Practical File

Lab Name: DS LAB ***Lab Code: KCS-351***

.

Estd. 2000

Name: ARADHY TRIPATHI

Adm.No: 2019B111014 **Univ. Roll No: 1900321290016**

Course: B.TECH **Branch: CEIT**

Sem: 3rd

ABES
Engineering College

(College Code-032)

NAAC Accredited, NBA Accredited Branches (CSE, ECE, EN & IT)

19th Km. Stone, NH-09, Ghaziabad - 201009 (UP), India
Phone : 0120-7135112, 9999889341 Fax : 0120-7135115 Website : www.abes.ac.in, Email: info@abes.ac.in

LIST OF PRACTICALS

1. Different operation perform on linked list.(creation , insertion , deletion)
2. Searching algorithms(linear search and binary search)
3. Implementation of stack using array and linked list.
4. Implementation of queue using array and linked list.
5. Sorting algorithms

a) Insertion sort
b) Selection sort
c) Bubble sort

6. Other sorting technique

a) Quick sort
b) Merge sort
c) Radix sort

7. Implementation of binary tree using array and linked list.
8. Implementation of BFS and DFS.
9. Implementation of spanning tree

PRACTICAL 1

1. **PRACTICAL STATEMENT OF PRACTICAL:**

Link list creation ,display ,insertion at begin,end,at,position,delete from end ,begin,at position

2. **OBJECTIVE OF PRACTICAL**

to create and display linked list and application using c language

3. **ALGORITHM / FLOW CHART**

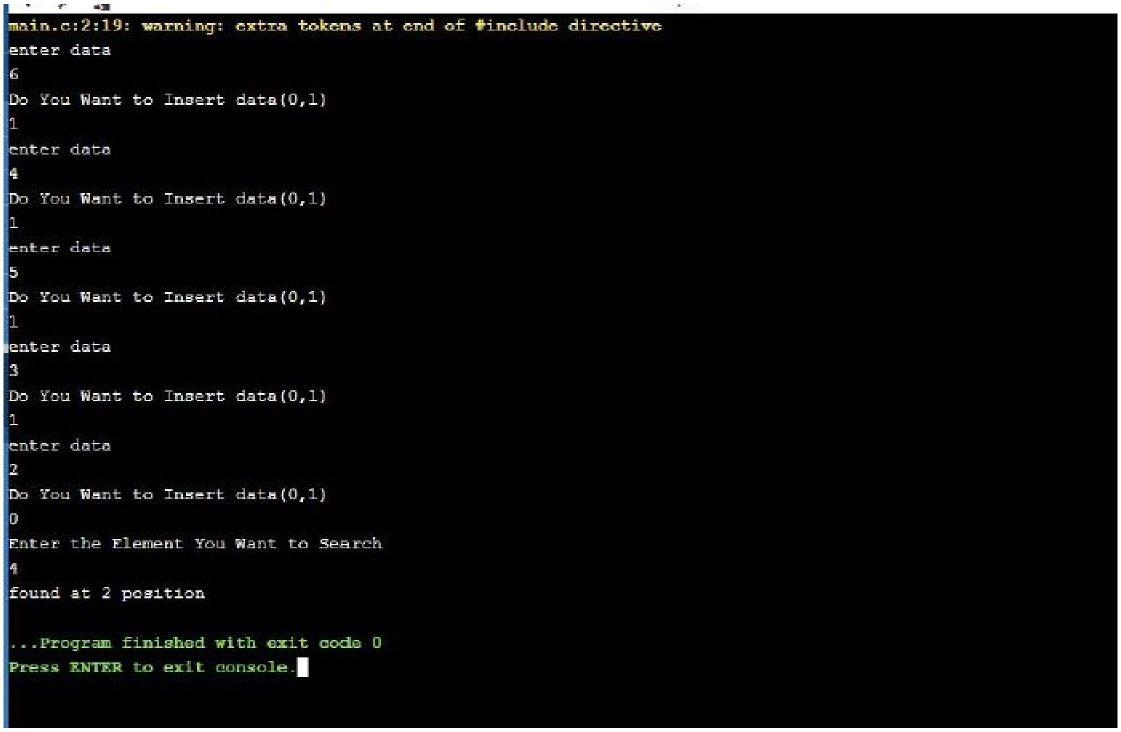
```
Struct node *temp,*ptr;
Temp=(struct node*) malloc(sizeof(struct node));
If(temp==NULL)
{
   Printf("out of memore space \n");
  Exit(0);
}
If(start==null)
{
   Start=temp;
}
Else
{
   Ptr=start;
   While(ptr->next!=null)
 {
  ptr=ptr->next;
 }
Ptr->next=temp
}
```

4. **IMPLEMENTATION**

```
     #include<stdio.h>

#include<stdlib.h>>

struct node

{
```

```
int data;
struct node *next;
};
struct node*head; void insertdata()
{
struct node*newnode,*temp; head=NULL;
int choice=1;
while(choice!=0)
{
newnode=(struct node*)malloc(sizeof(struct node));
printf("enter data \n");
scanf("%d",& newnode->data); newnode->next=NULL;
if (head==NULL)
{
temp=head=newnode;
}
else
{
temp->next =newnode; temp = newnode;
}
printf("Do You Want to Insert data(0,1)\n"); scanf("%d",&choice);
}

}

void linear_Search(){

int ele;

struct node *ptr; ptr=head; printf("Enter the Element You Want to Search \n"); scanf("%d",&ele);

int i=0;

if (ptr==NULL)

{

printf("Empty List"); return;

}

while(ptr->next!=NULL)

{ i++;

if(ptr->data==ele)

{

printf("found at %d position",i);

return ;

}

ptr=ptr->next;

}

}

int main()

{

insertdata(); linear_Search(); return 0;

}
```

5. Result /Output



```
main.c:2:19: warning: extra tokens at end of #include directive
enter data
6
Do You Want to Insert data(0,1)
1
enter data
4
Do You Want to Insert data(0,1)
1
enter data
5
Do You Want to Insert data(0,1)
1
enter data
3
Do You Want to Insert data(0,1)
1
enter data
2
Do You Want to Insert data(0,1)
0
Enter the Element You Want to Search
4
found at 2 position

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

PRACTICAL 2

1. PRACTICAL STATEMENT OF PRACTICAL:

Perform linear and binary search on the given array and linked list.

2. OBJECTIVE OF PRACTICAL :

Search a given element in the array using both linear and binary search.

Search a given element in the linked list using both linear and binary search.

3. ALGORITHM / FLOW CHART

ARRAY:

BINARY SEARCH:

1. Enter the array
2. Input the element you want to find
3. Find the mid of the array:

a. If the element matches with mid break

b. If mid is greater than element than shift to first half of the array and repeat from 3

c. If mid is smaller than element than shift to second half of the array and repeat from 3

4. IMPLEMENTATION:

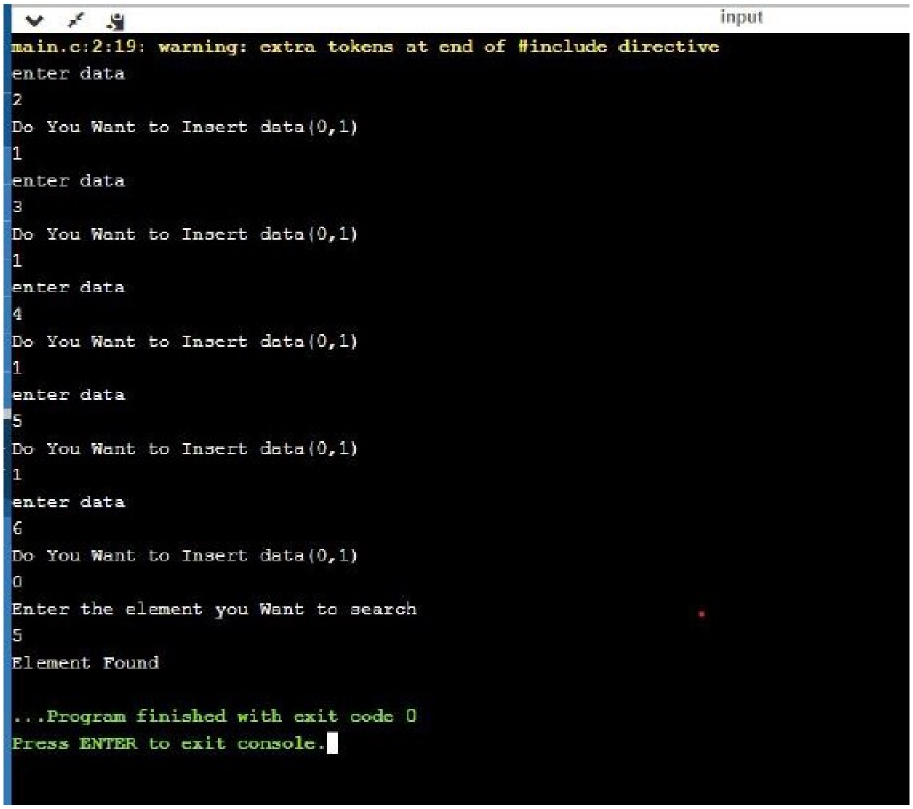
ARRAY:

BINARY SEARCH:

Code:

```
#include<stdio.h>
 main()
 {
 printf("enter size of array\n");
 int n,i,e,m;
 scanf("%d",&n);
 int a[n];
 printf("enter elements of array\n");
 for(i=0;i<n;i++)
{
        scanf("%d",&a[i]);
}
printf("Enter element to be searched\n");
scanf("%d",&e);
 //binary search
 i=0;
 while(i<=n)
 {
        m=(i+n)/2+1;
         if(e==a[m])
         {
              printf("element found at %d position",m+1);
              break;
         }
        Else if  (a[m]>e)
        {
               n=m-1;
         }
        else
        {
          i=m+1;
          {
      }
 }
```

5. Result /Output:



```
main.c:2:19: warning: extra tokens at end of #include directive
enter data
2
Do You Want to Insert data(0,1)
1
enter data
3
Do You Want to Insert data(0,1)
1
enter data
4
Do You Want to Insert data(0,1)
1
enter data
5
Do You Want to Insert data(0,1)
1
enter data
6
Do You Want to Insert data(0,1)
0
Enter the element you Want to search
5
Element Found

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

ALGORITHM:

ARRAY

LINEAR SEARCH:

1. Enter the array
2. Enter the element you want to find
3. Check the element with every element and print with found

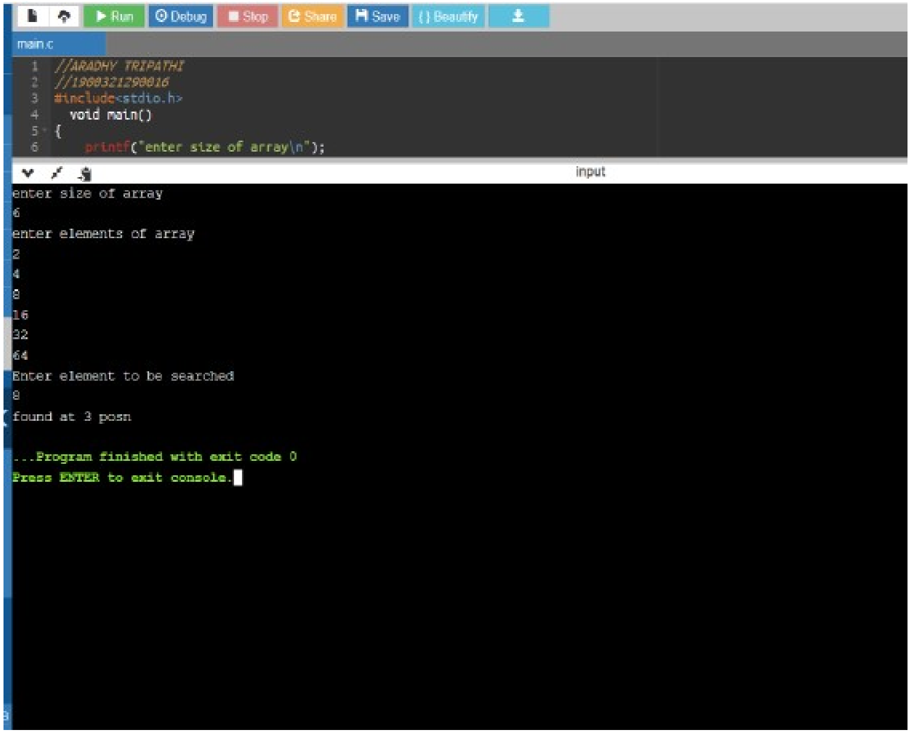
IMPLEMENTATION:

ARRAY
LINEAR SEARCH
CODE:

```
#include<stdio.h>
main()
{
    printf("enter size of array\n");
    int n,i,e,m;
    scanf("%d",&n);
    int a[n];
    printf("enter elements of array\n");
```

```
  for(i=0;i<n;i++)
  {
      scanf("%d",&a[i]);
  }
  printf("Enter element to be searched\n");
  scanf("%d",&e);
 //linear search
 for(i=0;i<n;i++)
 {
      if(e==a[i])
      {
           printf("found at %d posn",i+1);
       }
  }
}
```

RESULT:



```
//ARADHY TRIPATHI
//1900321290016
#include<stdio.h>
  void main()
{
    printf("enter size of array\n");
```

```
enter size of array
6
enter elements of array
2
4
8
16
32
64
Enter element to be searched
8
found at 3 posn

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

ALGORITHM:

LINKED LIST

LINEAR SEARCH:

1. Create a linked list
2. Enter the element
3. Traverse through the linked list and find the element and print if found

IMPLEMENTATION:

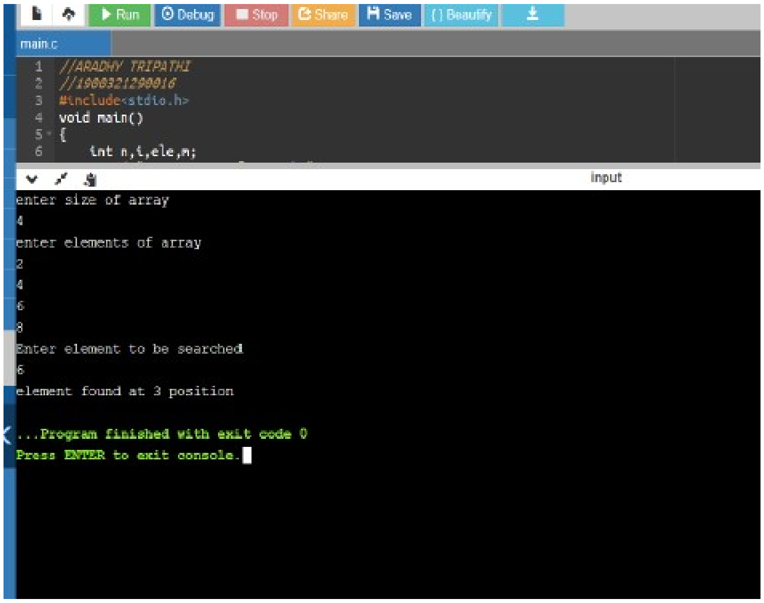
LINKED LIST

Linear search:

Code:

```
#include<stdio.h>
struct node
{
    int data;
    struct node *link;
 }*head=NULL,*ptr=NULL;
 main()
{
     printf("enter no of elements you want to insert\n");
     int n,i,e,m;
     scanf("%d",&n);
     int a[n];
     printf("enter elements of list\n");
     for(i=0;i<n;i++)
    {
           struct node *t=(struct node*)malloc(sizeof(struct node));
           scanf("%d",&m);
           t->data=m;
           t->link=NULL;
         if(head==NULL)
              {
                     head=t;
                     ptr=head;
               }
         else
              {
                    ptr->link=t;
                    ptr=t;
              }
       }
      printf("Enter element to be searched\n");
      scanf("%d",&e);
      ptr=head;
   //linear search
    for(i=0;i<n;i++,ptr=ptr->link)
    {
            if(ptr->data==e)
            printf("found at %d posn",i+1);
    }
 }
```

RESULT:



```
//ARADHY TRIPATHI
//1900321290016
#include<stdio.h>
void main()
{
    int n,i,ele,m;
```

```
enter size of array
4
enter elements of array
2
4
6
8
Enter element to be searched
6
element found at 3 position

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

ALGORITHM:

LINKED LIST

BINARY SEARCH:

```
1. Create a linked list
2. Traverse and count the no of elements
3. Enter the element you want to find
4. Find the mid of the array:
   a. If the element matches with mid break
   b. If mid is greater than element than shift to first half of the array and repeat from 3
   c. If mid is smaller than element than shift to second half of the array and repeat from 3
```

IMPLEMENTATION:

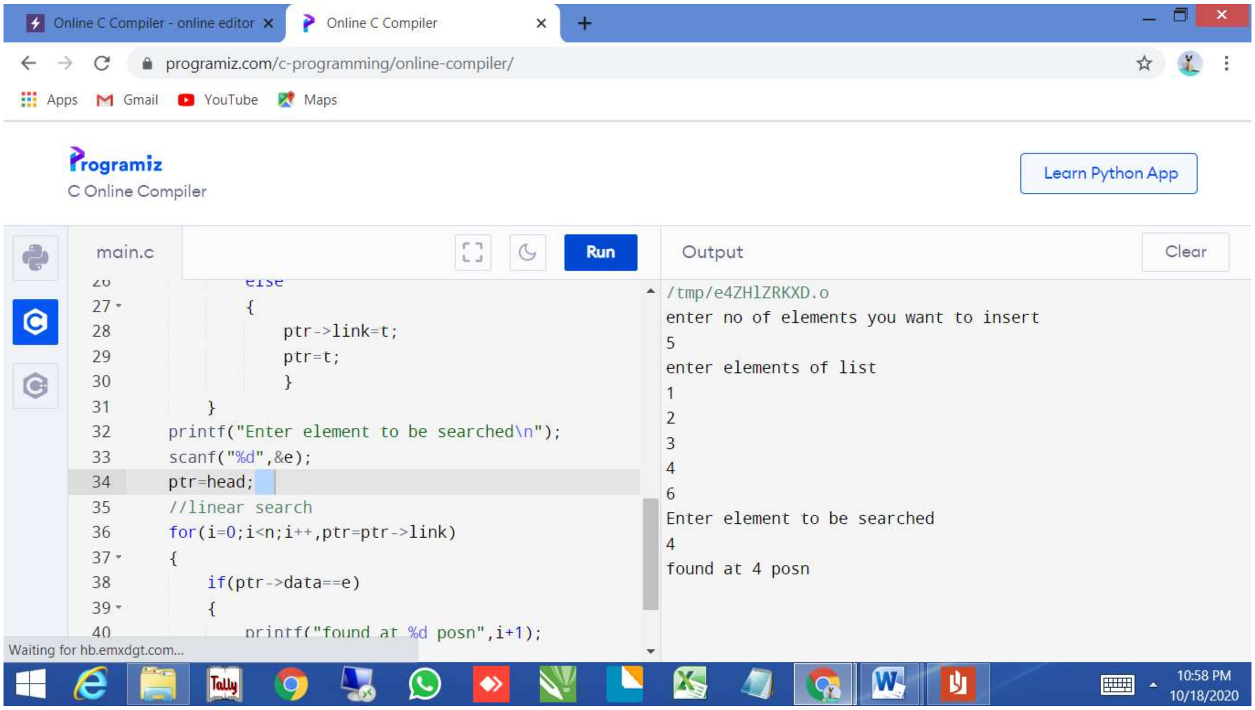
LINKED LIST:

Binary search:

Code:

```
 #include<stdio.h>
#include<stdlib.h>
struct node
{
  int data;
  struct node *link;
}*head=NULL,*ptr=NULL;
// function to find out middle element
struct node* middle(struct node* ptr,struct node *last)
{
   if (ptr == NULL)
   return NULL;
   struct node* slow = ptr;
    struct node* fast = ptr ->link;
   while (fast != last)
  {
      fast = fast ->link;
      if (fast != last)
     {
          slow = slow ->link;
          fast = fast ->link;
      }
   }
   return slow;
 }
main()
{
    printf("enter no of elements you want to insert\n");
    int n,i,e,m;
   scanf("%d",&n);
    int a[n];
    printf("enter elements of list\n");
    for(i=0;i<n;i++)
          {
              struct node *t=(struct node*)malloc(sizeof(struct node));
              scanf("%d",&m);
              t->data=m;
              t->link=NULL;
              if(head==NULL)
             {
                    head=t;
                   ptr=head;
              }
             else
             {
                  ptr->link=t;
                  ptr=t;
              }
             printf("Enter element to be searched\n");
             scanf("%d",&e);
             ptr=head;
             struct node *last=NULL;
    //binary search
    while (last == NULL || last != ptr)
   {
    // Find middle
          struct node* mid = middle(ptr,last);
       // If middle is empty
        if (mid == NULL)
                return NULL;
       // If value is present at middle
       if (mid -> data == e)
       {
               printf("found");
               break;
        }
       // If value is more than mid
      else if (mid -> data <e)
      ptr = mid ->link;
       // If the value is less than mid.
       else
       last = mid;
      }
}
```

RESULT:



```
            else
            {
                ptr->link=t;
                ptr=t;
                }
        }
    printf("Enter element to be searched\n");
    scanf("%d",&e);
    ptr=head;
    //linear search
    for(i=0;i<n;i++,ptr=ptr->link)
    {
        if(ptr->data==e)
        {
            printf("found at %d posn",i+1);
```

```
/tmp/e4ZHlZRKXD.o
enter no of elements you want to insert
5
enter elements of list
1
2
3
4
6
Enter element to be searched
4
found at 4 posn
```

.

PRACTICAL 3

1. ***PRACTICAL STATEMENT OF PRACTICAL:***

Stack implementation and conversion

(a) Stack Primitive Operations

(b) Postfix Evaluation

(c) Infix to Postfix Conversion

(d) Infix to Prefix Conversion

2. ***OBJECTIVE OF PRACTICAL***

conversion of different notation

3. IMPLEMENTATION

Stacks using array:-

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define SIZE 100
int stack[SIZE];
int top = -1;
```

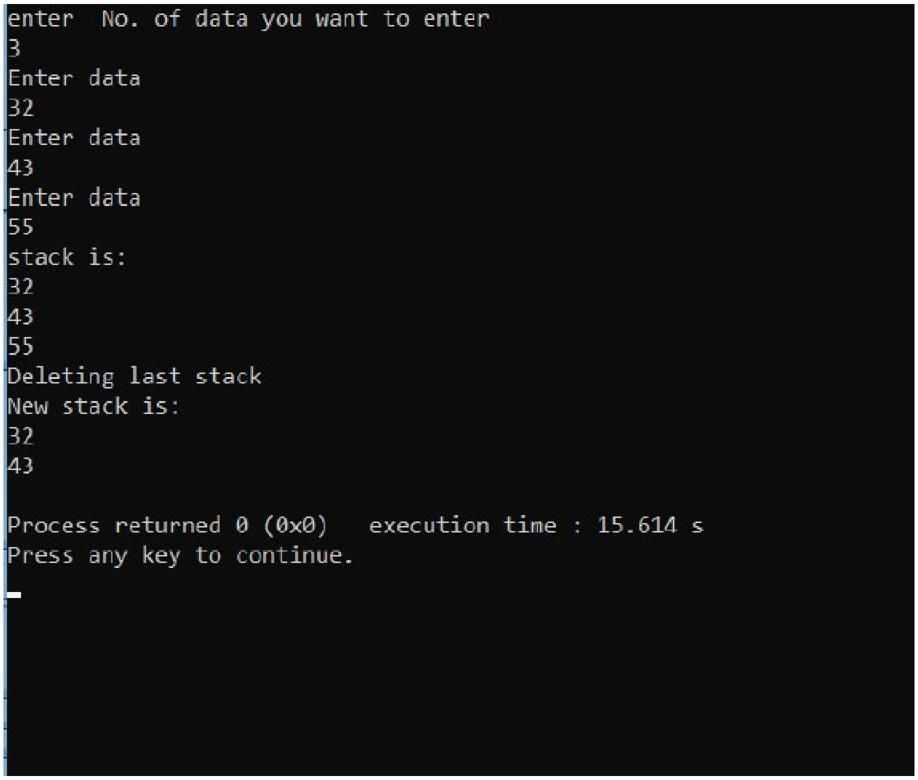
```
void push(int element);
int  pop();
int main()
{
   int choice, data;
   while(1)
   {
      printf("------------------------------------\n");
      printf("    STACK IMPLEMENTATION PROGRAM    \n");
      printf("------------------------------------\n");
      printf("1. Push\n");
      printf("2. Pop\n");
      printf("3. Size\n");
      printf("4. Exit\n");
      printf("------------------------------------\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
      switch(choice)
      {
         case 1:
            printf("Enter data to push into stack: ");
            scanf("%d", &data);
            push(data);
            break;
         case 2:
            data = pop();
            if (data != INT_MIN)
               printf("Data => %d\n", data);
            break;
         case 3:
            printf("Stack size: %d\n", top + 1);
            break;
         case 4:
            printf("Exiting from app.\n");
            exit(0);
            break;
         default:
            printf("Invalid choice, please try again.\n");
      }
      printf("\n\n");
   }
   return 0;
}
void push(int element)
{
   if (top >= SIZE)
   {
      printf("Stack Overflow, can't add more element element to stack.\n");
```

```
        return;
    }
    top++;
    stack[top] = element;
    printf("Data pushed to stack.\n");
}
int pop()
{
     if (top < 0)
    {
        printf("Stack is empty.\n");
        return INT_MIN;
    }
    return stack[top--];
}
```

Stacks using linked list:-

```
#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define CAPACITY 10000

struct stack

{

    int data;

    struct stack *next;

} *top;

int size = 0;

void push(int element);

int  pop();

int main()

{

    int choice, data;

    while(1)

    {

        printf("------------------------------------\n");

        printf("    STACK IMPLEMENTATION PROGRAM    \n");
        printf("----------------------------------------\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Size\n");
        printf("4. Exit\n");
        printf("----------------------------------------\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
                printf("Enter data to push into stack: ");
                scanf("%d", &data);
                push(data);
                break;
            case 2:
                data = pop();
                if (data != INT_MIN)
                    printf("Data => %d\n", data);
                break;
            case 3:
                printf("Stack size: %d\n", size);
                break;
            case 4:
                printf("Exiting from app.\n");
                exit(0);
                break;
            default:
                printf("Invalid choice, please try again.\n");
        }
        printf("\n\n");
    }
    return 0;
}
void push(int element)
{
    if (size >= CAPACITY)
    {
        printf("Stack Overflow, can't add more element to stack.\n");
        return;
    }
    struct stack * newNode = (struct stack *) malloc(sizeof(struct stack));
    newNode->data = element;
    newNode->next = top;
    top = newNode;
    size++;
    printf("Data pushed to stack.\n");
}
int pop()
{
    int data = 0;
    struct stack * topNode;
    if (size <= 0 || !top)
    {
        printf("Stack is empty.\n");
        return INT_MIN;
    }
    topNode = top;
    data = top->data;

    top = top->next;

    free(topNode);

    size--;

    return data;

}
```

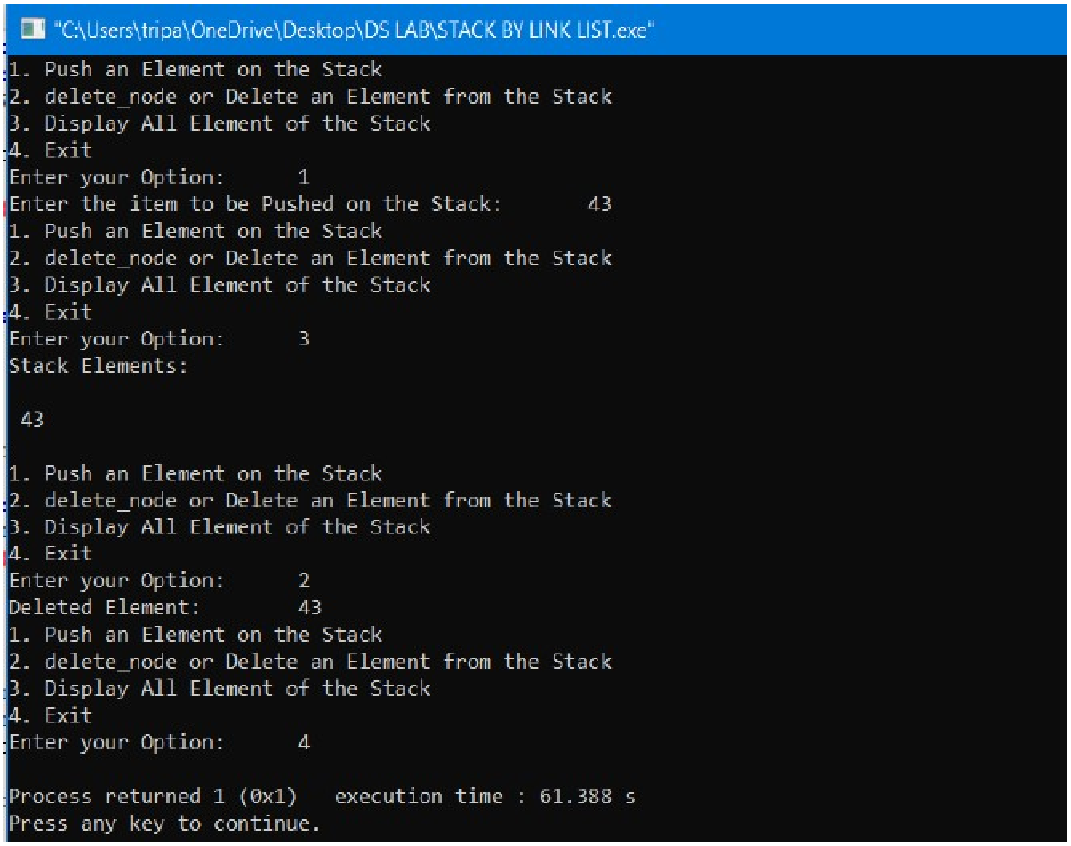
Output :



```
enter  No. of data you want to enter
3
Enter data
32
Enter data
43
Enter data
55
stack is:
32
43
55
Deleting last stack
New stack is:
32
43

Process returned 0 (0x0)   execution time : 15.614 s
Press any key to continue.
```

Stack using linked list :

Output:



"C:\Users\tripa\OneDrive\Desktop\DS LAB\STACK BY LINK LIST.exe"

```
1. Push an Element on the Stack
2. delete_node or Delete an Element from the Stack
3. Display All Element of the Stack
4. Exit
Enter your Option:      1
Enter the item to be Pushed on the Stack:       43
1. Push an Element on the Stack
2. delete_node or Delete an Element from the Stack
3. Display All Element of the Stack
4. Exit
Enter your Option:      3
Stack Elements:

 43

1. Push an Element on the Stack
2. delete_node or Delete an Element from the Stack
3. Display All Element of the Stack
4. Exit
Enter your Option:      2
Deleted Element:        43
1. Push an Element on the Stack
2. delete_node or Delete an Element from the Stack
3. Display All Element of the Stack
4. Exit
Enter your Option:      4

Process returned 1 (0x1)   execution time : 61.388 s
Press any key to continue.
```

PRACTICAL 4

1. PRACTICAL STATEMENT OF PRACTICAL:

   Queue implementation using Array and Linked List

2. OBJECTIVE OF PRACTICAL

   Queue implementation using Array and linked list in c

3. ALGORITHM / FLOW CHART

   USING ARRAY-

```
#include<stdio.h>
 #define n 1000
 int main()
 {
        int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;
        printf("Queue using Array");
        printf("\n1.Insertion \n2.Deletion \n3.Display \n4.Exit");
        while(ch)
        {
                printf("\nEnter the Choice:");
                scanf("%d",&ch);
                 switch(ch)
                {
                       case 1:
                                  if(rear==x)
                                  printf("\n Queue is Full");
                                  else
                                 {
                                         printf("\n Enter no %d:",j++);
                                         scanf("%d",&queue[rear++]);
                                 }
                                 break;
                        case 2:
                                 if(front==rear)
                                 {
                                         printf("\n Queue is empty");
                                 }
                               else
                              {
                                         printf("\n Deleted Element is %d",queue[front++]);
                                         x++;
                              }
                             break;
                         case 3:
                                printf("\nQueue Elements are:\n ");
                                if(front==rear)
                                  printf("\n Queue is Empty");
                                else
                               {
                                        for(i=front; i<rear; i++)
                                       {
                                            printf("%d ",queue[i]);
                                        }
                               break;
```

```
                    case 4:
                         exit(0);
                   default:
                        printf("Wrong Choice: please see the options");
                     }
                }
            }
      return 0;
}
```

USING LINKED LIST-

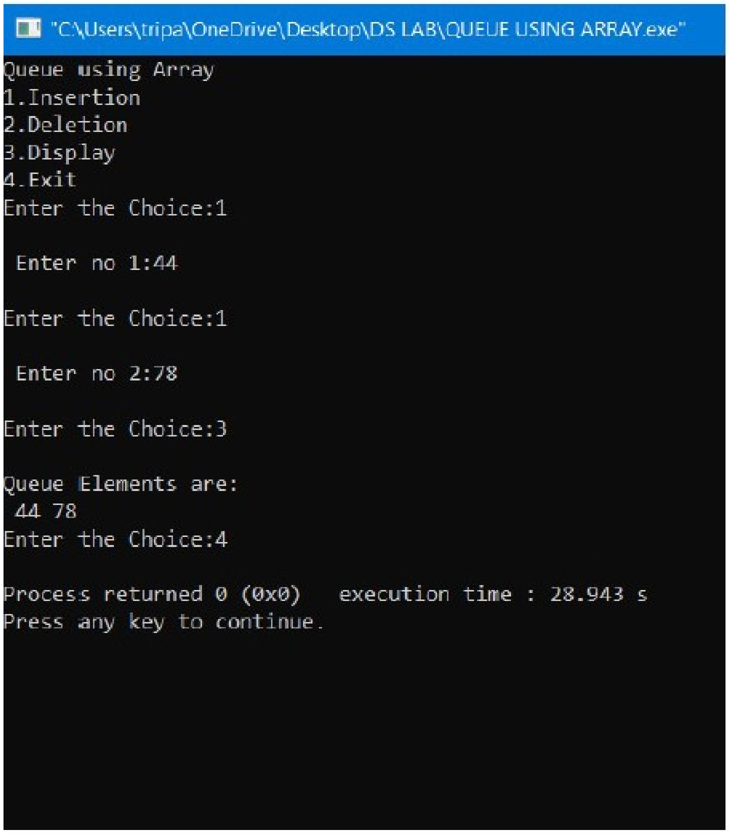
```
#include<stdio.h>
 struct Node
{
      int data;
     struct Node *link;
 };
struct Node* root=NULL;

void append();
void deletefirst();
 void display();
void length();

int main()
{
      int choice;
     printf("Queue Implementation using Linked List  ");
     while(1)
    {
           printf("1. Insert in Queue\n");
           printf("2. Delete From Queue\n");
           printf("3. Display Queue\n");
           printf("4. Front of the Queue\n");
           printf("5. Size of Queue\n");
             printf("6. Exit\n");
             printf("Enter your choice: ");
             scanf("%d",&choice);
             switch(choice)
             {
                   case 1: append();
                         break;
                   case 2: deletefirst();
```

```
                        break;
                  case 3: display();
                        break;
                  case 4: if(root==NULL)
                              printf("\n Queue is Empty!!!\n");
                              else
                                   printf("\n Data at front of the queue is %d \n",root->data);
                              break;
                  case 5: length();
                        break;
                  case 6: exit(0);
                 default:
                     printf("\n invalid number \n");
                 }
            }
    return 0;
 }

void append()
 {
    struct Node *temp;
    temp=(struct Node*)malloc(sizeof(struct Node));
    printf("Insert the value you want to enter: ");
    scanf("%d",&temp->data);
    temp->link=NULL;
    if(root==NULL)
   {
       root=temp;
   }
   else
  {
         struct Node *p;
          p=root;
          while(p->link!=NULL)
          {
                  p=p->link;
              }
             p->link=temp;
          }
         printf("\n Data inserted in Queue!!!\n");
   }

void display()
 {
```

```
 struct Node *temp;
 temp=root;
 if(root==NULL)
{
    printf("\n Queue is Empty!!!\n");
    return;
  }
 while(temp->link!=NULL)
 {
       printf("%d --> ",temp->data);
       temp=temp->link;
  }
 printf("%d\n",temp->data);
}


void length()
 {
      struct Node *temp;
      int count=0;
      temp=root;
      if(temp==NULL)
      {
           printf("\n Queue is Empty!!!\n");
           return;
       }
      while(temp->link!=NULL)
     {
          count++;
         temp=temp->link;
       }
    printf("\n Size of the queue is %d \n",count+1);
 }

void deletefirst()

 {
        struct Node *x;
         x=root;
         if(root==NULL)
        {
               printf("\n Queue is Empty!!!\n");
               return;
          }
        printf("\n Deleted element is: %d\n", root->data);
        root=root->link;
        free(x);
      }
```

5. RESULT

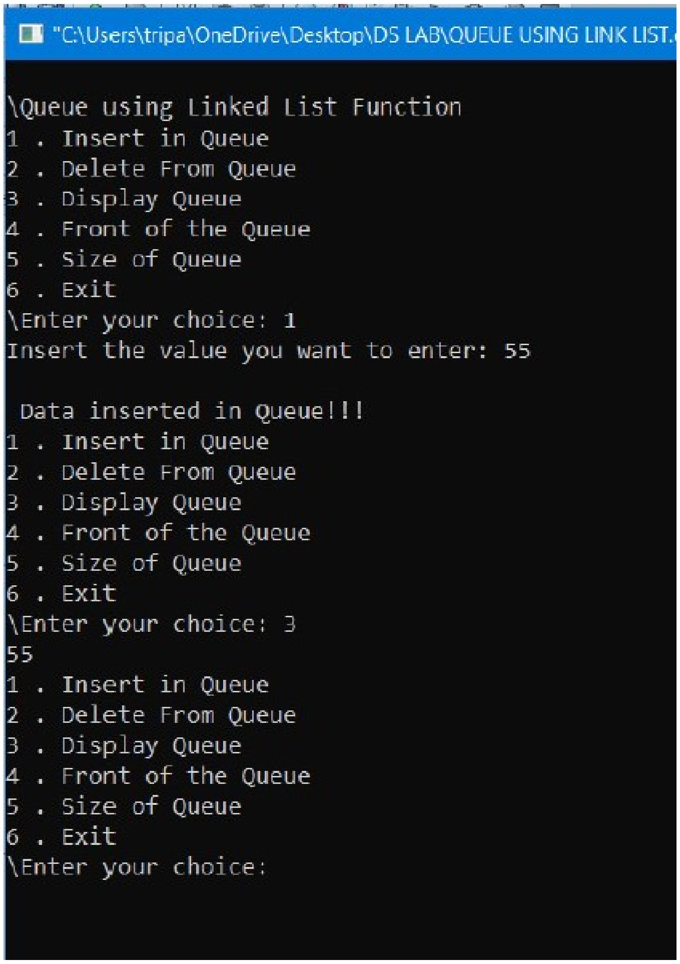
USING ARRAY



"C:\Users\tripa\OneDrive\Desktop\DS LAB\QUEUE USING ARRAY.exe"

```
Queue using Array
1.Insertion
2.Deletion
3.Display
4.Exit
Enter the Choice:1

 Enter no 1:44

Enter the Choice:1

 Enter no 2:78

Enter the Choice:3

Queue Elements are:
 44 78
Enter the Choice:4

Process returned 0 (0x0)   execution time : 28.943 s
Press any key to continue.
```

USING LINK LIST


"C:\Users\tripa\OneDrive\Desktop\DS LAB\QUEUE USING LINK LIST.

```
\Queue using Linked List Function
1 . Insert in Queue
2 . Delete From Queue
3 . Display Queue
4 . Front of the Queue
5 . Size of Queue
6 . Exit
\Enter your choice: 1
Insert the value you want to enter: 55

 Data inserted in Queue!!!
1 . Insert in Queue
2 . Delete From Queue
3 . Display Queue
4 . Front of the Queue
5 . Size of Queue
6 . Exit
\Enter your choice: 3
55
1 . Insert in Queue
2 . Delete From Queue
3 . Display Queue
4 . Front of the Queue
5 . Size of Queue
6 . Exit
\Enter your choice:
```

PRACTICAL 5

1. PRACTICAL STATEMENT OF PRACTICAL:

Implementation of Bubble Sort .

2. OBJECTIVE OF PRACTICAL

- Creation of Array .
- Implement Bubble Sort.
- Display sorted array.

3. ALGORITHM / FLOW CHART

- Starting with the first element(index = 0) .
- compare the current element with the next element of the array.
- If the current element is greater than the next element of the array, swap them.
- If the current element is less than the next element, move to the next element.
- Repeat the steps untill array is sorted.

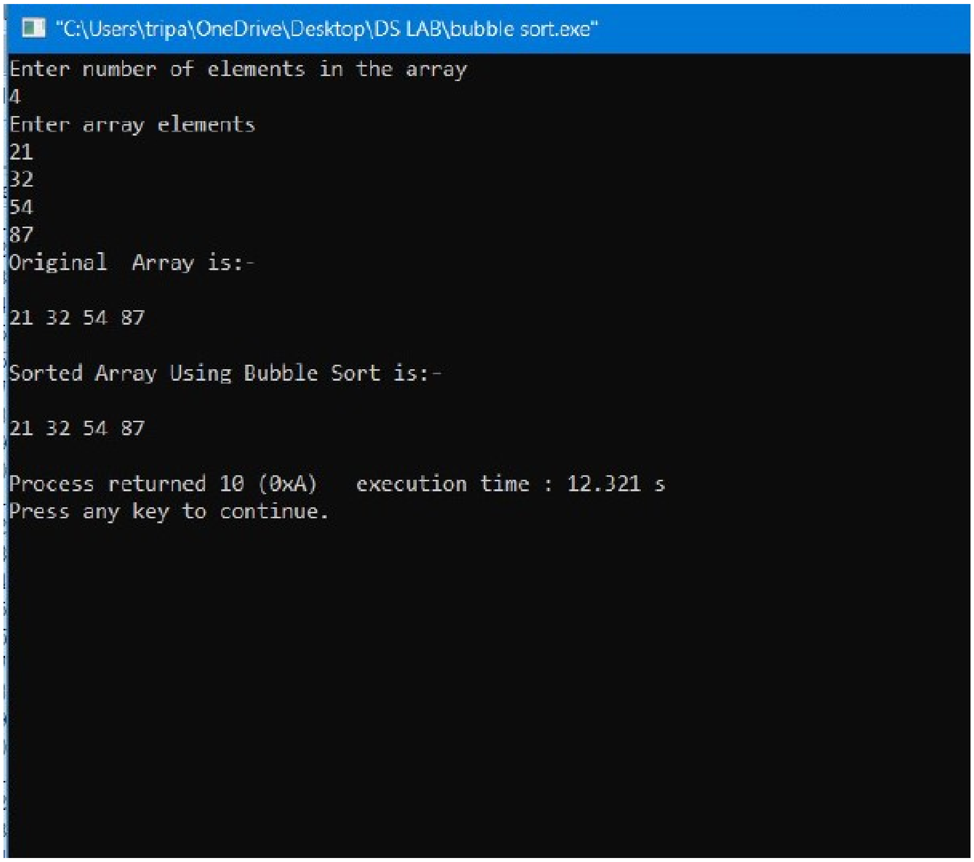
4. IMPLEMENTATION

Source Code:-

```
#include <stdio.h>
void main() {
intn,temp;
printf("Enter number of elements in the array\n");
scanf("%d",&n);
int arr[n];
printf("Enter array elements\n");
  for(inti=0;i<n;i++)
scanf("%d",&arr[i]);
printf("Original  Array is:-\n");
printf("\n");
  for (inti=0; i< n; i++)
printf("%d ", arr[i]);
printf("\n");
  for (inti = 0 ; i< n - 1; i++)
 {
   for (int j = 0 ; j < n - i - 1; j++)
   {
     if (arr[j] >arr[j+1])
     {
      temp      = arr[j];
arr[j]   = arr[j+1];
arr[j+1] = temp;
      }
    }
  }
printf("\n");
printf("Sorted Array Using Bubble Sort is:-\n");
printf("\n");
  for (inti=0; i< n; i++)
printf("%d ", arr[i]);
printf("\n");
}
```

5. Result /Output

The result of the above implementation is that we will be able to sort any array of integers in ascending order.



```
Enter number of elements in the array
4
Enter array elements
21
32
54
87
Original  Array is:-

21 32 54 87

Sorted Array Using Bubble Sort is:-

21 32 54 87

Process returned 10 (0xA)   execution time : 12.321 s
Press any key to continue.
```

PRACTICAL 6

1.PRACTICAL STATEMENT OF PRACTICAL:

Implementation of Selection Sort .

2.OBJECTIVE OF PRACTICAL

- Creation of Array .
- Implement Selection Sort.
- Display sorted array.

3.ALGORITHM / FLOW CHART

- Set MIN to location 0
- Search the minimum element in the list
- Swap with value at location MIN
- Increment MIN to point to next element
- Repeat until list is sorted

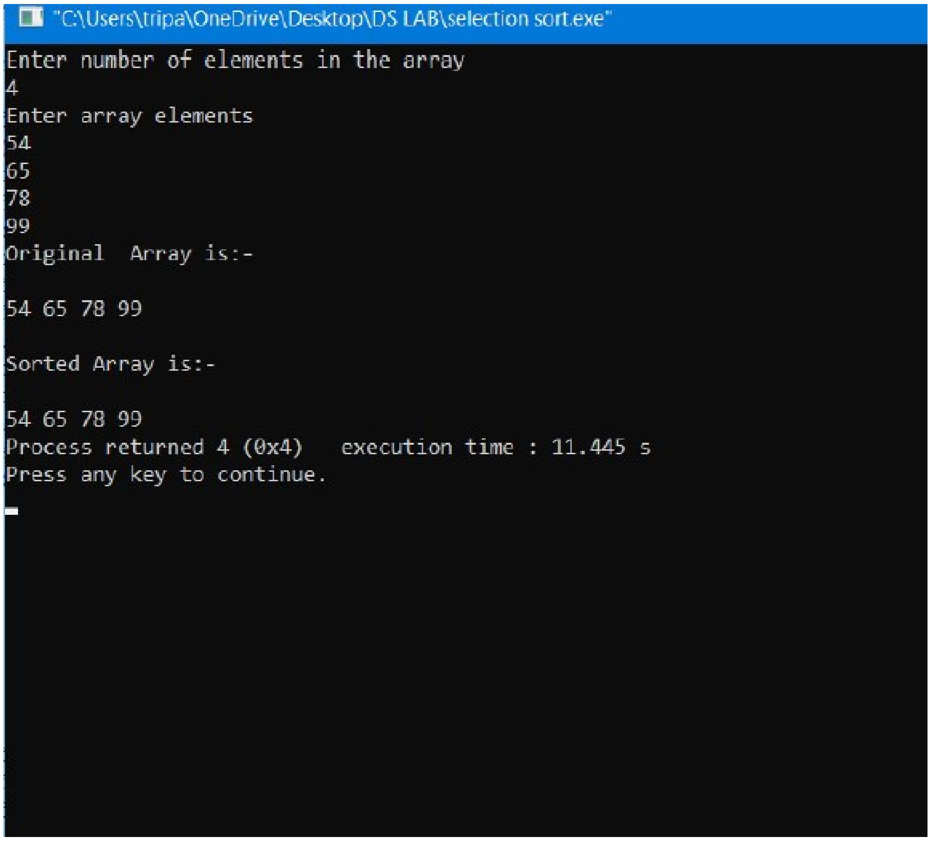
4. IMPLEMENTATION

Source Code:-

```
#include <stdio.h>
void main() {
intn,position, temp;
printf("Enter number of elements in the array\n");
scanf("%d",&n);
intarr[n];
printf("Enter array elements\n");
  for(inti=0;i<n;i++)
scanf("%d",&arr[i]);
printf("Original  Array is:-\n");
printf("\n");
  for (inti=0; i< n; i++)
printf("%d ", arr[i]);
printf("\n");
  for (inti = 0; i< (n - 1); i++) {
    position = i;
    for (int j = i + 1; j < n; j++) {
      if (arr[position] >arr[j])
        position = j;
    }
    if (position != i) {
      temp = arr[i];
arr[i] = arr[position];
arr[position] = temp;
    }
  }
printf("\n");
printf("Sorted Array is:-\n");
printf("\n");
  for (inti = 0; i< n; i++)
printf("%d ", arr[i]);
  return 0;
}
```

5.Result /Output

The result of the above implementation is that we will be able to sort any array of integers in ascending order.



"C:\Users\tripa\OneDrive\Desktop\DS LAB\selection sort.exe"

```
Enter number of elements in the array
4
Enter array elements
54
65
78
99
Original  Array is:-

54 65 78 99

Sorted Array is:-

54 65 78 99
Process returned 4 (0x4)   execution time : 11.445 s
Press any key to continue.
```

PRACTICAL 7

1.PRACTICAL STATEMENT OF PRACTICAL:

Implementation of Insertion Sort .

2.OBJECTIVE OF PRACTICAL

Creation of Array .
Implement Insertion Sort.
Display sorted array.

3. ALGORITHM / FLOW CHART

If it is the first element, it is already sorted. Return 1.
Pick next element.
Compare with all elements in the sorted sub-list.
Shift all the elements in the sorted sub-list that is greater than the value to be sorted.
Insert the value.
Repeat until list is sorted.

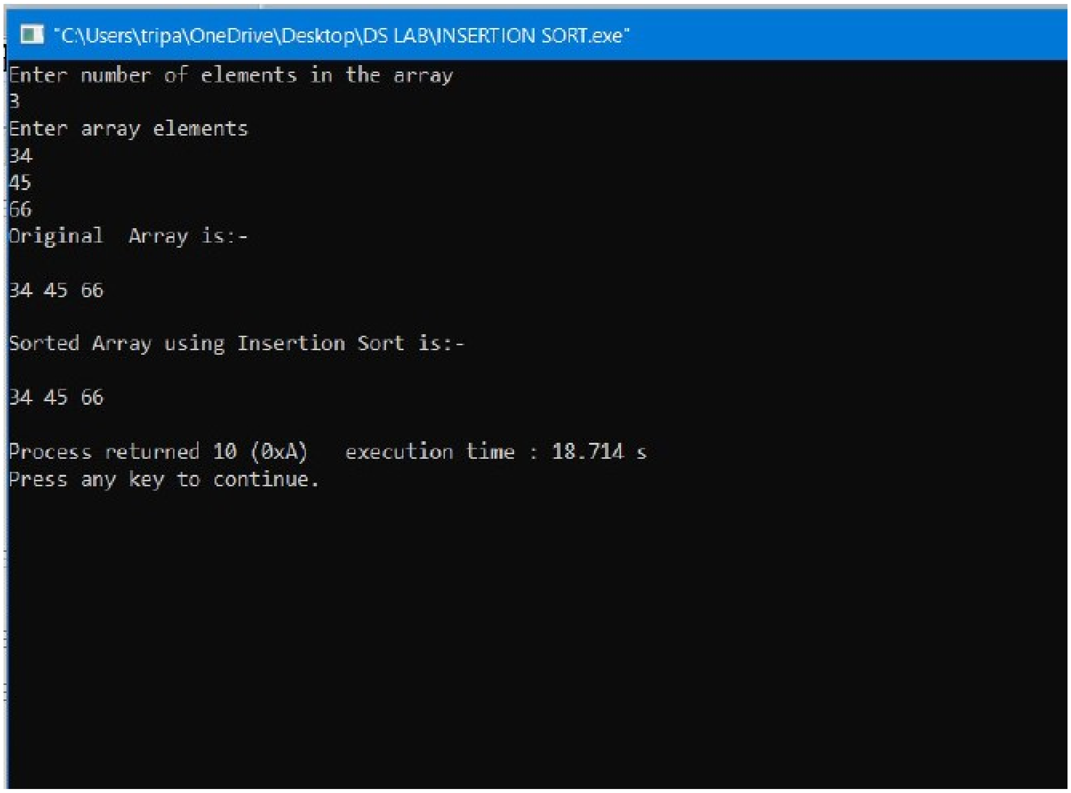
4. IMPLEMENTATION

Source Code:-

```
#include <stdio.h>
void main() {
intn,i,j,temp,flag=0;
printf("Enter number of elements in the array\n");
scanf("%d",&n);
intarr[n];
printf("Enter array elements\n");
  for(inti=0;i<n;i++)
scanf("%d",&arr[i]);
printf("Original  Array is:-\n");printf("\n");
  for ( i=0; i< n; i++)
printf("%d ", arr[i]);
printf("\n");
   for ( i = 1 ; i<= n - 1; i++)
   {
   temp = arr[i];
   for ( j = i - 1 ; j >= 0; j--)
   {
     if (arr[j] > temp)
     {
arr[j+1] = arr[j];
       flag = 1;
     }
     else
       break;
   }
   if (flag)
arr[j+1] = temp;}
printf("\n");
printf("Sorted Array using Insertion Sort is:-\n");printf("\n");
   for ( i=0; i< n; i++)
printf("%d ", arr[i]);
printf("\n");
}
```

5. Result /Output

The result of the above implementation is that we will be able to sort any array of integers in ascending order.



```
Enter number of elements in the array
3
Enter array elements
34
45
66
Original  Array is:-

34 45 66

Sorted Array using Insertion Sort is:-

34 45 66

Process returned 10 (0xA)   execution time : 18.714 s
Press any key to continue.
```

PRACTICAL 8

1. OBJECTIVE OF PRACTICAL

To sort elements using different sorting methods in C language

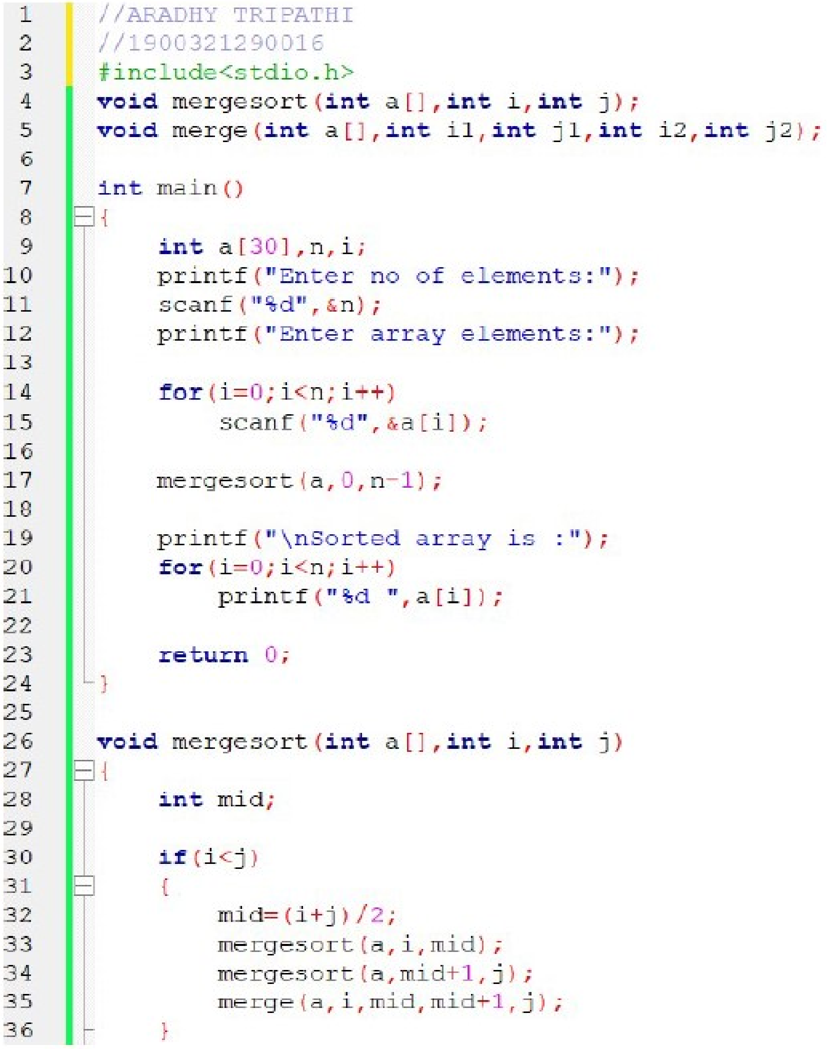
Quick Sort

2. ALGORITHM / FLOW CHART

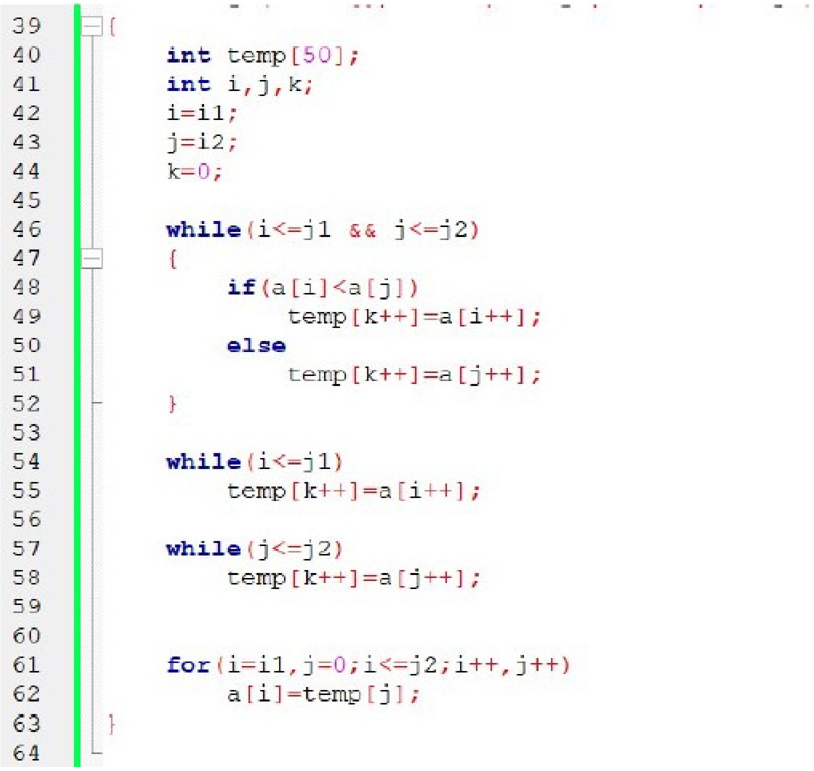
Quick Sorting

```
#include<stdio.h>
void quicksort(int number[25],int first,int last){
  int i, j, pivot, temp;

  if(first<last){
    pivot=first;
    i=first;
    j=last;

    while(i<j){
      while(number[i]<=number[pivot]&&i<last)
        i++;
      while(number[j]>number[pivot])
        j--;
      if(i<j){
        temp=number[i];
        number[i]=number[j];
        number[j]=temp;
      }
    }

    temp=number[pivot];
    number[pivot]=number[j];
    number[j]=temp;
    quicksort(number,first,j-1);
    quicksort(number,j+1,last);

  }
}

int main(){
  int i, count, number[25];

  printf("enter no. of element: ");
  scanf("%d",&count);

  printf("Enter %d elements: ", count);
  for(i=0;i<count;i++)
    scanf("%d",&number[i]);

  quicksort(number,0,count-1);

  printf("Order of Sorted elements: ");
  for(i=0;i<count;i++)
    printf(" %d",number[i]);

  return 0;
}
```

3. IMPLEMENTATION

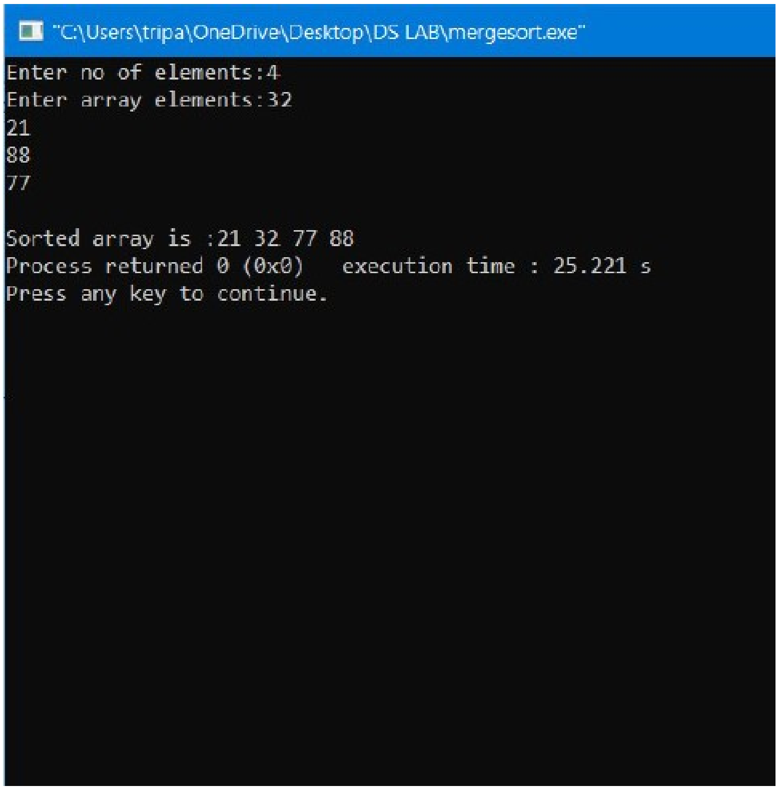


```c
//ARADHY TRIPATHI
//1900321290016
#include<stdio.h>
void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);

int main()
{
    int a[30],n,i;
    printf("Enter no of elements:");
    scanf("%d",&n);
    printf("Enter array elements:");

    for(i=0;i<n;i++)
        scanf("%d",&a[i]);

    mergesort(a,0,n-1);

    printf("\nSorted array is :");
    for(i=0;i<n;i++)
        printf("%d ",a[i]);

    return 0;
}

void mergesort(int a[],int i,int j)
{
    int mid;

    if(i<j)
    {
        mid=(i+j)/2;
        mergesort(a,i,mid);
        mergesort(a,mid+1,j);
        merge(a,i,mid,mid+1,j);
    }
```



```
{
    int temp[50];
    int i,j,k;
    i=i1;
    j=i2;
    k=0;

    while(i<=j1 && j<=j2)
    {
        if(a[i]<a[j])
            temp[k++]=a[i++];
        else
            temp[k++]=a[j++];
    }

    while(i<=j1)
        temp[k++]=a[i++];

    while(j<=j2)
        temp[k++]=a[j++];


    for(i=i1,j=0;i<=j2;i++,j++)
        a[i]=temp[j];
}
```

4. Results



```
Enter no of elements:4
Enter array elements:32
21
88
77

Sorted array is :21 32 77 88
Process returned 0 (0x0)   execution time : 25.221 s
Press any key to continue.
```

PRACTICAL 9

1. OBJECTIVE OF PRACTICAL

To sort elements using different sorting methods in C language

Merge Sort

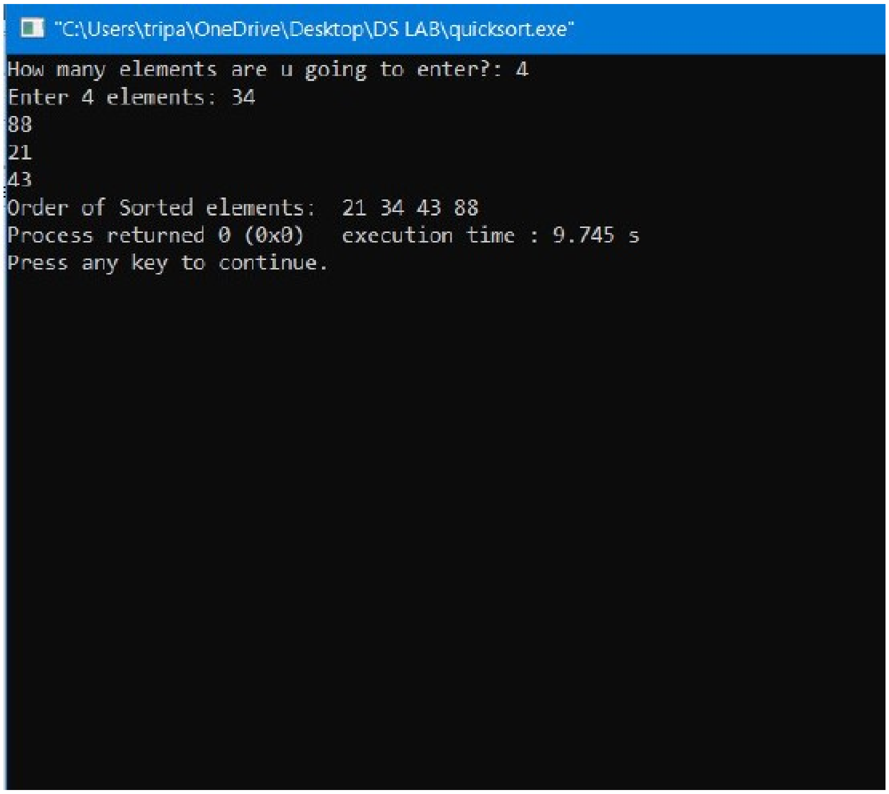
2. ALGORITHM / FLOW CHART

Merge Sorting

```
#include<stdio.h>

void main ()
{
   int i,n,a[20];
   printf("How many Elements you want to Enter: ");
   scanf("%d",&n);
   printf("Enter 5 Elements:\n");
   for(i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }

   mergeSort(a,0,n-1);
   printf("The Sorted Elements are: ");
   for(i=0;i<n;i++)
   {
      printf("%d ",a[i]);
   }

}
void mergeSort(int a[], int lb, int ub)
{
   int mid;
   if(lb<ub)
   {
      mid = (lb+ub)/2;
      mergeSort(a,lb,mid);
      mergeSort(a,mid+1,ub);
      merge(a,lb,mid,ub);
   }
}
void merge(int a[], int lb, int mid, int ub)
{
   int i=lb,j=mid+1,k,index = lb;
   int temp[10];
   while(i<=mid && j<=ub)
   {
      if(a[i]<a[j])
      {
         temp[index] = a[i];
         i++;
      }
      else
      {
         temp[index] = a[j];
         j++;
      }
      index++;
   }
   if(i>mid)
   {
      while(j<=ub)
      {
         temp[index] = a[j];
```

```
            index++;
            j++;
        }
    }
    else
    {
        while(i<=mid)
        {
            temp[index] = a[i];
            index++;
            i++;
        }
    }
    k = lb;
    while(k<index)
    {
        a[k]=temp[k];
        k++;
    }
}
```

3. Results



"C:\Users\tripa\OneDrive\Desktop\DS LAB\quicksort.exe"

```
How many elements are u going to enter?: 4
Enter 4 elements: 34
88
21
43
Order of Sorted elements:  21 34 43 88
Process returned 0 (0x0)   execution time : 9.745 s
Press any key to continue.
```

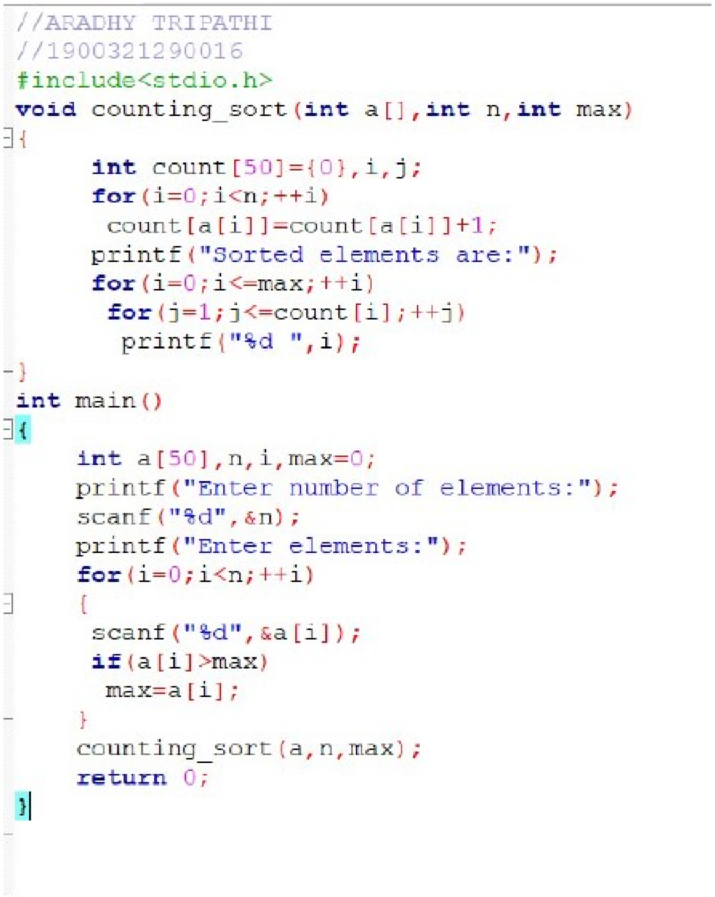
PRACTICAL 10

4. OBJECTIVE OF PRACTICAL

To sort elements using different sorting methods in C language

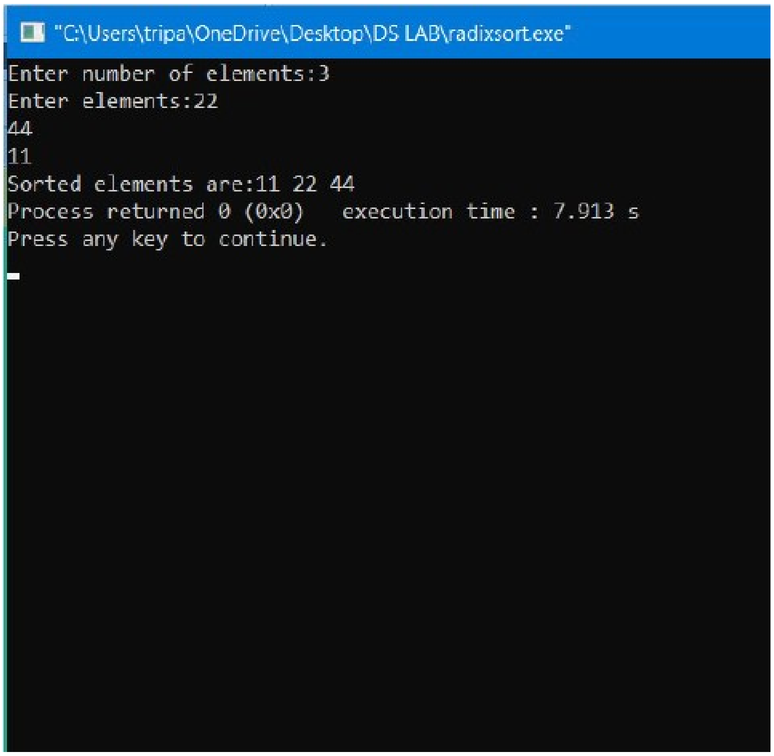
RADIX SORT

IMPLEMENTATION



```
//ARADHY TRIPATHI
//1900321290016
#include<stdio.h>
void counting_sort(int a[],int n,int max)
{
     int count[50]={0},i,j;
     for(i=0;i<n;++i)
      count[a[i]]=count[a[i]]+1;
     printf("Sorted elements are:");
     for(i=0;i<=max;++i)
      for(j=1;j<=count[i];++j)
       printf("%d ",i);
}
int main()
{
    int a[50],n,i,max=0;
    printf("Enter number of elements:");
    scanf("%d",&n);
    printf("Enter elements:");
    for(i=0;i<n;++i)
    {
     scanf("%d",&a[i]);
     if(a[i]>max)
      max=a[i];
    }
    counting_sort(a,n,max);
    return 0;
}
```

RESULT -



```
Enter number of elements:3
Enter elements:22
44
11
Sorted elements are:11 22 44
Process returned 0 (0x0)   execution time : 7.913 s
Press any key to continue.
```

PRACTICAL 11

PRACTICAL STATEMENT OF PRACTICAL:

::::::::::Implementation of Tree Structures, Binary Tree, Tree Traversal::::::::::::

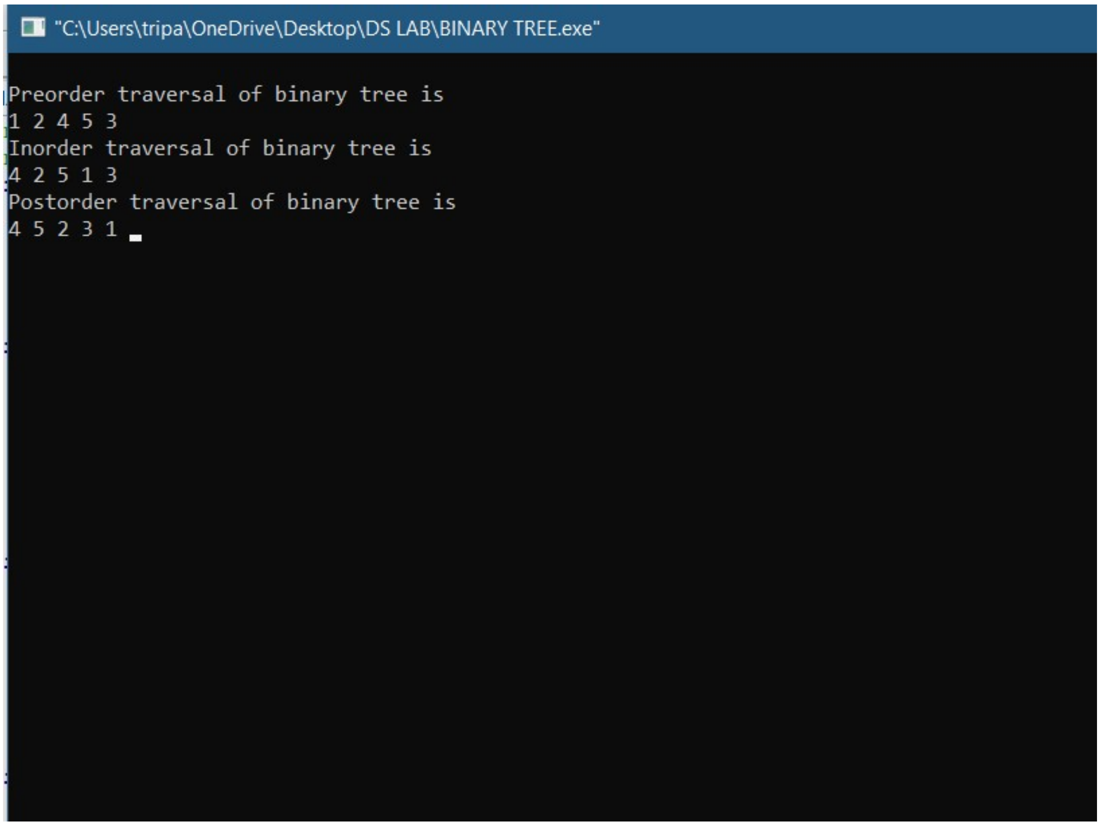
OBJECTIVE OF PRACTICAL :

::::::::::Implementation of Tree Structures, Binary Tree, Tree Traversal::::::::::::

PROGRAM:-

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
        int data;
        struct node* left;
        struct node* right;
};
struct node* newNode(int data)
{
        struct node* node = (struct node*)malloc(sizeof(struct node));
        node->data = data;
        node->left = NULL;
        node->right = NULL;
        return(node);
}
void printPostorder(struct node* node)
{
        if (node == NULL)
                return;
        printPostorder(node->left);
        printPostorder(node->right);
        printf("%d ", node->data);
}
void printInorder(struct node* node)
{
        if (node == NULL)
                return;
        printInorder(node->left);
        printf("%d ", node->data);
        printInorder(node->right);
}
void printPreorder(struct node* node)
{
        if (node == NULL)
                return;
        printf("%d ", node->data);
        printPreorder(node->left);
        printPreorder(node->right);
}
int main()
{
        struct node *root = newNode(1);
        root->left        = newNode(2);
        root->right       = newNode(3);
        root->left->left = newNode(4);
        root->left->right = newNode(5);
        printf("\nPreorder traversal of binary tree is \n");
        printPreorder(root);
        printf("\nInorder traversal of binary tree is \n");
        printInorder(root);
        printf("\nPostorder traversal of binary tree is \n");
        printPostorder(root);
        getchar();
        return 0;
}
```

4. OUTPUT SCREEN



```
Preorder traversal of binary tree is
1 2 4 5 3
Inorder traversal of binary tree is
4 2 5 1 3
Postorder traversal of binary tree is
4 5 2 3 1
```