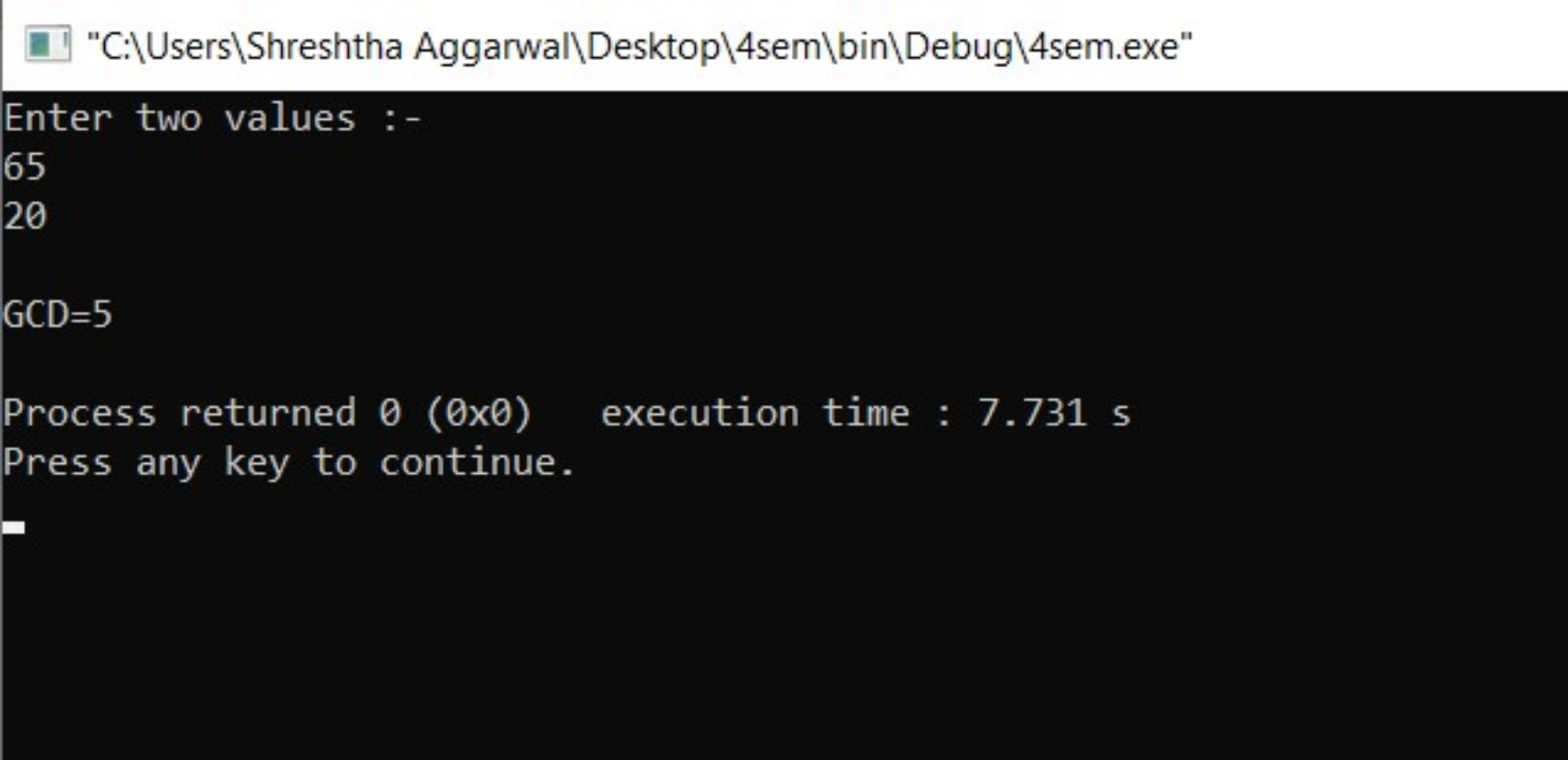
```c
//GCD using iteration
#include <stdio.h>
int main()
{
    int n2,n1,k;
    printf("Enter two values :-\n");
    scanf("%d %d",&n1,&n2);
    while(n2!=0){
        k=n2;
        n2=n1%n2;
        n1=k;
    }
    printf("\nGCD=%d\n",n1);
    return 0;
}
```


"C:\Users\Shreshtha Aggarwal\Desktop\4sem\bin\Debug\4sem.exe"
Enter two values :-
65
20
GCD=5
Process returned 0 (0x0)   execution time : 7.731 s
Press any key to continue.

```c
//GCD using Recursion
#include <stdio.h>
int gcd(int n1, int n2)
{
    if (n2 != 0)
        return gcd(n2,n1%n2);
    else
        return n1;
}
int main(){
    int n1, n2;
    printf("Enter two positive integers : \n");
    scanf("%d%d",&n1,&n2);
    printf("G.C.D of %d and %d is %d.",n1,n2,gcd(n1, n2));
    return 0;
}

```

```
Enter two positive integers :
24
18
G.C.D of 24 and 18 is 6.
Process returned 0 (0x0)   execution time : 10.153 s
Press any key to continue.
```

```c
//Linear and Binary Search using Recursion
#include <stdio.h>
#include<stdlib.h>
int linear(int a[],int l,int r,int key)
{
     if(r<l)
        return -1;
     if(a[l]==key)
        return l;
     if(a[r]==key)
        return r;
     return linear(a,l+1,r-1,key);
}
int binary(int a[],int first,int last,int key)
{
    if (last>=first)
    {
        int m=first+(last-first)/2;
        if(a[m]==key){;
            return m;
        }
        if(a[m]>key){
            return binary(a,first,m-1,key);
        }
        return binary(a,m+1,last,key);
    }
    return -1;
}
int main()
{
    int a[200],i,choice,key,n,res;
    printf("Enter the size of the array : ");
```

```c
    printf("Enter the size of the array : ");
    scanf("%d", &n);
    printf("Enter values of array in ascending order\n");
    for (i=0;i<n;i++){
        scanf("%d",&a[i]);
    }
    for(;;){
        printf("\nEnter value to find\n");
        scanf("%d",&key);
        printf("1.Linear Search\n2.Binary Search\n3.Exit\n");
        scanf("%d",&choice);
        switch(choice){
            case 1: printf("Linear search :- \n");
                    res=linear(a,0,n-1,key);
                    if(res!=-1){printf("%d is present at location %d",key,(res+1));}
                    else{printf("%d is not present\n",key);}
                    break;
            case 2: printf("Binary search :- \n");
                    res=binary(a,0,n-1,key);
                    if(res==-1){printf("%d is not present in the list\n",key);}
                    else{printf("%d is found at location %d\n",key,(res+1));}
                    break;
            case 3: exit(0);
            default:printf("Proper instruction not provided\n");
                    break;
        }
    }
    return 0;
}
```

"C:\Users\Shreshtha Aggarwal\Desktop\4sem\bin\Debug\4sem.exe"

```
Enter the size of the array : 5
Enter values of array in ascending order
12
45
55
56
89

Enter value to find
89
1.Linear Search
2.Binary Search
3.Exit
1
Linear search :-
89 is present at location 5
Enter value to find
56
1.Linear Search
2.Binary Search
3.Exit
2
Binary search :-
56 is found at location 4

Enter value to find
30
1.Linear Search
2.Binary Search
3.Exit
1
Linear search :-
30 is not present

Enter value to find
45
1.Linear Search
2.Binary Search
3.Exit
3

Process returned 0 (0x0)   execution time : 51.240 s
Press any key to continue.
```

```c
//Selection and Bubble Sort
#include<stdio.h>
#include<stdlib.h>
void swap(int *x, int *y)
{
    int temp;
    temp=(*x);
    (*x)=(*y);
    (*y)=temp;
}
void bubble(int a[],int n)
{
    int i=0,j=0;
    int swapped;
    while(i!=n-1){
        swapped = 0;
        for(j=0;j<n-i-1;j++){
            if(a[j]>a[j+1]){
                swap(&a[j],&a[j+1]);
                swapped = 1;
            }
        }
        i++;
        if(swapped==0)
            return;
    }
}
void selection(int a[],int n)
{
    int i,j,min;
    for (i=0;i<n-1;i++)
```

```
    for (i=0;i<n-1;i++)
    {
        min=i;
        for (j=i+1;j<n;j++){
            if(a[j]<a[min]){
                min=j;
            }
        }
        swap(&a[min],&a[i]);
    }
}
void display(int a[],int n)
{
    int i;
    for (i=0;i<n;i++){
        printf("%d ",a[i]);
    }
}
int main()
{
    int a[10],i,choice,n;
    for(;;){
        printf("\nEnter the size of the array : ");
        scanf("%d",&n);
        printf("Enter values of array\n");
        for (i=0;i<n;i++){
            scanf("%d",&a[i]);
        }
        printf("1.Bubble Sort\n2.Selection Sort\n3.Exit\n");
        scanf("%d",&choice);
        switch(choice){
```

```
182         switch(choice){
183             case 1: printf("Bubble Sort :- \n");
184                     bubble(a,n);
185                     display(a,n);
186                     break;
187             case 2: printf("Selection Sort :- \n");
188                     selection(a,n);
189                     display(a,n);
190                     break;
191             case 3: exit(0);
192             default:printf("Proper instruction not provided\n");
193                     break;
194         }
195     }
196     return 0;
197 }
198
199
200
```

"C:\Users\Shreshtha Aggarwal\Desktop\4sem\bin\Debug\4sem.exe"

```
Enter the size of the array : 3
Enter values of array
24
15
6
1.Bubble Sort
2.Selection Sort
3.Exit
1
Bubble Sort :-
6 15 24
Enter the size of the array : 4
Enter values of array
87
92
15
7
1.Bubble Sort
2.Selection Sort
3.Exit
2
Selection Sort :-
7 15 87 92
Enter the size of the array : 1
Enter values of array
14
1.Bubble Sort
2.Selection Sort
3.Exit
3

Process returned 0 (0x0)   execution time : 26.641 s
Press any key to continue.
```

```c
//Selection Sort with time complexity
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void swap(int *x, int *y)
{
    int temp;
    temp=(*x);
    (*x)=(*y);
    (*y)=temp;
}
void bubble(int a[],int n)
{
    int i=0,j=0;
    int swapped;
    while(i!=n-1){
        swapped = 0;
        for(j=0;j<n-i-1;j++){
            if(a[j]>a[j+1]){
                swap(&a[j],&a[j+1]);
                swapped = 1;
            }
        }
        i++;
        if(swapped==0)
            return;
    }
}
void selection(int a[],int n)
{
    int i,j,min;
    for (i=0;i<n-1;i++)
```

```
317     {
318         min=i;
319         for (j=i+1;j<n;j++){
320             if(a[j]<a[min]){
321                 min=j;
322             }
323         }
324         swap(&a[min],&a[i]);
325     }
326 }
327 void display(int a[],int n)
328 {
329     int i;
330     for (i=0;i<n;i++){
331         printf("%d ",a[i]);
332     }
333 }
334 int main()
335 {
336     int a[5000],i,c,n=0,j,k;
337     printf("1.Selection Sort\n2.Bubble Sort\n3.Exit\n");
338     scanf("%d",&i);
339     printf("Enter the no. of times u want sort : ");
340     scanf("%d",&j);
341     for(n;n<j;n=n+0){
342         printf("\nEnter array size : ");
343         scanf("%d",&k);
344         for (c=0;c<k;c++){
345             a[c]=rand();
346         }
347         double ss_time=0.0;
348         clock_t begin=clock();
349         if(i==1){
```

```
349         if(i==1){
350             printf("\n\n\n");
351             printf("Selection Sort :-\n");
352             selection(a,k);
353         }
354         else if(i==2){
355             printf("\n\n\n");
356             printf("Bubble Sort :-\n");
357             bubble(a,k);
358         }
359         else{
360             exit(0);
361         }
362         display(a,k);
363         clock_t end=clock();
364         ss_time+=(double)(end-begin)/CLOCKS_PER_SEC;
365         printf("\nn=%d :%f\n",k,ss_time);
366     }
367
368     return 0;
369 }
370
```

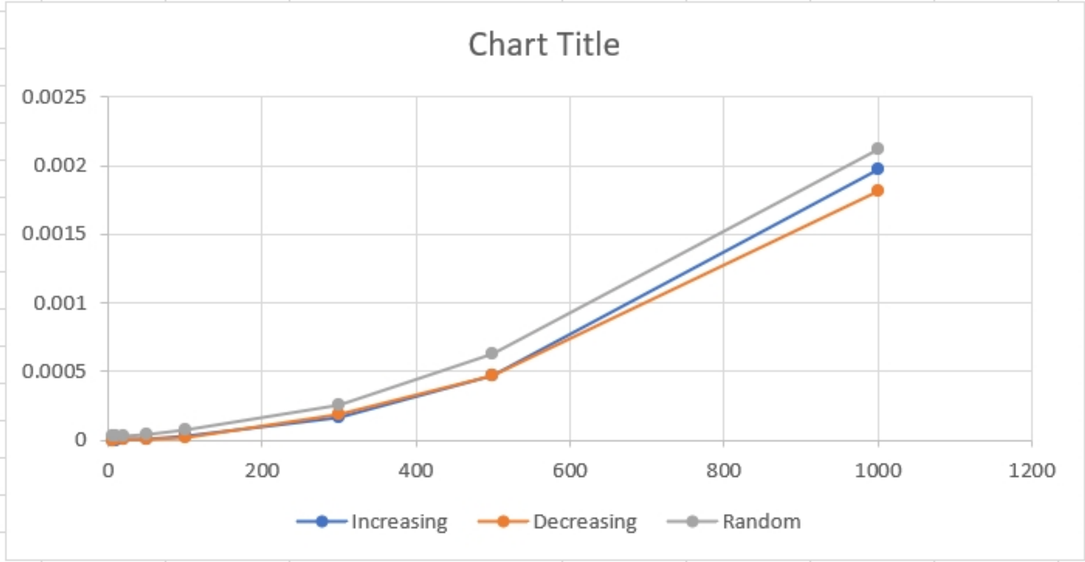
"C:\Users\Shreshtha Aggarwal\Desktop\4sem\bin\Debug\4sem.exe"

```
1.Selection Sort
2.Bubble Sort
3.Exit
1
Enter the no. of times u want sort : 1

Enter array size : 50



Selection Sort :-
41 153 292 491 1869 2995 3902 4664 4827 5436 5447 5705 6334 6868 7711 9894 9961 11478 11538 11942 12382 14604 14771 15141 15724 16827 17035 17421 17673 18467 18716 19169 19718 19895 19912 21726 23281 23811 2446
 25667 26299 26500 26962 28145 28253 28703 29358 30333 31322 32391
n=50    Time:0.005000

Process returned 0 (0x0)   execution time : 3.894 s
Press any key to continue.
```

"C:\Users\Shreshtha Aggarwal\Desktop\4sem\bin\Debug\4sem.exe"

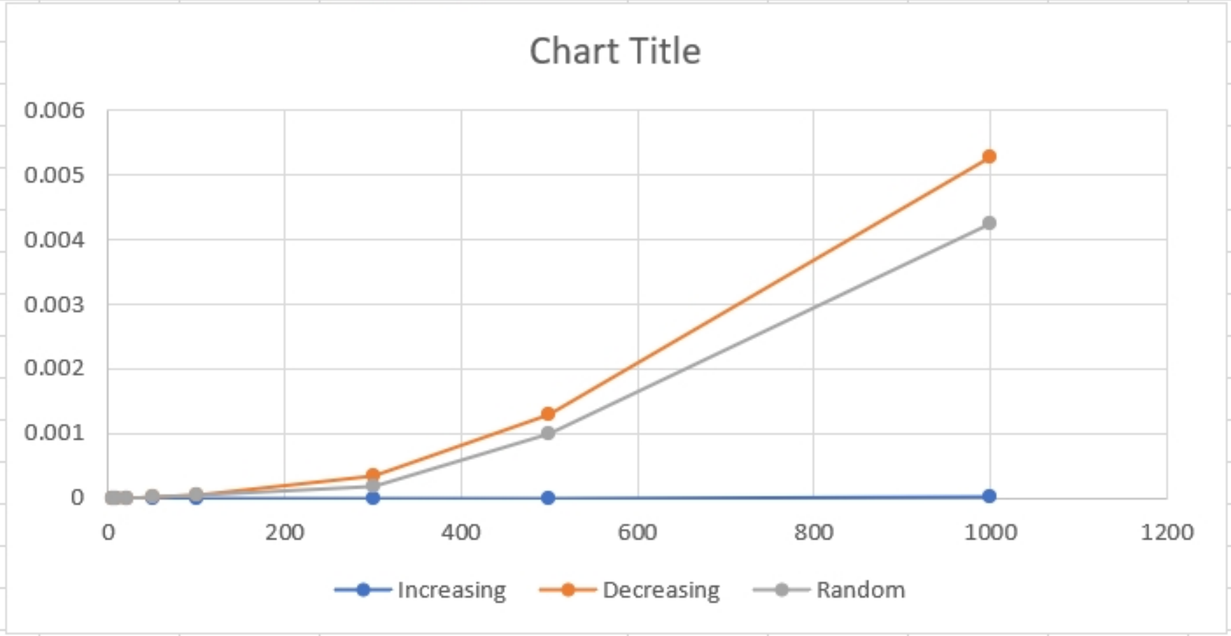
```
1.Selection Sort
2.Bubble Sort
3.Exit
2
Enter the no. of times u want sort : 2

Enter array size : 20



Bubble Sort :-
41 491 2995 4827 5436 5705 6334 9961 11478 11942 15724 16827 18467 19169 23281 24464 26500 26962 28145 29358
n=20    Time:0.001000

Enter array size : 100



Bubble Sort :-
153 288 292 778 1842 1869 2082 2306 3035 3548 3902 4639 4664 4833 4966 5021 5097 5447 5537 5829 6270 6729 6868 7376 7711 8723 8942 9040 9741 9894 9930 11323 11538 11840 12316 12382 12623 12859 13931 13977 14604
14771 15006 15141 15350 15573 15890 16118 16512 16541 16944 17035 17421 17673 18716 18756 19072 19264 19629 19718 19895 19912 19954 20037 21538 21726 22190 22386 22648 22704 22929 23805 23811 24084 24370 24393 2
4626 25547 25667 26299 26308 26777 26924 27446 27529 27644 28253 28703 28745 29658 30106 30333 31101 31115 31322 31673 32391 32439 32662 32757
n=100   Time:0.009000

Process returned 0 (0x0)   execution time : 8.115 s
Press any key to continue.
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1			Array Values							Selection Sort								
2	Array Size	Increasing	Decreasing	Random														
3	5	0.000002	0.000002	0.000025														
4	10	0.000002	0.000004	0.000027														
5	20	0.000005	0.000003	0.000028														
6	50	0.000007	0.000008	0.000038														
7	100	0.000025	0.000024	0.000074														
8	300	0.000171	0.000184	0.000262														
9	500	0.000476	0.000468	0.000631														
10	1000	0.001974	0.001816	0.002119														
11																		
12																		
13																		
14																		
15																		
16																		
17																		
18																		
19																		



		Array Values	
Array Size	Increasing	Decreasing	Random
5	0.000002	0.000002	0.000002
10	0.000002	0.000002	0.000003
20	0.000002	0.000004	0.000005
50	0.000003	0.000015	0.000015
100	0.000002	0.000056	0.000049
300	0.000003	0.000337	0.000176
500	0.000004	0.001288	0.000982
1000	0.000018	0.005268	0.004253

Bubble Sort



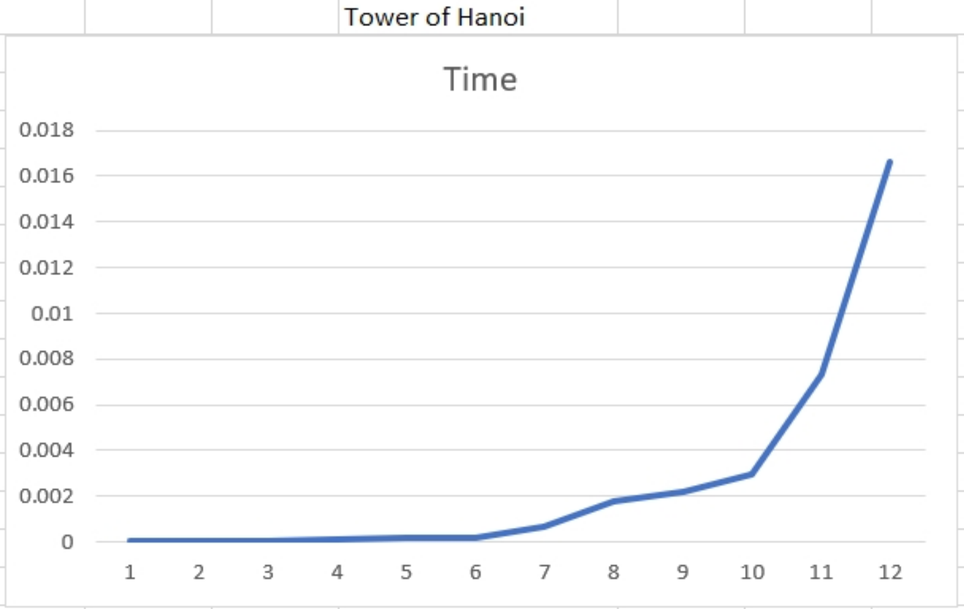
```c
//Tower of Hanoi
#include<stdio.h>
#include<time.h>
#include<math.h>
void towers(int n,char s, char t, char d)
{
    if(n==1){
        printf("Move disk 1 from %c to %c\n",s,d);
        return;
    }
    towers(n-1,s,d,t);
    printf("Move disk %d from %c to %c\n",n,s,d);
    towers(n-1,t,s,d);
}
int main()
{
    int n;
    printf("Enter no. of disks : ");
    scanf("%d",&n);
    double toh_t
    clock_t begin=clock();
    towers(n,'S','T','D');
    printf("\nTotal Steps : %lf",(pow(2,n)-1));
    clock_t end=clock();
    toh_time+=(double)(end-begin)/CLOCKS_PER_SEC;
    printf("\nn=%d\tTime:%f\n",n,toh_time);
    return 0;
}
```

int main::n

"C:\Users\Shreshtha Aggarwal\Desktop\4sem\bin\Debug\4sem.exe"

```
Enter no. of disks : 5
Move disk 1 from S to D
Move disk 2 from S to T
Move disk 1 from D to T
Move disk 3 from S to D
Move disk 1 from T to S
Move disk 2 from T to D
Move disk 1 from S to D
Move disk 4 from S to T
Move disk 1 from D to T
Move disk 2 from D to S
Move disk 1 from T to S
Move disk 3 from D to T
Move disk 1 from S to D
Move disk 2 from S to T
Move disk 1 from D to T
Move disk 5 from S to D
Move disk 1 from T to S
Move disk 2 from T to D
Move disk 1 from S to D
Move disk 3 from T to S
Move disk 1 from D to T
Move disk 2 from D to S
Move disk 1 from T to S
Move disk 4 from T to D
Move disk 1 from S to D
Move disk 2 from S to T
Move disk 1 from D to T
Move disk 3 from S to D
Move disk 1 from T to S
Move disk 2 from T to D
Move disk 1 from S to D

Total Steps : 31.000000
n=5     Time:0.015000

Process returned 0 (0x0)   execution time : 2.531 s
Press any key to continue.
```

Disks	Time
1	0.000031
2	0.000046
3	0.000079
4	0.000149
5	0.000179
6	0.000166
7	0.000692
8	0.001766
9	0.002185
10	0.002969
11	0.007319
12	0.016592



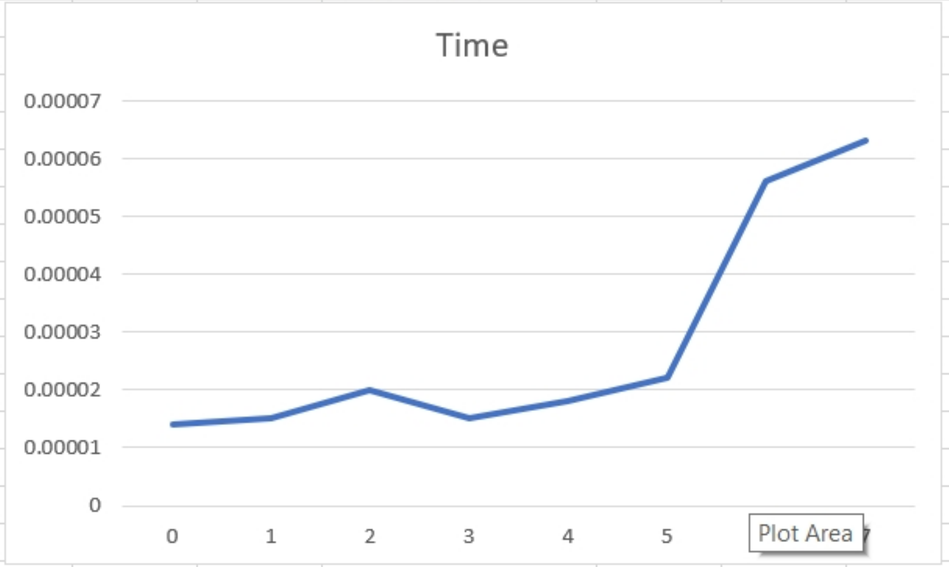
```c
//DFS Traversal
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int G[10][10],v[10],n,a[1][10];
void dfs(int i)
{
    int j;
    printf("\n%d",i);
    v[i]=1;
    for(j=0;j<n;j++){
       if(!v[j]&&G[i][j]==1)
            dfs(j);
    }
}
void dfs_c(int n,int G[10][10],int m,int s[])
{
    int y;
    s[m]=1;
    for(y=0;y<n;y++){
        if((G[m][y]==1)&&(!s[y]))
            dfs_c(n,G,y,s);
    }
}
int main()
{
    int i,j,con,s[10],flag;
    printf("Enter number of vertices : ");
    scanf("%d",&n);
    printf("\nEnter adjecency matrix of the graph :\n");
    for(i=0;i<n;i++)
    {
        printf("Enter row %d : -\n",i+1);
```

```
416         for(j=0;j<n;j++)
417             scanf("%d",&G[i][j]);
418      }
419     for(i=0;i<n;i++)
420         v[i]=0;
421     printf("DFS Traversal order :-\n");
422     double dfs_time=0.0;
423     clock_t begin=clock();
424     dfs(0);
425     con=0;
426     for(j=0;j<n;j++){
427         for(i=0;i<n;i++)
428             s[i]=0;
429         dfs_c(n,G,j,s);
430         flag=0;
431         for(i=0;i<n;i++){
432             if(s[i]==0)
433                 flag=1;
434         }
435         if(flag==0)
436             con=1;
437     }
438     if(con==1)
439        printf("\nGraph is connected\n");
440      else
441        printf("\nGraph is not connected\n");
442     clock_t end=clock();
443     dfs_time+=(double)(end-begin)/CLOCKS_PER_SEC;
444     printf("\nn=%d\tTime:%f\n",n,dfs_time);
445     return 0;
446 }
447
```

```
Enter number of vertices : 3

Enter adjecency matrix of the graph :
Enter row 1 : -
0
1
0
Enter row 2 : -
0
0
1
Enter row 3 : -
1
0
0
DFS Traversal order :-

0
1
2
Graph is connected

n=3     Time:0.000038


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

DFS Traversal

No. of Vertices	Time
0	0.000014
1	0.000015
2	0.00002
3	0.000015
4	0.000018
5	0.000022
6	0.000056
7	0.000063

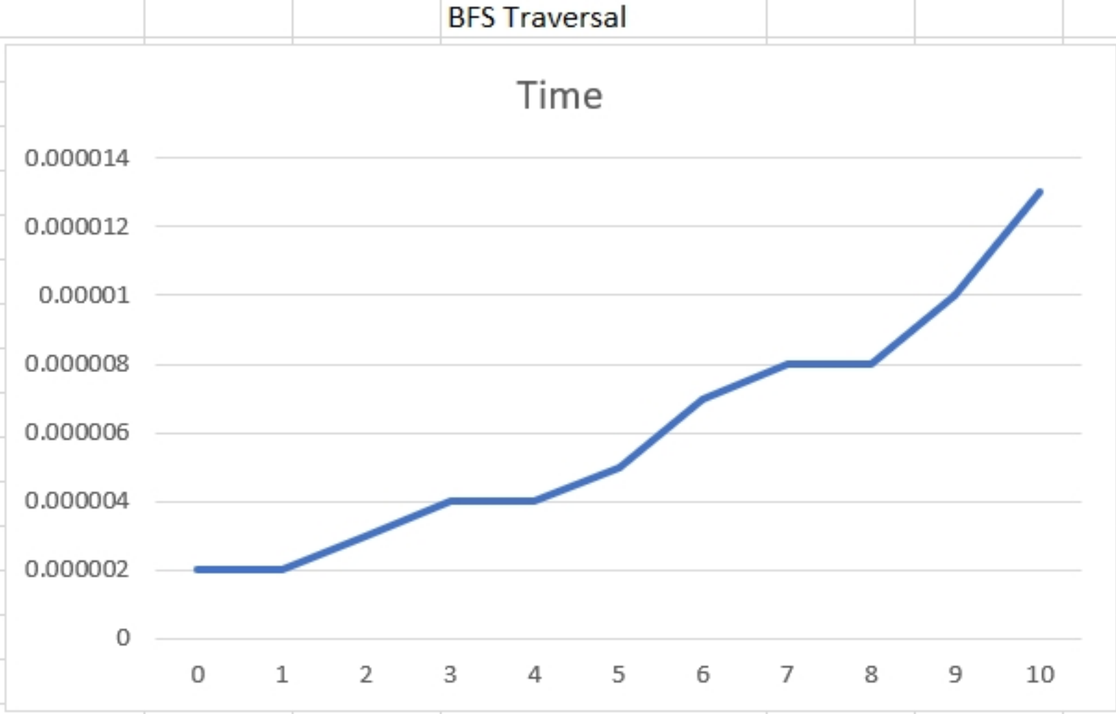


```c
//BFS Visitable Nodes
#include<stdio.h>
#include<time.h>
int a[10][10],q[10],visited[10],n,f=0,r=-1;
void bfs(int v){
    int i;
    for (i=1;i<=n;i++)
      if(a[v][i] && !visited[i])
       q[++r]=i;
    if(f<=r) {
        visited[q[f]]=1;
        bfs(q[f++]);
    }
}
int main() {
    int v,i,j;
    printf("\nEnter the number of vertices : ");
    scanf("%d",&n);
    printf("\nEnter matrix: \n");
    for (i=1;i<=n;i++){
      printf("Enter row %d : \n",i);
      for (j=1;j<=n;j++)
       scanf("%d",&a[i][j]);
    }
    for (i=1;i<=n;i++){
        q[i]=0;
        visited[i]=0;
    }
```

```
522     }
523     printf("\nEnter the beginning vertex : ");
524     scanf("%d",&v);
525     double bfs_time=0.0;
526     clock_t begin=clock();
527     bfs(v);
528     clock_t end=clock();
529     bfs_time+=(double)(end-begin)/CLOCKS_PER_SEC;
530     printf("Visitable Nodes are : \n");
531     for (i=1;i<=n;i++)
532       if(visited[i])
533        printf("%d ",i);
534       else
535        printf("\nBfs is not possible");
536     printf("\nn=%d\tTime:%f\n",n,bfs_time);
537     return 0;
538 }
539
540
```

```
Enter the number of vertices : 2

Enter matrix:
Enter row 1 :
0
1
Enter row 2 :
1
0

Enter the beginning vertex : 1
Visitable Nodes are :
1 2
n=2     Time:0.000002


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

No. of Vertices	Time
0	0.000002
1	0.000002
2	0.000003
3	0.000004
4	0.000004
5	0.000005
6	0.000007
7	0.000008
8	0.000008
9	0.00001
10	0.000013



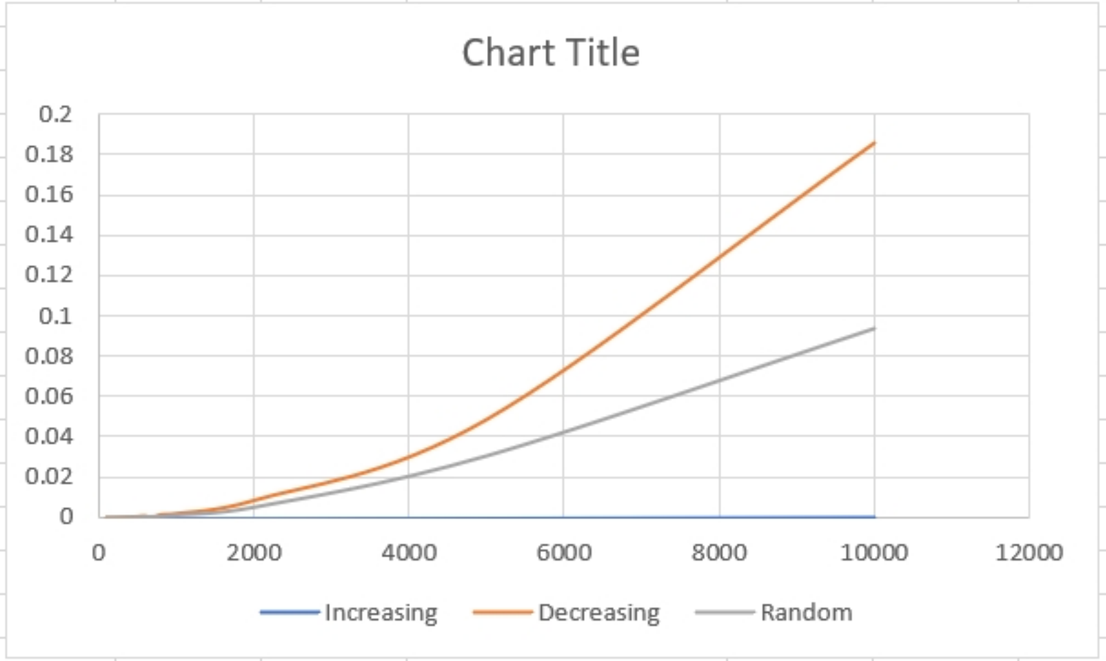
```c
//Insertion Sort
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void insertion(int a[],int n){
    for(int i=1;i<n;i++){
        int j,key;
        key=a[i];
        j=i-1;
        while(key<a[j]&&j>=0){
            a[j+1]=a[j];
            --j;
        }
        a[j+1]=key;
    }
}
void display(int a[],int n){
    for (int i=0;i<n;i++){
        printf("%d ",a[i]);
    }
}
int main(){
    int n,i,a[10];
    printf("\nEnter the size of the array : ");
    scanf("%d",&n);
    printf("Enter values of array\n");
    for(i=0;i<n;i++){
```

```
477     for(i=0;i<n;i++){
478         scanf("%d",&a[i]);
479     }
480     double is_time=0.0;
481     clock_t begin=clock();
482     insertion(a,n);
483     clock_t end=clock();
484     is_time+=(double)(end-begin)/CLOCKS_PER_SEC;
485     printf("Sorted array : \n");
486     display(a,n);
487     printf("\nn=%d\tTime:%f\n",n,is_time);
488     return n;
489 }
490
```

```
Enter the size of the array : 4
Enter values of array
25
41
7
8
Sorted array :
7 8 25 41
n=4     Time:0.000003


...Program finished with exit code 4
Press ENTER to exit console.
```

Insertion Sort

	Array Values		
Array Size	Increasing	Decreasing	Random
100	0.000002	0.000025	0.000015
200	0.000003	0.000087	0.000049
300	0.000003	0.000208	0.0001
400	0.000003	0.000366	0.000183
500	0.000004	0.000535	0.000306
600	0.000005	0.000812	0.000432
700	0.000005	0.0001108	0.000557
800	0.000005	0.001465	0.000707
900	0.000006	0.001652	0.001059
1000	0.000007	0.00199	0.001317
2000	0.000013	0.008508	0.005131
5000	0.000029	0.04864	0.030567
10000	0.000058	0.185927	0.093674



```c
//Topological using source removal
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int a[10][10],indegree[10],front=-1,rear=-1,stack[10],t[10],k,n;
void get(int n){
    int i,j;
    printf("Enter the adjacency matrix :\n");
    for(i=0;i<n;i++){
        printf("Enter row %d\n",(i+1));
        for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);
    }
}
void cal_indegree(int n){
    int i,j;
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            indegree[i]=indegree[i]+a[j][i];
        }
    }
}
void push(int x){
    if(front==-1&&rear==-1)
        front=rear=0;
    else if(rear==n-1)
        return;
    else{
        rear++;
    }
    stack[rear]=x;
}
int pop()
```

```
{
    int x;
    if(front==-1||front>rear){
        return -1;
    }
    x=stack[front];
    if(front==rear||front>rear){
        front=-1;
        rear=-1;
    }
    else{
      front++;
    }
    return x;

}
int main(){
    int i,v;
    printf("Enter the no of vertices :\n");
    scanf("%d",&n);
    get(n);
    for(i=0;i<n;i++){
        indegree[i]=0;
    }
    double t_time=0.0;
    clock_t begin=clock();
    cal_indegree(n);
    printf("\nThe topological order is : ");
    for(int i=0;i<n;i++){
        if(indegree[i]==0){
            push(i);
        }
    }
```

```
668     while(front!=-1){
669         int u=pop();
670         if(u==-1)
671             break;
672         t[k]=u;
673         k++;
674         for(v=0;v<n;v++){
675             if(a[u][v]==1){
676                 ((indegree[v])--);
677                 if(indegree[v]==0){
678                     push(v);
679                 }
680             }
681         }
682     }
683     clock_t end=clock();
684     t_time+=(double)(end-begin)/CLOCKS_PER_SEC;
685     printf("Topological Order :\n");
686     for(int i=0;i<k;i++){
687         printf("%d\t",t[i]);
688     }
689     printf("\nFor matrix n=%d\tTime:%f\n",n,t_time);
690     return 0;
691 }
692
```

```
Enter the no of vertices :
6
Enter the adjacency matrix :
Enter row 1
0 0 0 0 0 0
Enter row 2
0 0 0 0 0 0
Enter row 3
0 0 0 1 0 0
Enter row 4
0 1 0 0 0 0
Enter row 5
1 1 0 0 0 0
Enter row 6
1 0 1 0 0 0

The topological order is : Topological Order :
4       5       0       2       3       1
For matrix n=6  Time:0.000018


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

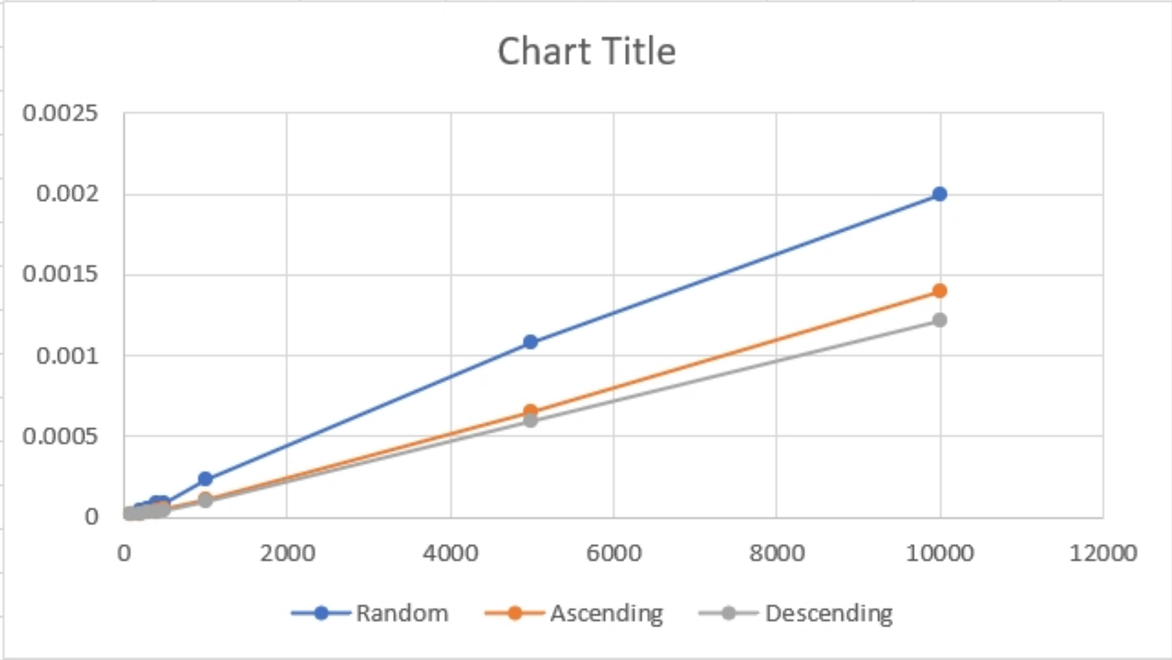
```
695  //Merge Sort
696  #include<stdio.h>
697  #include<stdlib.h>
698  #include<time.h>
699  void merge(int a[],int l,int mid,int r)
700  {
701      int c[10000],i,j,k;
702      i=k=l;
703      j=mid+1;
704      while(i<=mid&&j<=r){
705          if(a[i]<a[j]){
706              c[k]=a[i];
707              ++k;
708              ++i;
709          }
710          else{
711              c[k]=a[j];
712              ++k;
713              ++j;
714          }
715      }
716      if(i>mid){
717          while(j<=r){
718              c[k]=a[j];
719              ++k;
720              ++j;
721          }
722      }
723      if(j>r){
724          while(i<=mid){
725              c[k]=a[i];
726              ++k;
727              ++i;
```

```
728             }
729         }
730         for(i=l;i<=r;i++)
731             a[i]=c[i];
732     }
733     void mergeSort(int a[],int l,int r)
734     {
735         int mid;
736         if(l<r){
737             mid=(l+r)/2;
738             mergeSort(a,l,mid);
739             mergeSort(a,mid+1,r);
740             merge(a,l,mid,r);
741         }
742     }
743     int main(){
744         int a[10000],n,i;
745         printf("Enter the number of elements : ");
746         scanf("%d",&n);
747         printf("\nEnter array elements :\n");
748         for(i=0;i<n;i++)
749             a[i]=rand();
750         clock_t begin=clock();
751         mergeSort(a,0,n-1);
752         clock_t end=clock();
753         double m_time=0.0;
754         m_time+=(double)(end-begin)/CLOCKS_PER_SEC;
755         printf("\nSorted array : ");
756         for(i=0;i<n;i++)
757             printf("%d ",a[i]);
758         printf("\n\nTime taken for %d numbers is %f",n,m_time);
759         return 0;
760     }
```

```
Enter the number of elements : 5

Enter array elements :

Sorted array : 846930886 1681692777 1714636915 1804289383 1957747793

Time taken for 5 numbers is 0.000004

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

	Array Values		
Array Size	Random	Ascending	Descending
100	0.000022	0.000014	0.000014
200	0.000037	0.000017	0.000017
300	0.000057	0.000027	0.000027
400	0.000084	0.000038	0.000035
500	0.000092	0.000056	0.000046
1000	0.000236	0.000105	0.000104
5000	0.001079	0.000651	0.000596
10000	0.001996	0.001402	0.001215

Merge Sort



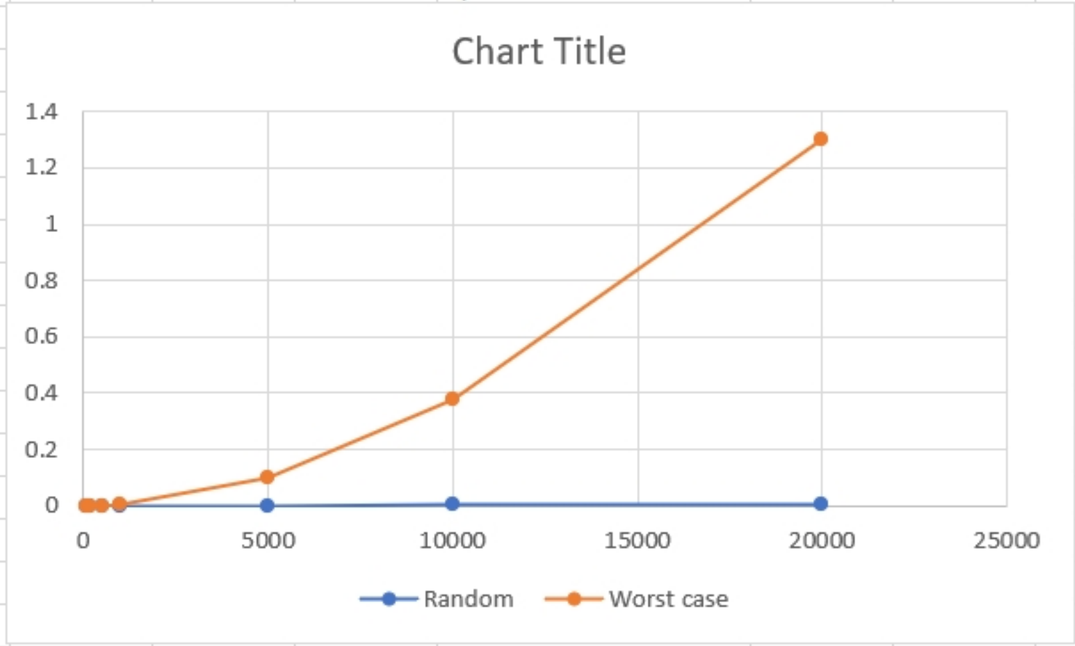
```c
//Quick Sort
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void swap(int *a,int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
int partition(int a[],int l,int r)
{
    int pivot=a[r];
    int pindex=l;
    for(int i=l;i<r;i++){
        if(a[i]<=pivot){
            swap(&a[pindex],&a[i]);
            pindex++;
        }
    }
    swap(&a[pindex],&a[r]);
    return pindex;
}
void quicksort(int a[],int l,int r)
{
    if(l<r)
    {
        int p=partition(a,l,r);
        quicksort(a,l,p-1);
        quicksort(a,p+1,r);
    }
}
```

```
797  int main()
798  {
799      int n,a[100];
800      printf("Enter the number of the elements in the array :- ");
801      scanf("%d",&n);
802      printf("Enter The Elements of the array :-\n");
803      for(int i=0;i<n;i++){
804          scanf("%d",&a[i]);
805      }
806      double q=0.0;
807      clock_t begin=clock();
808      quicksort(a,0,n-1);
809      clock_t end=clock();
810      q=((double)end-begin)/CLOCKS_PER_SEC;
811      printf("Elements of the sorted array :-\n");
812      for(int i=0;i<n;i++){
813          printf("%d ",a[i]);
814      }
815      printf("\n\nTime taken for %d elements = %f\n",n,q);
816      return 0;
817  }
818
819
```

```
Enter the number of the elements in the array :- 5
Enter The Elements of the array :-
12
45
7
8
65
Elements of the sorted array :-
7 8 12 45 65

Time taken for 5 elements = 0.000001


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

	Array Values	
Array Size	Random	Worst case
100	0.000032	0.000055
200	0.000023	0.00022
500	0.000074	0.001207
1000	0.000134	0.004812
5000	0.000927	0.096289
10000	0.001843	0.376258
20000	0.004324	1.299804

Quick Sort



```c
//Floyd
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
#define MAX 10000
int n;
void display(int d[][n]){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            if(d[i][j]==MAX)
                printf("%6s","INF");
            else
                printf ("%6d", d[i][j]);
        }
        printf("\n");
    }
}
void floyd(int graph[][n])
{
    int d[n][n], i, j, k;
    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            if(graph[i][j]==-1)
                d[i][j]=MAX;
            else
                d[i][j]=graph[i][j];
        }
    }
    printf("Matrix: D(0)\n");
    display(d);
    for(k=0;k<n;k++){
        for(i=0;i<n;i++){
            for(j=0;j<n;j++){
```

```
                if(d[i][j] > d[i][k]+d[k][j])
                    d[i][j]=d[i][k] + d[k][j];
            }
        }
        printf("\nD(%d) Solution Matrix : \n",k+1);
        display(d);
    }
}
int main()
{
    printf("\n Enter the Number of vertices :- ");
    scanf("%d",&n);
    int D[n][n];
    printf("\nEnter the adjacency matrix (enter -1 if no direct path) :- \n");
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            scanf("%d",&D[i][j]);
        }
    }
    floyd(D);
    return 0;
}
```

```
 Enter the Number of vertices :- 4

Enter the adjacency matrix (enter -1 if no direct path) :-
0 -1 3 -1
2 0 -1 -1
-1 7 0 1
6 -1 -1 0
Matrix: D(0)
     0   INF     3   INF
     2     0   INF   INF
   INF     7     0     1
     6   INF   INF     0

D(1) Solution Matrix :
     0   INF     3   INF
     2     0     5   INF
   INF     7     0     1
     6   INF     9     0

D(2) Solution Matrix :
     0   INF     3   INF
     2     0     5   INF
     9     7     0     1
     6   INF     9     0

D(3) Solution Matrix :
     0    10     3     4
     2     0     5     6
     9     7     0     1
     6    16     9     0

D(4) Solution Matrix :
     0    10     3     4
     2     0     5     6
     7     7     0     1
     6    16     9     0

Process returned 0 (0x0)   execution time : 81.630 s
Press any key to continue.
```

```c
//Warshall
#include<stdio.h>
#include<stdlib.h>
const int MAX = 100;
void Warshall(int graph[MAX][MAX],int n)
{
    int i,j,k;
    for (k=0; k<n; k++){
        for (i=0; i<n; i++){
            for (j=0; j<n; j++){
                if (graph[i][j] || (graph[i][k] && graph[k][j]))
                    graph[i][j] = 1;
            }
        }
    }
}
int main()
{
    int i,j,n;
    int graph[MAX][MAX];
    printf("Enter the number of vertices : ");
    scanf("%d",&n);
    printf("Enter the adjacency matrix :-\n");
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            scanf("%d",&graph[i][j]);
    Warshall(graph,n);
    printf("\nThe transitive closure for the given graph is :-\n");
    for (i=0; i<n; i++){
        for (j=0; j<n; j++)
            printf("%d\t",graph[i][j]);
        printf("\n");
    }
    return 0;
}
```

```
Enter the number of vertices : 4
Enter the adjacency matrix :-
0 0 1 0
0 0 0 1
1 0 0 0
0 1 0 0

The transitive closure for the given graph is :-
1       0       1       0
0       1       0       1
1       0       1       0
0       1       0       1

Process returned 0 (0x0)   execution time : 33.908 s
Press any key to continue.
```

```c
//Knapsack
#include <stdio.h>
#include<conio.h>
int max(int a,int b)
{
    return (a > b) ? a : b;
}
int knapSack(int W,int wt[],int val[],int n)
{
    int i, w;
    int K[n+1][W+1];
    for (i=0;i<=n;i++){
        for (w=0;w<=W;w++){
            if(i==0 || w==0)
                K[i][w]=0;
            else if(wt[i-1]<=w)
                K[i][w]=max(K[i-1][w],val[i-1]+K[i-1][w-wt[i-1]]);
            else
                K[i][w] = K[i-1][w];
        }
    }
    return K[n][W];
}
int main()
{
    int n,max;
```

```
1091  int main()
1092  {
1093      int n,max;
1094      printf("\nEnter the No of Items: ");
1095      scanf("%d",&n);
1096      int val[n],wt[n];
1097      printf("\nEnter the Weight and Profit of Items :- \n");
1098      for(int i=0;i<n;i++){
1099          printf("Weight of Item %d :- ",(i+1));
1100          scanf("%d",&wt[i]);
1101          printf("Value of Item %d :- ",(i+1));
1102          scanf("%d",&val[i]);
1103          printf("\n");
1104      }
1105      printf("\nEnter the Capacity :- ");
1106      scanf("%d",&max);
1107      printf("Maximum Profit :- %d",knapSack(max,wt,val,n));
1108      return 0;
1109  }
1110
1111
1112
1113
```

```
Enter the No of Items: 4

Enter the Weight and Profit of Items :-
Weight of Item 1 :- 5
Value of Item 1 :- 25

Weight of Item 2 :- 1
Value of Item 2 :- 32

Weight of Item 3 :- 9
Value of Item 3 :- 20

Weight of Item 4 :- 4
Value of Item 4 :- 30


Enter the Capacity :- 12
Maximum Profit :- 87
Process returned 0 (0x0)   execution time : 37.282 s
Press any key to continue.
```

```c
//Prims
#include<stdio.h>
#include<stdlib.h>
int a,b,u,t,m,n,i,j;
int v[10],c=1,min=0,cost[10][10];
int main(){
    printf("Enter the 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",&cost[i][j]);
            if(cost[i][j]==0)
                cost[i][j]=999;
        }
    }
    v[1]=1;
    while(c<n){
        for (i=1,m=999;i<=n;i++){
            for (j=1;j<=n;j++){
                if(cost[i][j]<m){
                    if(v[i]!=0){
                        m=cost[i][j];
                        a=u=i;
                        b=t=j;
                    }
```

```
1139                        b=t=j;
1140                    }
1141                }
1142            }
1143        }
1144        if(v[u]==0 || v[t]==0){
1145            printf("\nEdge %d:(%d %d) cost:%d",c++,a,b,m);
1146            min+=m;
1147            v[b]=1;
1148        }
1149        cost[a][b]=cost[b][a]=999;
1150    }
1151    printf("\nMinimun cost=%d\n",min);
1152    return 0;
1153 }
1154
1155
```

```
Enter the number of vertices :- 3

Enter the adjacency matrix:
0 2 5
2 0 0
5 0 0

Edge 1:(1 2) cost:2
Edge 2:(1 3) cost:5
Minimun cost=7

Process returned 0 (0x0)   execution time : 21.159 s
Press any key to continue.
```

```c
//Kruskal
#include<stdio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[10][10],parent[10];
int main()
{
    printf("Enter the no. 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",&cost[i][j]);
            if(cost[i][j]==0)
                cost[i][j]=999;
        }
    }
    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;
                }
            }
        }
        while(parent[u])
            u=parent[u];
```

```
1185            while(parent[u])
1186                u=parent[u];
1187            while(parent[v])
1188                v=parent[v];
1189            if(u!=v){
1190                printf("Edge %d:%d,%d) Cost:%d\n",ne++,a,b,min);
1191                mincost+=min;
1192                parent[v]=u;
1193            }
1194            cost[a][b]=cost[b][a]=999;
1195        }
1196        printf("\nMinimum cost = %d\n",mincost);
1197        return 0;
1198    }
1199
1200
1201
1202
1203
```

```
Enter the no. of vertices :- 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:(4,6) Cost:2
Edge 3:(1,2) Cost:3
Edge 4:(2,5) Cost:3
Edge 5:(3,6) Cost:4

Minimum cost = 13

Process returned 0 (0x0)   execution time : 48.654 s
Press any key to continue.
```

```
//Heap Sort
#include<stdio.h>
int temp;

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

void heapSort(int arr[], int size)
{
   int i;
   for (i = size / 2 - 1; i >= 0; i--)
      heapify(arr, size, i);
   for (i=size-1; i>=0; i--)
   {
      temp = arr[0];
      arr[0]= arr[i];
      arr[i] = temp;
      heapify(arr, i, 0);
   }
}

int main()
{
   int n;
   printf("Enter n :- ");
   scanf("%d",&n);
   int arr[n];
   printf("Enter elements :- \n");
   for(int i=0;i<n;i++){
      scanf("%d",&arr[i]);
   }
   int i;
   int size = sizeof(arr)/sizeof(arr[0]);
   heapSort(arr, size);
   printf("Sorted elements :- \n");
   for (i=0; i<size; ++i)
      printf("%d\n",arr[i]);
   return 0;
}
```

```
Enter n :- 4
Enter elements :-
12
84
32
44
Sorted elements :-
12
32
44
84


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

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

int minDistance(int dist[], int sptSet[],int V)
{
   int min = 999, min_index;
  int v;
   for ( v = 0; v < V; v++)
      if (sptSet[v] == 0 && dist[v] <= min)
         min = dist[v], min_index = v;

   return min_index;
}
int printSolution(int dist[],int V )
{ int i;
   printf("Vertex \t\t Distance from Source\n");
   for (i = 0; i < V; i++)
      printf("%d \t\t %d\n", i, dist[i]);
}
void dijkstra(int graph[10][10], int src,int V)
{
   int dist[V];
  int i,count,u,v;
   int sptSet[V];
   for ( i = 0; i < V; i++)
      dist[i] = 999, sptSet[i] = 0;

   dist[src] = 0;

   for ( count = 0; count < V - 1; count++) {

      u = minDistance(dist, sptSet,V);
      sptSet[u] = 1;

      for (v = 0; v < V; v++)

         if (!sptSet[v] && graph[u][v] && dist[u] != 999
            && dist[u] + graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
   }

   printSolution(dist,V);
}

int main()
{
   int i,j,V;

   int graph[10][10];
   printf("Enter number of vertices\n");
   scanf("%d",&V);
   printf("Enter adjacency matrix\n");
   for(i=0;i<V;i++)
   {
```

```
        for(j=0;j<V;j++)
    scanf("%d",&graph[i][j]);
    }

    dijkstra(graph, 0,V);

    return 0;
}
```

```
Enter number of vertices
5
Enter adjacency matrix
0 3 0 7 0
3 0 4 2 0
0 4 0 5 6
7 2 5 0 4
0 0 6 4 0
Vertex            Distance from Source
0                 0
1                 3
2                 7
3                 5
4                 9


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

```
//Sum of Subset
#include<stdio.h>
#include<conio.h>
#define TRUE 1
#define FALSE 0
int inc[50],w[50],sum,n;
int promising(int i,int wt,int total) {
 return(((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)));
}
void sumset(int i,int wt,int total) {
 int j;
 if(promising(i,wt,total)) {
  if(wt==sum) {
   printf("\n{\t");
   for (j=0;j<=i;j++)
       if(inc[j])
        printf("%d\t",w[j]);
   printf("}\n");
  } else {
   inc[i+1]=TRUE;
   sumset(i+1,wt+w[i+1],total-w[i+1]);
   inc[i+1]=FALSE;
   sumset(i+1,wt,total-w[i+1]);
  }
 }
}
int main() {
 int i,j,n,temp,total=0;
 printf("\n Enter how many numbers:\n");
 scanf("%d",&n);
 printf("\n Enter %d numbers to the set:\n",n);
 for (i=0;i<n;i++) {
  scanf("%d",&w[i]);
  total+=w[i];
 }
 printf("\n Input the sum value to create sub set:\n");
 scanf("%d",&sum);
 for (i=0;i<=n;i++)
   for (j=0;j<n-1;j++)
    if(w[j]>w[j+1]) {
  temp=w[j];
  w[j]=w[j+1];
  w[j+1]=temp;
 }
 printf("\n The given %d numbers in ascending order:\n",n);
 for (i=0;i<n;i++)
   printf("%d \t",w[i]);
 if((total<sum))
   printf("\n Subset construction is not possible"); else {
  for (i=0;i<n;i++)
     inc[i]=0;
  printf("\n The solution using backtracking is:\n");
  sumset(-1,0,total);
 }
```

```
 return 0;
}
```

```
 Enter how many numbers:
3

 Enter 3 numbers to the set:
4
12
84

 Input the sum value to create sub set:
16

 The given 3 numbers in ascending order:
4        12       84
 The solution using backtracking is:

{        4        12       }


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

```c
//n-queen
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int a[50],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) {
```

```
    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--;
    }
}
int main(){
    int n;
    printf("Enter the number of Queens :- ");
    scanf("%d",&n);
    queen(n);
    printf("\nTotal solutions=%d",count);
    return 0;
}
```

```
Enter the number of Queens :- 4


Solution 1 :
-       Q       -       -
-       -       -       Q
Q       -       -       -
-       -       Q       -


Solution 2 :
-       -       Q       -
Q       -       -       -
-       -       -       Q
-       Q       -       -

Total solutions=2
Process returned 0 (0x0)   execution time : 1.903 s
Press any key to continue.
```