PONDICHERRY UNIVERSITY(A CENTRAL UNIVERSITY)

SCHOOL OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

M.C.A – FIRST SEMESTER2021 – 2023

RECORD NOTE BOOK

DATA STRUCTURES AND ALGORITHMS LAB (CSCA414)

NAME: V. SivaDinesh

REGISTER NUMBER: 21352080

Table of Contents:

SEARCHING ALGORITHM - SEQUENTIAL SEARCH

EX: NO: 1

AIM:

To find an element of the given array by using Sequential Search

ALGORITHM:

Step1: Start

Step2: Read the target element from the user.

Step3: Compare the target element with each element of arr[].

Step4: If target element matches with an element in array, return the index.

Step5: If the target element doesn't match with any of element in array, then return -1.

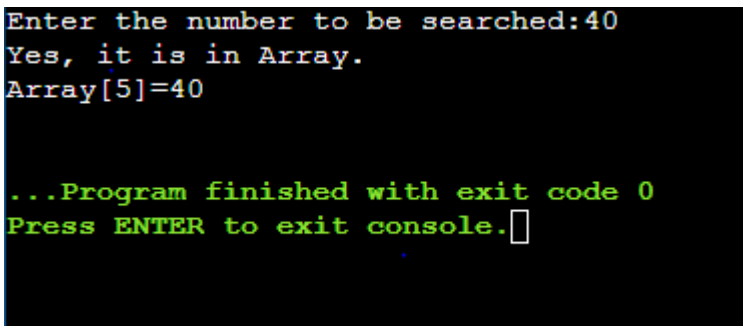
Step6: If the search function return -1, then print "element not found" otherwise print Element found at index value.

Step 7: Stop.

Source Code:

```
# include<stdio.h>
int main()
{
 int i;
 int Array[ ] = {5,2,13,15, 4,40 ,60, 30,85,90};
 int key ;
 int Size = sizeof(Array) / sizeof(int);
 printf("Enter the number to be searched:");
 scanf("%d", &key);
 for( i =0; i< Size; i++)
 if( Array[i]== key)
 {
 printf("Yes, it is in Array. \nArray[%d]=%d\n", i, Array[i]);
 goto end;
 }
 printf("The number is not in Array.\n");
 end:
 return 0;
}
```

Output:



SEARCHING ALGORITHM - BINARY SEARCH

EX: NO: 1

AIM:

To find an element of the given array using Binary search algorithm.

ALGORITHM:

Step1 : Start

Step2 : ~~Compare~~ Read the target element from the user.

Step3 : Compare the target element with the middle element of the sorted array.

Step4 : If the target element matches with the middle element, then returns the middle index

Step5 : Else if the target is greater than the middle element, then narrow the array to the right half after the middle element.

Step6 : Else, narrow the array to the left half before the middle element.

Step7 : Repeatedly check until the value is found or the interval is empty.

Step8 : If the target found in the array, print Element found otherwise "NOT FOUND".

Step9 : Stop.

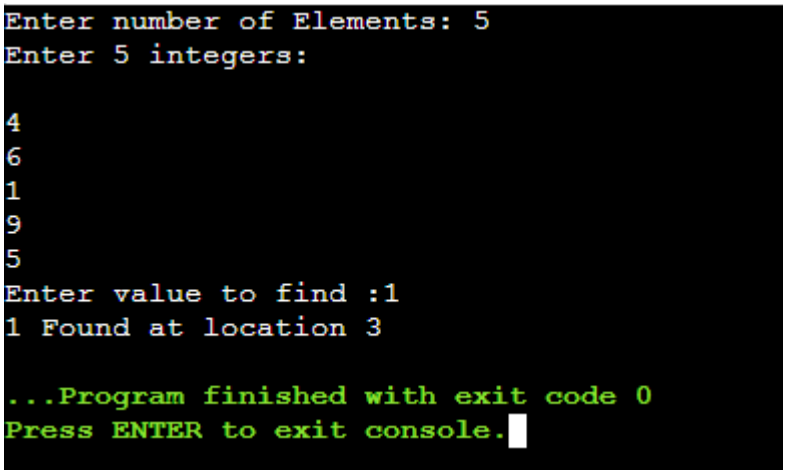
p

Source Code:

```
#include <stdio.h>
int main()
{
int i, low, high, mid, n, key, array[100];
printf("Enter number of Elements: ");
scanf("%d",&n);
printf("Enter %d integers: \n",n);
for(i = 0; i < n; i++)
scanf("%d",&array[i]);
printf("Enter value to find :");
scanf("%d", &key);
low = 0;
high = n - 1;
mid = (low+high)/2;
while (low <= high)
{
if(array[mid] < key)
low = mid + 1;
else if (array[mid] == key)
{
printf("%d Found at location %d", key, mid+1);
break;
}
else
high = mid - 1;
```

```
mid = (low + high)/2;

}

if(low > high)

printf("Not found! %d isn't present in the list", key);

return 0;

}
```

Output:



EX:NO:1

SEARCHING ALGORITHM - FIBONACCI SEARCH

AIM:

To search an element of the given array by using Fibonacci search algorithm.

ALGORITHM:

Step 1: Start

Step 2: First, we need to find F(K) whish is K^{th} fibonacci number which is greater than or equal to the size of array (n).

Step 3: If F(K) = 0, then we need to stop here and print the message as "Element not found".

Step 4: Set variable offset = -1.

Step 5: We need to set i = min(offset + F(K-2), (n-1))

Step 6: Check

* If search element (s) == Array [i] then, return i and stop the search.
* If search element (s) > Array [i] then, K = K-1, offset = I and repeat step 4, 5.
* If search element (s) < Array [i] then, k = K-2, repeat step 4, 5.

Step 7: Stop.

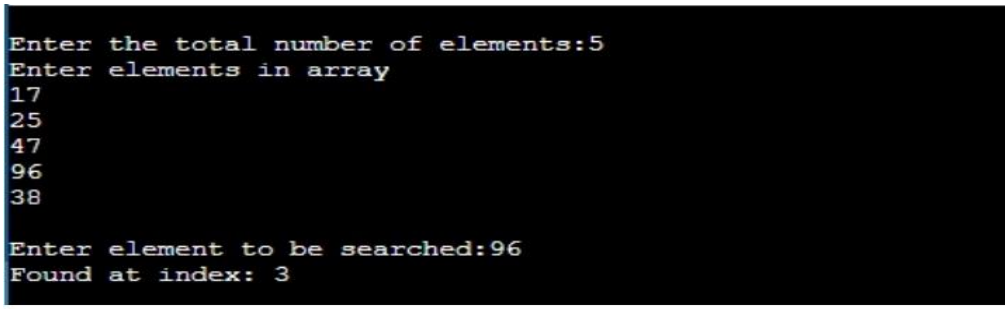
Source Code:

```
#include<stdio.h>
#include<conio.h>
int min(int x, int y)
{
 return (x<=y)? x : y;
}
int fibonaccianSearch(int arr[], int x, int n)
{
int fbK2 = 0;
int fbK1 = 1;
int fbK = fbK2 + fbK1;
int offset = -1;
while (fbK < n)
 {
 fbK2 = fbK1;
 fbK1 = fbK;
 fbK = fbK2 + fbK1;
 }
while (fbK > 1)
 {
int I = min(offset+fbK2, n-1);
if (arr[i] < x)
 {
 fbK = fbK1;
 fbK1 = fbK2;
```

```
 fbK2 = fbK – fbK1;
 offset = I;
 }
else if (arr[i] > x)
 {
 fbK = fbK2;
 fbK1 = fbK1 – fbK2;
 fbK2 = fbK – fbK1;
 }
else return I;
 }
if(fbK1 && arr[offset+1] == x)
return offset+1;
return -1;
}
int main(void)
{
int l,i;
printf("\nEnter the total number of elements:");
scanf("%d",&l);
int arr[l];
printf("Enter elements in array\n");
for(i=0;i<l;i++)
 {
scanf("%d",&arr[i]);
 }
int n = sizeof(arr)/sizeof(arr[0]);
```

```
int x;
printf("\nEnter element to be searched:");

scanf("%d",&x);
printf("Found at index: %d",fibonaccianSearch(arr, x, n));
getch();
return 0;
}
```

Output:



EX: NO: 2

EVALUATION OF ARITHMETIC EXPRESSION

AIM:

To convert the infix expression to postfix expression by using stack.

ALGORITHM:

Step 1: Start

Step 2: Scan the infix expression from left to right

Step 3: If the scanned character is an operand, then print it.

Step 4: Else,

* If the precedence of the scanned operator is greater than the precedence of the operator in the stack, then push it.
* Else, pop all the operators from the stack which are greater than or equal to precedence than that of the scanned operator. After doing that push the scanned operator to the stack.

Step 5: If the scanned character is an '(', then push it to the stack.

Step 6: If the scanned character is an ')', then pop the stack and print it until a '(' is encountered and discard both the parenthesis.

Step 7: Repeat step 3 to 6 until infix expression is scanned.

Step 8: Print the output.

Step 9: Pop and paint from the stack until it is not empty.

Step 10: Stop.

Source Code:

```
#include<stdio.h>
#include<ctype.h>
#include<conio.h>
int ch ,addi , subs , mult , didv;
void add();
void sub();
void multiply();
void divied();
void main()
{
while(1)
{
printf("\n *****ARITHMATIC OPERATIONS*****");
printf("\n 1.ADDITION");
printf("\n 2.SUBTRACTION");
printf("\n 3.MULTIPLICATION");
printf("\n 4.DIVIED");
printf("\n 5.Exit");
printf("\n Enter your choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:add();
break;
case 2:sub();
break;
```

```
case 3:multiply();
break;
case 4:divied();
break;
case 5:return;
}
}
}
void add()
{
int a , b;
printf("Enter the first number : ");
scanf("%d", &a);
printf("Enter the second number : ");
scanf("%d", &b);
addi = a+b;
printf("\nThe value is %d\n",addi);
}
void sub()
{
int a , b;
printf("Enter the first number : ");
scanf("%d", &a);
printf("Enter the second number : ");
scanf("%d", &b);
subs = a-b;
printf("\nThe value is %d\n",subs);
```

```
}
void multiply()
{
int a , b;
printf("Enter the first number :");
scanf("%d", &a);
 printf("Enter the second number : ");
scanf("%d", &b);
mult = a*b;
printf("\nThe value is %d\n",mult);
}
void divied()
{
int a , b;
printf("Enter the first number : ");
scanf("%d", &a);
 printf("Enter the second number : ");
scanf("%d", &b);
didv = a/b;
printf("\nThe value is %d\n",didv);
}
```

Output:

```
 *****ARITHMATIC OPERATIONS*****
 1.ADDITION
 2.SUBTRACTION
 3.MULTIPLICATION
 4.DIVIED
 5.Exit
 Enter your choice:1
Enter the first number : 45
Enter the second number : 65

The value is 110

 *****ARITHMATIC OPERATIONS*****
 1.ADDITION
 2.SUBTRACTION
 3.MULTIPLICATION
 4.DIVIED
 5.Exit
 Enter your choice:2
Enter the first number : 45
Enter the second number : 65

The value is -20

 *****ARITHMATIC OPERATIONS*****
 1.ADDITION
 2.SUBTRACTION
 3.MULTIPLICATION
 4.DIVIED
 5.Exit
 Enter your choice:3
```

```
Enter the first number :45
Enter the second number : 65

The value is 2925

 *****ARITHMATIC OPERATIONS*****
 1.ADDITION
 2.SUBTRACTION
 3.MULTIPLICATION
 4.DIVIED
 5.Exit
 Enter your choice:4
Enter the first number : 45
Enter the second number : 65

The value is 0

 *****ARITHMATIC OPERATIONS*****
 1.ADDITION
 2.SUBTRACTION
 3.MULTIPLICATION
 4.DIVIED
 5.Exit
 Enter your choice:5


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

IMPLEMENTATION OF STACK

EX:NO:3

AIM:

To implement the concept of stack and perform its operation.

ALGORITHM:

Step 1: Start

Step 2: To perform a push operation
* Check if the stack is full.
* If the stack is full, print "STACK IS OVERFLOW".
* If the stack is not full, increment top and add element to the stack.

Step 3: To perform a pop operation.
* Check if the stack is empty.
* If the stack is empty, print "STACK IS UNDERFLOW".
* If the stack is not empty, decrease the top by 1.

Step 4: To perform a peek operation
* Return to top element of the stack.

Step 5: To perform a is Full operation
* Check if the top equals to Max size
* If equals then return True
* Otherwise return False.

Step 6: To perform is Empty Operation
* Check if the top less than zero.
* If so, then return True.
* Otherwise, return False.

Step 7: Stop.

Source code:

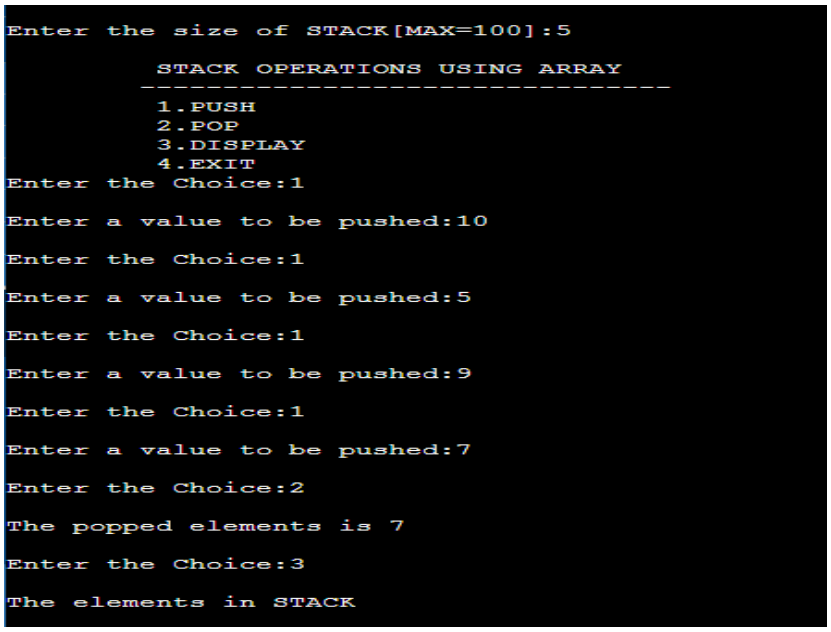
```
#include<stdio.h>
int stack[100],choice,n,top,x,I;
void push(void);
void pop(void);
void display(void);
int main()
{
top=-1;
printf("\nEnter the size of STACK[MAX=100]:");
scanf("%d",&n);
printf("\n\t STACK OPERATIONS USING ARRAY");
printf("\n\t--------------------------------");
printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
do
 {
 printf("\nEnter the Choice:");
 scanf("%d",&choice);
switch(choice)
 {
 case 1:
 {
 push();
 break;
 }
 case 2:
 {
```

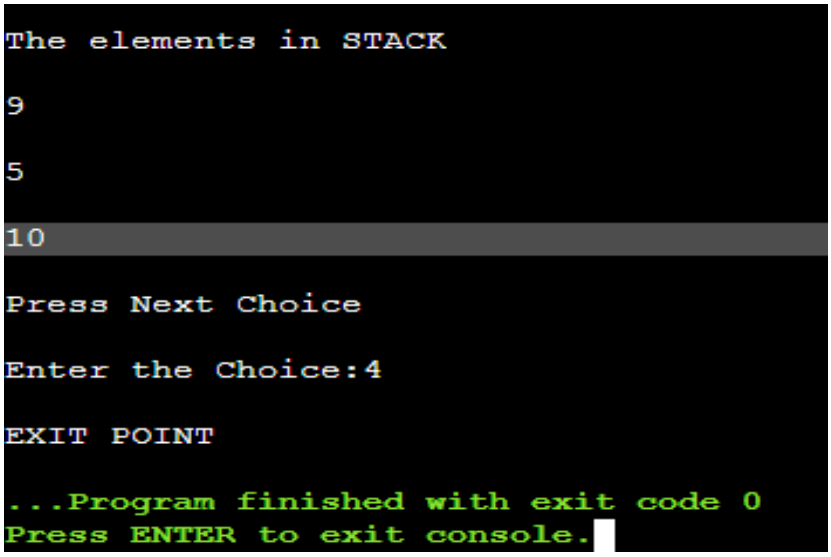
```
 pop();
 break;
 }
 case 3:
 {
 display();
 break;
 }
 case 4:
 {
 printf("\nEXIT POINT ");
 break;
 }
 default:
 {
 printf ("\nPlease Enter a Valid Choice(1/2/3/4)");
 }
 }
 }
 while(choice!=4);
 return 0;
}
void push()
{
 if(top>=n-1)
 {
 printf("\nSTACK is over flow");
```

```
 }
 else
 {
 printf("\nEnter a value to be pushed:");
 scanf("%d",&x);
 top++;
 stack[top]=x;
 }
}
void pop()
{
 if(top<=-1)
 {
 printf("\nStack is under flow\n");
 }
 else
 {
 printf("\nThe popped elements is %d\n",stack[top]);
 top--;
 }
}
void display()
{
 if(top>=0)
 {
 printf("\nThe elements in STACK\n");
 for(int i=top; i>=0; i--)
```

```
 printf("\n%d\n",stack[i]);
 printf("\nPress Next Choice\n");
 }
 else
 {
 printf("\nThe STACK is empty\n");
 }
}
```

Output:


Enter the size of STACK[MAX=100]:5
STACK OPERATIONS USING ARRAY
1.PUSH
2.POP
3.DISPLAY
4.EXIT
Enter the Choice:1
Enter a value to be pushed:10
Enter the Choice:1
Enter a value to be pushed:5
Enter the Choice:1
Enter a value to be pushed:9
Enter the Choice:1
Enter a value to be pushed:7
Enter the Choice:2
The popped elements is 7
Enter the Choice:3
The elements in STACK


The elements in STACK
9
5
10
Press Next Choice
Enter the Choice:4
EXIT POINT
...Program finished with exit code 0
Press ENTER to exit console.

EX:NO:3

IMPLEMENTATION OF QUEUE

AIM: To implement the concept of Queue and perform its operation.

ALGORITHM:

Step 1: Start.

Step 2: To perform Enqueue operation

* Allocate the space for the new node PTR.
* Set PTR → data = Value.
* Check if front = Null, then
  set front = rear = PTR and
  set front → next = rear → next = Null
* Else
  set rear → next = PTR and
  set rear = PTR
  set rear → next = Null

Step 3: To perform Dequeue operations

* Check if front = Null,
* If true then print "UNDERFLOW"
* Else
  set PTR = front
  set front = front → Next
  Free PTR

Step 4: Stop.

Source Code:

```
#include <stdio.h>
# define SIZE 100
void enqueue();
void dequeue();
void show();
int inp_arr[SIZE];
int Rear = - 1;
int Front = - 1;
void main()
{
int ch;
while (1)
{
printf("1.Enqueue Operation\n");
printf("2.Dequeue Operation\n");
printf("3.Display the Queue\n");
printf("4.Exit\n");
printf("Enter your choice of operations : ");
scanf("%d", &ch);
switch (ch)
{
case 1:enqueue();
break;
case 2:
dequeue();
```

```
break;
case 3:
show();
break;
case 4:
return;
default:
printf("Incorrect choice \n");
}
}
}
void enqueue()
{
int insert_item;
if (Rear == SIZE - 1)
printf("Overflow \n");
else
{
if (Front == - 1)
Front = 0;
printf("Element to be inserted in the Queue : ");
scanf("%d", &insert_item);
Rear = Rear + 1;
inp_arr[Rear] = insert_item;
}
}
void dequeue()
```

```
{
if (Front == - 1 || Front > Rear)
{
printf("Underflow \n");
return ;
}
else
{
printf("Element deleted from the Queue: %d\n", inp_arr[Front]);
Front = Front + 1;
}
}
void show()
{
if (Front == - 1)
printf("Empty Queue \n");
else
{
printf("Queue: \n");
for (int i = Front; i <= Rear; i++)
printf("%d ", inp_arr[i]);
printf("\n");
}
}
```

Output:

```
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 1
Element to be inserted in the Queue : 10
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 1
Element to be inserted in the Queue : 15
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 1
Element to be inserted in the Queue : 18
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 2
Element deleted from the Queue: 10
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 3
Queue:
15 18
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
```

```
1.Enqueue Operation
2.Dequeue Operation
3.Display the Queue
4.Exit
Enter your choice of operations : 4


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

EX:NO:3

IMPLEMENTATION OF CIRCULAR QUEUE

AIM:

To implement the concept of circular queue and its operation.

ALGORITHM:

Step 1: Start

Step 2: To perform Enqueue operation

* Create a struct node type node.
* Insert the given data in the newnode data section and NULL in address section.
* If queue is empty then initialize front and rear next and rear from new node.
* Else, initialize rear next and rear from new node
* New node next initialize from front.

Step 3: To perform Dequeue operation.

* Check if queue is empty.
* If empty, then print "UNDERFLOW".
* Else, initialize temp from front.
* If front is equal to the rear, then initialize front and rear from NULL.
* Print data of temp and free temp memory.
* If there is more than one node in queue, then make front next to front then initialize rear next front.
* Print temp and free temps.

Step 4: To show the element in queue.

* Check if there is some data in the queue or not.
* If the queue is empty then print "NO DATA IN THE QUEUE".
* Else define a node pointer and initialize it with front.
* Print data of node pointer untill the next of node pointer because NULL.

Step 5: Stop.

Source Code:

```
#include <stdio.h>
# define max 6
int queue[max];
int front=-1;
int rear=-1;
void enqueue(int element)
{
 if(front==-1 && rear==-1)
 {
 front=0;
 rear=0;
 queue[rear]=element;
 }
 else if((rear+1)%max==front)
 {
 printf("Queue is overflow..");
 }
 else
 {
 rear=(rear+1)%max;
 queue[rear]=element;
 }
}
int dequeue()
{
```

```
 if((front==-1) && (rear==-1))
 {
 printf("\nQueue is underflow..");
 }
else if(front==rear)
{
 printf("\nThe dequeued element is %d\n ", queue[front]);
 front=-1;
 rear=-1;
}
else
{
 printf("\nThe dequeued element is %d\n ", queue[front]);
 front=(front+1)%max;
}
}
 void display()
{
 int i=front;
 if(front==-1 && rear==-1)
 {
 printf("\n Queue is empty..");
 }
 else
 {
 printf("\nElements in a Queue are :");
 while(i<=rear)
```

```
 {
 printf("%d,",queue[i]);
 i=(i+1)%max;
 }
 }
}
void main()
{
 int choice=1,x,n4;
 while(choice<n4 && choice!=0)
 {
 printf("\n1.Insert an element");
 printf("\n2.Delete an element");
 printf("\n3.Display the element");
 printf("\n4.Exit");
 printf("\nEnter your choice: ");
 scanf("%d", &choice);
 switch(choice)
 {
 case 1:
 printf("Enter the element which is to be inserted: ");
 scanf("%d", &x);
 enqueue(x);
 break;
 case 2:
 dequeue();
 break;
```

```
case 3:
display();
break;

case 4:
return;
}
}
```

Output:

```
1.Insert an element
2.Delete an element
3.Display the element
4.Exit
Enter your choice: 1
Enter the element which is to be inserted: 10

1.Insert an element
2.Delete an element
3.Display the element
4.Exit
Enter your choice: 1
Enter the element which is to be inserted: 12

1.Insert an element
2.Delete an element
3.Display the element
4.Exit
Enter your choice: 1
Enter the element which is to be inserted: 15

1.Insert an element
2.Delete an element
3.Display the element
4.Exit
Enter your choice: 1
Enter the element which is to be inserted: 20

1.Insert an element
2.Delete an element
3.Display the element
4.Exit
Enter your choice: 2
```

```
The dequeued element is 10

1.Insert an element
2.Delete an element
3.Display the element
4.Exit
Enter your choice: 3

Elements in a Queue are :12,15,20,
1.Insert an element
2.Delete an element
3.Display the element
4.Exit
Enter your choice: 4


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

EX: NO: 3

IMPLEMENTATION OF PRIORITY QUEUE

AIM: To implement the concept of priority queue and its operations.

ALGORITHM:

Step 1: Start

Step 2: Check if there is no node.

Step 3: If there is no node, then create a new node

Step 4: Else, insert the new node at the end.

Step 5: Heapify the array.

Step 6: To perform deletion operation

* If the nodetobeDeleted is leafNode then ~~remov~~ remove the node.
* Else, swap the nodetobeDeleted with the leastLeafNode.
* Remove notetobeDeleted.
* Heapify the array.

Step 7: To perform peak operation, return root node.

Step 8: Step.

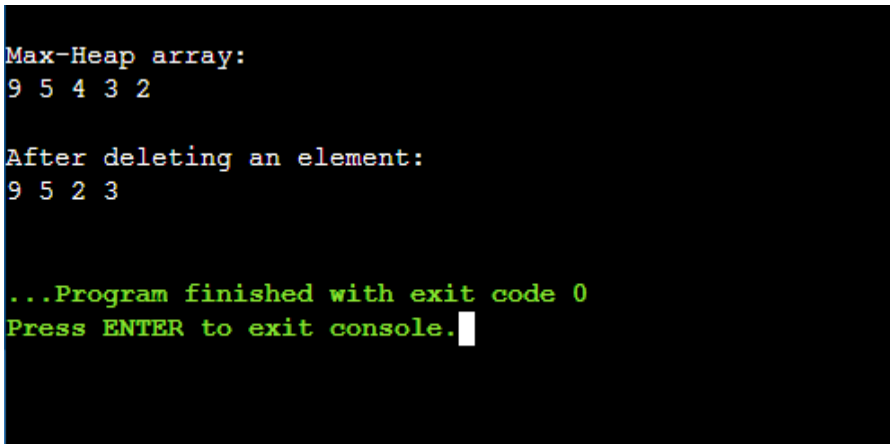
Source code:

```
#include <stdio.h>
int size = 0;
void swap(int *a, int *b) {
 int temp = *b;
 *b = *a;
 *a = temp;
}
void heapify(int array[], int size, int i) {
 if (size == 1) {
 printf("Single element in the heap");
 } else
 {
 int largest = i;
 int l = 2 * i + 1;
 int r = 2 * i + 2;
 if (l < size && array[l] > array[largest])
 largest = l;
 if (r < size && array[r] > array[largest])
 largest = r;
 if (largest != i) {
 swap(&array[i], &array[largest]);
 heapify(array, size, largest);
 }
 }
}
```

```
void insert(int array[], int newNum) {
 if (size == 0) {
 array[0] = newNum;
 size += 1;
 } else {
 array[size] = newNum;
 size += 1;
 for (int i = size / 2 - 1; i >= 0; i--) {
 heapify(array, size, i);
 }
}
}
void deleteRoot(int array[], int num) {
 int i;
 for (i = 0; i < size; i++) {
 if (num == array[i])
 break;
 }
 swap(&array[i], &array[size - 1]);
 size -= 1;
 for (int i = size / 2 - 1; i >= 0; i--) {
 heapify(array, size, i);
 }
}
void printArray(int array[], int size) {
 for (int i = 0; i < size; ++i)
 printf("%d ", array[i]);
```

```
 printf("\n");
}
int main() {
 int array[10];
 insert(array, 3);
 insert(array, 4);
 insert(array, 9);
 insert(array, 5);
 insert(array, 2);
 printf("\nMax-Heap array: \n");
 printArray(array, size);
 deleteRoot(array, 4);
 printf("\nAfter deleting an element: \n");
 printArray(array, size);
}
```

Output:



IMPLEMENTATION OF SINGLY LINKED LIST

AIM:

To implement the concept of singly linked list and its operations.

ALGORITHM:

Step 1: Start

Step 2: To perform the insertion operation

* Allocate the space for the new node
* Check if PTR = NULL
* If true, then print "OVERFLOW"
* Else

set newnode = PTR
set PTR = PTR → next
set new node → data = value
set new node → next = Head
set head = new node

Step 3: To perform the insert at the Middle

* Allocate memory and store data for new node
* Traverse to node just before the required position of new node.
* Change next pointers of includes new nodes in between.

Step 4: To perform the insertion at end.

* Allocate the memory for new node.
* Store data in new node.

* Traverse the list to last node
* Change next of last node to recently created node.

Step 5: To delete from begining, point head to the second node.

Step 6: To delete from end
* Traverse the list to second last element.
* change its next pointer to NULL.

Step 7: To Delete from middle
* Traverse to element before the element to be deleted.
* change next pointers to exclude the node from the list.

Step 8: Stop.

Source Code:

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
struct node {
 int value;
 struct node *next;
};
void insert();
void display();
void del();
int count();
typedef struct node DATA_NODE;
DATA_NODE *head_node, *first_node, *temp_node = 0, *prev_node, next_node;
int data;
int main() {
int option = 0;
 printf("\nSingly Linked List Example - All Operations\n");
 while (option < 5) {
 printf("\nOptions\n");
 printf("1 : Insert into Linked List \n");
 printf("2 : Delete from Linked List \n");
 printf("3 : Display Linked List\n");
 printf("4 : Count Linked List\n");
 printf("Others : Exit()\n");
 printf("Enter your option: ");
 scanf("%d", &option);
```

```
 switch (option) {
 case 1:
 insert();
 break;
 case 2:
 del();
 break;
 case 3:
 display();
 break;
 case 4:
 count();
 break;
 default:
 break;
 }
 }
 return 0;
}
void insert()
{
 printf("\nEnter Element for Insert Linked List : ");
 scanf("%d", &data);
 temp_node = (DATA_NODE *) malloc(sizeof (DATA_NODE));
 temp_node->value = data;
 if (first_node == 0)
 {
```

```
 first_node = temp_node;
 }
 else
 {
 head_node->next = temp_node;
 }
 temp_node->next = 0;
 head_node = temp_node;
 fflush(stdin);
}
void del()
{
 int countvalue, pos, i = 0;
 countvalue = count();
 temp_node = first_node;
 printf("\nDisplay Linked List : ");
 printf("\nEnter Position for Delete Element : ");
 scanf("%d", &pos);
 if (pos > 0 && pos <= countvalue) {
 if (pos == 1) {
 temp_node = temp_node -> next;
 first_node = temp_node;
 printf("\nDeleted Successfully\n");
 }
 else {
 while (temp_node != 0) {
 if (i == (pos - 1)) {
```

```
 prev_node->next = temp_node->next;
 if(i == (countvalue - 1))
 {
 head_node = prev_node;
 }
 printf("\nDeleted Successfully\n ");
 break;
 }
 else {
 i++;
 prev_node = temp_node;
 temp_node = temp_node -> next;
 }
 }
 }
 }
 else
 printf("\nInvalid Position\n ");
}
void display()
{
 int count = 0;
 temp_node = first_node;
 printf("\nDisplay Linked List :\n ");
 while (temp_node != 0) {
 printf("\t%d", temp_node->value);
 count++;
```

```
 temp_node = temp_node -> next;
 }
 printf("\nNo Of Items In Linked List : %d\n", count);
}
int count() {
 int count = 0;
 temp_node = first_node;
 while (temp_node != 0) {
 count++;
 temp_node = temp_node -> next;
 }
 printf("\nNo Of Items In Linked List : %d\n", count);
 return count;
}
```

Output:

```
Singly Linked List Example - All Operations

Options
1 : Insert into Linked List
2 : Delete from Linked List
3 : Display Linked List
4 : Count Linked List
Others : Exit()
Enter your option: 1

Enter Element for Insert Linked List : 10

Options
1 : Insert into Linked List
2 : Delete from Linked List
3 : Display Linked List
4 : Count Linked List
Others : Exit()
Enter your option: 1

Enter Element for Insert Linked List : 15

Options
1 : Insert into Linked List
2 : Delete from Linked List
3 : Display Linked List
4 : Count Linked List
Others : Exit()
Enter your option: 1

Enter Element for Insert Linked List : 20
```

IMPEMENTATION OF DOUBLE LINKED LIST

AIM: To implement the concept of doubly linked list and its operations.

ALGORITHM:

Step 1: Start

Step 2: To insert at begening:

* Allocate memory for new node and assign the data to new Node.
* point next of new Node to the first node of the doubly linked list.
* point prev of the list node to new Node and point head to new node.

Step 3: To insert in between two nodes:

* Allocate memory for new Node and assign data
* Assign the value of next from previous node to the address of new node.
* Assign the values of prev node to the prev of new node and assign address.

Step4: To insert at end

* Create a new node.
* If the Doubly linked list is empty then make the new node has head.
* Else, traverse to the end of the doubly linked list.

Step 6: To delete the node in last

* Find the node before the last node.
* store address of next node to last node
* free the memory of last
* Make temp as last node

Step 7: To delete the node in between two nodes

* Travel to node to deleted
* Let the node that before in node to Bedeleted as temp
* Store address of next node to nodetobedeleted in temp
* Free the memory of nodetobedeleted.

Step 8: Stop.

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node*llink,*rlink;
};
struct node*head=NULL,*tail=NULL,*temp,*t,*p;
int opt,c=0,item;
void create();
void dele();
void display();
void insert();
void main()
{
printf("\n\t\t\tDOUBLY LINKED LIST");
create();
display();
while(1)
{
printf("\n\tMENU");
printf("\n\t_____");
printf("\n\t1.insert");
printf("\n\t2.delete");
printf("\n\t3.display");
```

```
printf("\n\t4.exit");
printf("\nEnter your choice:");
scanf("%d",&opt);
switch(opt)
{
case 1: insert();
display();
break;
case 2: dele();
display();
break;
case 3: display();
break;
case 4: exit(0);
break;
}
}
}
void create()
{
int cap,i;
printf("\nEnter the list size:");
scanf("%d",&cap);
while(cap<=0)
{
printf("\nInvalid size");
printf("\nEnter the list size:");
```

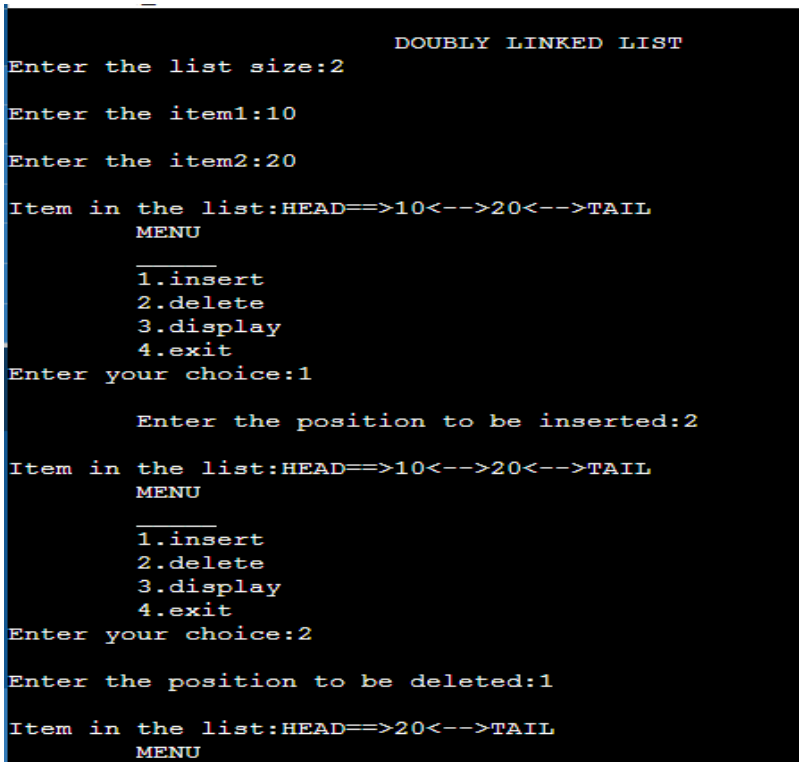
```
scanf("%d",&cap);
}
for(i=1;i<=cap;i++)
{
printf("\nEnter the item%d",i);
printf(":");
scanf("%d",&item);
if(head==NULL)
{
temp=(struct node*)malloc(sizeof(struct node));
temp->data=item;
head=temp;
tail=temp;
temp->llink=NULL;
}
else
{
t=(struct node*)malloc(sizeof(struct node));
t->data=item;
temp->rlink=t;
t->llink=temp;
temp=t;
tail=t;
}
c++;
temp->rlink=NULL;
}
```

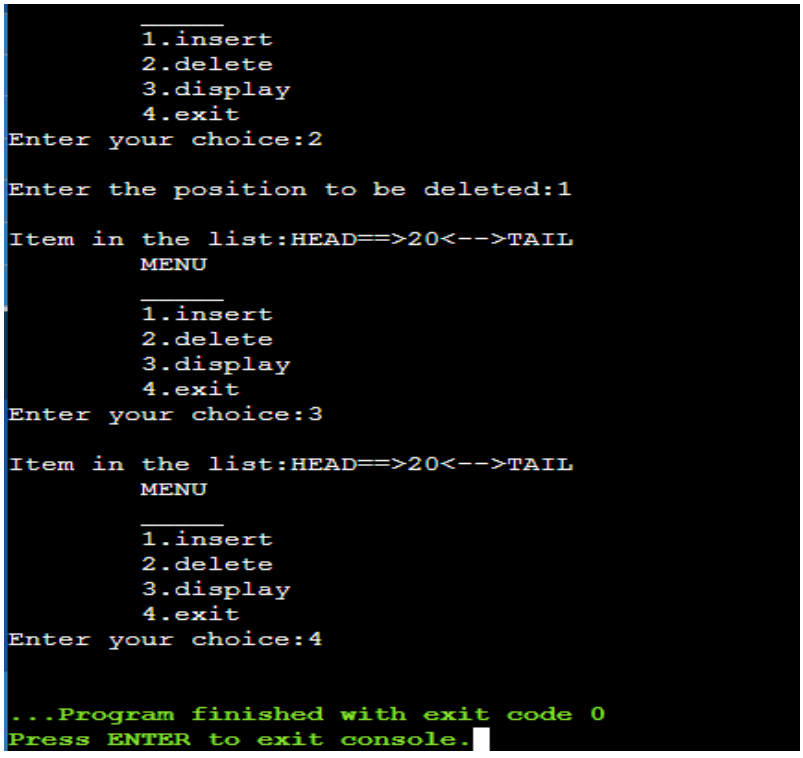
```
}
void insert()
{
int pos,k;
t=head;
printf("\n\tEnter the position to be inserted:");
scanf("%d",&pos);
if(pos>c+1)
{
for(k=1;k<pos-1;k++)
t=t->rlink;
temp->llink=t;
temp->rlink=t->rlink;
t->rlink->llink=temp;
t->rlink=temp;
}
c++;
}
void dele()
{
int pos,k;
temp=t=head;
if(temp==NULL)
{
printf("\n\tNo item");
}
else
```

```
{
printf("\nEnter the position to be deleted:");
scanf("%d",&pos);
if(pos>c)
{
printf("\n\tInvalid position");
return;
}
if(pos==1)
{
head=head->rlink;
head->llink=NULL;
free(temp);
}
else if(pos==c)
{
p=tail;
p->llink->rlink=NULL;
free(p);
}
else
{
for(k=1;k<pos;k++)
t=t->rlink;
t->llink->rlink=t->rlink;
t->rlink->llink=t->llink;
free(t);
```

```
}
}
c--;
}
void display()
{
temp=head;
printf("\nItem in the list:");
printf("HEAD==>");
while(temp!=NULL)
{
printf("%d",temp->data);
printf("<-->");
temp=temp->rlink;
}
printf("TAIL");
}
```

Output:


DOUBLY LINKED LIST
Enter the list size:2
Enter the item1:10
Enter the item2:20
Item in the list:HEAD==>10<-->20<-->TAIL
MENU
1.insert
2.delete
3.display
4.exit
Enter your choice:1
Enter the position to be inserted:2
Item in the list:HEAD==>10<-->20<-->TAIL
MENU
1.insert
2.delete
3.display
4.exit
Enter your choice:2
Enter the position to be deleted:1
Item in the list:HEAD==>20<-->TAIL
MENU


1.insert
2.delete
3.display
4.exit
Enter your choice:2
Enter the position to be deleted:1
Item in the list:HEAD==>20<-->TAIL
MENU
1.insert
2.delete
3.display
4.exit
Enter your choice:3
Item in the list:HEAD==>20<-->TAIL
MENU
1.insert
2.delete
3.display
4.exit
Enter your choice:4
...Program finished with exit code 0
Press ENTER to exit console.

EX NO: 4

IMPLEMENTATION OF CIRCULAR LINKED LIST

AIM: To implement the concept of circular linked list and its operations.

ALGORITHM:

Step 1: Start

Step 2: To insert at beginning
* Store the address of current first node in new node.
* Point the last node to new node.

Step 3: To insert in between two nodes
* Travel to node given and let this node be P.
* point the next of newnode to node next to P.
* Store the address of new node at next to P

Step 4: To insert at end.
* Store the address of the head
* Point the current last node to new node.
* Make new node as the last node.

Step 5: To delete the node.
* Free the memory occupied by node and store NULL in last.

Step 6: To delete the node in last
* Find the node before the last node.
* store address of next node to last node
* free the memory of last
* Make temp as last node

Step 7: To delete the node in between two nodes
* Travel to node to deleted
* Let the node that before in node to Bedeleted as temp
* Store address of next node to nodetobedeleted in temp
* Free the memory of nodetobedeleted.

Step 8: Stop.

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
  int data;
  struct Node*next;
};
struct Node*head = NULL;
void insert(int newdata){
struct Node*newnode =(struct Node*)malloc(sizeof(struct Node));
struct Node*ptr = head;
newnode->data = newdata;
newnode->next= head;
  if(head!= NULL)
  {
    while(ptr->next!= head)
    ptr = ptr->next;
    ptr->next= newnode;
  }
else
  newnode->next= newnode;
  head = newnode;
}
void display()
{
```

```
struct Node* ptr;
  ptr = head;
  do
  {
     printf("%d->",ptr->data);
     ptr = ptr->next;
  }
while(ptr != head);
}
int main()
{
  insert(10);
  insert(16);
  insert(7);
  insert(2);
  insert(9);
  printf(" \n CIRCULAR LINKED LIST \n");
  printf("\n The circular linked list is: \n \n");
  display();
  return 0;
}
```

Output:

```
 CIRCULAR LINKED LIST

 The circular linked list is:

9->2->7->16->10->

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

EX:NO:5

TREE TRAVERSAL TECHNIQUES

AIM: To perform Inorder, Preorder, Postorder in Tree Traversal.

ALGORITHM:

Step 1: Start

Step 2: For Inorder Traversal, until all nodes traversed
* recursively traverse left subtree
* Visit root node.
* recursively traverse right subtree

Step 3: For Preorder, Traverse,
* Visit root node
* Recursively traverse left subtree
* recursively traverse right subtree

Step4: For Post order Traversal,
* Recursively traversal left subtree
* Recursively traversal right subtree
* Visit root node.

Step 5: Stop.

Source code:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
 int data;
 struct node *leftChild;
 struct node *rightChild;
};
struct node *root = NULL;
void insert(int data)
{
 struct node *tempNode = (struct node*) malloc(sizeof(struct node));
 struct node *current;
 struct node *parent;
 tempNode->data = data;
 tempNode->leftChild = NULL;
 tempNode->rightChild = NULL;
 if(root == NULL) {
 root = tempNode;
 }
 else
 {
 current = root;
 parent = NULL;
 while(1)
```

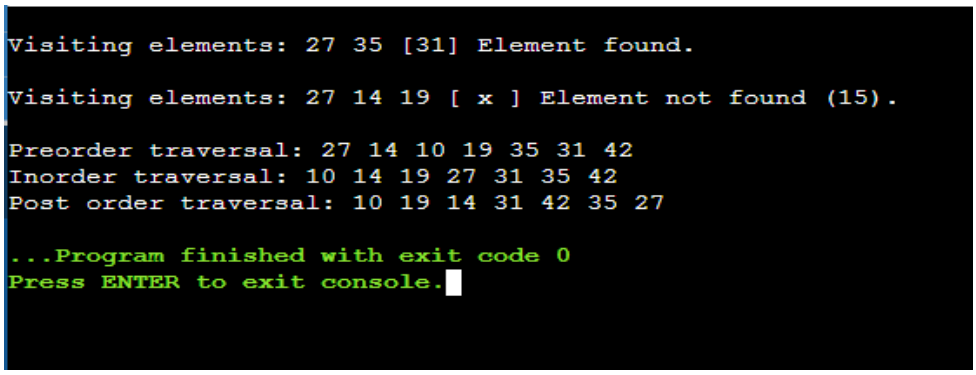
```
 {
 parent = current;
 if(data < parent->data) {
 current = current->leftChild;
 if(current == NULL) {
 parent->leftChild = tempNode;
 return;
 }
 }
 else {
 current = current->rightChild;
 if(current == NULL) {
 parent->rightChild = tempNode;
 return;
}
 }
 }
 }
}
struct node* search(int data)
{
 struct node *current = root;
 printf("\nVisiting elements: ");
 while(current->data != data)
 {
 if(current != NULL)
 printf("%d ",current->data);
```

```
 if(current->data > data)
 {
 current = current->leftChild;
 }
 else
 {
 current = current->rightChild;
 }
 if(current == NULL)
 {
 return NULL;
 }
 }
 return current;
}
void pre_order_traversal(struct node* root)
{
 if(root != NULL)
 {
 printf("%d ",root->data);
 pre_order_traversal(root->leftChild);
 pre_order_traversal(root->rightChild);
 }
}
void inorder_traversal(struct node* root)
{
 if(root != NULL)
```

```
 {
 inorder_traversal(root->leftChild);
 printf("%d ",root->data);
inorder_traversal(root->rightChild);
 }
}
void post_order_traversal(struct node* root)
{
 if(root != NULL)
 {
 post_order_traversal(root->leftChild);
 post_order_traversal(root->rightChild);
 printf("%d ", root->data);
 }
}
int main()
{
 int i;
 int array[7] = { 27, 14, 35, 10, 19, 31, 42 };
 for(i = 0; i < 7; i++)
 insert(array[i]);
 i = 31;
 struct node * temp = search(i);
 if(temp != NULL)
 {
 printf("[%d] Element found.", temp->data);
 printf("\n");
```

```
 }
 else
 {
 printf("[ x ] Element not found (%d).\n", i);
 }
 i = 15;
 temp = search(i);
 if(temp != NULL)
 {
 printf("[%d] Element found.", temp->data);
 printf("\n");
 }
 else
 {
 printf("[ x ] Element not found (%d).\n", i);
 }
 printf("\nPreorder traversal: ");
 pre_order_traversal(root);
 printf("\nInorder traversal: ");
 inorder_traversal(root);
printf("\nPost order traversal: ");
 post_order_traversal(root);
 return 0;
}
```

Output:


Visiting elements: 27 35 [31] Element found.
Visiting elements: 27 14 19 [x] Element not found (15).
Preorder traversal: 27 14 10 19 35 31 42
Inorder traversal: 10 14 19 27 31 35 42
Post order traversal: 10 19 14 31 42 35 27
...Program finished with exit code 0
Press ENTER to exit console.

EX: NO: 6

GRAPH TRAVERSAL TECHNIQUE BREADTH FIRST TRAVERSAL

AIM: To perform breadth First Traversal in Graph.

ALGORITHM:

Step 1: Stoat

Step 2: Read the number of vertices to total

Step 3: Read the adjacency matrix graph [][]

Step 4: Read the search element

Step 5: Start the traversal from the source note

Step 6: Visit the contiguous universal vertex

Step 7: Mark it as visited.

Step 8: Display it. If this is the required key and exit.

Step 9: Repeat step 3 to 8 until the queue is empty.

Step 10: Else, add it in a queue.

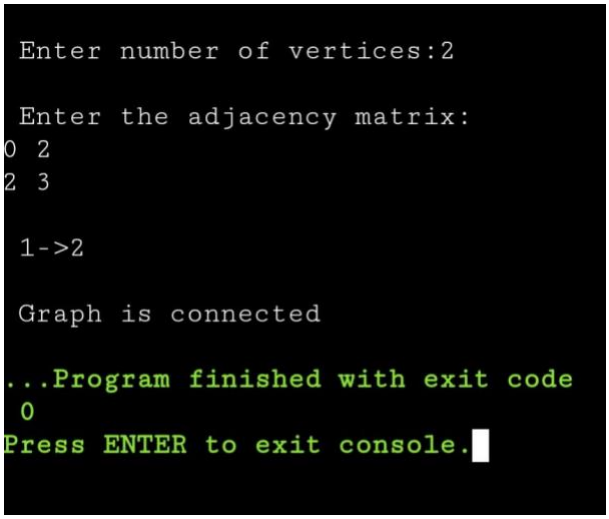
Step 11: On the off chance that no neighbouring vertex is discovered, expel the first vertex from the queue.

Step 12: Stop.

Source Code:

```
#include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(int v)
 {
        int i;
        reach[v]=1;
        for (i=1;i<=n;i++)
         if(a[v][i] && !reach[i])
 {
                printf("\n %d->%d",v,i);
                dfs(i);
}
}
void main()
{
        int i,j,count=0;
        printf("\n Enter number of vertices:");
        scanf("%d",&n);
        for (i=1;i<=n;i++)
{
                reach[i]=0;
                for (j=1;j<=n;j++)
```

```
            a[i][j]=0;
        }
        printf("\n Enter the adjacency matrix:\t");
        for (i=1;i<=n;i++)
         for (j=1;j<=n;j++)
         scanf("%d",&a[i][j]);
        dfs(1);
        printf("\n");
        for (i=1;i<=n;i++)
{
            if(reach[i])
             count++;
        }
        if(count==n)
         printf("\n Graph is connected");
            else
         printf("\n Graph is not connected");
        getch();
          }
```


Enter number of vertices:2
Enter the adjacency matrix:
0 2
2 3
1->2
Graph is connected
...Program finished with exit code
0
Press ENTER to exit console.

Output:

EX:NO:7

DIJKSTRA'S ALGORITHM TO OBTAIN THE SHORTEST PATHS

AIM: To obtain the shortest path using Dijkstra's Algorithm in True.

ALGORITHM:

Step1: Start

Step2: Create a set shortpath set that keeps track of vertices included in the shortest - path tree, i.e, whose minimum distance from the source is calculated and finalized. Initially this set is empty.

Step 3: Assign a distance value to all vertices in output graph. Initialize all distance value as INFINITE. Assign distance values as 0 for the source vertex so that it is picked first.

Step4: While shortpath set doesn't include all vertices repeat step 4, 5 and 6.

Step 5: Pick vertex u which is not there in shortpathSet and has minimum distance value.

Step6: Include u to shortpathSet.

Step 7: Update distance value of all adjacent vertices of u. to update distance values, through all adjace values. For every adjacent vertex v, if the sum of distance value of u and weight of edge u-v is less than the distance value of u, then update the distance value of v.

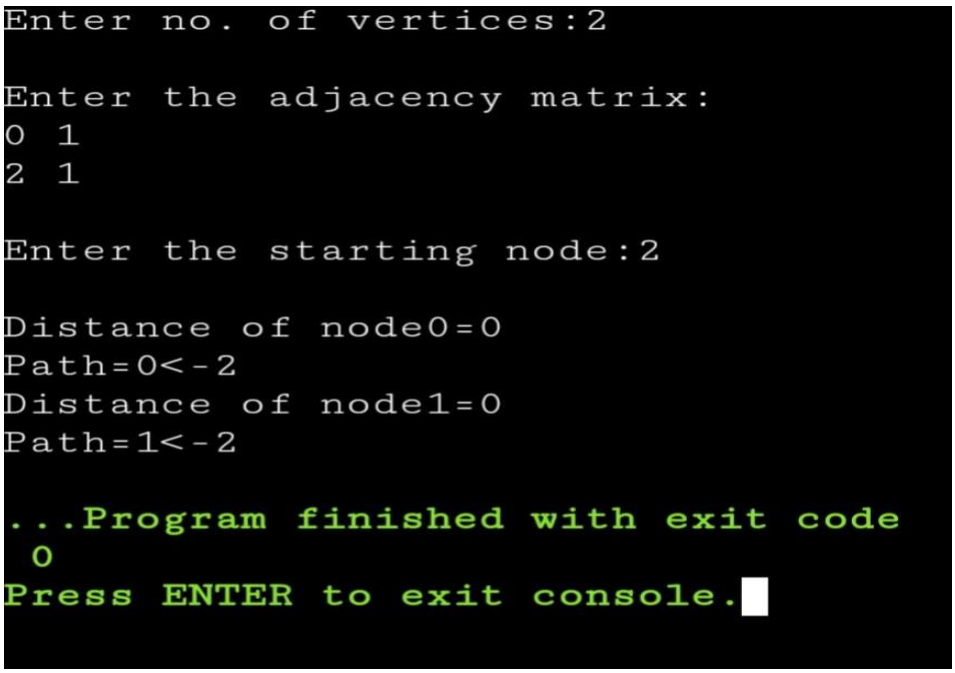
Step 8: STOP.

Source Code:

```
#include<stdio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX],int n,int startnode);
int main()
{
int G[MAX][MAX],i,j,n,u;
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
printf("\nEnter the starting node:");
scanf("%d",&u);
dijkstra(G,n,u);
return 0;
}
void dijkstra(int G[MAX][MAX],int n,int startnode)
{
int cost[MAX][MAX],distance[MAX],pred[MAX];
```

```
int visited[MAX],count,mindistance,nextnode,i,j;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{ if(G[i][j]==0)
cost[i][j]=INFINITY;
else
cost[i][j]=G[i][j];
}
for(i=0;i<n;i++)
{
distance[i]=cost[startnode][i];
pred[i]=startnode;
visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count<n-1)
{
mindistance=INFINITY;
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
```

```
}
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{
distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\nPath=%d",i);
j=i;
do
{
j=pred[j];
printf("<-%d",j);
}while(j!=startnode);
}
}
```


Enter no. of vertices:2
Enter the adjacency matrix:
0 1
2 1
Enter the starting node:2
Distance of node0=0
Path=0<-2
Distance of node1=0
Path=1<-2
...Program finished with exit code
0
Press ENTER to exit console.

Output:

EX:NO:8

BINARY SEARCH USING DIVIDE AND CONQUER TECHNIQUE

Aim: To search the element in the given array by using Binary Search with Divide and Conquer Technique.

ALGORITHM:

Step1: Start

Step 2: Compare target with middle element.

Step 3: If target matches with the middle element, we return the mid index.

Step 4: Else if target is greater than the mid element, then target can only lie in the right half sub array after the mid element. So, we recur for the right half.

Step 5: Else recur for the left half.

Step 6: Stop.

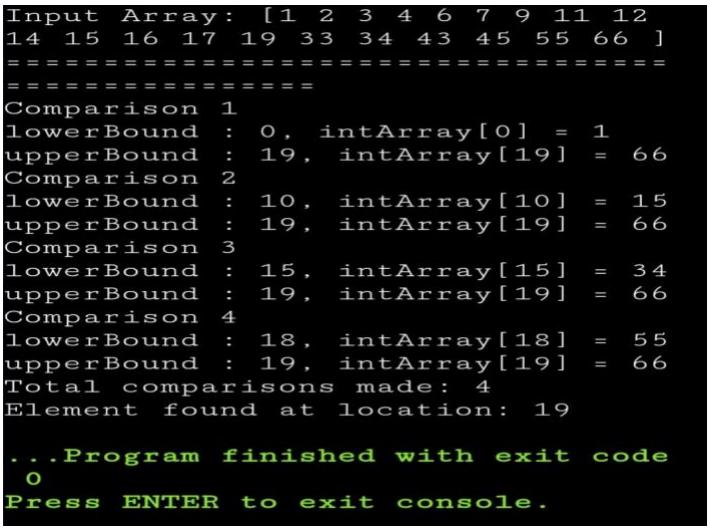
Source Code:

```
#include <stdio.h>
#define MAX 20
int intArray[MAX] = {1,2,3,4,6,7,9,11,12,14,15,16,17,19,33,34,43,45,55,66};
void printline(int count) {
   int i;
   for(i = 0;i <count-1;i++) {
      printf("=");
   }

   printf("=\n");
}
int find(int data) {
   int lowerBound = 0;
   int upperBound = MAX -1;
   int midPoint = -1;
   int comparisons = 0;
   int index = -1;

   while(lowerBound <= upperBound) {
      printf("Comparison %d\n" , (comparisons +1) );
      printf("lowerBound : %d, intArray[%d] = %d\n",lowerBound,lowerBound,
         intArray[lowerBound]);
      printf("upperBound : %d, intArray[%d] = %d\n",upperBound,upperBound,
         intArray[upperBound]);
      comparisons++;
```

```
      midPoint = lowerBound + (upperBound - lowerBound) / 2;
      if(intArray[midPoint] == data) {
        index = midPoint;
        break;
      } else {
        if(intArray[midPoint] < data) {
          lowerBound = midPoint + 1;
        }
        else {
          upperBound = midPoint -1;
        }
      }
   }
   printf("Total comparisons made: %d" , comparisons);
   return index;
}

void display() {
   int i;
   printf("[");
   for(i = 0;i<MAX;i++) {
      printf("%d ",intArray[i]);
   }

   printf("]\n");
```

```
}

void main() {
   printf("Input Array: ");
   display();
   printline(50);
   int location = find(55);
   if(location != -1)
      printf("\nElement found at location: %d" ,(location+1));
   else
      printf("\nElement not found.");
}
```

Output:



EX:NO:9

SORTING ALGORITHM USING DIVIDE AND CONQUER TECHNIQUE

Aim: To implement a sorting algorithm using divide and conquer technique.

ALGORITHM:

Step 1: Start

Step 2: Choose the highest index value has pivot

Step 3: Take two variable to point left and right of the list excluding pivot.

Step 4: Left points to the low index.

Step 5: Right points to the high

Step 6: While value at left is less than pivot move right

Step 7: While value at right is greater than pivot move left.

Step 8: If both step 6 and 7 does not match swap left and right.

Step 9: If left ≥ right, the point where they met is new point.

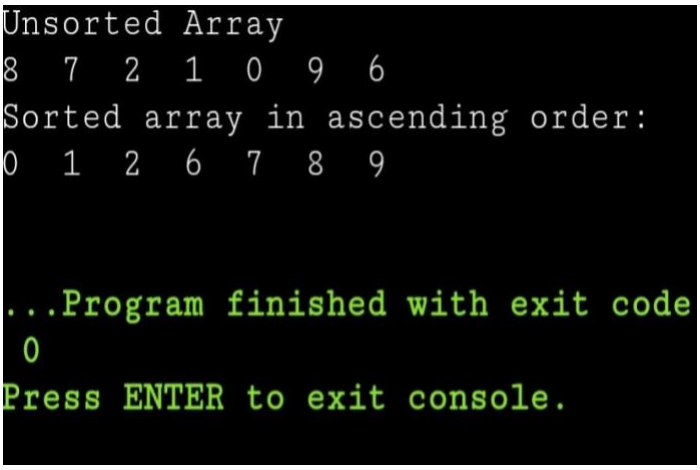
Step 10: Stop.

Source Code:

```
#include <stdio.h>
void swap(int *a, int *b) {
  int t = *a;
  *a = *b;
  *b = t;
}
int partition(int array[], int low, int high) {
  int pivot = array[high];
  int i = (low - 1);
  for (int j = low; j < high; j++) {
    if (array[j] <= pivot) {
      i++;
      swap(&array[i], &array[j]);
    }
  }
  swap(&array[i + 1], &array[high]);
  return (i + 1);
}
void quickSort(int array[], int low, int high) {
  if (low < high) {
    int pi = partition(array, low, high);
    quickSort(array, low, pi - 1);
    quickSort(array, pi + 1, high);
```

```
  }
}
void printArray(int array[], int size) {
  for (int i = 0; i < size; ++i) {
    printf("%d  ", array[i]);
  }
  printf("\n");
}
int main() {
  int data[] = {8, 7, 2, 1, 0, 9, 6};
  int n = sizeof(data) / sizeof(data[0]);
  printf("Unsorted Array\n");
  printArray(data, n);
  quickSort(data, 0, n - 1);
  printf("Sorted array in ascending order: \n");
  printArray(data, n);
}
```

Output:


Unsorted Array
8 7 2 1 0 9 6
Sorted array in ascending order:
0 1 2 6 7 8 9
...Program finished with exit code
0
Press ENTER to exit console.

Ex: No: 10

KNAPSACK USING GREEDY TECHNIQUE

Aim: To obtain a maximum profit by using the concept of KNAPSACK with Greedy Technique.

ALGORITHM:

Step 1: Start

Step 2: Read the number of items to n.

Step 3: Read the items and profit.

Step 4: Read the capacity of knapsack.

Step 5: Repeat the step 6 for i=1 to n.

Step 6: ratio [i] = profit [i] / weight [i];

Step 7: Repeat the step 8-18 for i=0 to i<n

Step 8: Repeat the step 9-18 for j=i+1 to j<n

Step 9: if (ratio [i] < ratio [j]), then repeat step 11-9

Step 10: Set temp = ratio [j];

Step 11: Set ratio [j] = ratio [i];

Step 12: Set ratio [i] = temp;

Step 13: Set temp = weighted [j];

Step 14: Set weight [j] = temp;

Step 15: Set weight [i] = temp;

Step 16: Set temp = profit [j];

Step 17: Set profit [j] = profit [i];

Step 18: Set profit [i] = temp;

Step 19: Repeat the steps 20-23.

Step 20: if (weight [i] > capacity) then exit else goto step 21

Step21: Set Total Value = Total Value + profit [i] and
Capacity = Capacity - weight [i];

Step22: if (i<n) then goto step23.

Step 23: Total Value = Total Value + (ratio [i] * Capacity);

Step 24: Print Total Value

Step25: Stop.

Source Code:

```
# include<stdio.h>
void knapsack(int n, float weight[], float profit[], float capacity) {
   float x[20], tp = 0;
   int i, j, u;
   u = capacity;
   for (i = 0; i < n; i++)
      x[i] = 0.0;
   for (i = 0; i < n; i++) {
      if (weight[i] > u)
         break;
      else {
         x[i] = 1.0;
         tp = tp + profit[i];
         u = u - weight[i];
      }
   }
   if (i < n)
      x[i] = u / weight[i];
   tp = tp + (x[i] * profit[i]);
   printf("\nThe result vector is:- ");
   for (i = 0; i < n; i++)
      printf("%f\t", x[i]);
   printf("\nMaximum profit is:- %f", tp);
}
```

```
int main() {
   float weight[20], profit[20], capacity;
   int num, i, j;
   float ratio[20], temp;
   printf("\nEnter the no. of objects:- ");
   scanf("%d", &num);
   printf("\nEnter the wts and profits of each object:- ");
   for (i = 0; i < num; i++) {
      scanf("%f %f", &weight[i], &profit[i]);
   }
   printf("\nEnter the capacityacity of knapsack:- ");
   scanf("%f", &capacity);
   for (i = 0; i < num; i++) {
      ratio[i] = profit[i] / weight[i];
   }
   for (i = 0; i < num; i++) {
      for (j = i + 1; j < num; j++) {
         if (ratio[i] < ratio[j]) {
            temp = ratio[j];
            ratio[j] = ratio[i];
            ratio[i] = temp;
            temp = weight[j];
            weight[j] = weight[i];
            weight[i] = temp;
            temp = profit[j];
```

```
            profit[j] = profit[i];
            profit[i] = temp;
          }
        }
    }
    knapsack(num, weight, profit, capacity);
    return(0);
}
```

```
Enter the no. of objects:- 2

Enter the wts and profits of each
object:-
0 2
6 9

Enter the capacityacity of knapsac
k:- 9

The result vector is:- 1.000000 1.
000000
Maximum profit is:- 11.000000

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

Output:

EX:NO:11

TRAVELLING SALESMAN ALGORITHM USING DYNAMIC PROGRAMMING TECHNIQUES.

AIM: To implement the concept of Travelling salesman algorithm using Dynamic programming techniques.

ALGORITHM:

Step 1: Stoot

Step 2: Read the number of villages to n.

Step 3: Read the cost matrix to a[][]

Step4: Select an arbitary vertex

Step5: Find the vertex that is nearest to this starting vertex to form a initial path of one degree edge

Step6: Repeat step 6-9 untill all the vertics of graph G are included.

Step 7: Let V denote the latest vertex that was added to path

Step 8: Select the closest one to V.

Step 9: Add path, edge- connecting V and this vertex

Step 10: Join starting vertex and last vertex added by edge and form the circuit.

Step11: Print path and cost of the shortest path obtained

Step 12: Stop.

Source Code:

```
#include<stdio.h>
int ary[10][10],completed[10],n,cost=0;
void takeInput()
{
int i,j;
printf("Enter the number of villages: ");
scanf("%d",&n);
printf("\nEnter the Cost Matrix\n");
for(i=0;i < n;i++)
{
printf("\nEnter Elements of Row: %d\n",i+1);
for( j=0;j < n;j++)
scanf("%d",&ary[i][j]);
completed[i]=0;
}
printf("\n\nThe cost list is:");
for( i=0;i < n;i++)
{
printf("\n");
for(j=0;j < n;j++)
printf("\t%d",ary[i][j]);
}
}
void mincost(int city)
```

```
{
int i,ncity;
completed[city]=1;
printf("%d--->",city+1);
ncity=least(city);
if(ncity==999)
{
ncity=0;
printf("%d",ncity+1);
cost+=ary[city][ncity];
return;
}
mincost(ncity);
}
int least(int c)
{
int i,nc=999;
int min=999,kmin;
for(i=0;i < n;i++)
{
if((ary[c][i]!=0)&&(completed[i]==0))
if(ary[c][i]+ary[i][c] < min)
{
min=ary[i][0]+ary[c][i];
kmin=ary[c][i];
```

```
nc=i;
}
}
if(min!=999)
cost+=kmin;
return nc;
}
int main()
{
takeInput();
printf("\n\nThe Path is:\n");
mincost(0);
printf("\n\nMinimum cost is %d\n ",cost);
return 0;
}
```

Output:

```
Enter the number of villages: 2

Enter the Cost Matrix

Enter Elements of Row: 1
0 3

Enter Elements of Row: 2
30 6


The cost list is:
        0       3
        30      6

The Path is:
1--->2--->1

Minimum cost is 33


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

EX:NO:12 EIGHT QUEENS WITH THE DESIGN OF BACKTRACKING

Aim: To implement the concept of Eight Queen with the Design of Backtracking.

ALGORITHM:

Step 1: Start

Step 2: Begin from the leftmost column.

Step 3: If all queens are placed, return TRUE and print configuration.

Step 4: Check for all rows in the current column

*If queen placed safely mark row and column, and recursively check if we approach in current configuration, do we obtain a solution or not.

*If placing yields a solutions, return TRUE.

*If placing does not yield a solution, unmark and try other rows.

Step 5: If all rows tried and solution not obtained return false and backtrack.

Step 6: Stop.

Source Code:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
int a[30],count=0;
int place(int pos)
 {
   int i;
   for (i=1;i<pos;i++) {
   if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
   return 0;
}
return 1;
}
void print_sol(int n) {
int i,j;
        count++;
        printf("\n\nSolution #%d:\n",count);
        for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++) {
        if(a[i]==j)
        printf("Q\t"); else
```

```
        printf("*\t");
        }
        printf("\n");
        }
}
void queen(int n) {
        int k=1;
        a[k]=0;
        while(k!=0) {
        a[k]=a[k]+1;
while((a[k]<=n)&&!place(k))
        a[k]++;
        if(a[k]<=n) {
        if(k==n)
        print_sol(n); else {
        k++;
        a[k]=0;
        }
        } else
          k--;
        }
}
void main() {
        int i,n;
        printf("Enter the number of Queens\n");
```

```
    scanf("%d",&n);
    queen(n);
    printf("\nTotal solutions=%d",count);
    getch();
}
```

Output: