Recursive GCD

```
#include <stdio.h>
int gcd (int x, int y);
int main()
{
    int x, y
    printf ("Enter the first number \n");
    scanf ("%d", &x);
    printf ("Enter second number \n")
    scanf ("%d", &y);
    int ans = gcd(x, y)
    printf ("GCD = %d", ans);
}
int gcd (int x, int y)
{
    int rem,
    int temp
    if (x>y)
    {
        rem = x % y;
    }
    else
    {
        temp = y
        y = x
        x = temp
        rem = x % y
    }
    if (rem==0)
    {
    return y;
    }
```

```
else
{
  return gcd(y, rem);
}
}
```

Linear GCD

```
#include<stdio.h>
int gcd(int x, int y)
int main()
{
   int x, y
   printf("first number\n");
   Scanf("%d", &x);
   print("second number\n");
   Scanf("%d", &y);
   int ans= gcd(x, y)
   printf("GCD is %d\n", ans);
}
int gcd(int x, int y)
{
   int rem, temp
   if (x>y)
   {
     rem = x % y;
   }
   else
   {
     temp = x
     x = y
     y = temp
     rem = x % y;
   }
```

```
if (rem == 0)
    return y
}
else
{
    while (rem != 0)
    {
        x = y
        y = rem
        rem = x % y
    }
    return y
}
```

Linear & Binary Searched

```
#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, m+1, last, key);
   }
   return -1;
 }
   int main()
   {
    int a[200], i, choice, key, n, res;
  Printf(" Enter the size of the array :");
  Scanf("%d", &n);
  Printf(" Enter the values of array in ascending order:");
```

```
for(i=0 ; i<n ; i++)
   scanf("%d", &a[i]);
}
for(;;){
 printf("Enter the 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 ", 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);
   }
 default : exit(0);
   }
   }
   }
```

Bubble Sort and Selection Sort

```
#include <stdio.h>

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

void bubbleSort (int arr[], int n)
{
    int i, j
    for (i=0; i<n-1 ; i++)
        for (j=0; j<n-i-1 ; j++)
            if (arr[j] > arr[j+1])
                swap (&arr[j], &arr[j+1])

}
void selectionSort (int arr[], int n)
{
    int i, j, min_idx
    for (i=0; i<n-1 ; i++)
    {
        min_idx = i
        for (j=i+1; j<n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;
```

```
        swap(&arr[min_idx], &arr[i]);
    }
}

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

int main()
{
    int n, option;
    printf("Enter the size of the array\n");
    scanf("%d", &n);
    int arr[n];
    printf("Enter element\n");
    scanf("%d", &a[i]);

    do {
        printf("1: Bubble Sort\n");
        printf("2: Selection Sort\n");
        printf("3: exit\n");
        printf("Enter Your option\n");
        scanf("%d", &option);
```

```
switch (option)
{
    case 1:
        bubbleSort(arr, n);
        printf("Sorted array:\n");
        printArray(arr);
        break;

    case 2:
        selectionSort(arr, n);
        printf("Sorted array:\n");
        printArray(arr);
        break;

}
} while (option != 3);

}
```

Depth First Search

```
#include<stdio.h>
#include<time.h>
void dfs(int);
int G[10][10], visited[10], n;
void main()
{
    int i, j;
    clock_t start, end;
    printf("Enter number of vertices");
    scanf("%d", &n);
    printf("Enter adjacency matrix\n");
    for(i=0; i<n; i++)
    for(j=0; j<n; j++)
            scanf("%d", &G[i][j])

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

    start=clock()
    dfs(0)
    end=clock()
    printf("\n Time taken is %f\n", (double)(end-start
                                    /(CLOCKS_PE
                                    -SEC));
}
```

```
void dfs(int i)
{
    int j
    printf("\n%d", i);
    visited[i] = 1
    for(j=0; j<n; j++)
        if(!visited[j] && G[i][j] == 1)
            Dfs(j)
}
```

Tower of Hanoi:

```
#include<stdio.h>
#include<time.h>
void toh(char src, char dest, char sp, int n)
int main()
{
    clock_t start, end;
    int a[3] ≠ {3,3}
    int n;
    printf("Enter the number of Vertices\n");
    scanf("%d", &n);
    start = clock();
    toh('A', 'C', 'B', n);
    end = clock()
    printf("Time taken %lf\n", (float)(end-start)
                                / CLOCKS_PER_SEC);
}
```

```
Void toh ( char src, char dest, char sp, int n)
{
    if (n==1)
    {
      printf ("Move from %c to %c\n", src, dest);
      return
    }
  else
  {
    toh (src, sp, dest, n-1);
    toh ( src, dest, sp, 1);
    toh (sp, dest, src, n-1));
  }
}
```

BFS Traversal

```
#include<stdio.h>
#include<math.h>
int n;
int adj[10][10]
int visited[50]
int q[20]
int front=-1;
int rear=-1;
Void enque(int v)
{
    if (front==0 && rear=n-1)
    {
        printf("QueueFull\n");
    }
  else if (front==-1 && rear==-1)
  {
     f. front=rear=0;
  }
  else
    rear++;
  q[rear]=v
}
```

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

}
```

```
void bfs (int v)
{
    for (int i=0; i<n; i++)
    {
        if (adj[v][i]==1 && visited[i]==
        {
            enqueue(i)
            printf("%d", i);
            visited[i]=1;
        }
    }
    int val = dequeue()
    if (val!=-1)
        bfs(i)
}
int main()
{
    int flag=1, v;
    printf("Enter Number of vertices\n")
    scanf("%d", &n);
    printf("Enter adjacent matrix\n");
    for (int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
            scanf("%d", &a[i][j]);
    }
    printf("Enter start vertex\n");
    scanf("%d", &v);
    printf("FOREST\n");
    printf("%d\t", v);
    visited[v]=1;
```

```
bfs(v)
printf("FOREST2\n");
for(int i=0; i<n; ++i)
{
    if(visited[i]==0)
    {
        flag=0;
        printf("%d",i);
        visited[i]=1;
        bfs(i);
    }
}
if(flag==1)
{
    printf("GRAPH CONNECTED");
}
}
```

Expt. No. ---------- Page No. ----------

Insertion Sort

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
void sort [int a[], int n]
int main ()
{
    clock_t t;
    int n;
    printf ("Enter Number of element \n");
    Scanf ("%d", &n);
    int a[n]
    printf ("Enter element of array \n");
    for (int i=0; i<n; i++)
        Scanf ("%d", &a[i]);
    t = clock()
    Sort (a, n);
    t = clock() - t;
    double time_taken = ((double)t) [clock
                                    -
    printf ("Time taken = %d \n", time_take
    printf ("Final Sorted order (\n");
    for (int i=0; i<n; i++)
    {
        printf ("%d \n", a[i]);
    }
}
```

Teacher's Signature : _____

```
Void Sort(int a[], int n)
{
    int v, j;
    for(int i=1; i<=n-1; i++)
    {
        v = a[i];
        j = i-1
        while (j>=0 && a[j] > v)
        {
            a[j+1] = a[j]
            j = j-1;
        }
        a[j+1] = v
    }
}
```

TOPOLOGICAL SORTING

```
#include<stdio.h>
#include<math.h>
int front=-1, rear=-1;
void push(int)
int pop();
int st[10];
int adj[10][10]
int indegree[10];
int t[10];
int k
int n

Void push(int x)
{
   if(front==-1 && rear==-1)
      front=rear=0

   else if(rear==n-1)
      return

   else
   {
      rear++;
   }
   st[rear]=x;

}
```

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

int main()
{
    int sum=0;
    printf("Enter number of vertices");
    scanf("%d", &n);
    printf("Enter the adjacency matrix\n");

    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
```

```
    {
        scanf("%d", &adj[i][j]);
    }
}
for(int i=0; i<n; i++)
{
    sum=0;
    for(int j=0; j<n; j++)
    {
        sum = sum + adj[i][i]
    }
    indegree[i] = sum;
}
for(int i=0; i<n; i++)
{
    if(indegree[i] == 0)
        push(i);
}
while(front != -1)
{
    int u = pop()
    if(u == -1)
        break;
    t[k] = u
    k++;
```

```
for(int j=0; j<n; j++)
    if (adj[v][i] == 1)
        indegree[j] --
        if (indegree[j] == 0)
            push(j);
    }
}
printf("Final Solution:");
for(int i=0; i<k; i++)
    printf("%d\t", t[i]);
}
}
```

Merge Sort

```
#include<stdio.h>
#include<time.h>
void split(int n, int a[])
{
    int mid,i,j
    if (n<2)
        return
    mid = n/2;
    int left[mid]
    int right[n-mid]
    for(i=0; i<mid; i++)
        left[i] = a[i]
    for(j=mid; j<n; j++)
        right[j-mid] = a[j]

    split( sizeof(left)/sizeof(int), left)
    split ( sizeof(right)/sizeof(int), right)
mergesort(left, right, a, sizeof(left)/sizeof(int)
                , sizeof(right)/sizeof
                (int))

}
```

```
void mergsort(int left[], int right[], int a
{
    int i=j=k=0
    while (i<nl && j<nr)
    {
        if (left[i]<right[i])
        {
            a[k] = left[i]
            i++;
        }
        else
        {
            a[k] = right[j]
            j++;
        }
        k++
    }
    while (i<nl)
    {
        a[k] = left[i]
        i++
        k++
    }
    while (i<nr)
    {
        a[k] = right[j]
        j++
        k++
    }
}
```

, int nl, int nr)

int main()
{
clock_t t;
int n;
printf("Size\n");
scanf("%d", &n);
int a[n];
printf("Elements\n");
for(int i=0; i<n; i++)
scanf("%d", &a[i]);
t=clock();
split(n, a);
t = clock() - t;
double time_taken = ((double)t)/CLOCKS_PER_SEC;
printf("%f\n", time_taken);
printf("order is\n");
for(int i=0; i<n; i++)
printf("%d\t", a[i]);

}

Quick Sort

```
void swap (int *x, int *y)
{
    int t = *x
    *x = *y
    *y = t
}

int partition (int a[], int start, int end)
{
    if (start <= end)
    int pivot = a[end]
    int pindex = start
    for(int i = start; i<end; i++)
    {
        if (a[i] <= pivot)
        {
            swap (&a[pindex], &a[i])
            pindex++;
        }
    }
    swap (&a[pindex], &a[end])
    return pindex
}
```

```
Void Quicksort(int a[], int start, int end)
{
    if (start<end){
        int pi = partition(a, start, end);
        Quicksort(a, start, pi-1);
        Quicksort(a, pi+1, end);
    }
}
```

Heap Sort

```
#include<stdio.h>
#include<math.h>
Void swap(int *x, int *y)
{
    int temp = *x;
   *x = *y;
   *y = temp;
}
Void heapify(int a[], int n)
{
  for(int i = n/2; i>=1; i--)
  {
      int k=i;
      int v= a[k];
      int heap=0;
      while(!heap && 2*k<=n)
      {
          int j = 2*k
          if(j<n)
          {
              if(a[j]<a[j+1])
              {
                  j+=1;
              }
          }
```

```
if (v>a[i])
    heap=1
else
{
  a[k]=a[j]
  k=j
}
a[k]=v
}
}
}
int main()
{
  int n
  printf("Enter the number of elements\n")
  scanf("%d", &n)
  int a[n+1];
  printf("Enter elements\n")
  for(int i=1; i<(n+1); i++)
      scanf("%d", &a[i])
  heapify(a,n)
  for(int i=n; i>=1; i--)
  {  swap(&a[1], &a[i])
     heapify(a,i-1);
  }
```

Knapsack

```
#include<stdio.h>
int max(int a, int b)
{
    if (a>b)
        return a
    else
        return b
}
int main()
{
    int cap, n, count;
    printf("Enter knapsack capacity");
    Scanf("%d", &cap);
    printf("Enter the number of instance\n");
    Scanf("%d", &n);
    int w[n+1];
    int v[n+1];
    printf("Enter the weight of instances\n");
    for (int i=1; i<=n; i++)
    {
        printf("Enter the weight of instance %d[n", i)
        Scanf("%d", &w(i));
    }
    for (int i=1; i<=n; i++)
    {
        printf("Enter the value of instance %d[n");
        Scanf("%d", &V[i]);
    }
```

```
int a[n+1][cap+1]
for(int i=0; i<=n; i++)
    for(int j=0; j<=cap; j++)
        a[i][j]=0
    }
}
if int optimal=0;
   int ins =0
   for(int i=1; i<=n; i++)
       for(int j=1; j<=cap; j++)
           a[i][i] = max( a[i]
           if (j - w[i] >=0)
               a[i][i] = max( a[i-1][i], v[i] + a[i-1][j-wi]
               if (a[i][i] > optimal)
                   optimal = a[i][i]
                   ins = i
               }
           }
       else
           a[i][i]= a[i-1][j]
       }
   }
}
```

```
printf("Final Solution matrix is \n");
for(int i=1; i<=n; i++)
{
    for(int j=1; j<=cap; j++)
        printf("%d\t", a[i][j]);
    printf("\n");
}

printf("optimal solution %d \n", optimal);

v[ins] = 0
wl = W - w[ins]
while(wl>0)
{
    optimal = 0
    for(int i=1; i<=n; i++)
    {
        if(v[i] != 0)
        {
            if(a[i][wl] > optimal)
            {
                ins = i;
                optimal = a[i][wl];
            }
        }
    }
    wl = wl - w[ins]
    v[ins] = 0
}

printf("Instances included are %d\n");
for(int i=1; i<=n; i++)
{
    if(v[i]==0)
    {
        printf("%d\t", i);
        count++;
    }
}
```

printf("instance included \n");

WARSALL;

```
#include<stdio.h>
int main()
{
    int n;
    printf("Enter the number of vertices\n");
    scanf("%d", &n);
    printf("Enter the adjacency matrix\n");
    int a[n+1][n+1]
    for(int i=1; i<=n; i++)
    {
        for(int j=1; j<=n; j++)
            scanf("%d", &a[i][j])
    }
    for(int k=1; k<=n; k++)
    {
        for(int i=1; i<=n; i++)
        {
            for(int j=1; j<=n; j++)
            {
                if(a[i][j]=1)
                a[i][j]= a[i][j]

                else
                if(a[i][k]==1 && a[k][j]=1)
                {
                    a[i][j]=1
                }
            }
        }
    }
```

```
printf("Transitive closure is \n")
for(int i=1; i<=n; i++)
{
    for(int j=1; j<=n; j++)
        printf("%d\t", a[i][j])
    printf("\n");
}
int v[n+1]
for(int i=1; i<=n; i++)
{
    v[i] = 1
}
for(int i=1; i<=n; i++)
{
    for(int j=i+1; j<=n; j++)
    {
        if(a[i][j]==1 && a[j][i]==1)
        {
            v[i]=0
            v[j]=0
        }
    }
}
printf("Vertex forming cycle include \n")
for(int i=1; i<=n; i++)
{
    if(v[i]==0)
        printf("%d \t", i);
}
}
```

FLOYD

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

```
int main()
{
    int n;
    printf("\n Enter number of Vertices\n");
    scanf("%d", &n);
    int a[n+1][n+1];
    printf("\n Enter distance matrix\n");
    printf("999 - Infinity\n");
    for(int i=1; i<=n; i++)
    {
        for(int j=1; j<=n; j++)
            scanf("%d", &a[i][j]);
    }
    int i, j, k;
    for(k=1; k<=n; k++)
        for(i=1; i<=n; i++)
            for(j=1; j<=n; j++)
                a[i][j] = min(a[i][j], (a[i][k]
                          + a[k][j]));

    printf("\n Distance Matrix is \n");
    for(int i=1; i<=n; i++) {
        for(int j=1; j<=n; j++) {
            printf("%d", a[i][j]);
        }
        printf("\n");
    }
}
```

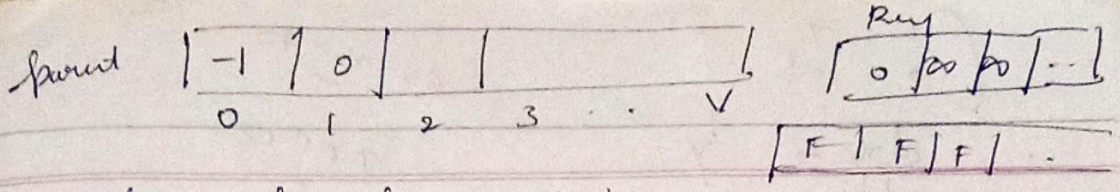
Prim's algorithm:

```
#include<limits.h>   → for obtaining infinity
#include<stdbool.h>  → include boolean datatype
int main()
{
    int graph[][] = ...

    primMST(graph)

    return 0
}
void primmst(int graph[V][V])
{
    int parent[V]
    int key[V]
    bool mstset[V]   → To check if vertex has been included
    for(int i=0; i<V; i++)
    {
        key[i] = INT_MAX
        mstset[i] = false
    }
    key[0] = 0
    parent[0] = -1
    for(int count=0; count<V; count++)
    {
        int u = minkey(key, mstset)
        mstset[u] = true

        for(int v=0; v<V; v++)
            if(graph[u][v] && mstset[v]==false graph[u][v]<
            {
                parent[v] = u
                key[v] = graph[u][v]
            }
        }
        printmst(graph)
}
```




```
int minKey(int key[], bool mstSet[])
{
    int min = INT_MAX
    min_index
    for (int v=0; v<V; v++)
        if (mstSet[v]==false && key[v] < min)
        {
            min = key[v]
            min_index = v;
        }
    return min_index
}
```

Here approach used here is the *

assign

```
void printMST(int parent[], int graph[V][V])
{
    printf("Edge \t Weight \n");
    for (int i=1; i<V; i++)
        printf("%d - %d \t %d \n", parent[i],
               i, graph[i][p]
                          parent[i]
```

key[v])

Kruskal's algorithm.

```
#include<stdio.h>
int c[10][10], n
Void main()
{
    int i, j;
    clrscr();
    printf("enter the number of vertices\t");
    scanf("%d", &n);
    printf("Enter the cost matrix\n");
    for(int i=1; i<=n; i++)
    {
        for(j=1; j<=n; j++)
        {
            scanf("%d", &c[i][j]);
        }
    }
    kruskals();
    getch()
}

Void kruskals()
{
    int i, j, u, v, a, b, min
    int ne=0; mincost=0
    int parent[10];
    for(int i=1; i<=n; i++)
    {
        parent[i]=0;
    }
    while(ne!=n-1)
    {
        min=9999;
        for(i=1; i<=n; i++)
```

```
}
    for(j=1; j<=n; j++)
    {
        if (c[i][j] < min)
        {
            min = c[i][j]
            u=a= i
            v=b= j
        }
    }
}
while (parent[u] != 0)
{
    u= parent[u]
}
while (parent[v] != 0)
{
    v= parent[v]
}
if (u!=v)
{
    printf ("\n%d - %d\t %d\n", a, b, min);
    parent[v]= u;
    ne = ne+1
    mincost = mincost + min
}
    c[a][b] = c[b][a] = 9999
    print
}
    printf ("\n min cost = %d", mincost);
}
```

DIJKSITRA'S ALGORITHM

```
# include < limit.h >
#include < stdio.h >
# define V 5
int main()
{
    int int adj

    int graph[V][V]
    printf ("Enter the distance matrix")
    for (int i=0; i<V; i++)
    {
        for(int j=0; j<V; j++)
            Scanf ("%d", & graph[i][j]);

    }
    dijkstra ( graph)

}

Void dijkstra ( int graph[V][V] )
{
    int parent[V]
    int key [V]
    bool mstset[V]
    for(int i=0; i<V; ;++){
        parent[i] = 0
        key [i] = INI_MAX
        mstset [i] = false
    }
    key [0] = 0
    parent[0] = -1
    for(int Count=0; Count< V-1 ; Count++)
    {
```

```
int u = minkey(key, mstset);
mstset[u] = true

for(int v=0; v<V; v++)
{
    if(graph[u][v] && mstset[v]==false
                   && key[v] != INT_MAX
                   && key[u] + graph[u][v]
                        < key[v])
    {
        key[v] = key[u] + graph[u][v]
    }
}
print(key);

int minkey(int k[], bool &mstset[])
{
    int min = INT_MAX;
    int minIndex;
    for(int i=0; i<V; i++)
    {
        if (key[i] < min)
        {
            min = key[i]
            minIndex = i
        }
    }
return minIndex.
}
```

Ba N-Queens problem:

Code

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

int x[10];
int main()
{
    int n;
    printf("Enter the number of Queens\n");
    scanf("%d", &n);
    for(int k=1; k<=n; k++)
        nQueens(k, n);
}

nQueens(int k, int n)
{
    for(int i=1; i<=n; i++)
    {
        if(ifBlock(k, i))
            x[k] = i;
            if(k==n)
                print(x, i);
            else
                nQueens(k+1, n));
        }
    }
}
```

```
int place(int k, int i)
{
    for (int j = 1; j <= k-1; j++)
    {
        if ((x[j] == i) || (abs(x[j] - i) == abs(j - k)))
        {
            return 0;
        }
    }
    return 1;
}
```

```
}
#include <stdio.h>
int d;
int x[10]
int w[10]
int n;

int main()
{
    int r=0
    printf("Enter target weight \n")
    scanf("%d", &d);
    printf("Enter the number of weight \n")
    scanf("%d", &n);
    printf("Enter the weight \n")

    for(int i=1; i<=n; i++){
        scanf("%d", &w[i])
        r+ = w[i]
        x[i] = 0
    }
    int cs=0;
    sumofsubs(cs, 1, r)

}

void sumofsubs(int cs, int k, int r)
{
    if(k<=n)
    {
        if(cs+w[k]==d)
        x[k]=1
        if((cs+w[k]==d)
        {
```

```
for(int i=1; i<=k; i++)
    if (x[i]==1)
        printf("%d\t", w[i]);
}
else if (cs + w[k] + w[k+1] <= d)
    SumOfSubs(cs+w[k], k+1, r-w[k]);

if ((cs + r-w[k] >= m) && (cs + w[k+1] <= d))
{
    x[k]=0;
    SumOfSubs(cs, k+1, r-w[k]);
}
}
```