ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E. Computer Science and Engineering (Data Science)

III – Semester

21DSCP309 – Data Structures and Algorithms Lab

Name : _____

Reg. No.: _____

ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E. Computer Science and Engineering (Data Science)

III – Semester

Certified that this is the bonafide record of work done by Mr./Ms._____

Reg. No._____

of B.E. Computer Science and Engineering (Data Science) in the 21DSCP309 – Data Structures and Algorithms lab during the odd semester of the academic year 2023 – 2024.

Staff in-charge Internal Examiner

Annamalai Nagar External Examiner

Date: ..… /.… / 2023.

21DSCP309	DATA STRUCTURES AND ALGORITHMS LAB	L	T	P	C
		0	0	3	1.5

Course Objectives:

- To understand and implement linear data structures such as linear list, stack and queue.
- To implement non-linear data structures such as linear trees and graphs.
- To understand algorithms that use data structures for operations such as sorting and searching.
- To study algorithm design methods such as the greedy method, divide and conquer, dynamic programming and backtracking.

LIST OF EXERCISES

(The Exercises are to be done in C++)

CYCLE-1

1. Write a program to create a Stack and perform insertion and deletion operations on it.
2. Write a program to create a List and perform operations such as insert, delete, update and reverse.
3. Write a program to create a Queue and perform operations such as insertion and deletion.
4. Using iteration and recursion concepts write programs for finding the element in the array using the Binary Search method.
5. Write a program and simulate various graph traversing techniques.
6. Write a program to Implement Binary Search Tree.
7. Write a program to simulate Bubble sort algorithm

CYCLE-2

8. Write a program to implement separate chaining to handle collisions in hashing.
9. Write a program to Implement Heaps using Priority Queues.
10. Implement the Quick sort algorithm to illustrate Divide and Conquer method.
11. Using Dynamic programming method implement Dijkstra's shortest path Algorithm.
12. Write a program to simulate the n-Queens problem using backtracking approach.
13. Implement the Selection sort algorithm to illustrate Greedy Approach.

Course Outcomes:

At the end of the course, the students will be able to

1. Develop a C++ program to build the basic data structures like stack, queue and list.
2. Develop a C++ program for searching and sorting algorithms using iteration and recursion concept.
3. Demonstrate an ability to listen and answer the viva questions related to programming skills needed for solving real-world problems in Computer Science andEngineering.

Mapping of Course Outcomes with Program Outcomes												
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO1	1	2	3	2	-	-	-	-	-	-	-	-
CO2	1	2	3	2	-	-	-	-	-	-	-	-
CO3	2	2	-	-	-	-	-	-	-	2	-	2

Vision-Mission of the Department of Computer Science and Engineering

Vision

To provide a congenial ambience for individuals to develop and blossom as academically superior, socially conscious and nationally responsible citizens.

Mission

- Impart high quality computer knowledge to the students through a dynamic scholastic environment wherein they learn to develop technical, communication and leadership skills to bloom as a versatile professional.
- Develop life-long learning ability that allows them to be adaptive and responsive to the changes in career, society, technology, and environment.
- Build student community with high ethical standards to undertake innovative research and development in thrust areas of national and international needs.
- Expose the students to the emerging technological advancements for meeting the demands of the industry.

Program Educational Objectives (PEOs)

PEO	PEO Statements
PEO1	To prepare the graduates with the potential to get employed in the right role and/or become entrepreneurs to contribute to the society.
PEO2	To provide the graduates with the requisite knowledge to pursue higher education and carry out research in the field of Computer Science.
PEO3	To equip the graduates with the skills required to stay motivated and adapt to the dynamically changing world so as to remain successful in their career.
PEO4	To train the graduates to communicate effectively, work collaboratively and exhibit high levels of professionalism and ethical responsibility.

Rubric for CO3 in Laboratory Course

Rubric for CO3 in Laboratory Courses				
Course Outcome	**Distribution of 10 Marks for IE/SEE out of 40/60 Marks**			
	Up to 2 Marks	**Up to 5 Marks**	**Up to 7 Marks**	**Up to 10 marks**
Demonstrate an ability to listen and answer the viva questions related to programming skills needed for solving real-world problems in Computer Science and Engineering.	Poor listening and communication skills. Failed to relate the programming skills needed for solving the problem	Showed better communication skill by relating the problem with the programming skills acquired but the description showed serious errors.	Demonstrated good communication skills by relating the problem with the programming skills acquired with few errors	Demonstrated excellent communication skills by relating the problem with the programming skills acquired and have been successful in tailoring the description

CONTENTS

EX.NO:1	STACK
DATE:	

AIM:

To write a C++ program to create a Stack and perform insertion and deletion operations on it.

Procedure:

Member variables in the class stack

size, top as integer
StackData as array of integer of size 100

Member function in the class stack

Stack() {top=-1;}	- Constructor to set top as -1
void SetSize(intn)	- To set the size of stack
void Push(intx)	- To insert value x in to the stack
int Pop(void)	- To remove a value from the stack
void Display(void)	- To display all elements of the stack

Algorithm:

Stack()

1. Initialize top=–1.void

SetSize(int n)

1. Set the size of the Stack as n

void Push(int x)

1. Check whether the top is less than the size of the stack.
2. If true (there is space in the stack), then increment the top by one and place the newelement in the position pointed to by top.
3. Otherwise Print “Stack Overflow”.

int Pop(void)

1. Check whether the top is greater than –1.
2. If true (there is an element) then, store the element present at the location pointed to bytop in x and decrement the value of the pointer by one and return the value of x.
3. Otherwise Print “Stack Underflow”.

void Display(void)

1. Display the elements present in the locations pointed by top to location zero.

Otherwise, display “The Stack is empty”.**int main()**

1. Display menu to perform the operations on the Stack.
2. Get user choice and perform the corresponding operation.

Source Code:

```
//Stack Operations     #include<iostream>
 #include<conio.h>
 #include<stdlib.h>
 #define CLRSCR system("cls");
 inline void foo() { CLRSCR }
 using namespace std;
```

```
class Stack {
		int StackData[100], size, top;
	public:
		Stack() {
			top = -1;
		}
		void SetSize(int n) {
			size=n;
		}
		void Push(int x);
		int Pop(void);
		void Display(void);
};
void Stack::Push(int x) {
	if(top>=size-1) {
		cout<<"\n\t\t Stack over flow...";
	} else {
		top++;
		StackData[top]=x;
		cout<<"\n\t\t"<<x<<" is pushed \n\n\n";
	}
}
int Stack::Pop(void) {
	int x;
	if(top<=-1) {
		cout<<"\n\t\t Stack under flow";
		return -9999;
	} else {
		x=StackData[top];
		top--;
		return x;
	}
}
void Stack::Display() {
	if(top>=0) {
		cout<<"\n The elements in stack are: ";
		cout<<"\n~~~~~~~~~~~~~~~~~~~~~~~~";
		for(int i=top; i>=0; i--)
			cout<<"\n\n DATA "<<i+1<<"="<<StackData[i];
		cout<<"\n\n\n\n Press any key to continue...";
	} else {
		cout<<"\n The Stack is empty.";
	}
}
int main() {
	Stack s;
	int option, n;
	cout<<"\n Enter the size of stack [MAX=100] : ";
	cin>>n;
	s.SetSize(n);
	while(option != 4) {
		system("cls");
		cout<<"\n\t\t Stack Operation";
```

```
		cout<<"\n\t\t ~~~~~~~~~~~~~~~~";
		cout<<"\n\t1.PUSH\n\t2.POP\n\t3.DISPLAY\n\t4.EXIT";
		cout<<"\n\n Enter your option : ";
		cin>>option;
		int x, topvalue;
		switch(option) {
			case 1:
				cout<<"\n Enter a value to be pushed : ";
				cin>>x;
				s.Push(x);
				s.Display();
				getch();
				break;
			case 2:
				topvalue=s.Pop();
				if (topvalue!=-9999)
					cout<<"\n\t\t The popped element is "<<topvalue;
				s.Display();
				getch();
				break;
			case 3:
				s.Display();
				getch();
				break;
			case 4:
				cout<<"\n\n\t\t You are on EXIT...";
				getch();
				break;
			default:
				cout<<"\n\n\t Enter only1,2,3,4.";
				getch();
		}
	};
	return 0;

}
```

Sample Input/Output:

```
Enter the size of stack [MAX=10] : 5
Stack Operation
~~~~~~~~~~~~
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option : 1
Enter a value to be pushed : 5
5 is Pushed
The elements in stack are:
~~~~~~~~~~~~~~~~
DATA1 =5
Press any key to continue...
Stack Operation
~~~~~~~~~~~~
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option : 1
Enter a value to be pushed : 4
4 is Pushed
The elements in stack are:
~~~~~~~~~~~~~~~~~~
DATA2 =4
DATA1 =5
Press any key to continue...
Stack Operation
~~~~~~~~~~~~
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option : 1
Enter a value to be pushed : 3
3 is Pushed
The elements in stack are:
~~~~~~~~~~~~~~~~~
DATA 3 =3
DATA2 =4
DATA1 =5
Press any key to continue...
Stack Operation
~~~~~~~~~~~~
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option : 1
```

Enter a value to be pushed: 22
22 is Pushed
The elements in stack are:
~~~~~~~~~~~~~~~~
DATA4=22
DATA 3 =3
DATA2 =4
DATA1 =5
Press any key to continue...
Stack Operation
~~~~~~~~~~~~
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option : 1
Enter a value to be pushed: 11
11 is Pushed
The elements in stack are:
~~~~~~~~~~~~~~~~~
DATA5=11
DATA4=22
DATA 3 =3
DATA2 =4
DATA1 =5
Press any key to continue...
Stack Operation
~~~~~~~~~~~~
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option : 1
Stack over flow...
Press any key to continue...
Stack Operation
~~~~~~~~~~~~
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option : 2
The popped element is 11.
The elements in stack are:
~~~~~~~~~~~~~~~~
DATA4=22
DATA 3 =3
DATA2 =4
DATA1 =5
Press any key to continue...
Stack Operation
~~~~~~~~~~~~
1. PUSH
2. POP
~~~~~~~~~~~~

```
3. DISPLAY
4. EXIT
Enter your option : 2
The popped element is 22.
The elements in stack are:
~~~~~~~~~~~~~~~~
DATA 3 =3
DATA2 =4
DATA1 =5
Press any key to continue...
Stack Operation
~~~~~~~~~~~~
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option : 2
The popped element is3.
The elements in stack are:
~~~~~~~~~~~~~~~~
DATA2 =4
DATA1 =5
Press any key to continue...
Stack Operation
~~~~~~~~~~~~
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option : 2
The popped element is4.
The elements in stack are:
~~~~~~~~~~~~~~~~
DATA1 =5
Press any key to continue...
Stack Operation
~~~~~~~~~~~~
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your option : 4
You are on EXIT…
```

Result:

Thus, a C++ program was written and executed to create a stack data structure and perform the operations on Stack.

EX NO:2	LINKED LIST
DATE:	

Aim:

To write a C++ program to create a List and perform operations such as insert, delete, update and reverse.

Procedure:

Create user defined data type called node using structure in C++ with the following members

data as integer

link as a pointer to hold the address of the next node

Member variables in the class LinkedList

Head as pointer to the node
Count variable to count number of elements.

Member functions in the LinkedList

LinkedList() - Constructor to initialize Head as NULL and Count as 0.
~LinkedList(){delete Head;} - Destructor to release the memory of Head void
Append(int x); - To add a node at the end of the linked list void
Add After(int loc, int x); - To add a node after the specific location in the linked list
void AddAtBegin(int x); - To add a node at the beginning of the linked list
void Display(int flag=1); - To display all nodes of linked list
void Count(); - To count number of nodes in the linked list
void Delete(int x); - To delete a node in the linked list
void Reverse(); - To reverse the linked list

Algorithm:

LinkedList()

1. Initialize Head as NULL.

~LinkedList()

1. Release the memory allocated to Head.

void Append(int x)

1. Create a node temp and get the data in the data field of the temp node and nextaddress field of temp to hold NULL.
2. Make the next-address field of the last node to hold the address of the temp node.

void AddAfter(int loc,int x)

1. Create a node temp and get the data in the data field of the temp node and next-address field of temp to hold NULL.
2. After traversing N nodes in the list i.e., specified by loc, make the Nth node nextaddress field to hold the address of the temp node and then make the next-address field of the temp node to hold the address of the (N+1)th node.

void AddAtBegin(int x)

1. Create a node temp and get the data in the data field of the temp node and next-address field of temp to hold NULL.

Make the temp node as front node by making the next-address field of the temp node to hold the address of the front node of the list. [If there is no element already present in the list then, the list would have only this one element in the data field and NULL in the address field.

Void Display(int flag=1)

1. Traverse the list from the first node to the last node by displaying the contents of its data

Field.

void Count()

1. Count the nodes by traversing the node one by one from the first node till the address field of the node contains NULL.

void Delete(int x)

1. Locate the node having the data x in its data field.
2. If the node to be deleted is present at the first location, then make the next (second) node as the first node by making the first node's next-address field to NULL and free the memory location of the deleted node.
3. If the node to be deleted is present as intermediate node (having previous node and next node in the list) then, make the previous node's next-address field to hold the address of the next node and free the memory location of the deleted node.
4. If the node to be deleted is present at the last in the list, then make the previous node address to hold the NULL value and free the memory location of the deleted node.
5. If the list consists of only one node then, make the list as Null and free the memory location of the deleted node.

void Reverse()

1. Set the node pointed to by the Head node as current node. Set prev, tmp nodes as NULL.
2. Traverse the list from the current node to the last node in the list (last node has NULL in the address field and do the following:
   (i) Save the current node's address field in temp.
   (ii) Save the prev node in the current node's address field.
   (iii) Set current node as prev node.
   (iv) Set temp node as current node.
3. Set the node pointed to be previous node as Head node.

int main()

1. Display menu to perform linked list operations.
2. Get user choice and perform corresponding operation.

Source Code:

```
#include<iostream>
using namespace std;

struct node {
	int data;
	struct node*ptr;
};

class LinkedList {
	private:
		node*head;
		int count;
	public:
		LinkedList() {
			head=NULL;
			count = 0;
		}
		void Append(int);
		void AddAfter(int,int);
		void AddAtBegin(int);
		void Display(int );
		void Count();
		void Delete(int);
		void Reverse();
```

```
        ~LinkedList() {
                delete[]head;
        }
};
void LinkedList::Append(int x) {
    node*new_node=new node();
    new_node->data=x;
    new_node->ptr=NULL;
    if(!head)
        head=new_node;
    else {
        node*temp=head;
        while(temp->ptr)
            temp=temp->ptr;
        temp->ptr=new_node;
    }
}
void LinkedList::AddAfter(int loc,int x) {
    node*new_node=new node(),*temp;
    new_node->data=x;
    new_node->ptr=NULL;
    temp=head;
    if(!temp) {
        cout<<"\n\tThere are less than "<<loc<<" nodes in the list "<<endl;
        delete[]new_node;
    } else {
        if(loc==1) {
            this->AddAtBegin(x);
            return;
        }
        for(int i=1; i<loc; i++) {
            if(!temp) {
                cout<<"\n\tThere are less than "<<loc<<" nodes in the list "<<endl;
                delete[]new_node;
                return;
            }
            temp=temp->ptr;
        }
        new_node->ptr=temp->ptr;
        temp->ptr=new_node;
    }
}
void LinkedList::AddAtBegin(int x) {
    node*new_node=new node();
    new_node->data=x;
    new_node->ptr=head;
    head=new_node;
}
void LinkedList::Display(int flag=1) {
    node*temp=head;
    count=0;
    if(flag)
        cout<<"\nNodes are => \n\t\t";
    while(temp) {
```

```
            if(flag)
                cout<<temp->data<<"--> ";
            temp=temp->ptr;
            count++;
        }
        cout<<" NULL";
        cout<<endl;
}
void LinkedList::Count() {
        this->Display(0);
        cout<<"There are "<<count<<" nodes in the list \n";
}
void LinkedList::Delete(int x) {
        node*temp=head,*prev_node;
        while(temp) {
            if(temp->data==x) {
                if(temp==head) {
                    head=head->ptr;
                    delete[]temp;
                    return;
                } else {
                    prev_node->ptr=temp->ptr;
                    delete[]temp;
                    return;
                }
            }
            prev_node=temp;
            temp=temp->ptr;
        }
        cout<<"\n\n\tInvalid data \n";
}
void LinkedList::Reverse() {
        node*current=head,*prev=NULL,*next=NULL;
        while(current) {
            next=current->ptr;
            current->ptr=prev;
            prev=current;
            current=next;
        }
        head=prev;
}
int main() {
        LinkedList L;
        int choice,value,loc;
        do {
            cout<<"\n\t\t\t Singly Linked List";
            cout<<"\n\t\t\t~~~~~~~~~~~~~~~~~~~~~";
            cout<<"\n\t1. CREATE A NODE";
            cout<<"\n\t2. ADD AT THE BEGINING";
            cout<<"\n\t3. ADD AT END";
            cout<<"\n\t4. ADD AFTER A SPECIFIC LOCATION";
            cout<<"\n\t5. COUNT NO OF NODES";
            cout<<"\n\t6. DELETE A SPECIFIED NODE";
            cout<<"\n\t7. REVERSE THE LIST";
```

```
		cout<<"\n\t8. EXIT";
		cout<<"\n\n\n Enter your choice : ";
		cin>>choice;
		switch(choice) {
			case 1 :
				cout<<"\n Enter a value : ";
				cin>>value;
				L.Append(value);
				L.Display();
				break;
			case 2 :
				cout<<"\n Enter a value : ";
				cin>>value;
				L.AddAtBegin(value);
				L.Display();
				break;
			case 3 :
				cout<<"\n Enter a value : ";
				cin>>value;
				L.Append(value);
				L.Display();
				break;
			case 4 :
				cout<<"\n Enter the location : ";
				cin>>loc;
				cout<<"\n Enter a value : ";
				cin>>value;
				L.AddAfter(loc,value);
				L.Display();
				break;
			case 5 :
				L.Count();
				L.Display();
				break;
			case 6 :
				cout<<"\n Enter a value to delete : ";
				cin>>value;
				L.Delete(value);
				L.Display();
				break;
			case 7:
				cout<< "\n The list before reversal is: \n";
				L.Display();
				L.Reverse();
				cout<<"\n The reversed list is: \n";
				L.Display();
				break;
			case 8 :
				cout<<"\n\n\n\t\t You are on Exit . . . ";
				break;
		}
	} while(choice!=8);
	return 0;
}
```

Sample Input/Output:

```
                Singly Linked List
               ~~~~~~~~~~~~~~~~~~~
    1. CREATE A NODE
    2.ADD AT THE BEGINING
    3. ADD AT END
    4. ADD AFTER A SPECIFIC LOCATION
    5. DISPLAY
    6. COUNT NO OF NODES
    7. DELETE A SPECIFIED NODE
    8. REVERSE THE LIST
    9. EXIT
 Enter your choice : 1

 Enter a value : 1

Nodes are =>
        1-->  NULL

                Singly Linked List
               ~~~~~~~~~~~~~~~~~~~
    1. CREATE A NODE
    2.ADD AT THE BEGINING
    3. ADD AT END
    4. ADD AFTER A SPECIFIC LOCATION
    5. DISPLAY
    6. COUNT NO OF NODES
    7. DELETE A SPECIFIED NODE
    8. REVERSE THE LIST
    9. EXIT


 Enter your choice : 2

Enter a value : 11

Nodes are =>
        11--> 1-->  NULL

                Singly Linked List
               ~~~~~~~~~~~~~~~~~~~
    1. CREATE A NODE
    2.ADD AT THE BEGINING
    3. ADD AT END
    4. ADD AFTER A SPECIFIC LOCATION
    5. DISPLAY
    6. COUNT NO OF NODES
    7. DELETE A SPECIFIED NODE
    8. REVERSE THE LIST
```

9. EXIT

Enter your choice : 3

Enter a value : 33

Nodes are =>
11--> 1--> 33--> NULL

Singly Linked List
~~~~~~~~~~~~~~~~~~~

1. CREATE A NODE
2.ADD AT THE BEGINING
3. ADD AT END
4. ADD AFTER A SPECIFIC LOCATION
5. DISPLAY
6. COUNT NO OF NODES
7. DELETE A SPECIFIED NODE
8. REVERSE THE LIST
9. EXIT

Enter your choice : 4

Enter the location : 2

Enter a value : 5

Nodes are =>
11--> 1--> 5--> 33--> NULL

Singly Linked List
~~~~~~~~~~~~~~~~~~~

1. CREATE A NODE
2.ADD AT THE BEGINING
3. ADD AT END
4. ADD AFTER A SPECIFIC LOCATION
5. DISPLAY
6. COUNT NO OF NODES
7. DELETE A SPECIFIED NODE
8. REVERSE THE LIST
9. EXIT

Enter your choice : 5

Nodes are =>
11--> 1--> 5--> 33--> NULL

Singly Linked List

~~~~~~~~~~~~~~~~~~~

1. CREATE A NODE
2.ADD AT THE BEGINING
3. ADD AT END
4. ADD AFTER A SPECIFIC LOCATION
5. DISPLAY
6. COUNT NO OF NODES
7. DELETE A SPECIFIED NODE
8. REVERSE THE LIST
9. EXIT

Enter your choice : 6

There are 4 nodes in the list

Nodes are =>

11--> 1--> 5--> 33--> NULL

Singly Linked List

~~~~~~~~~~~~~~~~~~~

1. CREATE A NODE
2.ADD AT THE BEGINING
3. ADD AT END
4. ADD AFTER A SPECIFIC LOCATION
5. DISPLAY
6. COUNT NO OF NODES
7. DELETE A SPECIFIED NODE
8. REVERSE THE LIST
9. EXIT

Enter your choice : 7

Enter a value to delete : 5

Nodes are =>

11--> 1--> 33--> NULL

Singly Linked List

~~~~~~~~~~~~~~~~~~~

1. CREATE A NODE
2.ADD AT THE BEGINING
3. ADD AT END
4. ADD AFTER A SPECIFIC LOCATION
5. DISPLAY
6. COUNT NO OF NODES
7. DELETE A SPECIFIED NODE
8. REVERSE THE LIST
9. EXIT
~~~~~~~~~~~~~~~~~~~

Enter your choice : 8

The list before reversal is:

Nodes are =>
11--> 1--> 33--> NULL

The reversed list is:
Nodes are =>
33--> 1--> 11--> NULL

Singly Linked List
~~~~~~~~~~~~~~~~~~

1. CREATE A NODE
2.ADD AT THE BEGINING
3. ADD AT END
4. ADD AFTER A SPECIFIC LOCATION
5. DISPLAY
6. COUNT NO OF NODES
7. DELETE A SPECIFIED NODE
8. REVERSE THE LIST
9. EXIT

Enter your choice : 9

You are on Exit . . .

Result:

Thus, a C++ program was written and executed to create a list and perform operations on the list.
~~~~~~~~~~~~~~~~~~

EX NO:3	QUEUE
DATE:	

Aim:

To write a C++ program to create a queue and perform operations on the queue.

Procedure:

Member Variables in the class Queue

size, front, rear as integer

QueueData as array of integer of size 100

Member Function in the class Queue

Queue() – Constructor to set front and rear as 0

void SetSize(intn) – To set the size of queue

void Enque(int); - To insert value x in to queue

int Deque(void); - To remove a value from queue

void View(void); - To display all elements of queue

Algorithm:

Queue()

1. Initialize the front and rear to 0.

Void SetSize(int n)

1. Set the size of the Queue as n.

void Enque(int x)

1. Check whether there is any space for new element to be added.
2. If there is any space then, place the new element in the position pointed by rear and increment the rear by one.
1. Otherwise, display “Queue is overflow”.

Int Deque(void)

1. Check whether there is any element present in the queue.
2. If there is an element then, store the element present at the location pointed by front in x and increment the front by one and return the value of x.
1. Otherwise, display “Queue is underflow”.

Void Display(void)

1. If there is element, then Display the elements present in the locations pointed by front to rear.
2. Otherwise, display “Queue is empty”.

Int main()

1. Display menu to perform queue operations.
2. Get user choice and perform corresponding operation.

Source Code:

```
//Queue Operation
 #include<iostream>
#include<conio.h>
using namespace std;

class Queue {
        int QueueData[100],size,front,rear;
    public:
        Queue() {
            front=0;
            rear=0;
        }
        void SetSize(int n) {
            size=n;
        }
        void Enque(int);
        int Deque(void);
        void View(void);
};
void Queue::Enque(int x) {
    if((rear-front+1)>size) {
        cout<<
        "\n\t\t Queue over flow...";
    } else {
        QueueData[rear]=x;
        cout<<"\n\t\t"<<x<<" is added.\n";
        rear++;
    }
}
int Queue::Deque(void) {
    int x;
    if(front>=rear) {
        cout<<"\n\t\t Queue under flow...";
        return -9999;
    } else {
        x=QueueData[front];
        front++;
        return x;
    }
}
void Queue::View() {
    int i=front,j=rear;
    if(i>=j) {
        cout<<"\n\t\t Queue is empty...";
    } else {
cout<<"\n\n\t Data in Queue:";
        for(; i<=j-1; i++)
            cout<<" "<<QueueData[i] << ",";
    }
}
int main() {
    int n;
cout<<"\n Enter the size of Queue[MAX=100] : ";
```

```
	cin>>n;
	Queue Q;
	Q.SetSize(n);
	int choice;
	do {
		cout<<"\n\t\t Queue Operation";
		cout<<"\n\t\t ~~~~~~~~~~~~~~~~";
		cout<<"\n\n\t1.INSERT\n\t2.DELETE\n\t3.VIEW\n\t4.EXIT";
		cout<<"\n\n Enter your choice :";cin>>choice;
		int x;
		switch(choice) {
			case 1:cout<<"\n Enter a value : ";
				cin>>x;
				Q.Enque(x);
				Q.View();
				getch();
				break;
			case 2:
				x=Q.Deque();
				if (x!=-9999)
					cout<<"\n\n\t"<<x<<" is deleted.\n\n";
				Q.View();
				getch();
				break;
			case 3:
				Q.View();
				getch();
				break;
			case 4:
				cout<<"\n\n\t\t Exit...";
				getch();
				break;
		}
	} while(choice!=4);
	return 0;
}
```

Sample Input/Output:

```
Enter The size of Queue [MAX=100] : 3
Queue Operations
~~~~~~~~~~~~~~~~
1. INSERT
2. DELETE
3. VIEW
4. EXIT
Enter your choice: 1
Enter a value: 11
11 is added.
Data in Queue : 11
Queue Operations
~~~~~~~~~~~~~~~~
1. INSERT
2. DELETE
3. VIEW
4. EXIT
Enter your choice: 1
Enter a value: 22
22 is added.
Data in Queue : 11 22
Queue Operations
~~~~~~~~~~~~~~~~
1. INSERT
2. DELETE
3. VIEW
4. EXIT
Enter your choice: 1
Enter a value : 33
33 is added.
Data in Queue : 11 22 33
Queue Operations
~~~~~~~~~~~~~~~~
1. INSERT
2. DELETE
3. VIEW
4. EXIT
Enter your choice : 1
Queue over flow...
Data in Queue : 11 22 33
Queue Operations
~~~~~~~~~~~~~~~~
1. INSERT
2. DELETE
3. VIEW
4. EXIT
Enter your choice : 3
Data in Queue : 11 22 33
Queue Operations
```

~~~~~~~~~~~~~~~~
1. INSERT
2. DELETE
3. VIEW
4. EXIT
Enter your choice: 2
11 is deleted.
Data in Queue : 22 33
Queue Operations
~~~~~~~~~~~~~~~~
1. INSERT
2. DELETE
3. VIEW
4. EXIT
Enter your choice: 2
22 is deleted.
Data in Queue : 33
Queue Operations
~~~~~~~~~~~~~~~~
1. INSERT
2. DELETE
3. VIEW
4. EXIT
Enter your choice: 2
33 is deleted.
Queue is empty...
Queue Operations
~~~~~~~~~~~~~~~~
1. INSERT
2. DELETE
3. VIEW
4. EXIT
Enter your choice: 2
Queue under flow... Queue is empty…
Queue Operations
~~~~~~~~~~~~~~~~
1. INSERT
2. DELETE
3. VIEW
4. EXIT
Enter your choice : 4
Exit...

Result:

Thus, a C++ program was written and executed to create a queue and perform operations on queue.
~~~~~~~~~~~~~~~~

EX NO:4(a)	BINARY SEARCH (ITERATION)
DATE:	

Aim:

To write a C++ program with iterative loop to search an element from the given list using binary search technique.

Procedure:

Member Variables in the class Binary

n and key as integer

a as array of integer of size100

Member Function in the class Binary

void GetData(void) - To get the input from user

void Sort(void) - To sort the input in ascending order

void Display(void) - To display all elements of the list

void BinarySearch(void) - To search the key in the given array a using binary search technique

Algorithm:

void GetData(void)

1. Get the list of numbers.
2. Get the Key value to be searched.

Void Sort(void)

1. for i=0 to n-1 do step2
2. for j=i+1 to n do step3
3. if(a[i]>a[j]) exchange a[i] and a[j]

void Display(void)

1. Display elements of the list one by one.

Void BinarySearch(void)

1. Assign:
   a. high= total number of elements in the list-1. Low=0.
2. Until low <= high repeat the following steps,
   a. mid= (high +low)/2.
   b. Check the key value with the value present at the middle index position of the
   c. sorted list.
   d. If present, display its position in the list and stop the process.
   e. Else if key value < the value present in the middle position, then
   f. high=mid+1;
   g. Else if key value > the value present in the middle position, then
   h. low=mid+1;
3. Display “The key value is not present in the list”.

Int main()

1. Create an object B of class Binary.
2. Call member function GetData() of object B.
3. Call member function Sort() of object B.
4. Call member function Display() of object B.
5. Call member function BinarySearch() of object B.

Source Code:

```
// Binary Search
#include <iostream>
using namespace std;

class Binary {
        int a[100],n,key;
    public:
        void GetData(void);
        void Sort(void);
        void Display(void);
        void BinarySearch(void);
};
void Binary::GetData(void) {
cout<<"\nEnter the size of the list to be searched :";
    cin>>n;
cout<<"\nEnter the elements of the list :";
    for(int i=0; i<n; i++) {
        cin>>a[i];
    }
cout<<"Enter the key value to be searched in the above list :"; cin>>key;
}
void Binary::Sort(void) {
    for(int i=0; i<n-1; i++)
        for(int j=i+1; j<n; j++) {
            if(a[i]>a[j]) {
                int temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
        }
}
void Binary::Display(void) {
    for(int i=0; i<n; i++)
        cout<<" Element "<< i+1<<"is"<< a[i]<<endl;
}
void Binary::BinarySearch(void) {
    int high,low,mid;
    high=n-1;
    low=0;
    while (low<=high) {
        mid=(low+high)/2;
        if(a[mid]==key) {
            cout<<"\n The key value "<< key <<" is present at location "<<mid+1;
            return;
        } else if (key<a[mid]) {
            high=mid-1;
        } else {
            low=mid+1;
        }
    }
    cout<<"\n The key value "<< key<<" is not present in the above list";
}
```

```
int main() {
    Binary B;
    B.GetData();
    B.Sort();
    cout<<"\n The Sorted List \n";
    cout<<" \n";
    B.Display();
    B.BinarySearch();
    return 0;
}
```

Sample Input/Output:

Enter the size of the list to be searched: 11
Enter the elements of the list.: -12 2 3 1 -15 3 52 4 4 66 -11
Enter the key value to be searched in the above list: 3
The Sorted List
Element 1 is -15.
Element 2 is -12.
Element 3 is -11.
Element 4 is 1.
Element 5 is 2.
Element 6 is 3.
Element 7 is 4.
Element 8 is 4.
Element 9 is 5.
Element 10 is 23.
Element 11 is 66.
The key value 3 is present at location 6.

Result:

Thus, a C++ program to search a key element from the given list by binary search technique was written, executed and output was verified.

EX NO:4(b)	BINARY SEARCH (RECURSION)
DATE:	

Aim:

To write a C++ program with recursion to search an element from the given list using binary search technique.

Procedure:

Member variables in the class Binary

n and key as integer
a as array of integer of size100

Member Functions in the class Binary

int GetData(void)	- To get the input from user and return the number of elements in the list
void Sort(void)	- To sort the input in ascending order
void Display(void)	- To display all elements of the list
void RecursiveBsearch(int, int)	- To search the key in the given array a using recursive binary search

Algorithm:

int GetData(void)

1. Get the list of numbers.
2. Get the Key value to be searched.
3. Return the number of elements in the list.

Void Sort(void)

1. for i=0 to n-1 do step 2
2. for j=i+1 to n do step 3
3. if(a[i]>a[j]) exchange a[i] and a[j]

void Display(void)

0. Display elements of the list one by one.

Int RecursiveBsearch(int start, int end)

1. If start > end, then return -1, since element is not found.
2. Else do the following:
   (i) Find the position of middle element as mid= (start +end)/2
   (ii) If middle element is equal to the key, then return the position of middle element.
   (iii) Else if key is lesser than middle element, then search only the lower (left) half of the list by calling the function recursively as RecursiveBsearch(start, mid- 1)
   (iv) Else if key is greater than middle element, then search only the upper (right) half of the list by calling the function recursively as RecursiveBsearch(mid+1, end)

int main()

1. Create an object B of class Binary.
2. Call member function GetData() of object B.
3. Call member function Sort() of object B.
4. Call member function Display() of object B.
5. Call member function RecursiveBsearch(0,N-1) of object B.

Source Code:

```
#include <iostream>
using namespace std;

class Binary {
        int a[100],n,key;
    public:
        int GetData(void);
        void Sort(void);
        void Display(void);
        int RecursiveBsearch(int, int);
};
int Binary::GetData(void) {
    cout<<"\nEnter the size of the list to be searched :";
    cin>>n;
    cout<<"\n Enter the elements of the list :";
    for(int i=0; i<n; i++) {
        cin>>a[i];
    }
    cout<<"Enter the key value to be searched in the above list :";
    cin>>key;
    return n;
}
void Binary::Sort(void) {
    for(int i=0; i<n-1; i++)
        for(int j=i+1; j<n; j++) {
            if(a[i]>a[j]) {
                int temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
        }
}
void Binary::Display(void) {
    for(int i=0; i<n; i++)
        cout<<" Element "<< i+1<<"is "<< a[i]<<"\n";
}
int Binary::RecursiveBsearch(int start, int end) {
    if(start>end)
        return -1;
    int mid = (start+end)/2;
    if( a[mid] == key )
        return mid;
```

```
        else if( key < a[mid] )
                RecursiveBsearch(start, mid-1);
        else
                RecursiveBsearch(mid+1, end);
        return 0;
  }
  int main() {
        Binary B;
        int index = -1, N;
        N=B.GetData();
        B.Sort();
        cout<<"\n The Sorted List \n";
        cout<<"\n";
        B.Display();
        index=B.RecursiveBsearch(0,N-1);
        if (index>=0)
                cout<<"\n The key value is present at location "<<index+1;
        else
                cout<<"\n The key value is not present in the list";
        return 0;
  }
```

Sample Input/Output:

```
Enter the size of the list to be searched: 11
Enter the elements of the list.: -12 2 3 1 -15 3 52 4 4 66 -11
Enter the key value to be searched in the above list: 3
The Sorted List
Element 1 is -15.
Element 2 is -12.
Element 3 is -11.
Element 4 is 1.
Element 5 is 2.
Element 6 is 3.
Element 7 is 4.
Element 8 is 4.
Element 9 is 5.
Element 10 is 23.
Element 11 is 66.
The key value is present at location 6.
```

Result:

Thus, a C++ program to search a key element from the given list by binary search technique was written, executed and output was verified.

EX NO:5(a)	GRAPH TRAVERSAL(DEPTH FIRST SEARCH)
DATE:	

Aim:

To write a program to traverse a graph using Depth First Search (DFS) Algorithm.

Procedure:

Member Variables in the class Graph

n and s as integer

a as two dimensional array of integer of size[10][10]

Member Function in the class Graph

void getdata() - To get the input from user

void dfs_traverse() - To traverse a graph from one node to another node in depth first order.

Algorithm:

void getdata()

1. Get the number of vertices in the graph.
2. Get the adjacency matrix of the graph.
3. Get the starting vertex for traversal.

Void dfs_traverse()

1. Push starting vertex into a stack and make it visited.
2. Pop a vertex from stack.
3. Find all adjacent vertices of popped vertex in the reverse order in such a way that it is not already visited.
4. Push them into stack and make them visited.
5. Repeat steps 2 to 4 until the stack is empty.

Int main()

1. Call the getdata( ) function to get input from user.
2. Call the dfs_traverse( ) function to do the traversal operation.

Source Code:

```
//Depth First Search
#include<iostream>
using namespace std;

class Graph {
        int a[10][10],n,s;
    public:
        void getdata();
        void dfs_traverse();
};
void Graph::getdata() {
cout<<"\n Enter the number of vertices in the graph:";cin>>n;
cout<<"\n Enter the adjacency matrix of graph:"<<endl;
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
```

```
                cin>>a[i][j];
cout<<"\n Enter the vertex from which you want to traverse:";cin>>s;
}
void Graph::dfs_traverse() {
    int *visited=new int[n];
    int stack[10],top=-1,I;
    for(int j=0; j<n; j++)
        visited[j]=0;
cout<<"\n The Depth First Search Traversal :"<<endl;
    int i=stack[++top]=s;
    visited[s]=1;
    while(top>=0) {
        i=stack[top];
        cout<<stack[top--]<<endl;
        for(int j=n-1; j>=0; j--)
            if(a[i][j]!=0&&visited[j]!=1) {
                stack[++top]=j;
                visited[j]=1;
            }
    }
}
int main() {
    Graph DFS;
    DFS.getdata();
    DFS.dfs_traverse();
    return 0;
}
```

Sample Input/ Output:

```
Enter the number of vertices in the graph: 6
Enter the adjacency matrix of graph:
0 1 1 0 0 0
1 0 0 1 1 0
1 0 0 0 1 0
0 1 0 0 1 1
0 1 1 1 0 1
0 0 0 1 1 0
Enter the vertex from which you want to traverse: 0
The Depth First Search Traversal:
0
1
3
5
4
2
```

Result:

Thus, a C++ program to traverse a graph using Depth First Search algorithm was written, executed and output was verified.

EX NO:5(b)	GRAPH TRAVERSAL (BREADTH FIRST SEARCH)
DATE:	

Aim:

To write a C++ program to traverse a graph using Breadth First Search (BFS) algorithm.

Procedure:

Member Variables in the class Bgraph

n and start as integer

a as two dimensional array of integer of size[10][10]

Member Function in the class Bgraph

voidgetdata() - To get the input from user

voidbfs_traverse() - To traverse a graph from one node to another node in breadth first order

Algorithms:

void getdata()

1. Get the number of vertices in the graph.
2. Get the adjacency matrix of the graph.
3. Get the starting vertex for traversal.

Void bfs_traverse()

1. Insert starting vertex into a queue and make it visited.
2. Remove a vertex from queue.
3. Find all adjacent vertices of removed vertex in such a way that it is not already visited.
4. Insert them into queue and make them visited.
5. Keep repeating steps 2 to 4 until the queue is empty.

Int main()

1. Call the getdata( ) function to get input from user.
2. Call the bfs_ traverse( ) function to do the traversal operation.

Source Code:

```
//Breadth First Search
#include<iostream>
using namespace std;

class Bgraph {
        int a[10][10],n,start;
    public:
        void getdata();
        void bfs_traverse();
};
void Bgraph::getdata() {
    cout<<"\n Enter the number of vertices in the graph:";
    cin>>n;
    cout<<"\n Enter the adjacency matrix of graph:"<<endl;
```

```
        for(int i=0; i<n; i++)
                for(int j=0; j<n; j++)
                        cin>>a[i][j];
        cout<<"\n Enter the vertex from which you want to traverse:";
        cin>>start;
}
void Bgraph::bfs_traverse() {
        int *visited= new int[n];
        int queue[10],front=-1,rear=0,I;
        for(int j=0; j<n; j++)
                visited[j]=0;
        cout<<"\n Traversing the graph using breadth first search algorithm :"<<endl;
        queue[rear]=start;
        visited[start]=1;
        while(front!=rear) {
                cout<<queue[++front]<<endl;
                int i=queue[front];
                for(int j=0; j<n; j++)
                        if(a[i][j]!=0&&visited[j]!=1) {
                                queue[++rear]=j;
                                visited[j]=1;
                        }
        }
}
int main() {
        Bgraph bfs;
        bfs.getdata();
        bfs.bfs_traverse();
        return 0;
}
```

Sample Input/Output:

Enter the number of vertices in the graph: 5
Enter the adjacency matrix of graph:
0 1 1 0 0
1 0 0 1 0
1 0 0 1 1
0 1 1 0 0
0 0 1 1 0
Enter the vertex from which you want to traverse: 1
Traversing the graph using breadth first search algorithm:
1
0
3
2
4

Result:

Thus, a C++ program to traverse a graph using Breadth First Search algorithm was written, executed and output was verified.

EX NO:6	BINARY SEARCH TREE
DATE:	

Aim:

To write a C++ program to create a binary search tree and perform in-order, pre-order and post-order traversals.

Procedure:

Create user defined data type called node using structure in C++ with the following members:
data as integer
left as a pointer to hold the address of the left child.
right as a pointer to hold the address of the right child.

Member Variables in the class BST

Root as pointer to the root node.

Member function in the class BST

BinarySearchTree() - Constructor to initialize Root as NULL
~BinarySearchTree() - Destructor to release the memory of Root
BSTnode*GetRoot(void) - To know the Root
void Insert(int) - To insert a new node in binary search tree
voidPreorder(Node*temp) - To traverse in preorder
voidInorder(Node*temp) - To traverse in inorder void
Postorder(Node *temp) - To traverse in preorder void
Display(Node *temp,int row,int low,int high) -To display the nodes in tree structure

Algorithm:

BinarySearchTree()

1. Initialize Root as NULL

~BinarySearchTree()

1. Release the memory of Root

BSTnode *GetRoot(void)

1. Return the Root

void Insert(int)

1. If the tree is empty or left-address field and right-address field is holding NULL values, then store the value in root node itself.
2. Compare the new with the root for its place until the leaf node is reached by traverse through left child or right child based on the root data.
3. If the new value is greater than the leaf node value then, place the data in a new node and make the right-address field of the leaf node to hold the address of the new node.
4. If the new value is lesser than the leaf node value then, place the data in a new node and make the left-address field of the leaf node to hold the address of the new node.

void Preorder(Node *temp)

1. Display the present root node's value
2. Traverse the left sub-tree by taking the left node as new root node in Preorder.
3. Traverse the right sub-tree by taking the left node as new root node in Preorder.

void Inorder(Node *temp)

1. Traverse the left sub-tree by taking the left node as new root node in Inorder.
2. Display the present root node's value.
3. Traverse the right sub-tree by taking the left node as new root node in Inorder.

void Postorder(Node *temp)

1. Traverse the left sub-tree by taking the left node as new root node in Postorder.
2. Traverse the right sub-tree by taking the left node as new root node in Postorder.
3. Display the present root node's value.

void Display(Node *temp,int row,int low,int high)

1. Find the place to print the data and print it to display like a binary search tree.

int main()

1. Display menu to perform binary search tree creation and it's traversals.
2. Get user choice and perform corresponding operation.

Source Code:

```
#include<iostream>
#include<conio.h>
using namespace std;

typedef struct Node {
   int data;
   Node *LNode, *RNode, *left, *right;
} Node;

class BST {
public:
   Node* root;
   BST() {
      root = NULL;
   }

   void Insert(int);
   void Preorder(Node*);
   void Inorder(Node*);
   void Postorder(Node*);
   void Display(Node*, int);

   Node* getRoot() {
      return root;
   }

   ~BST() {
      delete root;
   }
};

void BST::Insert(int x) {
   Node* temp = new Node();
   temp->data = x;
   temp->LNode = NULL;
   temp->RNode = NULL;
```

```
    if (root == NULL) {
        root = temp;
        return;
    } else {
        Node* prev, *current;
        current = root;
        while (current != NULL) {
            prev = current;
            if (current->data == x)
                return;
            else if (current->data > x)
                current = current->LNode;
            else
                current = current->RNode;
        }
        if (prev->data > x)
            prev->LNode = temp;
        else
            prev->RNode = temp;
    }
}

void BST::Preorder(Node* temp) {
    if (temp != NULL) {
        cout << "-> " << temp->data << " ";
        Preorder(temp->LNode);
        Preorder(temp->RNode);
    }
}

void BST::Inorder(Node* temp) {
    if (temp != NULL) {
        Inorder(temp->LNode);
        cout << "-> " << temp->data << " ";
        Inorder(temp->RNode);
    }
}

void BST::Postorder(Node* temp) {
    if (temp != NULL) {
        Postorder(temp->LNode);
        Postorder(temp->RNode);
        cout << "-> " << temp->data << " ";
    }
}

void BST::Display(Node* temp, int level) {
```

```
    if (temp != NULL) {
        Display(temp->RNode, level + 1);

        for (int i = 0; i < level; i++) {
            cout << "      ";
        }

        cout << temp->data << endl;

        Display(temp->LNode, level + 1);
    }
}

int main() {
    BST B;
    int ch;
    while (1) {
        cout << "\n\t\t\tBinary Search Tree";
        cout << "\n\t\t\t ~~~~~~~~~~~~~~~~~~";
        cout << "\n1-Create";
        cout << "\n2-Preorder";
        cout << "\n3-Inorder";
        cout << "\n4-Postorder";
        cout << "\n5-Display";
        cout << "\n6-Exit\n";
        cout << "\nEnter your choice : ";
        cin >> ch;
        switch (ch) {
        case 1:
            cout << "\nEnter elements 1 by 1 (0 to stop entering)\n";
            int x;
            do {
                cin >> x;
                B.Insert(x);
            } while (x != 0);
            break;
        case 2:
            cout << "\n~~~~ PreOrder Traversal ~~~~ \n\nThe Tree is:\n";
            B.Preorder(B.getRoot());
            break;
        case 3:
            cout << "\n~~~Inorder Traversal~~~\n\nThe Tree is:\n";
            B.Inorder(B.getRoot());
            break;
        case 4:
            cout << "\n~~~Postorder Traversal~~~\n\nThe Tree is:\n";
            B.Postorder(B.getRoot());
```

```
            break;
        case 5:
            cout << "\n~~~~~DISPLAY~~~~~~~~\n\n";
            B.Display(B.getRoot(), 0);
            break;
        case 6:
            cout << "\n~~~Exit~~~\n";
            return 0;
        default:
            cout << "\nEnter between 1 to 5\n\n";
            break;
        }
    }
}
```

Sample Input/Output:

```
Binary Search Tree
~~~~~~~~~~~~~~
1- Create
2- Preorder
3- Inorder
4- Postorder
5- Display
6- Exit
Enter your choice: 1
Enter elements1 by1:(0to stopentering)
100
70
120
50
80
110
130
5
55
75
85
105
115
125
135
0
Binary Search Tree
~~~~~~~~~~~~~~
1- Create
2- Preorder
3-Inorder
4-Postorder
5-Display
6-Exit
Enter your choice : 2
~~~Preorder Traversal~~~
The Tree is:->100->70->50->5->55->80->75->85->120->110->105->115->130->125 ->135
Binary Search Tree
~~~~~~~~~~~~~~~
1- Create
2- Preorder
3- Inorder
4- Postorder
5- Display
6- Exit
Enter your choice : 3
~~~Inorder Traversal~~~
The Tree is:->5->50->55->70->75->80->85->100->105->110->115->120->125->130->135
```

```
Binary Search Tree
~~~~~~~~~~~~~~
1- Create
2- Preorder
3- Inorder
4- Postorder
5- Display
6- Exit
Enter your choice :4
~~~Postorder Traversal~~~
The Tree is:->5->55->50->75->85->80->70->105->115->110->125->135->130->120 ->100
Binary Search Tree
~~~~~~~~~~~~~~
1- Create
2- Preorder
3- Inorder
4- Postorder
5- Display
6- Exit
Enter your choice : 5
                    135
              130
                    125
        120         115
              110
                    105
100
                     85
               80
         70          75
                     55
               50
                     5
Binary Search Tree
~~~~~~~~~~~~~~
1- Create
2- Preorder
3- Inorder
4- Postorder
5- Display
7- Exit
Enter your choice : 6
~~~Exit~~~
```

Result:

Thus, a C++ program to create the binary search tree and perform in-order, pre-order and post-order traversals was written, executed and output was verified

EX NO:7	BUBBLE SORT
DATE:	

Aim:

To write a C++ program to sort the given list of numbers using bubble sort technique.

Procedure

Member Variables in the Base class DataArray

n as integer

a as array ofinteger of size100

Member Function in the Base class DataArray

voidGetData(void) - To get the input from user

voidDisplay(void) - To display elements of the list

Algorithms:

void GetData(void)

1. Get the list of

numbers.void Display(void)

1. Display all elements of the list one by one.

Create a derived class Bubble, derived from the base class DataArray with public access.

Member variable in the derived class Bubble is None

Member Function in the derived class Bubble

void BubbleSort(void) - To sort elements in ascending order

Algorithms:

void BubbleSort(void)

1. Compare pairs of adjacent elements from the first location towards right for theentire list with the size of the list decreasing by one for each repetition.
2. For each comparison, if the first element is greater than the second element theninterchange it's position.
3. Repeat the above steps 1 and 2 until no interchanges has happened during anycomparison from left to right

int main()

1. Create an object B of classBubble.
2. Call member function GetData() of object B.
3. Call member function BubbleSort() of object B.
4. Call member function Display() of object B.

Source Code:

```
// Bubble sorting
#include<iostream>
using namespace std;
class DataArray {
        protected:
                int a[100],n;
        public:
                void GetData(void);
                void Display(void);
};
```

```
void DataArray::GetData(void) {
        cout<<"\n Enter the number of elements to be sorted : ";
        cin>>n;
        for(int i=0; i<n; i++) {
                cout<<" Enter element no."<<i+1<<" :";
                cin>>a[i];
        }
}
void DataArray::Display(void) {
        for(int i=0; i<n; i++)
                cout<<" "<<a[i];
        cout<<endl;
}
class Bubble : public DataArray {
        public:
                void BubbleSort(void);
};
void Bubble::BubbleSort(void) {
        int flag=1,i,j,temp;
        for(int i=0; i<n-1; i++) {
                for(int j=0; j<n-i-1; j++)
                        if(a[j]>a[j+1]) {
                                temp=a[j];
                                a[j]=a[j+1];
                                a[j+1]=temp;
                                flag=0;
                        }
                cout<<"\n Array after iteration "<<i<<": ";
                Display();
                if(flag)
                        break;
                else
                        flag=1;
        }
}
int main() {
        Bubble B;
        B.GetData();
        B.BubbleSort();
        cout<<"\n\t The Sorted Array";
        cout<<"\n\t ~~~~~~~~~~~~~~~~\n";
        B.Display();
        return 0;
    }
```

Sample Input/Output:

```
Enter the number of elements to be sorted:
10Enter element no. 1: 19
Enter element no. 2: 77
Enter element no. 3: -60
Enter element no. 4: -12
Enter element no. 5: 7
Enter element no. 6: 41
Enter element no. 7: 2
Enter element no. 8: 1
Enter element no. 9: 99
Enter element no. 10: 0
Array after iteration 0: 19 -60 -12 7 41 2 1 77 0 99
Array after iteration 1: -60 -12 7 19 2 1 41 0 77 99
Array after iteration 2: -60 -12 7 2 1 19 0 41 77 99
Array after iteration 3: -60 -12 2 1 7 0 19 41 77 99
Array after iteration 4: -60 -12 1 2 0 7 19 41 77 99
Array after iteration 5: -60 -12 1 0 2 7 19 41 77 99
Array after iteration 6: -60 -12 0 1 2 7 19 41 77 99
Array after iteration 7: -60 -12 0 1 2 7 19 41 77
99The Sorted Array
~~~~~~~~~~~~~
-60 -12 0 1 2 7 19 41 77 99
```

Result:

Thus, a C++ program to sort the given list by bubble sort technique was written,executed and output was verified.

EX NO:8	SEPARATE CHAINING IN HASHING
DATE:	

Aim:

To write a C++ program to implement separate chaining to handle collisions in hashing.

Procedure:

Classes

HashNode
HashMap

Data members in the class HashNode

key, value as integers
next as pointer to HashNode

Member function in the class HashNode

HashNode(int key, int value) – to initialize the node

Data members of the class HashMap

htable as pointer to HashNode

Member Function in the class HashMap

HashMap()	- to allocate memory and create new node in the Hash table
~HashMap()	- to delete the entries in the Hash table and deallocate memory
int HashFunc(int key)	- to compute the hash key
void Insert(int key, int value)	- to insert an element at key
void Remove(int key)	- to remove Element at a key
int Search(int key)	- to search an element at key

main() Function

1. For the class HashMap, create an object hash.
2. Display menu to accept choice from user.
3. Based on choice, invoke Insert or Remove or Search function for the object hash.

Algorithms:

A Hash Table in C/C++ is a data structure that maps keys to values. This uses a hash function to computed indexes for a key. Based on the Hash Table index, we can store the value at the appropriate location. If two different keys get the same index, we need to use other data structures (buckets) to account for these collisions. Open Hashing is a collision avoidance method which uses Array of linked list to resolve the collision. It is also known as the separate chaining method (each linked list is considered as a chain). The algorithms to insert/remove/search are written based on separate chaining method to resolve collision.

Insert data into the hash table:

1. Declare an array of a linked list with the hash table size.
2. Initialize an array of a linked list to NULL.
3. Find hash key. (where, hash key = value / table size)
4. If htable[hash_val] == NULL, make htable[hash_val] point to the key node.
5. Otherwise (i.e., there is collision), Insert the key node at the end of the htable[hash_val].

Searching a value in the hash table:

1. Get the value.
2. Compute the hash key.
3. Search the value in the entire hash table. i.e. htable[hash_val].
4. If found, print the element found at key.
5. Otherwise, print "No element found at key".

Deleting/Removing an element from the hash table:

1. Get the value
2. Compute the key.
3. Using linked list deletion algorithm, delete the element from the htable[hash_val] (by using
the Linked List Deletion Algorithm for deleting a node in the linked list)
4. If unable to delete, print "No element found at key".

Source Code:

```
#include<iostream>
#include<cstdlib>
#include<string.h>
#include<cstdio>
using namespace std;

const int TABLE_SIZE = 128;
bool value1 = false;
bool value2 = true;
class HashNode {
	public:
		int key;
		int value;
		HashNode* next;
		HashNode(int key, int value) {
			this->key = key;
			this->value = value;
			this->next = NULL;
		}
};
class HashMap {
	private:
		HashNode** htable;
	public:
		HashMap() {
			htable = new HashNode*[TABLE_SIZE];
			for (int i = 0; i < TABLE_SIZE; i++)
				htable[i] = NULL;
		}
		~HashMap() {
			for (int i = 0; i < TABLE_SIZE; ++i) {
				HashNode* entry = htable[i];
				while (entry != NULL) {
					HashNode* prev = entry;
					entry = entry->next;
```

```
                delete prev;
            }
        }
        delete[ ] htable;
}
int HashFunc(int key) {
        return key % TABLE_SIZE;
}
void Insert(int key, int value) {
        int hash_val = HashFunc(key);
        HashNode* prev = NULL;
        HashNode* entry = htable[hash_val];
        while (entry != NULL) {
            prev = entry;
            entry = entry->next;
        }
        if (entry == NULL) {
            entry = new HashNode(key, value);
            if (prev == NULL) {
                htable[hash_val] = entry;
            } else {
                prev->next = entry;
            }
        } else {
            entry->value = value;
        }
}
void Remove(int key) {
        int hash_val = HashFunc(key);
        HashNode* entry = htable[hash_val];
        HashNode* prev = NULL;
        if (entry == NULL || entry->key != key) {
            cout<<"No Element found at key "<<key<<endl;
            return;
        }
        while (entry->next != NULL) {
            prev = entry;
            entry = entry->next;
        }
        if (prev != NULL) {
            prev->next = entry->next;
        }
        delete entry;
        cout<<"Element Deleted"<<endl;
}
int Search(int key) {
        bool flag = false;
        int hash_val = HashFunc(key);
        HashNode* entry = htable[hash_val];
        while (entry != NULL) {
            if (entry->key == key) {
                cout<<entry->value<<" ";
                flag = true;
            }
```

```
                        entry = entry->next;
                }
                if (!flag)
                        return -1;
        }
};
int main() {
        HashMap hash;
        int key, value;
        int choice;
        while (1) {
                cout<<"\n ---------------------- "<<endl;
                cout<<"Operations on Hash Table"<<endl;
                cout<<"\n ---------------------- "<<endl;
                cout<<"1.Insert element into the table"<<endl;
                cout<<"2.Search element from the key"<<endl;
                cout<<"3.Delete element at a key"<<endl;
                cout<<"4.Exit"<<endl;
                cout<<"Enter your choice: ";
                cin>>choice;
                switch(choice) {
                        case 1:
                                cout<<"Enter element to be inserted: ";
                                cin>>value;
                                cout<<"Enter key at which element to be inserted: ";
                                cin>>key;
                                hash.Insert(key, value);
                                break;
                        case 2:
                                cout<<"Enter key of the element to be searched: ";
                                cin>>key;
                                cout<<"Element at key "<<key<<" : ";
                                if (hash.Search(key) == -1) {
                                        cout<<"No element found at key "<<key<<endl;
                                        continue;
                                }
                                break;
                        case 3:
                                cout<<"Enter key of the element to be deleted: ";
                                cin>>key;
                                hash.Remove(key);
                                break;
                        case 4:
                                exit(1);
                        default:
                                cout<<"Enter correct option\n";
                }
        }
        return 0;
}
```

Sample input/Output:

```
Operations on Hash Table
-------------------------
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 1
Enter element to be inserted: 12
Enter key at which element to be inserted: 1
-------------------------
Operations on Hash Table
-------------------------
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 1
Enter element to be inserted: 24
Enter key at which element to be inserted: 1
-------------------------
Operations on Hash Table
-------------------------
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 1
Enter element to be inserted: 36
Enter key at which element to be inserted: 1
-------------------------
Operations on Hash Table
-------------------------
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 1
Enter element to be inserted: 48
Enter key at which element to be inserted: 2
-------------------------
Operations on Hash Table
-------------------------
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 1
```

```
Enter element to be inserted: 60
Enter key at which element to be inserted: 2
-------------------------
Operations on Hash Table
-------------------------
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 2
Enter key of the element to be searched: 1
Element at key 1 : 12 24 36
-------------------------
Operations on Hash Table
-------------------------
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 2
Enter key of the element to be searched: 2
Element at key 2 : 48 60
-------------------------
Operations on Hash Table
-------------------------
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 3
Enter key of the element to be deleted: 4
No Element found at key 4
Operations on Hash Table
-------------------------
1. Insert element into the table
2. Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 2
Enter key of the element to be searched: 1
Element at key 1 : 12 24 36
-------------------------
Operations on Hash Table
-------------------------
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 3
Enter key of the element to be deleted: 1
Element Deleted
```

Operations on Hash Table

1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 2
Enter key of the element to be searched: 5
Element at key 5 : No element found at key 5

Operations on Hash Table

1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 2
Enter key of the element to be searched: 1
Element at key 1 : 12 24

Operations on Hash Table

1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 1
Enter element to be inserted: 36
Enter key at which element to be inserted: 1

Operations on Hash Table

1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 2
Enter key of the element to be searched: 1
Element at key 1 : 12 24 36

~~Operations on Hash Table ----~~

~~1.Insert element into the table --~~
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 2
Enter key of the element to be searched: 2
Element at key 2 : 48 60

```
-------------------------
Operations on Hash Table
-------------------------
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice:4
```

Result:

Thus, a C++ program was written to implement separate chaining on hash tables with singly linked lists and executed to verify the output.

EX NO:9	HEAP SORT
DATE:	

Aim:

To write a C++ program to implement a Heap and perform Heap Sort.

Procedure:

Variables used in the main() Function:

n - number of elements in the array
arr[] - array to hold the elements

Function Used:

void heapify(int arr[], int n, int i)	- To heapify a subtree rooted with node i which is an index in arr[], n is the size of heap
void heapSort(int arr[], int n)	- To do heap sort on the array elements
void printArray(int arr[], int n)	- To display the array elements
int main()	- To accept user input, invoke function to sort the heap, and display the sorted list of numbers.

Algorithms:

void heapify(int arr[], int n, int i)

1. Initialize largest as root
2. Set left = 2*i + 1, right = 2*i + 2
3. If left child is larger than root, then set largest = l
4. If right child is larger than largest so far, then set largest = r
5. If largest is not root, then
    i. Interchange arr[i] and arr[largest]
    ii. Recursively call heapify() function to heapify the affected sub-tree

void heapSort(int arr[], int n)

1. Build heap by calling the function heapify() to rearrange array elements.
2. Display the heap created.
3. One by one extract an element from heap by doing the following:
    i) Move current root to end
    ii) call max heapify on the reduced heap

void printArray(int arr[], int n)

Display the array elements stored in arr[].

int main()

1. Accept the number of elements and array elements from the user.
2. Invoke function heapSort(arr, n) to create and sort the heap elements.
3. Display the sorted list.

Source Code:

```
// C++ program for implementation of Heap Sort
#include <iostream>
```

```
using namespace std;

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 l = 2*i + 1;
	int r = 2*i + 2;
	if (l < n && arr[l] > arr[largest])
		largest = l;
	if (r < n && arr[r] > arr[largest])
		largest = r;
	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);
	cout << "The heap is : \n";
	for (int i = 0; i < n; ++i)
		cout << arr[i] << " ";
	cout << "\n";
	for (int i=n-1; i>0; i--) {
		swap(arr[0], arr[i]);
		heapify(arr, i, 0);
	}
}
void printArray(int arr[], int n) {
	for (int i=0; i<n; ++i)
		cout << arr[i] << " ";
	cout << "\n";
}
int main() {
	int n, i, arr[100];
	cout << "\n Enter the number of elements in the list: ";
	cin >> n;
	for (i=0; i<n; i++) {
		cout << "\n Enter the element no. " << i+1 << ": ";
		cin >> arr[i];
	}
	heapSort(arr, n);
	cout << "\n Sorted array is \n";
	printArray(arr, n);
	return(0);
}
```

Source Code:

Sample Input/Output:

```
Enter the number of elements in the list: 6
Enter the element no. 1: 12
Enter the element no. 2: 11
Enter the element no. 3: 13
Enter the element no. 4: 5
Enter the element no. 5: 6
Enter the element no. 6: 7
The Heap is:
13 11 12 5 6 7
Sorted array is
5 6 7 11 12 13
```

Result:

Thus, a C++ program was written to implement heap and perform heap sort on a list of numbers and executed to verify the output.

EX NO:10	QUICK SORT
DATE:	

Aim:

To write a C++ program to sort the given list by quick sort technique using divide and conquer method.

Procedure:

Create the base class DataArray with the following member variables and functions.

Member variables in the base class DataArray

n as integer
a as array of integer of size100

Member function in the base class DataArray

int Getn(void) - To return value of n
void GetData(void) - To get the input from user
void Display(void) - To display elements of the list

Algorithm:

void GetData(void)

1. Get the list of numbers.

void Display(void)

1. Display all elements of the list one by one.

Create a derived class Quick, derived from the base class DataArray with public access.

Member variables in the derived class Quick is none

Member Function in the derived class Quick

void QuickSort(int low, int high) - To divide the list into two part based on pivot element
void QuickDisplay(int l, int h) - To Display elements from l to h

Algorithm:

void QuickSort(int low, int high)

1. Set the first element in the list as pivot element.
2. Assume two pointers left and right moving from either end of the list towards each other
3. Move left pointer until finding the element larger than the pivot element's value.
4. Move the right pointer until finding the element not larger than the pivot element value.
5. Swap the contents of the location pointed by these two pointers.
6. Repeat the steps 3, 4 and 5 until the pointers crossed each other.
7. Swap the contents of the location pointed by pivot pointer and right pointer.
8. Repeat the above steps from 1 to 8 by considering two new lists, partitioned by the pivot pointer until further partitioning is not possible.

void QuickDisplay(int l, int h)

1. Display elements from l to h in the list one by one.

Source Code:

```
#include<iostream>
#include<conio.h>
using namespace std;

class data_array{
	protected:
		int a[100],n;
	public:
		int getn(void) {
			return n;
		}
		void getdata(void);
		void display(void);
};
void data_array::getdata(void) {
	cout<<"\n Enter the number of element to be sorted:";
	cin>>n;
	for(int i=0; i<n; i++) {
		cout<<"Enter the element number "<<i+1<<":";
		cin>>a[i];
	}
}
void data_array::display(void) {
	for( int i=0; i<n; i++)
		cout<<" "<<a[i];
	cout<<endl;
}
class quick:public data_array {
	public:
		void quicksort(int low,int high);
		void quickdisplay(int l,int h);
};
void quick::quicksort(int low,int high) {
	int left=low+1;
	int right=high;
	int temp=0;
	int pivot=a[low];
	if (low>high)return;
	do {
		while(a[left]<pivot&&left<high)left++;
		while(a[right]>=pivot&&right>low)right--;
		if(left<right) {
			temp=a[left];
			a[left]=a[right];
			a[right]=temp;
		} else break;
	} while(1);
	temp=a[low];
	a[low]=a[right];
	a[right]=temp;
	cout<<"left:";
	quickdisplay(low,right-1);
```

```
        cout<<"pivot:[ "<<pivot<<" ]";
        cout<<"right:";
        quickdisplay(right+1,high);
        cout<<endl;
        quicksort(low,right-1);
        quicksort(right+1,high);
}
void quick::quickdisplay(int l,int h) {
        cout<<"{";
        for(int i=l; i<=h; i++)
                cout<<" "<<a[i]<<"";
        cout<<" }";
}
int main() {
        quick q;
        q.getdata();
        cout<<"Original:";
        q.quickdisplay(0,q.getn()-1);
        cout<<endl;
        q.quicksort(0,q.getn()-1);
        cout<<"\n\t The Sorted Array ";
        cout<<"\n\t~~~~~~~~~~~~~~~~~\n";
        q.display();
        getch();
        return 0;
}
```

Sample input/Output:

```
Enter the number of elements to be sorted : 7
Enter element no. 1 : 30
Enter element no. 2 : 10
Enter element no. 3: 20
Enter element no. 4: 15
Enter element no. 5: 45
Enter element no. 6: 40
Enter element no. 7: 50
Original : { 30 10 20 15 45 40 50}
Left: { 15 10 20} Pivot: [30] Right : { 45 40 50}
Left: { 10} Pivot : [15] Right: { 20}
Left: {} Pivot: [10] Right : {}
Left: {} Pivot: [20] Right : {}
Left : { 40 } Pivot : [ 45 ] Right : { 50 }
Left : { } Pivot : [ 40 ] Right : {}
Left : { } Pivot : [ 50 ] Right : {}
The Sorted Array
~~~~~~~~~~~~~
10 15 20 30 40 45 50
```

Result:

Thus, a C++ program to sort the given list by quick sort technique using divide and conquer method was written, executed and output was verified.

EX NO:11	DJIKSTRA'S SHROTEST PATH ALGORITHM
DATE:	

Aim:

To write a C++ program to implement Dijkstra's shortest path algorithm.

Procedure:

Variables used:

N - as integer to store Number of vertices in the graph

graph[10][10] - as integer to store the adjacency matrix representation of graph

dist[10] - as integer, the output array, dist[i] will hold the shortest distance from src to i

visited[10] - as Boolean, it will be true if vertex i is included in shortest path tree or shortest distance from src to i is finalized

parent[10] - as integer to store parent information used to display the shortest path

Functions Used

void createGraph() - to accept the user input for number of nodes, edge and weight values to create the adjacency matrix representation of the graph.

int minDistance() - to find the vertex with minimum distance value, from the set of vertices not yet included in shortest path tree.

void printPath(int j) - to print the shortest path.

void dijkstra() - to implement Dijkstra's single source shortest path algorithm.

int main() - to invoke functions createGraph() and Dijkstra()

Algorithm:

Dijkstra's Single Source Shortest Path to calculate the shortest path is as follows:

1. Create a set visited[] (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.
2. Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
3. While visited doesn't include all vertices
    a) Pick a vertex u which is not there in visited[] and has minimum distance value.
    b) Include u to visited[].
    c) Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

Source Code:

```
#include <iostream>
using namespace std;

bool value1 = false;
bool value2 = true;

int N;
int graph[10][10];
int dist[10];
bool visited[10];
int parent[10];
void createGraph() {
	int i,j,max,u,v,w;
cout<<"Enter the number of nodes :";cin>>N;
	for(i=0; i<=N; i++)
		for(j=0; j<=N; j++)
			graph[i][j]=0;
	max=N*(N+1);
	for(i=0; i<max; i++) {
cout<<"Enter Edge and Weight :";cin>>u>>v>>w;
		if(u==-1) break;
		else {
			graph[u][v]=w;
			graph[v][u]=w;
		}
	}
}
int minDistance() {
	int min = 10000, minDist;
	for (int v = 0; v < N; v++)
		if (visited[v] == false && dist[v] <= min) {
			min = dist[v];
			minDist = v;
		}
	return minDist;
}
void printPath(int j) {
	if (parent[j]==-1)
		return;
	printPath(parent[j]);
	cout<<j<<" ";
}
```

```
void dijkstra() {
	int src;
cout<<"Enter the Source Node :";cin>>src;
	for (int i = 0; i < N; i++) {
		parent[0] = -1;
		dist[i] = 10000;
		visited[i] = false;
	}
	dist[src] = 0;
	for (int count = 0; count < N-1; count++) {
		int u = minDistance();
		for (int v = 0; v < N; v++)
			if (!visited[v] && graph[u][v] && dist[u] + graph[u][v] < dist[v]) {
				parent[v] = u;
				dist[v] = dist[u] + graph[u][v];
			}
	}
	cout<<"Src->Dest \t Distance \t Path"<<endl;
	for (int i = 1; i < N; i++) {
		cout<<src<<"->"<<i<<"\t "<<dist[i]<<"\t "<<src<<" ";
		printPath(i);
		cout<<endl;
	}
}
int main() {
	createGraph();
	dijkstra();
	return 0;
}
```

Sample Input/Output:

```
Enter the number of nodes : 5
Enter Edge and Weight : 0 1 3
Enter Edge and Weight : 1 2 4
Enter Edge and Weight : 1 3 2
Enter Edge and Weight : 0 3 7
Enter Edge and Weight : 2 3 5
Enter Edge and Weight : 3 4 4
Enter Edge and Weight : 2 4 6
Enter Edge and Weight : -1 -1 -1
Enter the Source Node : 0
Src->Dest Distance Path
0->1 3 0 1
0->2 7 0 1 2
0->3 5 0 1 3
0->4 9 0 1 3 4
```

Result:

Thus a C++ program was written to implement Dijkstra's algorithm to find the single source shortest path and executed to verify the output.

EX NO:12	N-QUEENS PROBLEMS
DATE:	

Aim:

To write a C++ program to solve N-queens problem using Back tracking Method.

Procedure:

Create user defined data typed called bool using enum in C++.

Member variables in the class N-queen:

Board as pointer to pointer to hold 2D array

Member Function in the class N-queen:

NQueen(int size)	- Constructor to create and initialize 2D array board
~NQueen	- Destructor to release the memory of board
void printsolution (void)	- To print the solution
bool Issafe(int row, int col)	- To check is the safety of the queen
bool SolveNQueen(int col)	- To solve N-queens placement

Algorithm:

NQueen(int size)

1. Create board as 2D array initialize values of 2D array board as zero.

~NQueen ()

1. Release the memory of board.

void printSolution()

1. Print solution board

bool Issafe(int row, int col)

1. Check the left side of row if found a queen return false.
2. Check upper diagonal on left side if found a queen return false.
3. Check lower diagonal on left side if found a queen return false
4. Return true.

bool SolveNQueen(int col)

1. Base case: if all queens are placed then return true.
2. Consider this column and try placing this queen in all rows one by one.
3. Check if queen can be placed on board[i][col]
4. If safe then place this queen in board[i][col]
5. Recur to place rest of the queens
6. If placing queen in board[i][col] doesn't lead to a solution, then remove queen from board[i][col]
7. If queen cannot be placed in any row in this column col then return false

int main()

1. Use solve NQueen() to solve the problem after getting size of the board.
2. If solve NQueen() reurns false then queens cannot be placed, otherwise print placement of queens in the form of Qs and 0s.

Source Code:

```
//program to solve N-queen problem using backtracking
    #include <iostream>
    using namespace std;

    bool value1 = false;
    bool value2 = true;

    class NQueen {
            int **board,N;
      public:
            NQueen(int size);
            ~NQueen() {
                    delete board;
            }
            void printSolution(void);
            bool IsSafe(int row,int col);
            bool SolveNQueen(int col);
    };
    NQueen::NQueen(int size) {
      int i,j;
      N=size;
      board=new int*[N];
      for (i=0; i<N; i++)
            board[i]=new int[N];
      for (i=0; i<N; i++)
            for (j=0; j<N; j++)
                    board[i][j]=0;
    }
    void NQueen::printSolution() {
      for (int i=0; i<N; i++) {
            for (int j=0; j<N; j++)
                    if(board[i][j]==1)
                            cout<<"Q"<<j+1<<" ";
                    else
                            cout<<" "<<board[i][j]<<" ";
            cout<<"\n";
      }
    }
    bool NQueen::IsSafe(int row,int col) {
      int i,j;
      for(i=0; i<col; i++)
            if(board[row][i]) return false;
      for(i=row, j=col; i>=0 && j>=0; i--,j--)
            if(board[i][j]) return false;
      for (i=row, j=col; j>=0 && i<N; i++, j--)
            if (board[i][j]) return false;
      return true;
    }
    bool NQueen::SolveNQueen(int col) {
      if(col>=N) return true;
      for(int i=0; i<N; i++) {
            if(IsSafe(i, col)) {
```

```
                board[i][col]=1;
                if(SolveNQueen(col+1)) return true;
                board[i][col] = 0;
            }
        }
        return false;
    }
    int main() {
        int size;
        cout<<"\n enter the size of the board:";
        cin>>size;
        NQueen Q(size);
        if (Q.SolveNQueen(0)==false) {
            cout<<"\n solution does not exist";
        }else {
            cout<<"\n queens can be placed as shown below \n";
              Q.printSolution();
        }
        return 0;
    }
```

Sample Input/Output:

Enter the size of the board:8

Queens can be placed as shown below

Q1 0 0 0 0 0 0 0

0 0 0 0 0 0 Q7 0

0 0 0 0 Q5 0 0 0

0 0 0 0 0 0 0 Q8

0 Q2 0 0 0 0 0 0

0 0 0 Q4 0 0 0 0

0 0 0 0 0 Q6 0 0

0 0 Q3 0 0 0 0 0

Result:

Thus, a C++ program to solve N-Queens problem using backtracking method was written, executed and output was verified.

EX NO:13	SELECTION SORT
DATE:	

Aim:

To write a C++ program to sort the given list of elements by selection sort technique.

Procedure:

Create the base class DataArray with the following member variables and functions.

Member variables of base class DataArray

n as integer
a as array of integer of size 100

Member Function of base class DataArray

void GetData(void) - To get the input from user
void Display(void) - To display elements of the list

Algorithm:

void GetData(void)

1. Get the list of numbers.

void Display(void)

1. Display all elements of the list one by one.

Create a derived class Selection, derived from the base class DataArray with public access.

Member variables of the class Selection is none

Member functions in the derived class Selection

void SelectionSort(void) - To sort elements in ascending order

Algorithm:

void SelectionSort(void)

1. Find the largest number and place it in last position in the list.
2. Consider rest of the elements only as new list and repeat the step 1 until there is only one element in the list.

int main()

1. Create an object S of class Selection.
2. Call member function GetData() of object S.
3. Call member function SelectionSort() of object S.
4. Call member function Display() of object S.

Source Code:

```
// Selection sorting
   #include<iostream>
   using namespace std;

   class DataArray {
      protected:
            int a[100], n;
      public:
            void GetData(void);
            void Display(void);
   };
   void DataArray::GetData(void) {
      cout<<"\n Enter the number of elements to be sorted : ";
      cin>>n;
      for(int i=0; i<n; i++) {
            cout<<" Enter element no."<<i+1<<" :";
            cin>>a[i];
      }
   }
   void DataArray::Display(void) {
      for(int i=0; i<n; i++)
            cout<<" "<<a[i];
      cout<<endl;
   }
   class Selection : public DataArray {
      public:
            void SelectionSort(void);
   };
   void Selection::SelectionSort(void) {
      int large,index,i,j;
      for(i=n-1; i>0; i--) {
            large=a[0];
            index=0;
            for(j=1; j<=i; j++) {
                  if(a[j]>large) {
                        large=a[j];
                        index=j;
                  }
            }
            a[index]=a[i];
            a[i]=large;
            cout<<"\n Array after iteration "<<n-i<<": ";
            Display();
      }
   }
```

```
int main() {
    Selection S;
    S.GetData();
    S.SelectionSort();
    cout<<"\n\t The Sorted Array";
    cout<<"\n\t ~~~~~~~~~~~~~~~~\n";
    S.Display();
    return 0;
}
```

Sample Input/Output:

```
Enter the number of elements to be sorted : 10
Enter element no. 1 :  19
Enter element no. 2  :  77
Enter element no. 3  : -60
Enter element no. 4  : -12
Enter element no. 5 :  7
Enter element no. 6 :  41
Enter element no. 7  :  2
Enter element no. 8  :  1
Enter element no. 9 :  99
Enter element no. 10:  0
Array after iteration 1: 19 77 -60 -12 7 41 2 1 0 99
Array after iteration 2: 19 0 -60 -12 7 41 2 1 77 99
Array after iteration 3: 19 0 -60 -12 7 1 2 41 77 99
Array after iteration 4: 2 0 -60 -12 7 1 19 41 77 99
Array after iteration 5: 2 0 -60 -12 1 7 19 41 77 99
Array after iteration 6: 1 0 -60 -12 2 7 19 41 77 99
Array after iteration 7: -12 0 -60 1 2 7 19 41 77 99
Array after iteration 8: -12 -60 0 1 2 7 19 41 77 99
Array after iteration 9: -60 -12 0 1 2 7 19 41 77 99
The Sorted Array
~~~~~~~~~~~~~~
-60 -12 0 1 2 7 19 41 77 99
```

Result:

Thus, a C++ program to sort the given list of numbers by selection sort technique was written, executed and output was verified.