

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

Analysis and Design of Algorithms

Submitted by

Amish B Harsoor(1BM21CS017)

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 **Amish B Harsoor (1BM21CS017)**, 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.

Sunayana S
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

INDEX

| 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. | 10 |
| 3 | Implement Johnson Trotter algorithm to generate permutations. | 13 |
| 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. | 17 |
| 5 | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | 20 |
| 6 | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. | 23 |
| 7 | Implement 0/1 Knapsack problem using dynamic programming. | 26 |
| 8 | Implement All Pair Shortest paths problem using Floyd's algorithm. | 29 |
| 9 | Find Minimum Cost Spanning Tree of a given undirected graph using Prim's and Kruskal's algorithm. | 31 |
| 10 | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. | 36 |
| 11 | Implement "N-Queens Problem" using Backtracking. | 38 |

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.

Code:

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

```
#include<conio.h>
```

```
int a[15][15],n;
```

```
void bfs(int);
```

```
void main() {
```

```
    int i,j,root;
```

```
    printf("\nEnter the no of nodes:\t");
```

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

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

```
    for(i=1;i<=n;i++)
```

```
        for(j=1;j<=n;j++)
```

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

```
    printf("\nEnter the source node:\t");
```

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

```
    bfs(root);
```

```
}
```

```
void bfs(int root) {
```

```

int q[15],f=0,r=-1,vis[15],i,j;
for(j=1;j<=n;j++)
    vis[j]=0;
vis[root]=1;
r=r+1;
q[r]=root;
while(f<=r) {
    i=q[f];
    f=f+1;
    for(j=1;j<=n;j++)
    {
        if(a[i][j]==1&&vis[j]!=1) {
            vis[j]=1;
            r=r+1;
            q[r]=j;
        }
    }
}
for(j=1;j<=n;j++) {
    if(vis[j]!=1)
        printf("\nNode %d is not reachable",j);
    else{
        printf("\nNode %d is reachable",j);
    }
}

```

Output:

```
Enter the no of nodes:  4

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

Enter the source node:  1

Node 1 is reachable
Node 2 is reachable
Node 3 is reachable
Node 4 is reachable

...Program finished with exit code 0
Press ENTER to exit console.□
```

B.

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

```
#include<conio.h>
```

```
int a[10][10],n,vis[10];
```

```
int dfs(int root){
```

```
    int j;
```

```
    vis[root]=1;
```

```
    for(j=1;j<=n;j++)
```

```
        if(a[root][j]==1&&vis[j]!=1)
```

```
            dfs(j);
```

```
    for(j=1;j<=n;j++) {
```

```
        if(vis[j]!=1)
```

```
            return 0;
```

```
    }
```

```
    return 1;
```

```
}
```

```
void main()
```

```
{
```

```
    int i,j,root,ans;
```

```
    for(j=1;j<=n;j++)
```

```
        vis[j]=0;
```

```
    printf("\nEnter the no of nodes:\t");
```

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

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

```
    for(i=1;i<=n;i++)
```

```
        for(j=1;j<=n;j++)
```



```
scanf("%d",&a[i][j]);
printf("\nEnter the source node:\t");
scanf("%d",&root);
ans=dfs(root);
if(ans==1)
printf("\nGraph is connected\n");
else
printf("\nGraph is not connected\n");
getch();
}
```

Output:

```
Enter the no of nodes: 4
Enter the adjacency matrix:
0 1 1 1
0 0 1 0
0 0 0 0
0 0 0 1
Enter the source node: 1
Graph is connected

...Program finished with exit code 0
Press ENTER to exit console. □
```

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

Code:

```
#include<stdio.h>
#include<conio.h>
void main(){
    int a[10][10],n,i,j;
    int indeg[10],flag[10],c=0;

    printf("Enter number of vertices \n");
    scanf("%d",&n);

    printf("Enter 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;

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

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            if(a[i][j]==1)
```

```

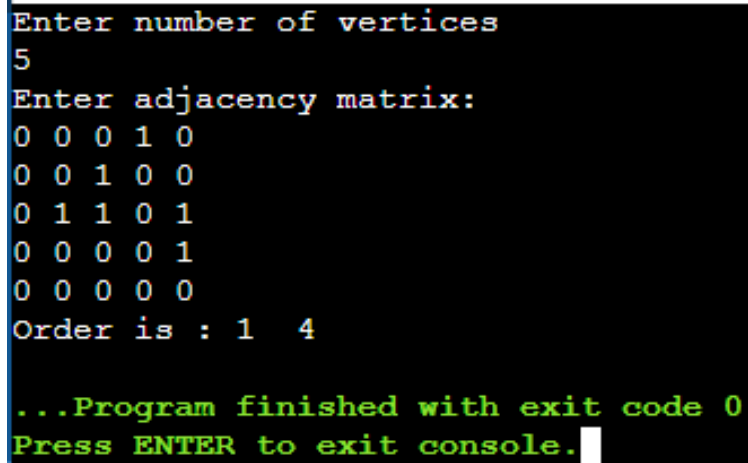
indeg[j]+=1;

printf("Order is : ");
while(c<=n)
{
    for(i=0;i<n;i++)
    {
        if(indeg[i]==0 && flag[i]==0)
        {
            printf("%d ",i+1);
            flag[i]=1;
        }
    }
    for(i=0;i<n;i++)
    {
        if(flag[i]==1)
        {
            for(j=0;j<n;j++)
            {
                if(a[i][j]==1)
                {
                    indeg[j]-=1;
                    a[i][j]=0;
                }
            }
        }
    }
}

```

```
        C++;  
    }  
}
```

Output:



A screenshot of a console window with a black background and white and green text. The text shows the program's execution flow: it prompts for the number of vertices (5), then for the adjacency matrix (a 5x5 grid of 0s and 1s), then displays the order (1 4), and finally shows a green message indicating the program finished with exit code 0 and prompts to press ENTER to exit the console.

```
Enter number of vertices  
5  
Enter adjacency matrix:  
0 0 0 1 0  
0 0 1 0 0  
0 1 1 0 1  
0 0 0 0 1  
0 0 0 0 0  
Order is : 1 4  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

3. Implement Johnson Trotter algorithm to generate permutations.

Code:

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

#define left_to_right true
#define right_to_left false

int getPosOfMobile(int a[], int n, int mobile) {
    for (int i = 0; i < n; i++) {
        if (a[i] == mobile)
            return i + 1;
    }
    return 0;
}

int getMobile(int a[], bool 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;
            }
        }

        // direction 1 represents LEFT TO RIGHT.
        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;
    else
        return mobile;
}

void produceOnePermutation(int a[], bool dir[], int n) {
    int mobile = getMobile(a, dir, n);
    int pos = getPosOfMobile(a, n, mobile);

    if (dir[a[pos] - 1] == right_to_left) {
        int temp = a[pos - 1];
        a[pos - 1] = a[pos - 2];
        a[pos - 2] = temp;
    } else if (dir[a[pos] - 1] == left_to_right) {
        int temp = a[pos];
        a[pos] = a[pos - 1];
        a[pos - 1] = temp;
    }

    // changing the directions for elements
    // greater than largest mobile integer.
    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 fact(int n)
{
    int result=1;
    for(int i=1;i<=n;i++)
    {
        result*=i;
    }
    return result;
}
```

```
void producePermutation(int n) {
    // To store the current permutation
    int a[n];

    // To store the current directions
    bool dir[n];

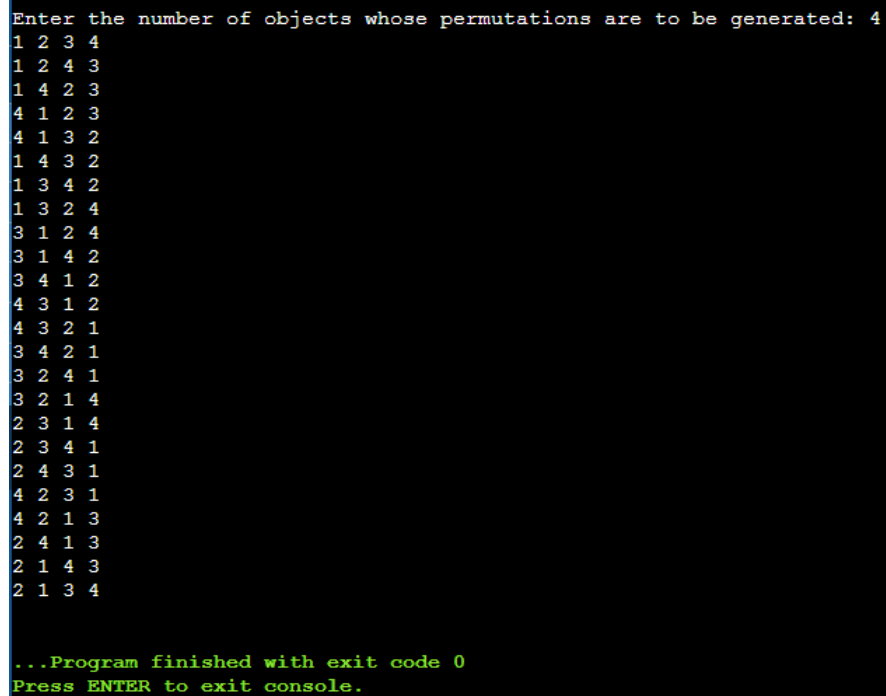
    // Storing the elements from 1 to n and
    // printing the first permutation.
    for (int i = 0; i < n; i++) {
        a[i] = i + 1;
        printf("%d ", a[i]);
    }
    printf("\n");

    // Initially all directions are set
    // to RIGHT TO LEFT i.e. 0.
    for (int i = 0; i < n; i++)
        dir[i] = right_to_left;

    // For generating permutations in order.
    for (int i = 1; i < fact(n); i++)
        produceOnePermutation(a, dir, n);
}
```

```
void main()
{
    int n;
    printf("\nEnter the number of objects whose permutations are to be generated: ");
    scanf("%d",&n);
    producePermutation(n);
}
```

Output:



```
Enter the number of objects whose permutations are to be generated: 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

...Program finished with exit code 0
Press ENTER to exit console.
```


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.

Code:

```
#include<stdio.h>
#include<time.h>
int n;
void merge_sort(int a[],int low,int high)
{
    int mid;
    if(low<high)
    {
        mid = (low+high)/2;
        merge_sort(a,low,mid);
        merge_sort(a,mid+1,high);
        merge(a,low,mid,high);
    }
}
void merge(int a[],int low,int mid,int high)
{
    int i,j,k,b[n];
    i = low;
    k = low;
    j = mid+1;

    while(i<=mid && j<=high)
    {
        if(a[i] < a[j])
        {
            b[k] = a[i];
            i++;
            k++;
        }
        else{
            b[k] = a[j];
            j++;
            k++;
        }
    }
}
```

```

        j++;
        k++;
    }
}
while(i<=mid)
{
    b[k] = a
    i++;
    k++;
}
while(j<=high)
{
    b[k] = a[j];
    j++;
    k++;
}
for(int i=low; i<=high; i++)
{
    a[i]=b[i];
}

}

int main()
{
    int low, high,n;
    printf("Enter size of array: ");
    scanf("%d", &n);
    int a[n];
    printf("Enter array elements: ");
    for(int i=0; i<n; i++){
        // scanf("%d",&a[i]);

        a[i] = rand()%10000;
        printf("%d\t",a[i]);
    }

    low =0;
    high = n-1;
    clock_t start,end;

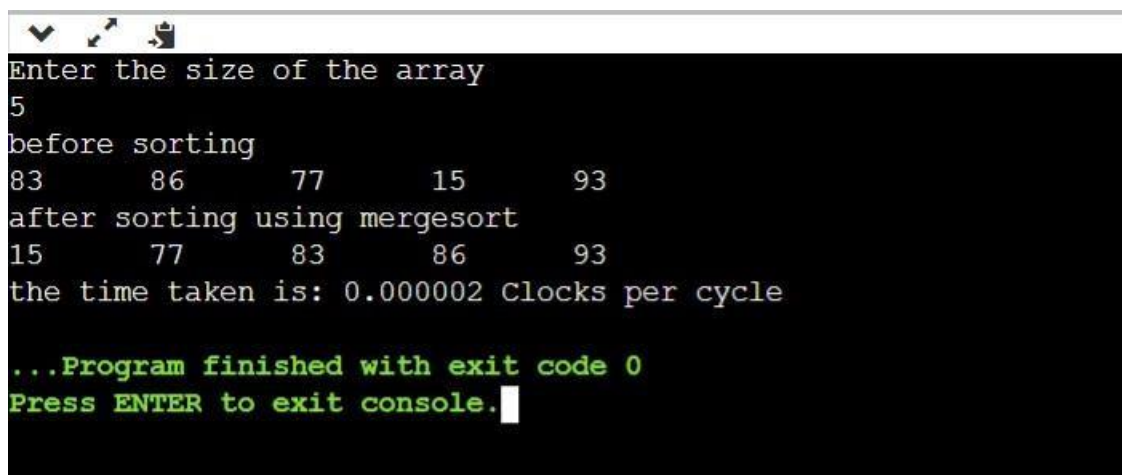
    start = clock();
    merge_sort(a,low,high);
    end = clock();

```

```
printf("\nSorted array elements: ");  
for(int i=0; i<n; i++)  
printf("%d\t",a[i]);  
printf("\nStart time: %f", (double)start);  
printf("\nEnd time: %f", (double)end);  
printf("\nTime take is %f ", (double)(end-start)/CLOCKS_PER_SEC );
```

Output:

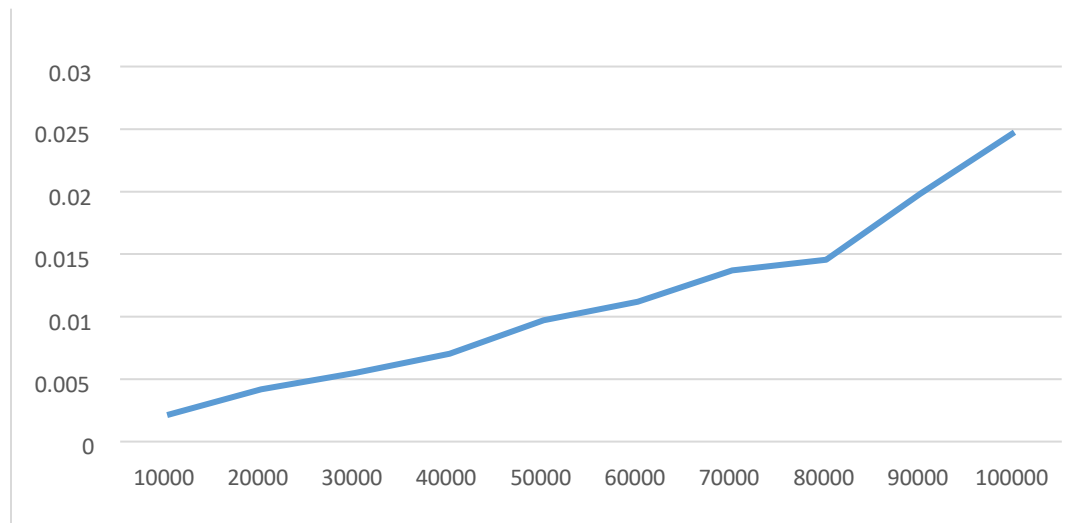
Graph



```
Enter the size of the array  
5  
before sorting  
83      86      77      15      93  
after sorting using mergesort  
15      77      83      86      93  
the time taken is: 0.000002 Clocks per cycle  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

| sizeofarray | timetaken |
|-------------|-----------|
| 10000 | 0.002114 |
| 20000 | 0.00418 |
| 30000 | 0.005486 |
| 40000 | 0.007019 |
| 50000 | 0.00969 |
| 60000 | 0.011191 |
| 70000 | 0.013704 |
| 80000 | 0.014539 |
| 90000 | 0.019828 |
| 100000 | 0.024749 |

Merge Sort



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

Code:

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

void quick_sort(int a[],int low,int high)
{
    int j;
    if(low<high)
    {
        j = partition(a,low,high);
        quick_sort(a,low,j-1);
        quick_sort(a,j+1,high);
    }
}
```

```

}

int partition(int a[],int low,int high)
{
    int i,j,pivot;

    i = low;
    j= high+1;
    pivot = a[low];
    while(i<=j)
    {
        do
        {
            i++;
        }while(pivot>=a[i]);

        do
        {
            j--;
        }while(pivot<a[j]);

        if(i<j)
            swap(&a[i],&a[j]);
    }
    swap(&a[low],&a[j]);
    return j;
}

void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b= temp;
}

int main()
{
    int n;
    printf("Enter size of array: ");
    scanf("%d", &n);
    int a[n], low, high;
    printf("Enter array elements: ");

```

```

for(int i=0; i<n; i++){
    // scanf("%d",&a[i]);

    a[i] = rand()%10000;
    // printf("%d\t",a[i]);
}

low =0;
high = n-1;

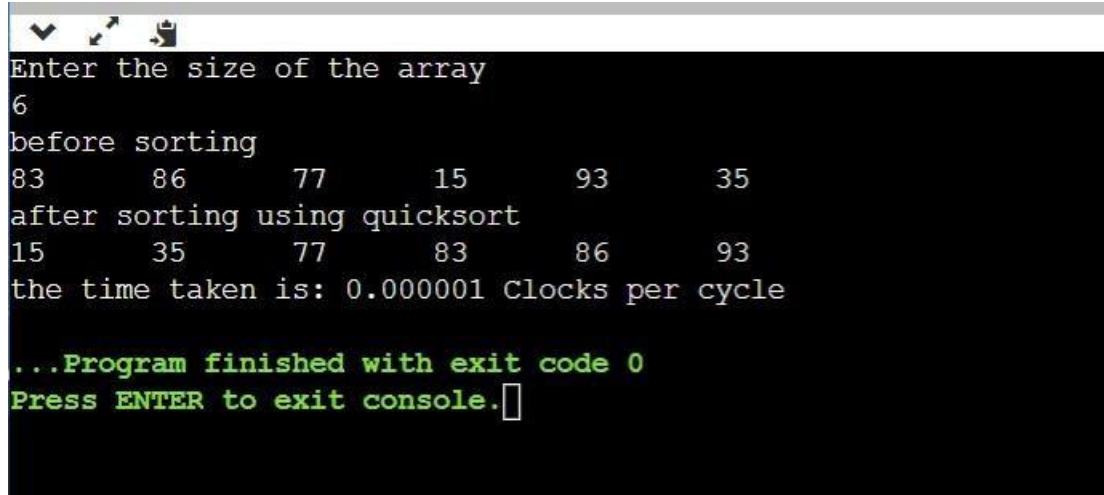
clock_t start,end;

start = clock();
quick_sort(a,low,high);
end = clock();

// printf("\nSorted array elements: ");
// for(int i=0; i<n; i++)
// printf("%d\t",a[i]);
printf("\nStart time: %f", (double)start);
printf("\nEnd time: %f", (double)end);
printf("\nTime take is %f ", (double)(end-start)/CLOCKS_PER_SEC );
}

```

Output:



```

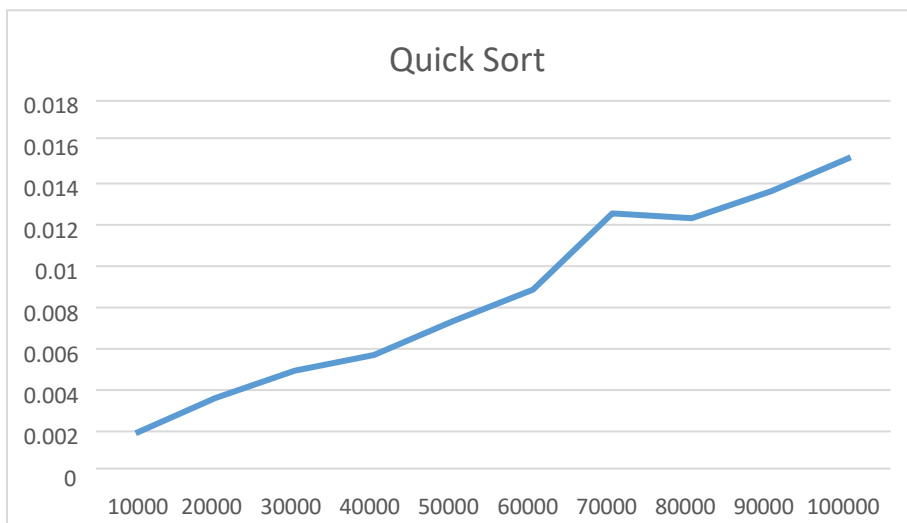
Enter the size of the array
6
before sorting
83      86      77      15      93      35
after sorting using quicksort
15      35      77      83      86      93
the time taken is: 0.000001 Clocks per cycle

...Program finished with exit code 0
Press ENTER to exit console.

```

Graph

| sizeofarray | timetaken |
|-------------|-----------|
| 10000 | 0.001908 |
| 20000 | 0.003618 |
| 30000 | 0.004931 |
| 40000 | 0.005698 |
| 50000 | 0.00735 |
| 60000 | 0.008865 |
| 70000 | 0.012559 |
| 80000 | 0.012323 |
| 90000 | 0.013631 |
| 100000 | 0.015273 |



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

Code

```
#include <stdio.h>
#include<time.h>
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");
}

int main()
{

    int N;
    printf("Enter no of nodes: ");
    scanf("%d",&N);
    int arr[N];
    printf("Enter elements: ");
    for (int i = 0; i < N; i++)
    {
        arr[i] = rand()%10000;
        printf("%d\t",arr[i]);
    }
}

```



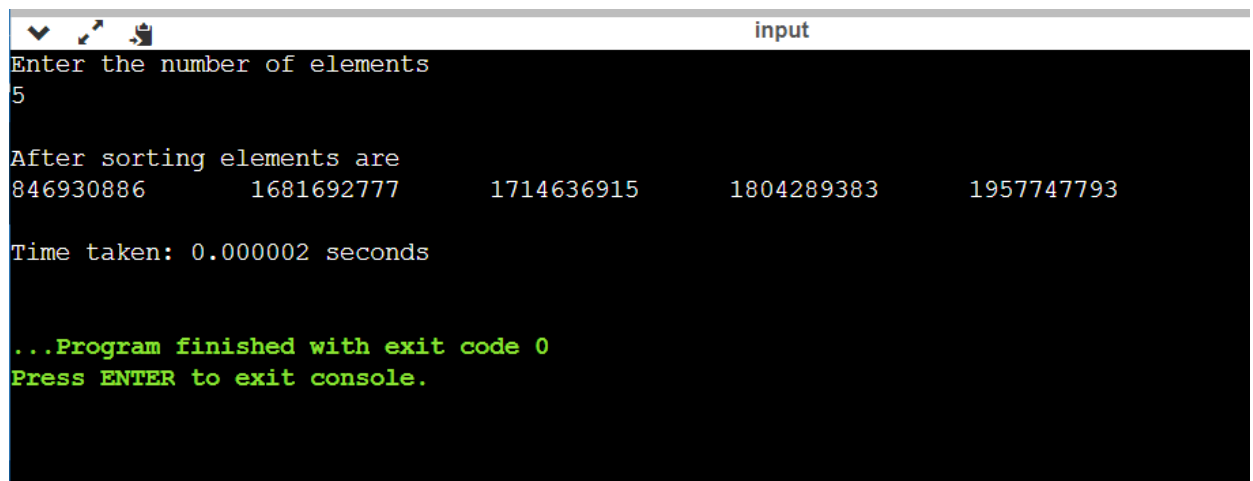
```
clock_t start,end;
start = clock();
heapSort(arr, N);
end = clock();

printf("\nSorted array is\n");
printArray(arr, N);

printf("\nStart time: %f", (double)start);
printf("\nEnd time: %f", (double)end);
printf("\nTime take is %f ", (double)(end-start)/CLOCKS_PER_SEC );
}
```

Output

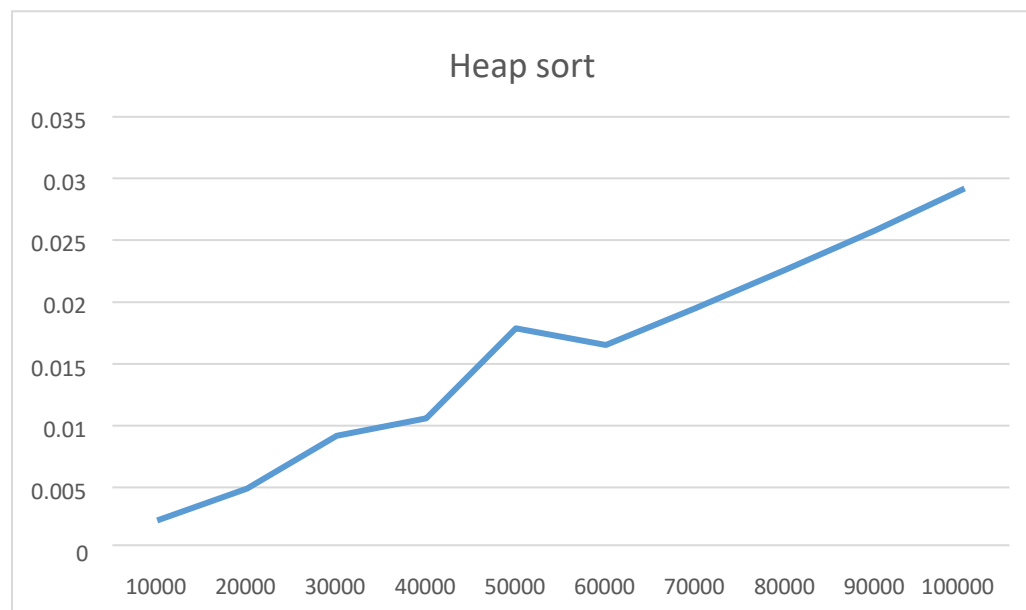
Graph



```
input
Enter the number of elements
5
After sorting elements are
846930886      1681692777      1714636915      1804289383      1957747793
Time taken: 0.000002 seconds

...Program finished with exit code 0
Press ENTER to exit console.
```

| sizeofarray | timetaken |
|-------------|-----------|
| 10000 | 0.002324 |
| 20000 | 0.004903 |
| 30000 | 0.009185 |
| 40000 | 0.010584 |
| 50000 | 0.017871 |
| 60000 | 0.016515 |
| 70000 | 0.019496 |
| 80000 | 0.022587 |
| 90000 | 0.025799 |
| 100000 | 0.029185 |



7. Implement 0/1 Knapsack problem using dynamic programming.

Code:

```
#include<stdio.h>
int n,m;
int max(int a,int b){
    if(a>b){
        return a;
    }
    else{
        return b;
    }
}

void selectObj(int v[n+1][m+1],int w[n],int n){
    int i,j,x[n];
    for(i=1;i<=n;i++){
        x[i]=0;
```

```

}
i=n;
j=m;
while(i!=0 && j!=0){
    if(v[i][j]!=v[i-1][j]){
        x[i]=1;
        j=j-w[i];
    }
    i--;
}
for(i=1;i<=n;i++){
    if(x[i]==1){
        printf("\n%d is selected",i);
    }
    else{
        printf("\n%d not selected",i);
    }
}
}

int* knapsack(int v[n+1][m+1],int w[n],int p[n]){
int i,j;

for(i=0;i<=n;i++){
    for(j=0;j<=m;j++){
        if(i==0 || j==0){
            v[i][j]=0;
        }
        else if(w[i]>j){
            v[i][j]=v[i-1][j];
        }
        else {
            int x,y;
            x=v[i-1][j];
            y=v[i-1][j-w[i]]+p[i];
            v[i][j]=max(x,y);
        }
    }
}
printf("\n");
for(i=0;i<=n;i++){
    for(j=0;j<=m;j++){
        printf("%d\t",v[i][j]);
    }
}

```

```

        printf("\n");
    }
    printf("\nOptimal solution: %d",v[n][m]);
    return v[n+1][m+1];
}

void main(){
    int i,j;
    printf("\n enter no. of items: ");
    scanf("%d",&n);
    int w[n],p[n];
    for(int i=1;i<=n;i++){
        printf("\n enter weight and value of %d: ",i);
        scanf("%d%d",&w[i],&p[i]);
    }
    printf("\n enter knapsack capacity: ");
    scanf("%d",&m);
    int v[n+1][m+1];
    v[n+1][m+1]=knapsack(v,w,p);
    selectObj(v,w,n);
}

```

Output:

```
Enter the number of items
4
enter the weight and profit of each item
2 12
1 10
3 20
2 15
enter the knapsack capacity
5

knapsack table
0      0      0      0      0      0
0      0      12     12     12     12
0      10     12     22     22     22
0      10     12     22     30     32
0      10     15     25     30     37

items selected are designated 1
1 1 0 1

...Program finished with exit code 0
Press ENTER to exit console.
```

8. Implement all pair shortest path problem using Floyd's Algorithm.

Code:

```
#include<stdio.h>
void main()
{
    int i,j,k,n,p[10][10],o[10][10];

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

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

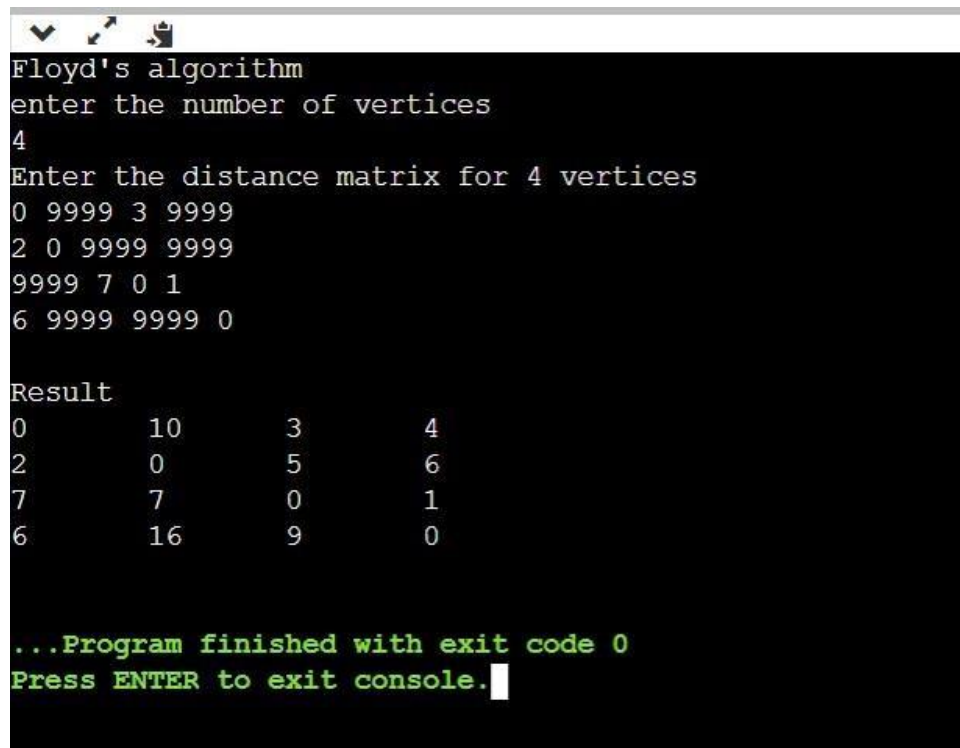
    for(i=0;i<n;i++)
    for(j=0;j<n;j++)
        o[i][j]=p[i][j];

    for(k=0;k<n;k++)
    for(i=0;i<n;i++)
    for(j=0;j<n;j++)
        if(p[i][j] > p[k][j]+p[i][k])
            p[i][j]=p[k][j]+p[i][k];

    printf("\nOriginal Adjacency Matrix \n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            printf("%d ",o[i][j]);
        printf("\n");
    }
    printf("\nUpdated Adjacency Matrix \n");
```

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

Output:



```
Floyd's algorithm
enter the number of vertices
4
Enter the distance matrix for 4 vertices
0 9999 3 9999
2 0 9999 9999
9999 7 0 1
6 9999 9999 0

Result
0      10      3      4
2      0      5      6
7      7      0      1
6      16     9      0

...Program finished with exit code 0
Press ENTER to exit console.
```

9. Find Minimum Cost Spanning Tree of a given undirected graph using prims and kruskals Algorithm.

Prims Code:

```
#include<stdio.h>
float cost[10][10];
int vt[10],et[10][10],vis[10],j,n;
float sum=0;
int x=1;
int e=0;
void prims();

void main()
{
    int i;
    printf("Enter the number of vertices\n");
    scanf("%d",&n);
    printf("Enter the cost adjacency matrix\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%f",&cost[i][j]);
        }
        vis[i]=0;
    }
    prims();
    printf("Edges of spanning tree\n");
    for(i=1;i<=e;i++)
    {
        printf("%d,%d\t",et[i][0],et[i][1]);
    }
    printf("weight=%f\n",sum);
}

void prims()
{
    int s,m,k,u,v;
    float min;
```



```

vt[x]=1;
vis[x]=1;
for(s=1;s<n;s++)
{
    j=x;
    min=999;
    while(j>0)
    {
        k=vt[j];
        for(m=2;m<=n;m++)
        {
            if(vis[m]==0)
            {
                if(cost[k][m]<min)
                {
                    min=cost[k][m];
                    u=k;
                    v=m;
                }
            }
        }
        j--;
    }
    vt[++x]=v;
    et[s][0]=u;
    et[s][1]=v;
    e++;
    vis[v]=1;
    sum=sum+min;
}
}

```

Output:

```
enter the number of vertices
5
enter the cost adjacency matrix
0 1 5 2 999
1 0 999 999 999
5 999 0 3 999
2 999 3 0 2
999 999 999 2 0
edges of spanning tree
1,2      1,4      4,5      4,3      weight=8

...Program finished with exit code 9
Press ENTER to exit console.
```

Kruskals Code:

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

int find(int v,int parent[10])
{
    while(parent[v]!=v)
    {
        v=parent[v];
    }
    return v;
}

void union1(int i,int j,int parent[10])
{
    if(i<j)
        parent[j]=i;
    else
        parent[i]=j;
}

void kruskal(int n,int a[10][10])
{
    int count,k,min,sum,i,j,t[10][10],u,v,parent[10];
    count=0;
```

```

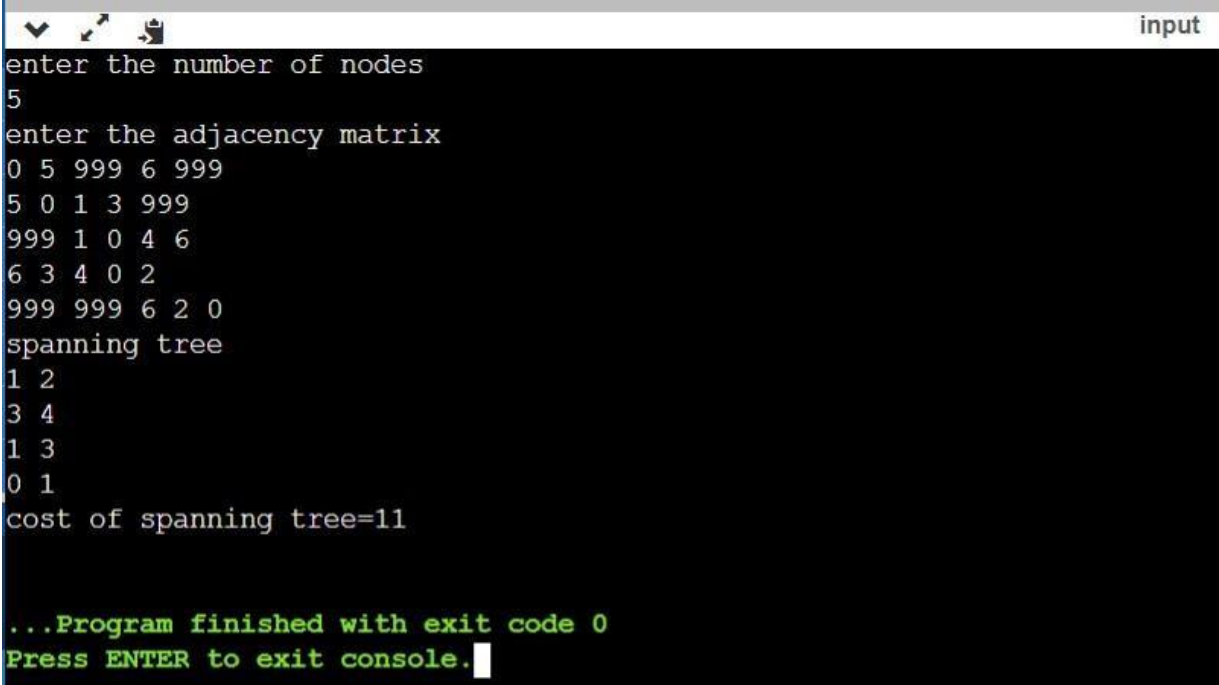
k=0;
sum=0;
for(i=0;i<n;i++)
    parent[i]=i;
while(count!=n-1)
{
    min=999;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            if(a[i][j]<min && a[i][j]!=0)
            {
                min=a[i][j];
                u=i;
                v = j;
            }
        }
    }
    i=find(u,parent);
    j=find(v,parent);
    if(i!=j)
    {
        union1(i,j,parent);
        t[k][0]=u;
        t[k][1]=v;
        k++;
        count++;
        sum=sum+a[u][v];
    }
    a[u][v]=a[v][u]=999;
}
if(count==n-1)
{
    printf("spanning tree\n");
    for(i=0;i<n-1;i++)
    {
        printf("%d %d\n",t[i][0]+1,t[i][1]+1);
    }
    printf("cost of spanning tree=%d\n",sum);
}
else
    printf("spanning tree does not exist\n");

```

```
}
```

```
void main()
{
    int n,i,j,a[10][10];
    printf("Kruskal's Algorithm");
    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",&a[i][j]);
    kruskal(n,a);
}
```

Output:



```
input
enter the number of nodes
5
enter the adjacency matrix
0 5 999 6 999
5 0 1 3 999
999 1 0 4 6
6 3 4 0 2
999 999 6 2 0
spanning tree
1 2
3 4
1 3
0 1
cost of spanning tree=11

...Program finished with exit code 0
Press ENTER to exit console.
```

10. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's Algorithm

Code:

```
#include<stdio.h>
#include<conio.h>
void dijkstras();
int c[10][10],n,src;
void main()
{
    int i,j;
    printf("\nEnter number of vertices:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&c[i][j]);
        }
    }
    printf("\nEnter the source node:\t");
    scanf("%d",&src);
    dijkstras();
    getch();
}

void dijkstras()
{
    int vis[10],dist[10],u,i,j,count,min;
    for(j=1;j<=n;j++)
    {
        dist[j]=c[src][j];
    }
    for(j=1;j<=n;j++)
    {
        vis[j]=0;
    }
    dist[src]=0;
    vis[src]=1;
    count=1;
    while(count!=n)
    {
```

```

min=9999;
for(j=1;j<=n;j++)
{
    if(dist[j]<min&&vis[j]!=1)
    {
        min=dist[j];
        u=j;
    }
}
vis[u]=1;
count++;
for(j=1;j<=n;j++)
{
    if(min+c[u][j]<dist[j]&&vis[j]!=1)
    {
        dist[j]=min+c[u][j];
    }
}
}
printf("\nThe shortest distance is:\n");
for(j=1;j<=n;j++)
{
    printf("\n%d to %d=%d ",src,j,dist[j]);
}
}

```

Output:

```

input
Enter no. of vertices:6
Enter the adjacency matrix:
0 25 100 35 9999 9999
9999 0 9999 27 14 9999
9999 9999 0 50 9999 9999
9999 9999 9999 0 29 9999
9999 9999 9999 9999 0 21
9999 9999 48 9999 9999 0
Enter the starting node:0
Distance of node1 = 25
Path = 1<-0
Distance of node2 = 100
Path = 2<-0
Distance of node3 = 35
Path = 3<-0
Distance of node4 = 39
Path = 4<-1<-0
Distance of node5 = 60
Path = 5<-4<-1<-0
...Program finished with exit code 0
Press ENTER to exit console.

```

11. Implement “N-Queen’s Problem” using backtracking

Code:

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

int board[20],count;

int main()
{
    int n,i,j;
    void queen(int row,int n);

    printf("\n\nEnter number of Queens:");
    scanf("%d",&n);
    queen(1,n);
    return 0;
}

void print(int n)
{
    int i,j;
    printf("\n\nSolution %d:\n\n",++count);

    for(i=1;i<=n;i++)
        printf(" %d",i);
    for(i=1;i<=n;i++)
    {
        printf("\n\n%d",i);
        for(j=1;j<=n;j++)
        {
            if(board[i]==j)
                printf(" Q");
            else
                printf(" -");
        }
    }
}

int place(int row,int column)
```

```

{
int i;
for(i=1;i<=row-1;i++)
{
    if(board[i]==column)
        return 0;
    else
        if(abs(board[i]-column)==abs(i-row))
            return 0;
}
return 1;
}
void queen(int row,int n)
{
int column;
for(column=1;column<=n;column++)
{
    if(place(row,column))
    {
        board[row]=column;
        if(row==n)
            print(n);
        else
            queen(row+1,n);
    }
}
}

```

Output:


```
- N Queens Problem Using Backtracking -
```

```
Enter number of Queens:4
```

```
Solution 1:
```

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | - | Q | - | - |
| 2 | - | - | - | Q |
| 3 | Q | - | - | - |
| 4 | - | - | Q | - |

```
Solution 2:
```

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | - | - | Q | - |
| 2 | Q | - | - | - |
| 3 | - | - | - | Q |
| 4 | - | Q | - | - |

```
...Program finished with exit code 0  
Press ENTER to exit console.
```