**BFS**

```
#include<iostream.h>
#include<conio.h>
class graph
{
 int array[10][10];
 int direction;
 int novertex;
 public:
 graph ();
 void create_mat();
 void print ();
};
   graph ::  graph()
 {
  cout<<"\n\t How many vertex you want in graph: ";
  cin>>novertex;
  cout<<"\n\t Enter direction of graph (1-Directed, 0-Undirected)";
  cin>>direction;
  int i,j;
  for (i=0;i<10;i++)
  {
          for(j=0;j<10;j++)
          array[i][j]=0;     // disjoint graph
  }
 }
```

```
   void graph:: create_mat()
 {
  int max_edge,from,to;
  if(direction==1)
  max_edge=(novertex*(novertex-1));
  else
  max_edge=(novertex*(novertex-1))/2;
  for(int i=1;i<=max_edge;i++)
  {
cout<<"\n \t Enter two vertices to be connected. Press 0 0 to Quit \n"<<endl;
          cout<<"\n\t from vertex="; //source vertex
          cin>>from;
          cout<<"\n\t To vertex ="; //destination vertex
          cin>>to;
  if(from==0||to==0)              //condition to quit loop
          break;
```

```
   if (from <0   ||  to<0   ||   from>novertex ||  to>novertex)
   {
          cout<<"\tGraph Not Possible,Invalid connections"<<endl;
          i--;
   }
   else
   {
          array[from][to]=1;
          if(direction==0)
          array[to][from]=1;
   }
   }
}
void graph :: print()
{
   int i,j;
   for (i=1;i<=novertex;i++)
   {
    cout<<"\t ";
    for(j=1;j<=novertex;j++)
    cout<<" "<<array[i][j];
    cout<<endl;
   }
}
void main()
          {
             clrscr();
             graph g;
             clrscr();
             g.create_mat();
             g.print();
             getch();
          }
```

**// BFS on Graph**

```
#define MAX 100
#define initial 1
#define waiting 2
#define visited 3
```

```
int n;
int adj[MAX][MAX];
int state[MAX];
void create_graph();
void  BF_Traversal();
void BFS(int v);

int queue[MAX], front = -1,rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();

int main()
{
        create_graph();
        BF_Traversal();
    return 0;
}

void BF_Traversal()
{
        int v;
                for(v=1; v<=n; v++)
                state[v] = initial;
        cout<<"Enter Start Vertex for BFS: \n";
        cin>>v;
        BFS(v);
}
void BFS (int v)
{
        int i;
        insert_queue(v);
        state[v] = waiting;
        while(!isEmpty_queue())
        {
                v = delete_queue( );
                cout<<v;
                state[v] = visited;

                for(i=1; i<=n; i++)
                {
                        if(adj[v][i] == 1 && state[i] == initial)
                        {
                                insert_queue(i);
                                state[i] = waiting;
                        }
                }
        }
```

```
        Cout<<"\n";
}

void insert_queue(int vertex)
{
        if(rear == MAX-1)
                cout<<"Queue Overflow\n";
        else
        {
                if(front == -1)
                        front =  0;
                rear = rear+1;
                queue[rear] = vertex ;
        }
}

int isEmpty_queue()
{
        if(front == -1 || front > rear)
                return 1;
        else
                return 0;
}

int delete_queue()
{
        int delete_item;
        if(front == -1 || front > rear)
        {
                Cout<<"Queue Underflow\n";
                exit(1);
        }
        delete item = queue [front];
        front = front+1;
        return delete_item;
}

void create_graph()
{
        int count,max_edge,origin,destin;
        cout<<"Enter number of vertices: “
        cin>>n;
        max_edge = n*(n-1);
        for (count=1; count<=max_edge; count++)
        {
                cout<<"Enter edge (-1 -1 to quit): "<<count;
                cin>>origin>>destin;
```

```
                if((origin == -1) && (destin == -1))
                        break;

                if(origin>n || destin>n || origin<0 || destin<0)
                {
                        cout<<"Invalid edge!\n";
                        count--;
                }
                else
                {
                        adj[origin][destin] = 1;
                }
        }
}
```

**// Minimum spanning Tree Using Prims Algorithm**

```
#include <iostream>
#define ROW 7
#define COL 7
#define infi 9999 //infi for infinity
using namespace std;
class prims
{
        int graph[ROW][COL],nodes;
        public:
                prims();
                void createGraph();
                void primsAlgo();
};

prims :: prims()
{
        for(int i=0;i<ROW;i++)
                for(int j=0;j<COL;j++)
                        graph[i][j]=0;
}

void prims :: createGraph()
{
        int i,j;
        cout<<"Enter Total Nodes : ";
        cin>>nodes;
        cout<<"\n\nEnter Adjacency Matrix : \n";
        for(i=0;i<nodes;i++)
                for(j=0;j<nodes;j++)
                        cin>>graph[i][j];
```

```
	//Assigning infinity to all graph[i][j] where weight is 0
	for(i=0;i<nodes;i++)
		for(j=0;j<nodes;j++)
			if(graph[i][j]==0)
				graph[i][j]=infi;

	//Printing graph in matrix form
	cout<<"Matrix is:"<<endl;
	for(i=0;i<nodes;i++)
		for(j=0;j<nodes;j++)
		{
		  cout<<" "<<graph[i][j];
		  if ((j+1)%nodes==0)
		  cout<<endl;
		}
}

void prims :: primsAlgo()
{
	cout<<"Minimum spanning tree is:" ;
	int selected[ROW],i,j,ne;
	int min,x,y;
	for(i=0;i<nodes;i++)
		selected[i]=false;
	selected[0]=true;
	ne=0;
	while(ne < nodes-1)
	{
		min=infi;
		for(i=0;i<nodes;i++)
		{
			if(selected[i]==true)
			{
				for(j=0;j<nodes;j++)
				{
					if(selected[j]==false)
					{
						if(min > graph[i][j])
						{
						  min=graph[i][j];
						  x=i;
						  y=j;
						}
					}
				}
			}
		}
```

```
                    selected[y]=true;
                    cout<<"\n"<<x+1<<" --> "<<y+1;
                    ne=ne+1;
  }
}

int main(){
   prims MST;

   cout<<"\nPrims Algorithm to find Minimum Spanning Tree\n";
   MST.createGraph();
   MST.primsAlgo();
  return 0;
}
```

**Hashing**

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

class hashMethod
{
   private:
      int arr[1000];
   public:
      void directM(void);
      void subtractionM(void);
      void moduloDivisionM(void);
      void moduloDivCollusionM(void);
      void digitExtraction_1(void);
      void digitExtractionCollusion_1();
      void digitExtraction_2(void);
      void digitExtractionCollusion_2();
      void digitExtraction_3(void);
      void digitExtractionCollusion_3();
      void displayArray(void);
      hashMethod()
      {
         int i;
         for(i=0;i<1000;i++)
         {
            arr[i]=0;
         }
      }
};

void hashMethod :: directM(void)
{
   int address, key;
   cout<<"Enter key value between 1 to 20: ";
   cin>>key;
   if((key>=0) && (key<20))
   {
      address=key;
      arr[address]=key;
   }
   else
   {
      cout<<"Invalid Input\nEnter value between 1 to 20 only."<<endl;
   }
}

void hashMethod :: subtractionM(void)
{
```

```
   int address, key;
   cout<<"Enter key value between 1 to 20: ";
   cin>>key;
   if((key>=0) && (key<100))
   {
      address=(100-key)%20;
      arr[address]=key;

   }
   else
   {
      cout<<"Invalid Input\nEnter value between 1 to 20 only."<<endl;
   }
}

void hashMethod :: moduloDivisionM(void)
{
   int address, key;
   cout<<"Enter key value between 1 to 100: ";
   cin>>key;
   if((key>=0) && (key<100))
   {
      address=key%20;
      arr[address]=key;
   }
   else
   {
      cout<<"Invalid Input\nEnter value between 1 to 20 only."<<endl;
   }
}

void hashMethod :: moduloDivCollusionM(void)
{
   int address, key;
   cout<<"Enter key value between 1 to 100: ";
   cin>>key;
   if((key>=0) && (key<100))
   {
      address=key%20;
      if(arr[address]==0)
      {
         arr[address]=key;
      }
      else
      {
         address+=1;
         arr[address]=key;
      }
```

```
    }
    else
    {
        cout<<"Invalid Input\nEnter value between 1 to 20 only."<<endl;
    }
}

void hashMethod :: digitExtraction_1(void)
{
    int address, key;
    cout<<"Enter key value between 1 to 100: ";
    cin>>key;
    if((key>=10) && (key<1000))
    {
        address=key%10;
        arr[address]=key;
    }
    else
    {
        cout<<"Invalid Input\nEnter value between 1 to 20 only."<<endl;
    }
}

void hashMethod :: digitExtractionCollusion_1(void)
{
    int address, key;
    cout<<"Enter key value: ";
    cin>>key;
    if((key>=10) && (key<1000))
    {
        address=key%10;
        if(arr[address]==0)
        {
            arr[address]=key;
        }
        else
        {
            address+=1;
            arr[address]=key;
        }
    }
    else
    {
        cout<<"Invalid Input\nEnter value between 1 to 20 only."<<endl;
    }
}

void hashMethod :: digitExtraction_2(void)
```

```
{
   int address, key;
   cout<<"Enter key value between 1 to 100: ";
   cin>>key;
   if((key>=10) && (key<1000))
   {
      address=key%100;
      arr[address]=key;
   }
   else
   {
      cout<<"Invalid Input\nEnter value between 1 to 20 only."<<endl;
   }
}

void hashMethod :: digitExtractionCollusion_2(void)
{
   int address, key;
   cout<<"Enter key value: ";
   cin>>key;
   if((key>=10) && (key<1000))
   {
      address=key%100;
      if(arr[address]==0)
      {
         arr[address]=key;
      }
      else
      {
         address+=1;
         arr[address]=key;
      }
   }
   else
   {
      cout<<"Invalid Input\nEnter value between 1 to 20 only."<<endl;
   }
}

void hashMethod :: digitExtraction_3(void)
{
   int address, key;
   cout<<"Enter key value between 1 to 100: ";
   cin>>key;
   if((key>=10) && (key<1000))
   {
      address=key%1000;
      arr[address]=key;
```

```
    }
    else
    {
        cout<<"Invalid Input\nEnter value between 1 to 20 only."<<endl;
    }
}

void hashMethod :: digitExtractionCollusion_3(void)
{
    int address, key;
    cout<<"Enter key value: ";
    cin>>key;
    if((key>=10) && (key<1000))
    {
        address=key%1000;
        if(arr[address]==0)
        {
            arr[address]=key;
        }
        else
        {
            address+=1;
            arr[address]=key;
        }
    }
    else
    {
        cout<<"Invalid Input\nEnter value between 1 to 20 only."<<endl;
    }
}

void hashMethod :: displayArray(void)
{
    for(int i =0; i<1000;i++)
    {
        if(arr[i]==0)
        {
            continue;
        }
        else
        {
            cout<<"arr["<<i<<"]="<<arr[i]<<endl;
        }
    }
}

void main(void)
{
```

```
clrscr();

hashMethod h;
char ch='y';
int option;
int i;
while(ch=='y')
{
   cout<<"Selection operation:"<<endl;
   cout<<"1. Display"<<endl;
   cout<<"2. Direct  Method"<<endl;
   cout<<"3. Subtraction Method" <<endl;
   cout<<"4. Modulo Division Method"<<endl;
   cout<<"5. Modulo Division Method with collusion"<<endl;
   cout<<"6. Last 1 Digit Extraction"<<endl;
   cout<<"7. Last 1 Digit Extraction with collusion"<<endl;
   cout<<"8. Last 2 Digit Extraction"<<endl;
   cout<<"9. Last 2 Digit Extraction with collusion"<<endl;
   cout<<"10. Last 3 Digit Extraction"<<endl;
   cout<<"11. Last 3 Digit Extraction with collusion"<<endl;
   cout<<"Enter an option: ";
   cin>>option;
   switch(option)
   {
      case 1:
      {
         h.displayArray();
         cout<<"\nDo you want to continue?";
         break;
      }

      case 2:
      {
         h.directM();
         cout<<"\nDo you want to continue?";
         break;
      }
      case 3:

      {
         h.subtractionM();
         cout<<"\nDo you want to continue?";
         break;
      }
      case 4:
      {
         h.moduloDivisionM();
         cout<<"\nDo you want to continue?";
```

```
            break;
        }
        case 5:
        {
            h.moduloDivCollusionM();
            cout<<"\nDo you want to continue?";
            break;
        }
        case 6:
        {
            h.digitExtraction_1();
            cout<<"\nDo you want to continue?";
            break;
        }
        case 7:
        {
            h.digitExtractionCollusion_1();
            cout<<"\nDo you want to continue?";
            break;
        }
        case 8:
        {
            h.digitExtraction_2();
            cout<<"\nDo you want to continue?";
            break;
        }
        case 9:
        {
            h.digitExtractionCollusion_2();
            cout<<"\nDo you want to continue?";
            break;
        }
        case 10:
        {
            h.digitExtraction_3();
            cout<<"\nDo you want to continue?";
            break;
        }
        case 11:
        {
            h.digitExtractionCollusion_3();
            cout<<"\nDo you want to continue?";
            break;
        }
      }
      ch = getch();
    }
}
```

**Binary Search :**

```
#include<iostream.h>
#include<conio.h>
class BSearch
{
int arr[10];
public:
void bsearch();
void getdata();
void display();
};
void BSearch::getdata()
{
cout<<"\nEnter 10 elements in ascending\n";
for(int i=0;i<10;i++)
cin>>arr[i];
}
void BSearch::display()
{
cout<<"\nElements in array\n";
for(int i=0;i<10;i++)
cout<<arr[i]<<"\t";
}
void BSearch::bsearch()
{
 int first=0;
 int last=10;
 int mid=(first+last)/2;
 int search;
 cout<<"\nEnter value to be sarched: ";
 cin>>search;
 while(first<=last)
 {
if(arr[mid]<search)
first=mid+1;
else if(arr[mid]==search)
{
cout<<"value found at pos: "<<mid+1;
break;
}
else
last=mid-1;
mid=(first+last)/2;
 }
 if(first>last)
cout<<"\nValue not found";
}
void main()
{
```

```
clrscr();
BSearch b;
b.getdata();
b.display();
b.bsearch();
getch();
}
```

**Linear Search :**

```
#include<iostream.h>
#include<conio.h>
class LSearch
{
        int arr[10];
        public:
                void getdata();
                void display();
                void search1();
                void search2(int);
                int search3();
                int search4(int);
} ;

void LSearch::getdata()
{
        int i;
        cout<<"\nEnter 10 elements in array\n";
        for(i=0;i<10;i++)
                cin>>arr[i];
}

void LSearch::display()
{
        int i;
        cout<<"\nElements in array: \n";
        for(i=0;i<10;i++)
                cout<<arr[i]<<"\t";
}

void LSearch::search1()
{
        int num,pos=1,flag=0;
        cout<<"\nEnter element to search: ";
        cin>>num;
        for(int i=0;i<10;i++)
        {
```

```
                if(arr[i]==num)
                {
                        flag=1;
                        break;
                }
                pos++;
        }
        if(flag==1)
        {
                cout<<"\nElement found at pos: "<<pos;
        }
        else
        {
                cout<<"\nElement not found";
        }
}

void LSearch::search2(int num)
{
        int flag=0,pos=1;
        for(int i=0;i<10;i++)
        {
                if(arr[i]==num)
                {
                        flag=1;
                        break;
                }
                pos++;
        }
        if(flag==1)
        {
                cout<<"\nElement found at pos: "<<pos;
        }
        else
        {
                cout<<"\nElement not found";
        }
}

int LSearch::search3()
{
        int num,pos=1,flag=0;
        cout<<"\nEnter element to search: ";
        cin>>num;
        for(int i=0;i<10;i++)
        {
                if(arr[i]==num)
                {
                        flag=1;
                        break;
                }
                pos++;
```

```
	}
	if(flag==1)
	{
		return pos;
	}
	else
	{
		return -1;
	}
}

int LSearch::search4(int num)
{
	int flag=0,pos=1;
	for(int i=0;i<10;i++)
	{
		if(arr[i]==num)
		{
			flag=1;
			break;
		}
		pos++;
	}
	if(flag==1)
	{
		return pos;
	}
	else
	{
		return -1;
	}
}

void main()
{
	clrscr();
	int opt,num,result;
	LSearch b;
	b.getdata();
	cout<<"\nElements without sorting:\n";
	b.display();
	cout<<"\n1.Search1\n2.search2\n3.Search3\n4.search4\n";
	cout<<"\nEnter the Option to perform the operation: ";
	cin>>opt;
	switch(opt)
	{
		case 1:
				b.search1();
				break;
		case 2:
				cout<<"\nEnter the element to search: ";
				cin>>num;
```

```
                        b.search2(num);
                        break;
                case 3:
                        result=b.search3();
                        if(result==-1)
                        {
                                cout<<"\nElement not found";
                        }
                        else
                        {
                                cout<<"\nElement found at pos: "<<result;
                        }
                        break;
                case 4:
                        cout<<"\nEnter the element to search: ";
                        cin>>num;
                        result=b.search4(num);
                        if(result==-1)
                        {
                                cout<<"\nElement not found";
                        }
                        else
                        {
                                cout<<"\nElement found at pos: "<<result;
                        }
                        break;
        }

        getch();
}
```