS\n");
printArray(arr, n);
time_taken=(double)(end-start);
time_taken=(time_taken/CLOCKS_PER_SEC);
printf("THE TIME TAKEN TO SORT AN ARRAY OF SIZE %d is: %f seconds",n,time_taken);
}
```

OUTPUT:


```
ENTER THE SIZE OF THE ARRAY:
5
THE GIVEN ARRAY IS:
41 18467 6334 26500 19169

SORTED ARRAY IS
41 6334 18467 19169 26500

THE TIME TAKEN TO SORT AN ARRAY OF SIZE 5 is: 0.000000 seconds
PS C:\Users\samri\Desktop\dsa> s
```


```
ENTER THE SIZE OF THE ARRAY:
10
THE GIVEN ARRAY IS:
41 18467 6334 26500 19169 15724 11478 29358 26962 24464

SORTED ARRAY IS
41 6334 11478 15724 18467 19169 24464 26500 26962 29358

THE TIME TAKEN TO SORT AN ARRAY OF SIZE 10 is: 0.000000 seconds
PS C:\Users\samri\Desktop\dsa>
```

7)

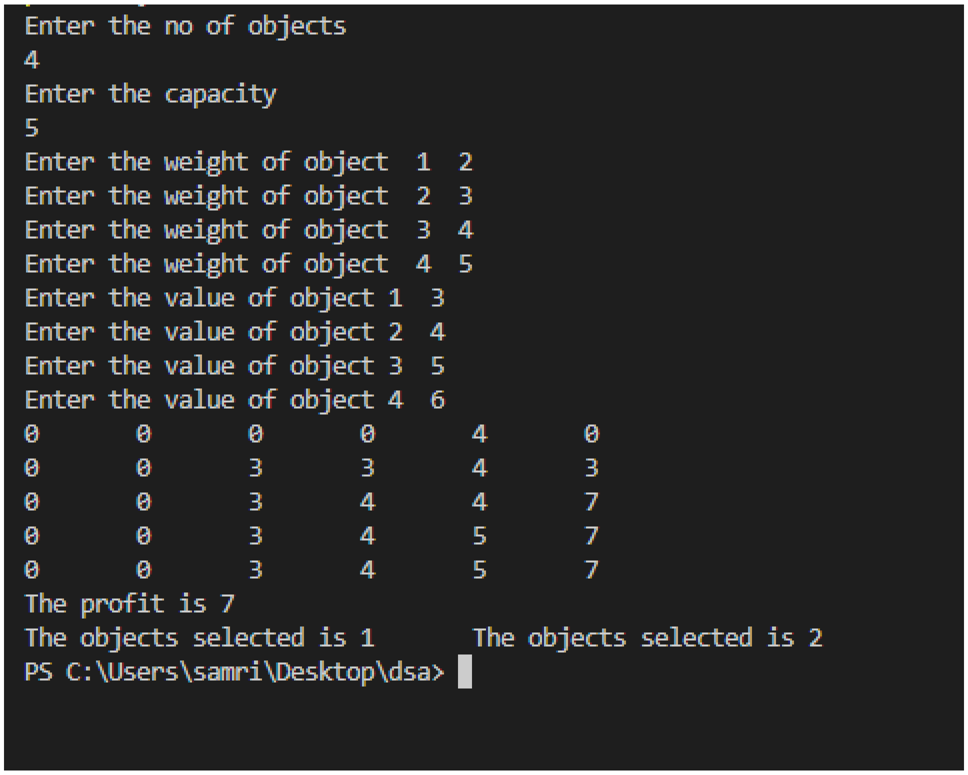
Implement 0/1 Knapsack problem using dynamic programming.

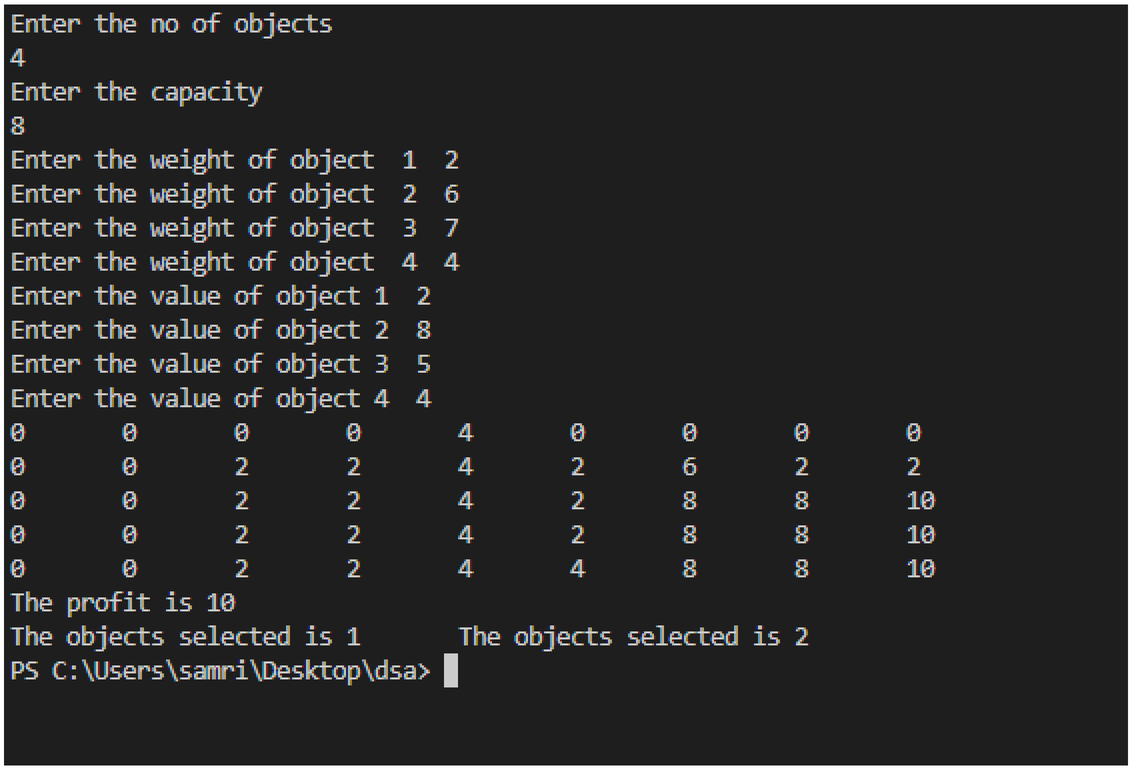
sol:

```
#include <stdio.h>
#include <conio.h>
int n, m, w[10], p[10], v[10][10];
int max(int a, int b)
{
   if (a > b)
      return a;
   else
      return b;
}

int knapsack()
{
   printf("The table\n");
   for (int i = 0; i <= n; i++)
   {
      for (int j = 0; j <= m; j++)
      {
         if (i == 0 && j == 0)
         {
            v[i][j] = 0;
         }
         else if (w[i - 1] > j)
         {
            v[i][j] = v[i - 1][j];
         }
         else
         {
            v[i][j] = max(v[i - 1][j], v[i - 1][j - w[i - 1]] + p[i - 1]);
         }
```

```
            printf("%d ",v[i][j]);

        }
        printf("\n");
    }
    return v[n][m];
}
void object_selected()
{
    int i = n, j = m;
    int x[10];
    for (int k = 1; k <= n; k++)
    {
        x[k] = 0;
    }
    while (i != 0 && j != 0)
    {
        if (v[i][j] != v[i - 1][j])
        {
            x[i] = 1;
            j = j - w[i - 1];
        }
        i = i - 1;
    }

    for (i = 1; i <= n; i++)
    {
        if (x[i] == 1)
        {
            printf("Object %d selected \n", i);
        }
    }
}
void main()
{
    printf("Enter the number of objects\n");
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        printf("Enter the weight of object  %d  ", i + 1);
        scanf("%d", &w[i]);
    }
    for (int i = 0; i < n; i++)
    {
        printf("Enter the value of object %d  ", i + 1);
        scanf("%d", &p[i]);
    }
    printf("Enter the capacity\n");
    scanf("%d", &m);
    int profit = knapsack();
    object_selected();

    printf("Maximum sum %d\n", profit);
}
```

OUTPUT :


```
Enter the no of objects
4
Enter the capacity
5
Enter the weight of object  1  2
Enter the weight of object  2  3
Enter the weight of object  3  4
Enter the weight of object  4  5
Enter the value of object 1  3
Enter the value of object 2  4
Enter the value of object 3  5
Enter the value of object 4  6
0       0       0       0       4       0
0       0       3       3       4       3
0       0       3       4       4       7
0       0       3       4       5       7
0       0       3       4       5       7
The profit is 7
The objects selected is 1       The objects selected is 2
PS C:\Users\samri\Desktop\dsa>
```


```
Enter the no of objects
4
Enter the capacity
8
Enter the weight of object  1  2
Enter the weight of object  2  6
Enter the weight of object  3  7
Enter the weight of object  4  4
Enter the value of object 1  2
Enter the value of object 2  8
Enter the value of object 3  5
Enter the value of object 4  4
0       0       0       0       4       0       0       0       0
0       0       2       2       4       2       6       2       2
0       0       2       2       4       2       8       8       10
0       0       2       2       4       2       8       8       10
0       0       2       2       4       4       8       8       10
The profit is 10
The objects selected is 1       The objects selected is 2
PS C:\Users\samri\Desktop\dsa>
```

8)

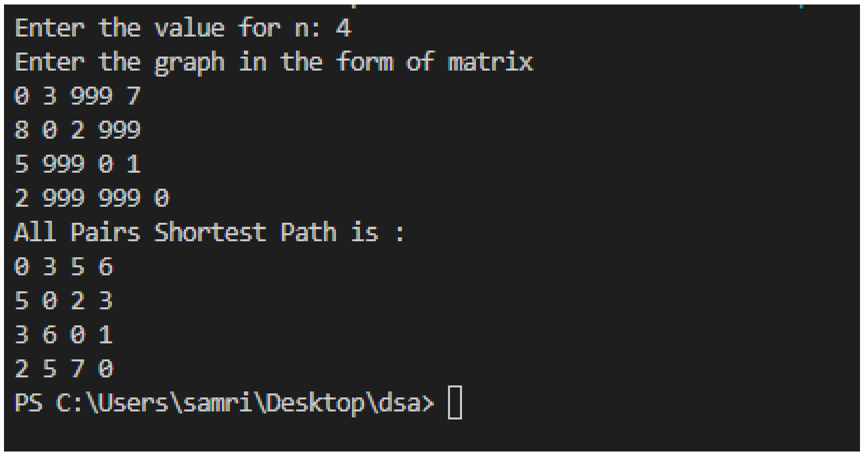
Implement All Pair Shortest paths problem using Floyd's algorithm

sol:

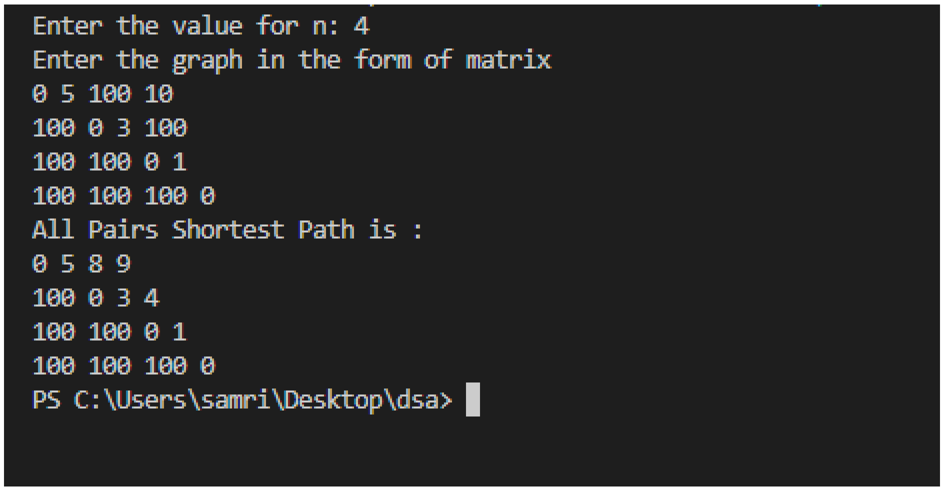
```
#include<stdio.h>


void floyd(int a[4][4], int n)
{
        for(int k=0;k<n;k++)
        {
                for(int i=0;i<n;i++)
                {
                        for(int j=0;j<n;j++)
                        {
                                if(a[i][j]>a[i][k]+a[k][j])
                                {
                                        a[i][j]=a[i][k]+a[k][j];
                                }
                          }
                  }
        }
        printf("All Pairs Shortest Path is :\n");
                for(int i=0;i<n;i++)
           {
                for(int j=0;j<n;j++)
                {
                        printf("%d ",a[i][j]);
                        }
                printf("\n");
                }
}
int main()
```

```
{
   int n ;
   printf("Enter the value for n: ");
   scanf("%d",&n);
   int mat[n][n];

   printf("Enter the graph in the form of matrix ");
   printf("\n");
   for(int i=0;i<n;i++)
                {
                         for(int j=0;j<n;j++)
                         {
                            scanf("%d",&mat[i][j]);
                         }
                }



         floyd(mat,n);
}
```

Output:



```
Enter the value for n: 4
Enter the graph in the form of matrix
0 3 999 7
8 0 2 999
5 999 0 1
2 999 999 0
All Pairs Shortest Path is :
0 3 5 6
5 0 2 3
3 6 0 1
2 5 7 0
PS C:\Users\samri\Desktop\dsa> 
```



```
Enter the value for n: 4
Enter the graph in the form of matrix
0 5 100 10
100 0 3 100
100 100 0 1
100 100 100 0
All Pairs Shortest Path is :
0 5 8 9
100 0 3 4
100 100 0 1
100 100 100 0
PS C:\Users\samri\Desktop\dsa>
```

9)

Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s and Kruskal’s algorithm.

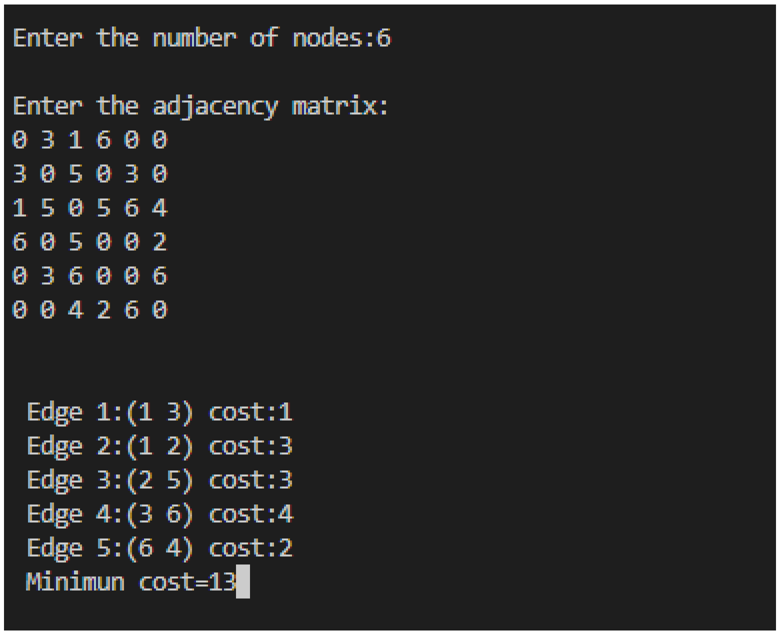
a) prims

```
#include<stdio.h>

#include<conio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

void main()

{

        clrscr();

        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 Edge %d:(%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);
        getch();
}
```

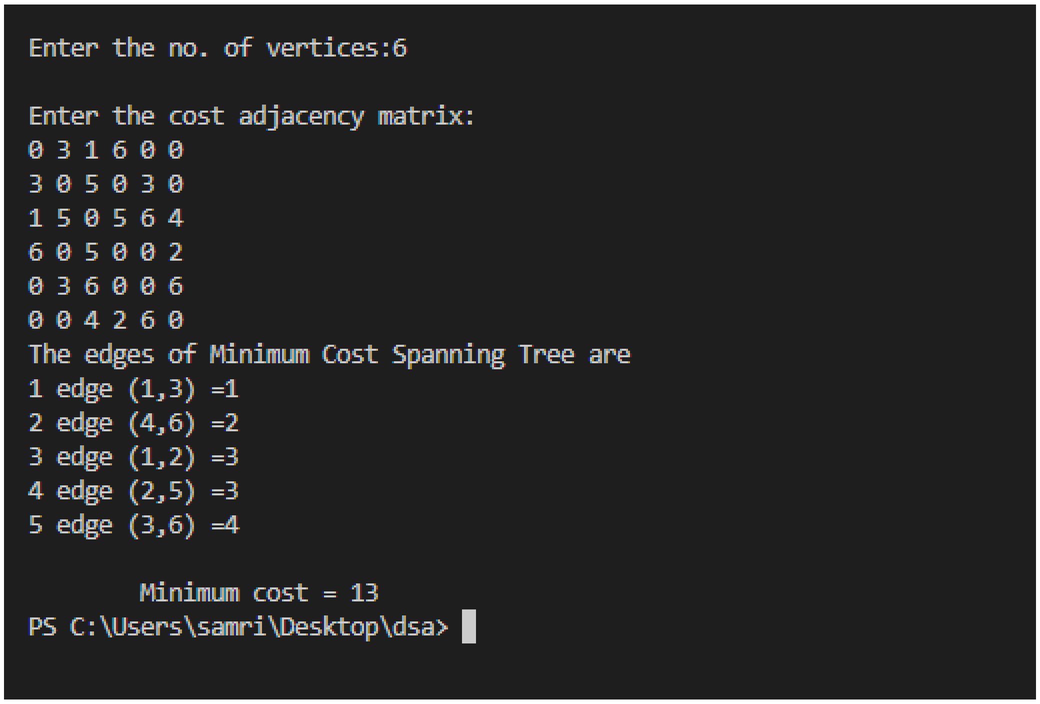
output:


```
Enter the number of nodes:6

Enter the 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


 Edge 1:(1 3) cost:1
 Edge 2:(1 2) cost:3
 Edge 3:(2 5) cost:3
 Edge 4:(3 6) cost:4
 Edge 5:(6 4) cost:2
 Minimun cost=13
```

b)krushkal

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.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()
{
        clrscr();
        printf("\n\tImplementation of Kruskal's algorithm\n");
        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("%d edge (%d,%d) =%d\n",ne++,a,b,min);
                mincost +=min;
            }
            cost[a][b]=cost[b][a]=999;
        }
        printf("\n\tMinimum cost = %d\n",mincost);
        getch();
}
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 edge (1,3) =1
2 edge (4,6) =2
3 edge (1,2) =3
4 edge (2,5) =3
5 edge (3,6) =4

        Minimum cost = 13
PS C:\Users\samri\Desktop\dsa>
```

10)

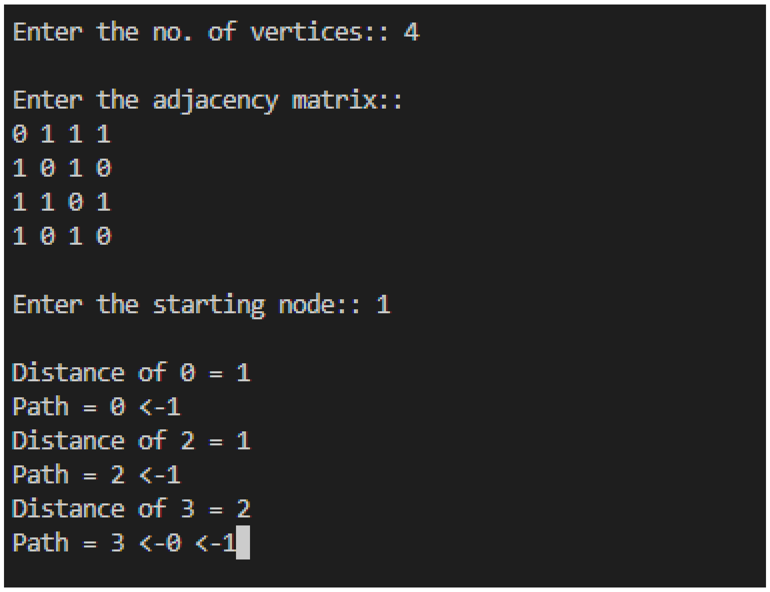
From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm

Sol:

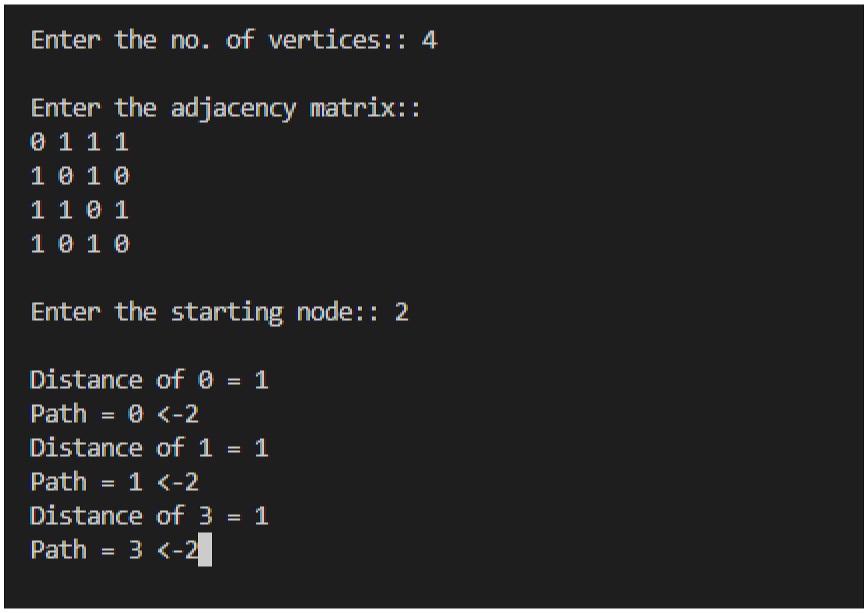
```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10

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

void main(){
        int G[MAX][MAX], i, j, n, u;
        clrscr();
        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);
        dijikstra(G,n,u);
        getch();
}

void dijikstra(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){
                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 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:: 4

Enter the adjacency matrix::
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0

Enter the starting node:: 1

Distance of 0 = 1
Path = 0 <-1
Distance of 2 = 1
Path = 2 <-1
Distance of 3 = 2
Path = 3 <-0 <-1
```



```
Enter the no. of vertices:: 4

Enter the adjacency matrix::
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0

Enter the starting node:: 2

Distance of 0 = 1
Path = 0 <-2
Distance of 1 = 1
Path = 1 <-2
Distance of 3 = 1
Path = 3 <-2
```

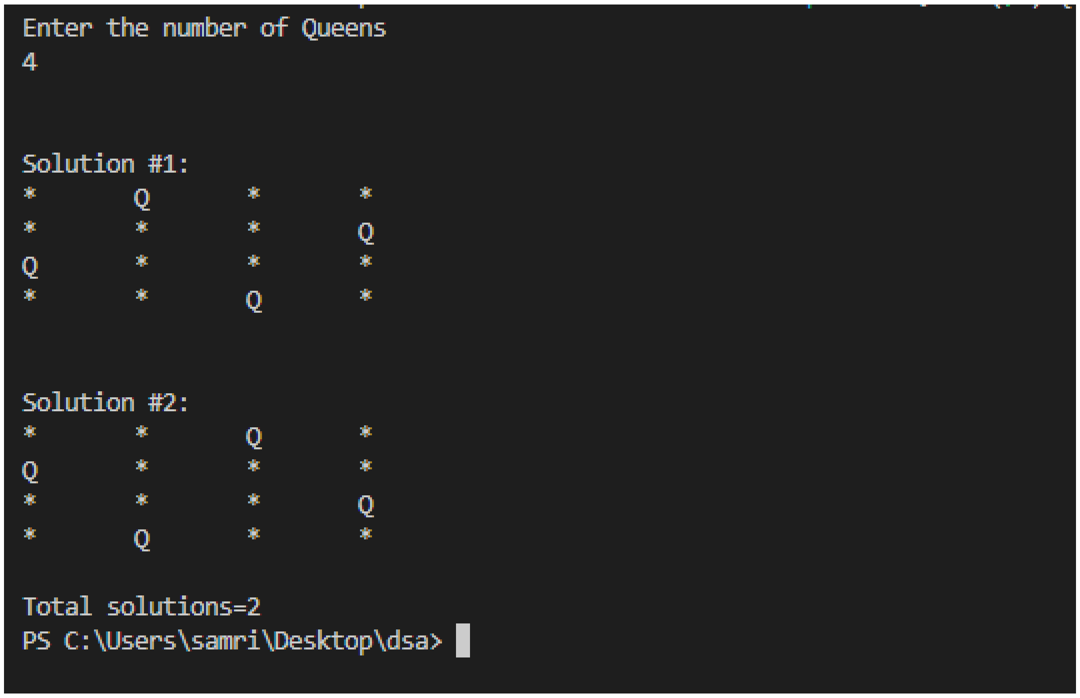
11)

Implement “N-Queens Problem” using Backtracking.

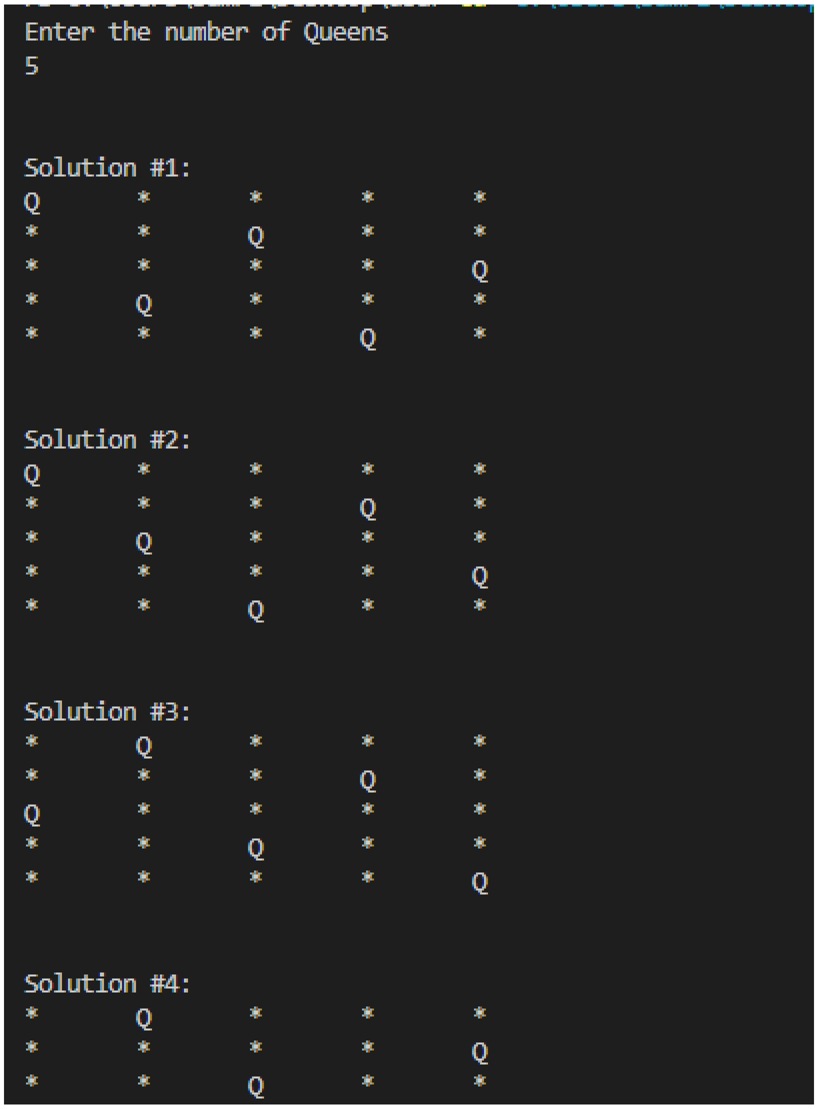
```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
int a[30],count=0;
int place(int pos) {
        int i;
        for (i=1;i<pos;i++) {
                if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
                  return 0;
        }
        return 1;
```

```
}
void print_sol(int n) {
        int i,j;
        count++;
        printf("\n\nSolution #%d:\n",count);
        for (i=1;i<=n;i++) {
                for (j=1;j<=n;j++) {
                        if(a[i]==j)
                           printf("Q\t"); else
                           printf("*\t");
                }
                printf("\n");
        }
}
void queen(int n) {
        int k=1;
        a[k]=0;
        while(k!=0) {
                a[k]=a[k]+1;
                while((a[k]<=n)&&!place(k))
                  a[k]++;
                if(a[k]<=n) {
                        if(k==n)
                           print_sol(n); else {
                                k++;
                                a[k]=0;
                        }
                } else
                  k--;
        }
}
void main() {
        int i,n;

        printf("Enter the number of Queens\n");
        scanf("%d",&n);
        queen(n);
        printf("\nTotal solutions=%d",count);
        getch();
}
```

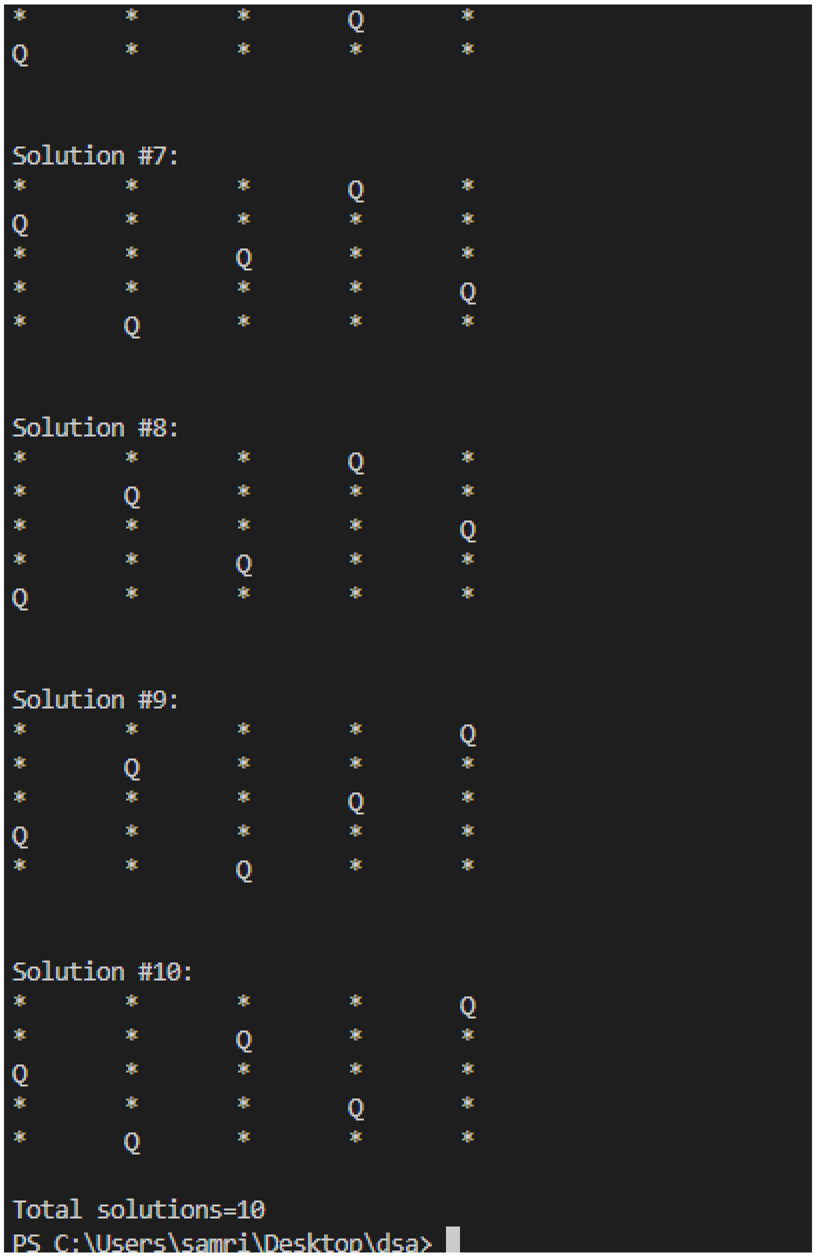
Output:



```
Enter the number of Queens
4


Solution #1:
*       Q       *       *
*       *       *       Q
Q       *       *       *
*       *       Q       *


Solution #2:
*       *       Q       *
Q       *       *       *
*       *       *       Q
*       Q       *       *

Total solutions=2
PS C:\Users\samri\Desktop\dsa>
```



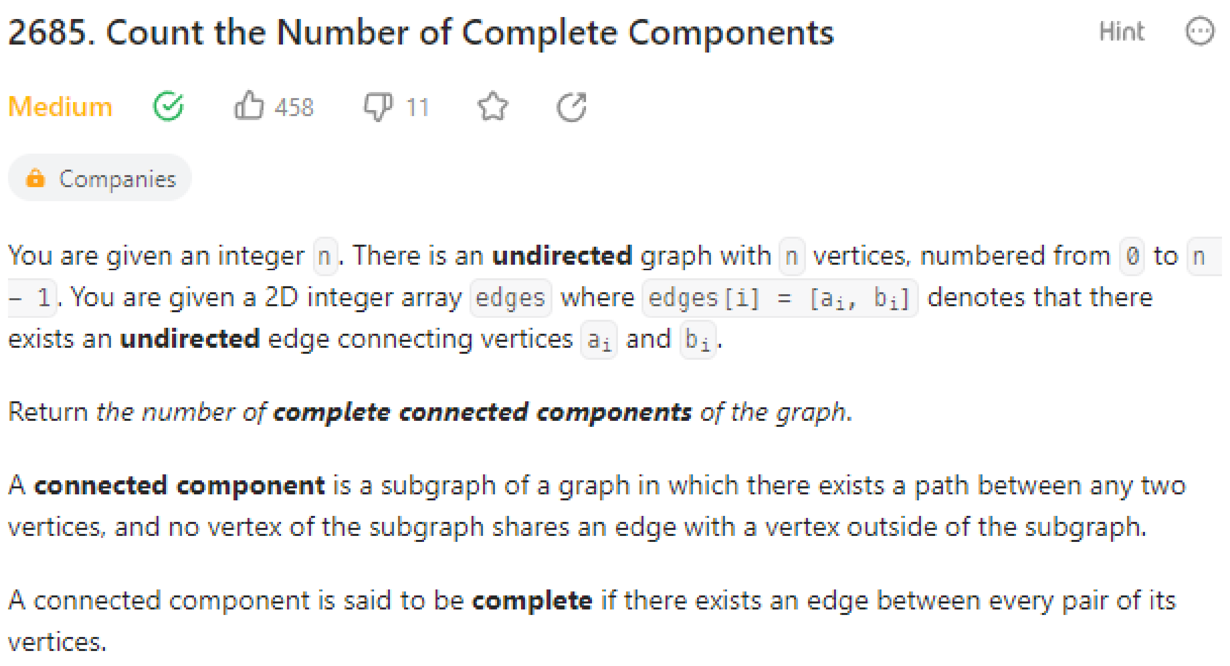
```
Enter the number of Queens
5


Solution #1:
Q       *       *       *       *
*       *       Q       *       *
*       *       *       *       Q
*       Q       *       *       *
*       *       *       Q       *


Solution #2:
Q       *       *       *       *
*       *       *       Q       *
*       Q       *       *       *
*       *       *       *       Q
*       *       Q       *       *


Solution #3:
*       Q       *       *       *
*       *       *       Q       *
Q       *       *       *       *
*       *       Q       *       *
*       *       *       *       Q


Solution #4:
*       Q       *       *       *
*       *       *       *       Q
*       *       Q       *       *
```



```
*       *       *       Q       *
Q       *       *       *       *


Solution #7:
*       *       *       Q       *
Q       *       *       *       *
*       *       Q       *       *
*       *       *       *       Q
*       Q       *       *       *


Solution #8:
*       *       *       Q       *
*       Q       *       *       *
*       *       *       *       Q
*       *       Q       *       *
Q       *       *       *       *


Solution #9:
*       *       *       *       Q
*       Q       *       *       *
*       *       *       Q       *
Q       *       *       *       *
*       *       Q       *       *


Solution #10:
*       *       *       *       Q
*       *       Q       *       *
Q       *       *       *       *
*       *       *       Q       *
*       Q       *       *       *

Total solutions=10
PS C:\Users\samri\Desktop\dsa>
```

Leetcode Problems

A)Problem 1



2685. Count the Number of Complete Components

You are given an integer `n`. There is an **undirected** graph with `n` vertices, numbered from `0` to `n - 1`. You are given a 2D integer array `edges` where `edges[i] = [a_i, b_i]` denotes that there exists an **undirected** edge connecting vertices `a_i` and `b_i`.

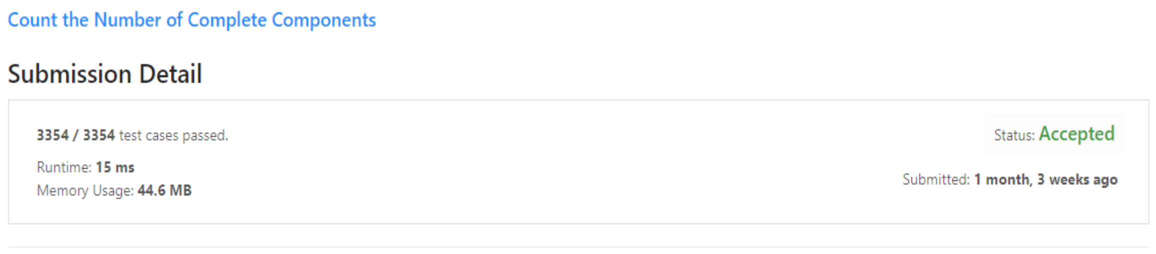
Return *the number of* ***complete connected components*** *of the graph.*

A **connected component** is a subgraph of a graph in which there exists a path between any two vertices, and no vertex of the subgraph shares an edge with a vertex outside of the subgraph.

A connected component is said to be **complete** if there exists an edge between every pair of its vertices.

sol:

```
class Solution {

  List<List<Integer>> adj;
  boolean visit[];
  int vcnt,ecnt;
  void dfs(int n){
    visit[n] = true;
    ++vcnt;
    for(var c:adj.get(n)){
      ecnt++;
      if(!visit[c]){
        dfs(c);
      }
    }
    }

    public int countCompleteComponents(int n, int[][] edges) {
            adj = new ArrayList<>();
        visit = new boolean[n];
        for(int i = 0;i < n;i++) adj.add(new ArrayList<>());
        for(var e:edges){
            adj.get(e[0]).add(e[1]);
            adj.get(e[1]).add(e[0]);
        }
        int ans = 0;
        for(int i = 0;i < n;i++){
            if(!visit[i]){
                vcnt = 0;
                ecnt = 0;
                dfs(i);
                if((vcnt * (vcnt - 1)) == ecnt) ++ans;
            }
        }
        return ans;


    }
}
```


Count the Number of Complete Components

Submission Detail

3354 / 3354 test cases passed.
Runtime: 15 ms
Memory Usage: 44.6 MB

Status: Accepted
Submitted: 1 month, 3 weeks ago

B) Problem 2

64. Minimum Path Sum

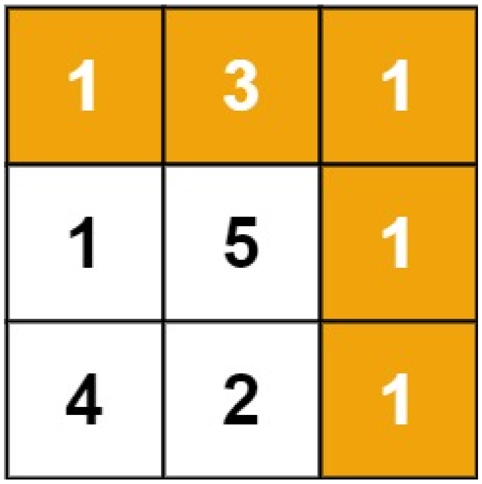
Medium 11.5K 149

Companies

Given a `m x n` `grid` filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

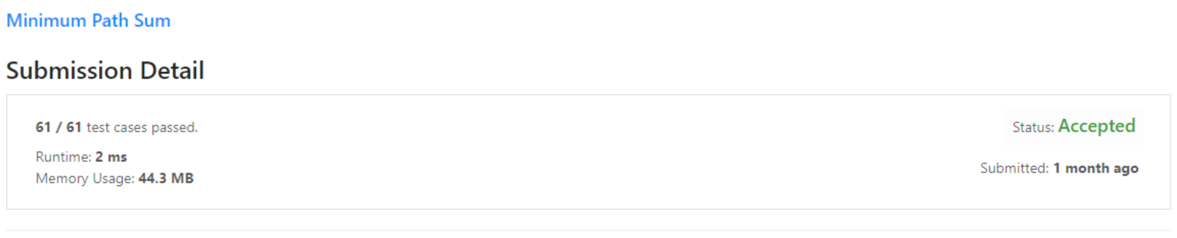
Example 1:



sol:

```
class Solution {



    public int minPathSum(int[][] grid) {



int min=-1;
for(int i=0;i<grid.length;i++){
```

```
for(int j=0;j<grid[0].length;j++){
if(i==0&&j!=0){
grid[i][j] +=grid[0][j-1];
}
else if(i!=0&&j==0){
grid[i][j] +=grid[i-1][0];
}else{
if(i!=0){
min=Math.min(grid[i-1][j],grid[i][j-1]);
grid[i][j] +=min;}
}
}
}
 return grid[grid.length-1][grid[0].length-1];

 }


}
```



Minimum Path Sum

Submission Detail

61 / 61 test cases passed.

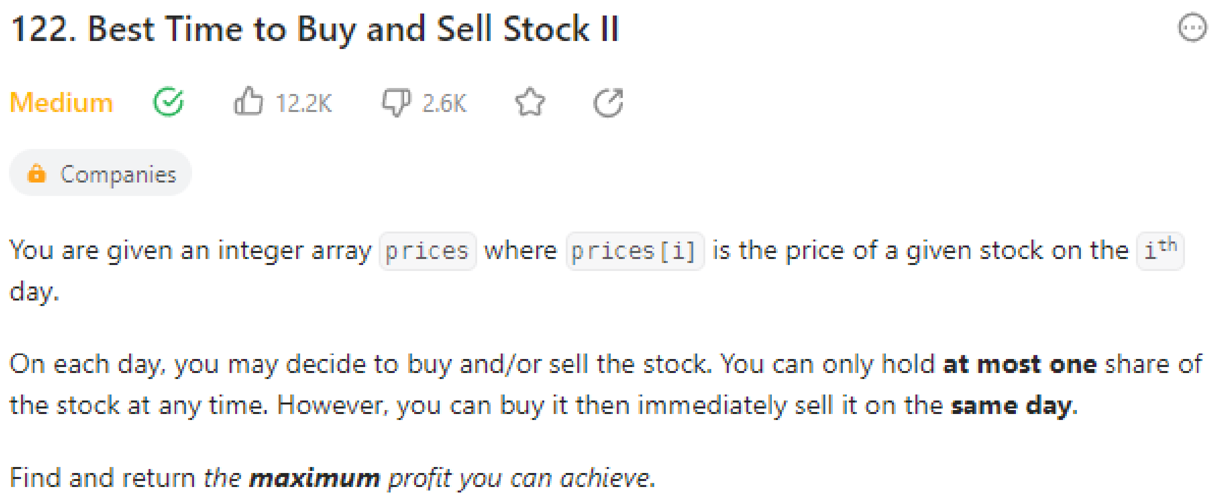
Runtime: 2 ms

Memory Usage: 44.3 MB

Status: Accepted

Submitted: 1 month ago

c) Problem 3



122. Best Time to Buy and Sell Stock II

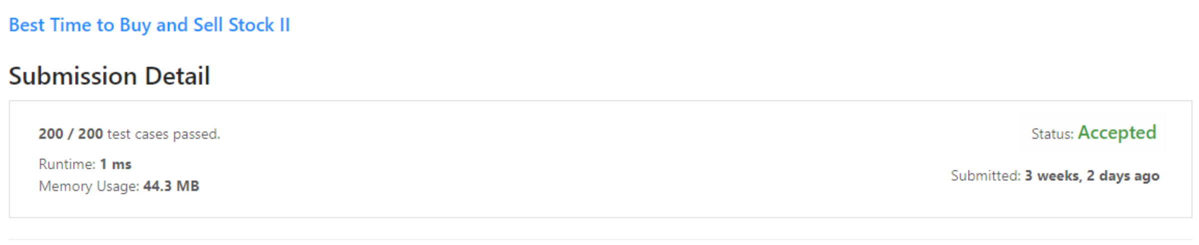
You are given an integer array `prices` where `prices[i]` is the price of a given stock on the `i`th day.

On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.

Find and return *the **maximum** profit you can achieve.*

sol:

```
class Solution {
   public int maxProfit(int[] prices) {

      int profit = 0;
      for(int i=0;i<prices.length-1;i++){
         if(prices[i+1]>prices[i]){

            profit=profit + prices[i+1]- prices[i];
         }

      }
      return profit;
   }
}
```



Best Time to Buy and Sell Stock II

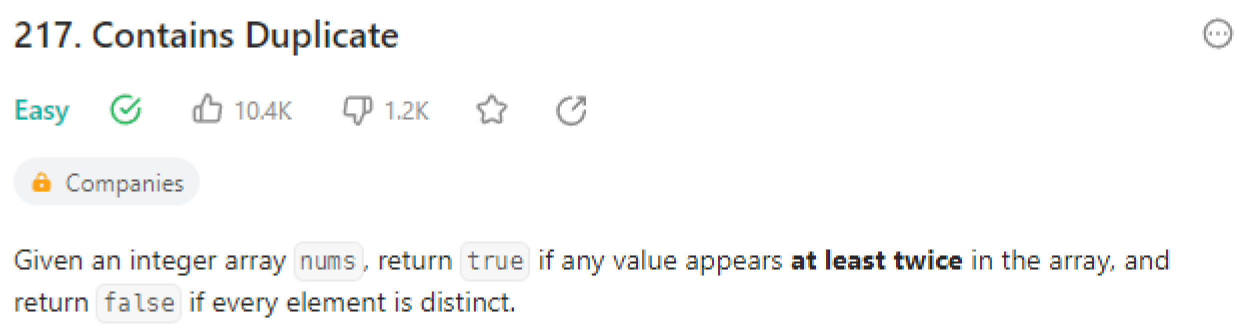
Submission Detail

200 / 200 test cases passed.
Runtime: 1 ms
Memory Usage: 44.3 MB

Status: Accepted
Submitted: 3 weeks, 2 days ago

D) Problem 4

```
class Solution {
```



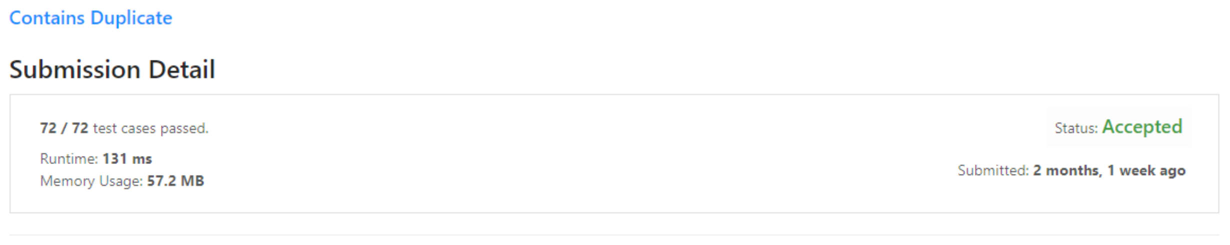
217. Contains Duplicate

Easy 10.4K 1.2K

Companies

Given an integer array `nums`, return `true` if any value appears **at least twice** in the array, and return `false` if every element is distinct.

```
public:
  bool containsDuplicate(vector<int>& nums) {
    sort(nums.begin(),nums.end());
    bool flag = false;
    for(int i =0;i<nums.size()-1;i++){
      if(nums[i] == nums[i+1]) return true;
    }
    return flag;
  }
};
```



Contains Duplicate

Submission Detail

72 / 72 test cases passed.
Runtime: **131 ms**
Memory Usage: **57.2 MB**

Status: Accepted
Submitted: **2 months, 1 week ago**