1)GCD

```
#include<stdio.h>
#include<stdlib.h>
int euclid(int m,int n)
{
int r,count=0;
while(n!=0)
{
count++;
r=m%n;
if(r==0)
{  break;  }
m=n;
n=r;
}
return count;
}
int consec(int m,int n)
{
int min,count=0;
min=m;
if(n<min)
{   min=n;  }
for(;;)
{
count++;
if(m%min==0)
{
count++;
if(n%min==0)
{  break;   }
min=min-1;
}
else
min=min-1;
}
return count;
}
int modifiedeuclid(int m,int n)
{
int temp,count=0;
while(n>0)
{
if(n>m)
{
temp=m;
m=n;
n=temp;
}
m=m-n;
count=count+1;
}
return count;
}
void analysis(int ch)
{
int m,n,i,j,k;
float mincount,maxcount,count;
FILE *fp1,*fp2;
for(i=10;i<10000;i*=10)
{
maxcount=0;
mincount=10000;
for(j=2;j<=i;j++)
{
for(k=2;k<=i;k++)
```

```
{
count=0;
m=j;
n=k;
switch(ch)
{
case 1:
count=euclid(m,n);
break;
case 2:
count=consec(m,n);
break;
case 3:
count=modifiedeuclid(m,n);
break;
}
if(count>maxcount)
{   maxcount=count;   }
if(count<mincount)
{   mincount=count;   }}}
switch(ch)
{
case 1:
fp2=fopen("ebest.txt","a");
fp1=fopen("eworst.txt","a");
break;
case 2:
fp2=fopen("cbest.txt","a");
fp1=fopen("cworst.txt","a");
break;
case 3:
fp2=fopen("mbest.txt","a");
fp1=fopen("mworst.txt","a");
break;
}
fprintf(fp2,"%d
%.2f\n",i,mincount);
fclose(fp2);
fprintf(fp1,"%d
%.2f\n",i,maxcount);
fclose(fp1);
}}
int main()
{
int choice;
for(;;)
{
printf("1.euclids\n2.con
\n3.modified \n");
scanf("%d",&choice);
switch(choice)
{
case 1:
case 2:
case 3:analysis(choice);
break;
default:system("gnuplot>load
'gcdplot.txt' ");exit(0);
}}}
plot
Set title ‘gcd’
Set xrange[10:1000]
Set yrange[0:1020]
Set xlabel ‘Input size’
Set ylabel ‘operation count’
Set style data linespoints
```

Plot ‘ebest.txt’ title ‘ebest case’,’cbest.txt’ title ‘cbest case’, ‘mbest.txt’ title ‘mbest case’,’eworst.txt’ title ‘eworst case’,’cworst.txt’ title ‘cworst case’, ‘mworst.txt’ title ‘mworst case’

Pause -1 ‘press any keyy to continue’

Analysis euclid

Input size: Two positive integers

Basic operation: Division

Best case: C(n)∈ θ(1) Constant

Worst case: C(n)∈ θ(logn) Logarithmic

Consecutive integer method

Basic operation: Division

Best caseC(n)∈ θ(1)Constant

Worst caseC(n)∈ θ(n) Linear

Modified Euclid

Basic operation: Subtraction

Best caseC(n)∈ θ(1)Constant

Worst caseC(n)∈ θ(n)Linear

2)a)linear search

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define n1 10
#define n2 100000
void search(int * ar,int n,int key)
{
  for(int i=0 ; i<n ; i++)
   {
     if( key == ar[i] )
      {  return ; }
   }}
void main()
{
  clock_t st,et;
  double ts;
  FILE *fp2,*fp1,*fp3;
  int i,n,* ar,key;
  srand(time(NULL));
 for( n=n1 ; n<=n2 ; n*=100)
  {
   fp2 =
fopen("worstlinear.txt","a");
fp1=fopen("bestlinear.txt","a");
ar = (int *)malloc(n*sizeof(int));
  for(i=0 ; i<n-1 ; i++)
{   ar[i] = rand()%1000 ;
}
  ar[n-1] = -9999;
  ar[0]=-1;
  key = -9999;
  st = clock();
  search(ar,n,key);
  et = clock();
  ts = (double)(et-
st)/CLOCKS_PER_SEC;
  fprintf(fp2,"%d %lf\n",n,ts);
  fclose(fp2);
   key=-1;
  st=clock();
  search(ar,n,key);
```

```
  et=clock();
  ts=(double)(et-
st)/CLOCKS_PER_SEC;
  fprintf(fp1,"%d %lf\n",n,ts);
  fclose(fp1);
 }
system("gnuplot>load
'commandlinear.txt'");
}
plot
set title 'Linear Search'
set xrange[10:100000]
set yrange[0:0.0004]
set xlabel 'Input size(n)'
set ylabel 'Operation Count'
set style data linespoints
plot 'worstlinear.txt' title 'Worst
case', 'bestlinear.txt' title 'Best
Case'
pause -1 'Hit any key to confirm'
```

Analysis

Input size: array of size n
Basic operation: Comparison
Best case-key is in the first position: C(n)∈ θ(1)
Worst case– key is in the last position Efficiency: C(n)∈ θ(n)

2)b)binary search recursive

```
#include<stdio.h>
#include<stdlib.h>
int binarysearch(int *ar,int key,int
low,int high)
{
int count=0;
int mid=(high+low)/2;
if(ar[mid]==key)
{  return 1;}
else if(ar[mid]<key)
{  low=mid+1;}
else
{  high=mid-1;  }
return
1+binarysearch(ar,key,low,high);
}
void main()
{
FILE *fp1,*fp2,*fp3;
fp1=fopen("bestbinaryrecursive.tx
t","a");
fp2=fopen("worstbinaryrecursive.
txt","a");
int *ar,key,i,avcount;
for(int n=10;n<=100000;n*=10)
{
ar=(int*)malloc(sizeof(int)*n);
for(i=0;i<n;i++)
{  ar[i]=i;}
key=ar[(n-1)/2];
fprintf(fp1,"%d
%d\n",n,binarysearch(ar,key,0,n-
1));
key=ar[n-1];
fprintf(fp2,"%d
%d\n",n,binarysearch(ar,key,0,n-
1));
```

```
}
fclose(fp1);
fclose(fp2);
system("gnuplot>load
commandbinaryrecursive.txt");
}plot
set title 'Binary Search Recursive'
set xrange[10:100000]
set yrange[0:20]
set xlabel 'Input size(n)'
set ylabel 'Operation Count'
set style data linespoints
plot 'worstbinaryrecursive.txt'
title 'Worst case',
'bestbinaryrecursive.txt' title 'Best
Case',
pause -1 'Hit any key to confirm'
```

Analysis

Input: Array of n elements

Basic Operation: Comparison

Best case: key is the middle element $C(n) \in \theta(1)$

Successful Worst case: key is at first or last element

Unsuccessful Worst case: When the key is greater than the last element or lesser than the first element $C(n) \in \theta(\log n)$

3)a)selection sort

```
#include<stdio.h>
#include<stdlib.h>
int selectionsort(int *ar,int n)
{
int cnt=0,min,temp;
for(int i=0;i<n-1;i++)
{
min = i;
for(int j=i+1;j<n;j++)
{
cnt++;
if(ar[j]<ar[min])
{
temp = ar[j];
ar[j] = ar[min];
ar[min] = temp;
}}}
return cnt;
}
int main()
{
FILE*fp = fopen("select.txt","a");
int *ar;
for(int n=10;n<=200;n+=10)
{
ar = (int *)malloc(sizeof(int)*n);
for(int i=0;i<n;i++)
ar[i] = n-i;
fprintf(fp,"%d\t
%d\n",n,selectionsort(ar,n));
free(ar);
}
fclose(fp);
system("gnuplot>load
selectcommand.txt");
return 0;
```

```
}plot
set title 'Selection Sort'
set xrange[0:200]
set yrange[0:20000]
set xlabel 'n'
set ylabel 'count'
set style data linespoints
plot 'select.txt' title 'General
CASE'
pause -1 ' any key to continue'
```

Analysis

Input: Array of n elements

Basic Operation: Comparison

There is no best case or worst case $C(n) \in \theta(n2)$

3)b)bubble sort

```
#include<stdio.h>
#include<stdlib.h>
#include <time.h>
int bubblesort(int *ar,int n)
{
  int cnt=0,flag=0,temp;
 for(int i=0 ; i<n-1 ; i++)
 {
  for(int j=0; j<n-i-1; j++)
  {
    cnt++;
    if( ar[j+1] < ar[j] )
    {
      flag=1;
      temp = ar[j+1];
      ar[j+1] = ar[j];
      ar[j] = temp;
    }  }
   if(flag == 0)
      return cnt;
  }
  return cnt;
}
void main()
{
 int * ar,i;
 FILE * fp1,* fp2;
 fp1 =
fopen("bestbubble.txt","a");
 fp2 =
fopen("worstbubble.txt","a");
 srand(time(NULL));
 for( int n=10; n<=100; n+=10)
 {
  ar = (int *)malloc(n*sizeof(int));
  for( i=0 ; i<n ; i++)
   ar[i] = i;
  fprintf(fp1,"%d
%d\n",n,bubblesort(ar,n));
   for( i=0 ; i<n ; i++)
    ar[i] = n-i;
  fprintf(fp2,"%d
%d\n",n,bubblesort(ar,n));
  free(ar);
 }
 fclose(fp1);
 fclose(fp2);
  system("gnuplot>load
commandbubble.txt");
```

```
}plot
set title 'Bubble '
set xrange[0:200]
set yrange[0:20000]
set xlabel 'n'
set ylabel 'count'
set style data linespoints
plot 'bestbubble.txt' title 'best
case','worstbubble.txt' title 'worst
case'
pause -1 'hit any key to continue'
```

Analysis

Input: Array of n elements

Basic Operation: Comparison

Best Case: Already sorted array

C(n)∈ θ(n)

Worst case: Unsorted array C(n)∈ θ(n2)

3)c)insertion sort

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int insertionsort(int * ar, int n)
{
 int j,i,temp,cnt=0;
 for( int i=1 ; i<n ; i++)
 {
  temp = ar[i];
  j = i-1;
  while(j>=0)
  {
   cnt++;
   if( ar[j] < temp )
      break;
    ar[j+1] = ar[j--];
  }
  ar[j+1] = temp;
 }
 return cnt;}
void main()
{
 FILE *fp1,*fp2,*fp3;
 int *ar,i;
 srand(time(NULL));
 fp1 = fopen("besti.txt","w");
 fp2 = fopen("worsti.txt","w");
 for( int n=10; n<=200 ; n+=10 )
 {
  ar = (int *)malloc(sizeof(int)*n);
  for( i=0 ; i<n ; i++)
     ar[i] = i+1;
  fprintf(fp1,"%d
%d\n",n,insertionsort(ar,n));
  for( i=0 ; i<n ;i++ )
     ar[i] = n-i;
  fprintf(fp2,"%d
%d\n",n,insertionsort(ar,n));
  free(ar);  }
 fclose(fp1);
 fclose(fp2);
 system("gnuplot>load
insertplot.txt");
}plot
set title 'Insertion Sort'
set xrange[0:200]
set yrange[0:20000]
```

```
set xlabel 'n'
set ylabel 'count'
set style data linespoints
plot 'besti.txt' title 'best
case','worsti.txt' title 'worst case',
 pause -1 'hit any key to continue'
```

Analysis

Input: Array of n elements Basic operation: Comparison

Best case: Previously sorted array $C(n) \in \theta(n)$

Worst case: Unsorted array $C(n) \in \theta(n2)$

4)string matching

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<time.h>
int stringmatch (char * txt,char *
pattern,int n,int m)
 {
  int i,j;
  int count;
    for(i=0;i<=n-m;i++)
      {   j=0;
while(j <m && txt[i+j] ==
pattern[j])
        {
         j++;
         count++; }
       if(j==m)
        return count;
       }
     return count;
  }
void main()
 {
   int i,j,m,n=1000;
   char *txt,*pattern;
   FILE *fp1,*fp2;
   srand(time(NULL));
txt=(char*)malloc(sizeof(char)*(n+
1));
           for(j=0;j<n;j++)
             txt[j]='a';
             txt[n]='\0';
for(m=100;m<=1000;m+=100)
         {
pattern=(char*)malloc(sizeof(char
)*(m+1));
for(j=0;j<m;j++)
pattern[j]='a';
pattern[m]='\0';
fp1=fopen("beststring.txt","a");
fprintf(fp1,"%d
%d\n",m,stringmatch(txt,pattern,
n,m));
fclose(fp1);
fp2=fopen("worststring.txt","a");
pattern[m-1]='b';
fprintf(fp2,"%d
%d\n",m,stringmatch(txt,pattern,
n,m));
fclose(fp2);
       free(txt);
```

```
system("gnuplot > load
'stringplot.txt'");  }plot
set title 'String Matching'
set xrange[100:1000]
set yrange[0:249999]
set xlabel 'Input size(n)'
set ylabel 'Operation Count'
set style data linespoints
plot 'worststring.txt' title 'Worst
case', 'beststring.txt' title 'Best
Case',
pause -1 'Hit any key to confirm'
```

Analysis

Input: String of length 'n' and pattern of length 'm'

Basic operation: Character pair comparison

Best case: Search pattern is in the first alignment C(n)∈ θ(m)

Worst case: Search pattern is in last alignment or not found C(n)∈ θ(mn)

5)a)merge sort

```
#include<stdio.h>
#include<stdlib.h>
void mergedata(int *arr1,int p,int
*arr2,int q)
{
  int i,j=0;
  for(i=p;i<q;i++)
  {
  arr1[i]=arr2[j++];
  }}
void datagenerator(int *arr,int n)
{
 if(n==1)   return;
 int i=0,j=(n/2),k=0;
 int dup=(int)malloc(sizeof(int)*n);
  for(int x=0 ;x<n ;x++)
    (dup+x)=(arr+x);
  while(i<(n/2) && (j<n) )
  {
   arr[i]=dup[k++];
   i++;
   arr[j]=dup[k++];
   j++;
  }
int *arr1=arr,*arr2=(arr+(n/2));
int p=(n/2),q=(n-n/2);
datagenerator(arr1,p);
datagenerator(arr2,q);
mergedata(arr1,p,arr2,q);
}
int mergesort(int *arr1,int p,int
*arr2,int q)
{
  int i=0,j=0,k=0,count=0,temp;
  int
dup=(int)malloc(sizeof(int)*(p+q))
;
  while((i<p) && (j<q))
  {
   count++;
   if((arr1+i)<(arr2+j))
   {
    (dup+k)=(arr1+i);
```

```
  i++;    k++;
  }
 else
 {
  (dup+k)=(arr2+j);
  j++;   k++;
 } }
 while(i<p)
 {
 (dup+k)=(arr1+i);
 i++;  k++;
 }
 while(j<q)
{
 (dup+k)=(arr2+j);
 j++;    k++;
}
 for(k=0;k<(p+q);k++)
  (arr1+k)=(dup+k);
 return count;
}
int dividemerge(int *arr,int n)
{
if(n==1)   return 0;
if(n!=1)
{
int *arr1=arr,*arr2=(arr+(n/2));
int p=(n/2),q=n-
n/2,cnt1,cnt2,count=0;
cnt1=dividemerge(arr1,p);
cnt2=dividemerge(arr2,q);
count=cnt1+cnt2+mergesort(arr1,
p,arr2,q);
return count;
}}
void main()
{
FILE *fp1,*fp2;
fp1=fopen("bestms.txt","w");
fp2=fopen("worstms.txt","w");
int *arr,i;
for(int n=2;n<=1024;n*=2)
{
arr=(int*)malloc(sizeof(int)*n);
for(i=0;i<n;i++)
{  arr[i]=i+1; }
fprintf(fp1,"%d
%d\n",n,dividemerge(arr,n));
datagenerator(arr,n);
fprintf(fp2,"%d
%d\n",n,dividemerge(arr,n));
}
fclose(fp1);
fclose(fp2);
system("gnuplot>load
mergeplot.txt");
}plot
set title 'Merge Sort'
set xrange[0:1024]
set yrange[0:10000]
set xlabel 'n'
set ylabel 'count'
set style data linespoints
plot 'bestms.txt' title 'Best
case','worstms.txt' title 'worst
case'
```

pause -1 'hit any key to continue'

Analysis

Input: Array of n elements

Basic Operation: Comparison

Best case: ascending or descending

C(n)=2 log2 -→nlogn

Worst case: Each element is coming from the alternate array

C(n)=nlogn−n+1→: nlogn

6)a)quick sort

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int count;
void swap( int * a,int * b){
  int temp=*a;
  *a = *b;
  *b = temp;}
int partition(int *ar,int l,int r)
{
   int pivot=l,b=l,e=r+1;
  do{
do{
     b++;     count++;
   }while( ar[b]<ar[pivot] );
   do  {    e--;     count++;
   }while( ar[e]>ar[pivot] );
    swap((ar+b),(ar+e));
   }while( b<e );
    swap((ar+b),(ar+e));
   swap((ar+pivot),(ar+e));
   return e;
}
int quicksort(int * ar, int l,int r)
{
  if( l<r )
  {
    int s = partition(ar,l,r);
    quicksort(ar,l,s-1);
    quicksort(ar,s+1,r);
  }}
void main()
{
  FILE *fp1,*fp2,*fp3;
  int *ar,i;
  srand(time(NULL));
  fp1 = fopen("bestqs.txt","w");
  fp2 = fopen("worstqs.txt","w");
  for( int n=10; n<=200 ; n+=10 )
  {
    ar = (int *)malloc(sizeof(int)*n);
     count = 0;
    for( i=0 ; i<n ; i++)
       ar[i] = 10;
    quicksort(ar,0,n-1);
    fprintf(fp1,"%d %d\n",n,count);
    count = 0;
    for( i=0 ; i<n ;i++ )
       ar[i] = i+1;
    quicksort(ar,0,n-1);
    fprintf(fp2,"%d %d\n",n,count);
    free(ar);
  }
  fclose(fp1);
  fclose(fp2);
```

```
  system("gnuplot>load
quickplot.txt");
}plot
set title 'Quick Sort'
set xrange[0:200]
set yrange[0:30000]
set xlabel 'n'
set ylabel 'count'
set style data linespoints
plot 'bestqs.txt' title 'best
case','worstqs.txt' title 'worst
case',
 pause -1 'hit any key to continue'
```

Analysis

Input: Array with n elements

Basic Operation: Comparison

Best case: It occurs when the partition process always picks middle element as pivot C(n)= nlogn

Worst case: When the partition process always picks greatest or smallest as pivotN2

7)DFS

```
#include <stdio.h>

Int graph[40][40], n,
visited[40]={0}, acyclic =1;

Void createGraph()

{

Printf("No. Of vertices>> ");

Scanf("%d", &n);

Printf("Enter adjacency
matrix:\n");

For(int i=0;i<n;i++){

For(int j=0;j<n;j++){

Scanf("%d",&graph[i][j]);

      }}}

Void dfs(int v){

Visited[v]=1;

For(int i=0;i<n;i++){

If (graph[v][i] && !visited[i]){

      Printf("%d→",v);

            Dfs(i);

            }

If (graph[v][i] && visited[i]){

            Acyclic=0;

            }}}

Void main()

{

Int i,count=0;

      createGraph();

      dfs(0);

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

      if (visited[i])

      count++;
```

```
    }
 (count==n) ? printf("\nConnected Graph\n") : printf("\nGraph not connected!\n");
 (acyclic) ? printf("Acyclic Graph\n") : printf("Graph not acyclic!\n");
    If (count!=n)
{
For(i=0;i<n;i++) visited[i]=0;
    For(i=0;i<n;i++){
    Dfs(i);
    Printf("\n");
    }   }}
```

Analysis
Input: Adjacency matrix
Basic Operation: Testing for Adjacency T(n)∈ O(v2)

8)BFS

```
#include <stdio.h>
Int graph[40][40], n, visited[40]={0}, acyclic =1, f=0, r=-1, q[40];
Void createGraph(){
Printf("No. Of vertices>> ");
Scanf("%d", &n);
Printf("Enter adjacency matrix:\n");
For(int i=0;i<n;i++)  {
For(int j=0;j<n;j++)  {
Scanf("%d",&graph[i][j]);
    }   }}
Void bfs(int v)
{
    Visited[v]=1;
    For(int i=0;i<n;i++){
    If (graph[v][i] && !visited[i]){
    Printf("%d→",v);
    Q[++r] = i;
    }
    If (graph[v][i] && visited[i]){
    Acyclic=0;
        }
    If(f<=r) {
    Visited[q[f]]=1;
    Bfs(q[f++]);
    }}}
Void main()
{
Int i,count=0;
```

```
createGraph();
        bfs(0);
        for(i=0;i<n;i++){
        if (visited[i])
        count++;
        }
 (count==n) ? printf("\nConnected Graph\n") : printf("\nGraph not connected!\n");
 (acyclic) ? printf("Acyclic Graph\n") : printf("Graph not acyclic!\n");
        If (count!=n)    {
For(i=0;i<n;i++) visited[i]=0;
        For(i=0;i<n;i++){
        Bfs(i);
        Printf("\n");
        }      }}
```

9)DFS TOPO

```
#include <stdio.h>
#include <stdlib.h>
 int n,stk[20],tos=-1;
void dfs(int graph[n][n],int cur,int *vis)
{
 vis[cur]=1;
 for(int next=0;next<n;++next)
{
if(graph[cur][next]&&!vis[next])
dfs(graph,next,vis);
}  stk[++tos] = cur;
}
int main()
{
 printf("enter number of verticies:");
 scanf("%d",&n);
 int graph[n][n];
 printf("enter adjacency matrix of DAG:\n");
 for(int i=0;i<n;i++)
 for(int j=0;j<n;j++)
 scanf("%d",&graph[i][j]);
 int vis = (int)calloc(n,sizeof(int));
 for(int i=0;i<n;i++)
  {
    if(!vis[i])
    dfs(graph,i,vis);
   }
    printf("Topological sorting:");
    for(int i=tos;i>-1;--i)
    printf("-->%c",stk[i]+65);
      free(vis);      return 0;
  }
```

10)source removal method

```
#include <stdio.h>
int graph[40][40], n,
visited[40]={0}, indegree[40]={0};
void createGraph(){
```

```
printf("No. of vertices>> ");
scanf("%d", &n);
printf("Enter adjacency
matrix:\n");
for(int i=0;i<n;i++){
for(int j=0;j<n;j++){
scanf("%d",&graph[i][j]);
}}}
void main()
{
int i,j,count=0;
createGraph();
for(i=0;i<n;i++){
for(j=0;j<n;j++){
if (graph[j][i]) indegree[i]++;
}}
printf("Topologically Sorted
Order:\n");
while(count<n){
for(i=0;i<n;i++){
if (!visited[i] && !indegree[i]){
printf("%d-->",i);
visited[i]=1;
for(j=0;j<n;j++){
if (graph[i][j]){
graph[i][j]=0;
indegree[j]--;
}}
count++;   break;
}}}
printf("\n");
}
```

11) heap

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
Int count;
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;
      Count++;
      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 main(){
    Int n,i,*arr;
    FILE *b,*w,*a;
System("rm a.txt b.txt w.txt");
    B = fopen("b.txt","a");
    W = fopen("w.txt","a");
    For(n=10;n<1000;n+=10){
    Arr =
(int*)malloc(n*sizeof(int));
        Count=0;
    For(i=0;i<n;i++){
    Arr[i] = 1;
        }
    heapSort(arr,n);
fprintf(b,"%d\t%d\n",n,count);
        Count=0;
    For(i=0;i<n;i++){
Arr[i] = (i==0) ? rand()%100 : arr[i-
1]+rand()%10;
        }
    heapSort(arr,n);
    fprintf(w,"%d\t%d\n",n,count
);
    Fclose(b); fclose(a); fclose(w);
}plot
Set title "Heap Sort Analysis"
Set xlabel "Input Size"
Set ylabel "Operation Count"
Set style data line
Set xrange [10:100]
Set yrange [10:1000]
Plot "b.txt" using 1:2 title "Best
Case "w.txt" using 1:2 title "Worst
Case"
```

Analysis

Input: Array of n elements

Basic Operation: Comparison

Best case: $C(n) \in \theta(n)$

Worst case: $C(n) \in \theta(n)$

Deletion: $C(n) \in \theta(n \log n)$

12)a)warshell

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
 int v;
 printf("\nEnter number of
verties: ");
 scanf("%d",&v);
 int c[v][v],j,k,i;
 printf("\nEnter adjacency matrix:
");
 for(i=0;i<v;i++)
 for(j=0;j<v;j++)
 scanf("%d",&c[i][j]);
 for(k=0;k<v;k++)
 for(i=0;i<v;i++)
 for(j=0;j<v;j++)
 if(c[i][j]||c[i][k]&&c[k][j])
 c[i][j]=1;
 printf("\nTransitive Closure
Matrix is\n");
 for(i=0;i<v;i++)
 {
```

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

12)b)Floyd

```
#include<stdio.h>
#include<stdlib.h>
int Min(int a,int b)
{
if(a<b)
 return a;
 return b;
}
void main()
{
 int v;
 printf("Enter the number of
Vertices: ");
 scanf("%d",&v);
 int c[v][v],d[v][v],i,j,k;
 printf("\nEnter the Cost
Matrix:\n ");
 for(i=0;i<v;i++)
 {
 for(j=0;j<v;j++)
 {
 scanf("%d",&c[i][j]);
 d[i][j]=c[i][j];
 }}
 for(k=0;k<v;k++)
 {
 for(i=0;i<v;i++)
 {
 for(j=0;j<v;j++)
 {
 d[i][j]=Min(d[i][j],d[i][k]+d[k][j]);
 } } }
 printf("\nThe Shortest distance
Matrix:\n");
 for(i=0;i<v;i++)
 {
 for(j=0;j<v;j++)
 printf("%d\t",d[i][j]);
 printf("\n");
 }}
```

13)a) knapsack

```
 #include <stdio.h>
#include <stdlib.h>
int max(int a, int b){
return (a>b) ? a : b;
}
void main(){
int t[100][100], v[100], w[100], n,
m, i, j;
printf("No. of Items>> ");
scanf("%d",&n);
printf("Sack Capacity>> ");
scanf("%d",&m);
printf("Weight\tValue\n");
for(i=1;i<n+1;i++){
scanf("%d\t%d",&w[i],&v[i]);
}
for(i=0;i<n+1;i++){
for(j=0;j<m+1;j++){
if (i==0||j==0)
t[i][j] = 0;
```

```
else if (j<w[i])
t[i][j] = t[i-1][j];
else
t[i][j] = max(t[i-1][j], v[i]+t[i-1][j-
w[i]]);
}}
printf("Maximum Value:
%d\n",t[n][m]);
printf("Composition:\n");
for(i=n;i>0;i--){
if (t[i][m] != t[i-1][m]){
printf("%d ",i);
m = m-w[i];
}}
printf("\n");
}
```

13)b)memory function knapsack

```
#include <stdio.h>
#include <stdlib.h>
int max(int a, int b){
return (a>b) ? a : b;
}
int t[100][100], v[100], w[100], n,
m, i, j;
int knap(int i, int j){
if (t[i][j]==-1){
if (j<w[i])
t[i][j] = knap(i-1,j);
else
t[i][j] = max(knap(i-
1,j),v[i]+knap(i-1,j-w[i]));
}
return t[i][j];
}
void main(){
printf("No. of Items>> ");
scanf("%d",&n);
printf("Sack Capacity>> ");
scanf("%d",&m);
printf("Weight\tValue\n");
for(i=1;i<n+1;i++){
scanf("%d\t%d",&w[i],&v[i]);
}
for(i=0;i<n+1;i++){
for(j=0;j<m+1;j++){
if (i==0||j==0)
t[i][j]=0;
else
t[i][j]=-1;
}}
printf("Maximum Value:
%d\n",knap(n,m));
printf("Composition:\n");
for(i=n;i>0;i--){
if (t[i][m] != t[i-1][m]){
printf("%d ",i);
m = m-w[i];
}}
printf("\n");
}
```

14)prims

```
#include <stdio.h>
int cost[40][40], n, visited[40]={0};
void createGraph()
{
printf("No. of vertices>> ");
```

```
scanf("%d", &n);
printf("Enter cost matrix:\n");
for(int i=0;i<n;i++){
for(int j=0;j<n;j++){
scanf("%d",&cost[i][j]);
}}}
void main()
{
int i,j,edges=0;
int a,b,min,min_cost = 0;
createGraph();
visited[0]=1;
while(edges<n-1){
min = 9999;
for(i=0;i<n;i++){
if(visited[i]){
for(j=0;j<n;j++){
if (cost[i][j] && min>cost[i][j] &&
!visited[j]){
min = cost[i][j];
a = i;  b = j;
}}}}
printf("%d-->%d | Cost:
%d\n",a,b,min);
visited[b] = 1;
min_cost += min;
edges++;
}
printf("Minimum Cost:
%d\n",min_cost);
}
```

15)dijktstra

```
#include<stdio.h>
#include<stdlib.h>
int v;
int select_min_vertex(int
selectted[],int value[])
{
 int min=999,vertex;
 for(int i=0;i<v;i++){
if(selectted[i]==0&&value[i]<=min
)
 {
 min=value[i];
 vertex=i;
 } }
 return vertex;
}
void main()
{
 printf("\nEnter the number of
vertices: ");
 scanf("%d",&v);
 int i,j,u,G[v][v],value[v];
 int selected[v];
 printf("\nEnter the adjacency
matrix:\n ");
 for(i=0;i<v;i++)
 for(j=0;j<v;j++)
 scanf("%d",&G[i][j]);
 for(i=0;i<v;i++)
 selected[i]=0,value[i]=999;
 for(i=0;i<v-1;i++)
 {
u=select_min_vertex(selected,val
ue);
```

```
 selected[u]=1;
 for(j=0;j<v;j++)
if(selected[j]==0&&G[i][j]&&value
[u]!=999&&value[u]+G[u][j]<valu
e[j])
 value[j]=value[u]+G[i][j];
 }
 printf("\nVertex\tDistance from
source\n");
 for(i=0;i<v;i++)
 printf("%d\t%d\n",i,value[i]);
}
```