**ABES Engineering College, Ghaziabad**

# **Affiliated to Dr. APJ AKTU Lucknow**

**Department of Computer science and engineering**



# **Lab Manual**

**Session 2020-21 (Odd Semester)**

**Name :** HARSH MOHAN

**Roll No. :** 1900320100065

**Subject Name :** Data Structure Using C Lab

**Subject Code :** KCS351

**Semester**   **:** III semester

**LIST OF EXPERIMENT**

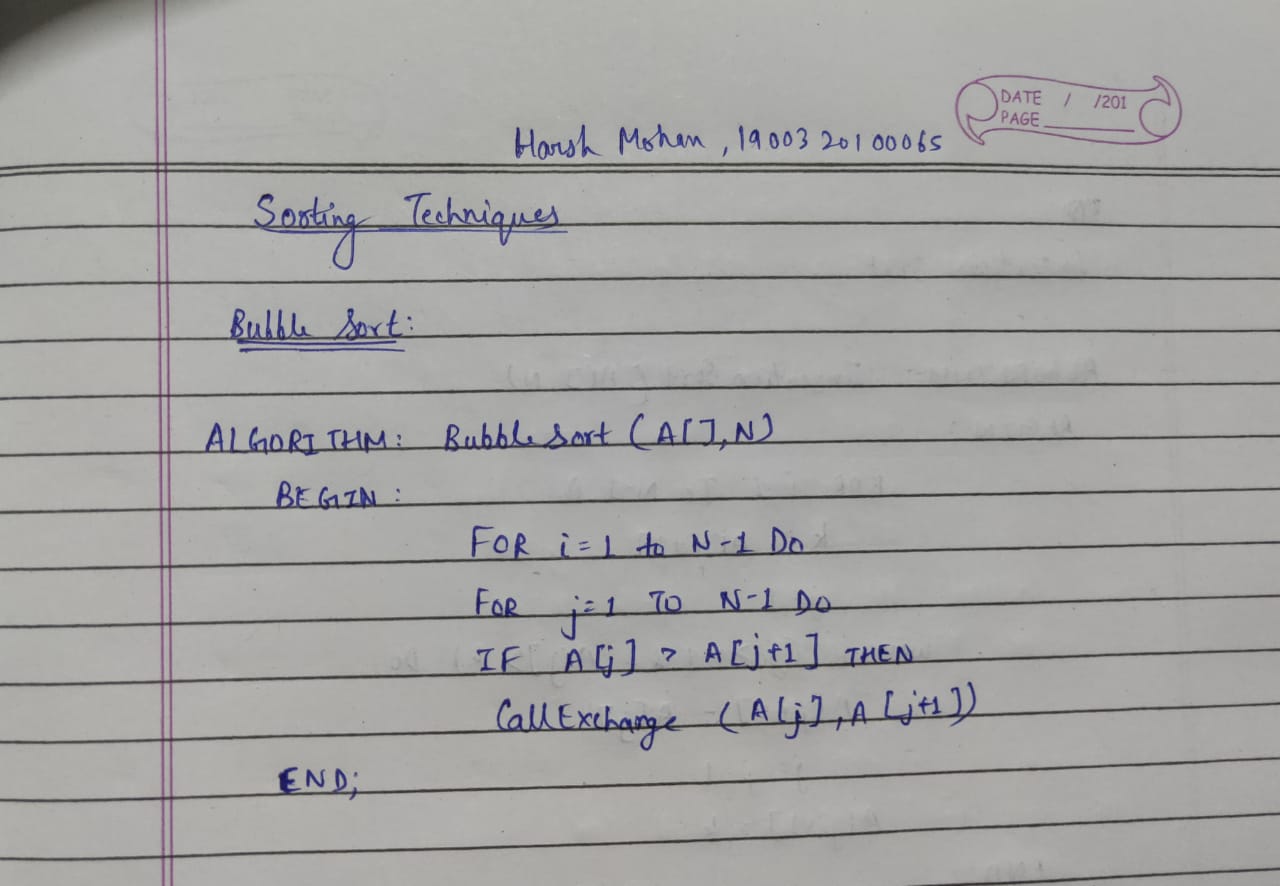
Write C programs to illustrate the concept of the following:

|  |  |
| --- | --- |
| **S. No.** | **LIST OF EXPERIMENTS** |
| 1. | Sorting Algorithms – Non-Recursive. |
| 2. | Sorting Algorithms – Recursive. |
| 3. | Searching Algorithm. |
| 4. | Implementation of Stack using Array. |
| 5. | Implementation of Queue using Array. |
| 6. | Implementation of Circular Queue using Array. |
| 7. | Implementation of Stack using Linked List. |
| 8. | Implementation of Queue using Linked List. |
| 9. | Implementation of Circular Queue using Linked List |
| 10. | Implementation of Tree Structures, Binary Tree, Tree Traversal, Binary Search Tree, Insertion and Deletion in BST. |
| 11. | Graph Implementation, BFS, DFS, Minimum cost spanning tree, shortest path algorithm. |

**EXPERIMENT NUMBER 1**

**Sorting Algorithms – Non-Recursive**

**TITLE OF EXPERIMENT: Bubble, Selection and Insertion Sort**

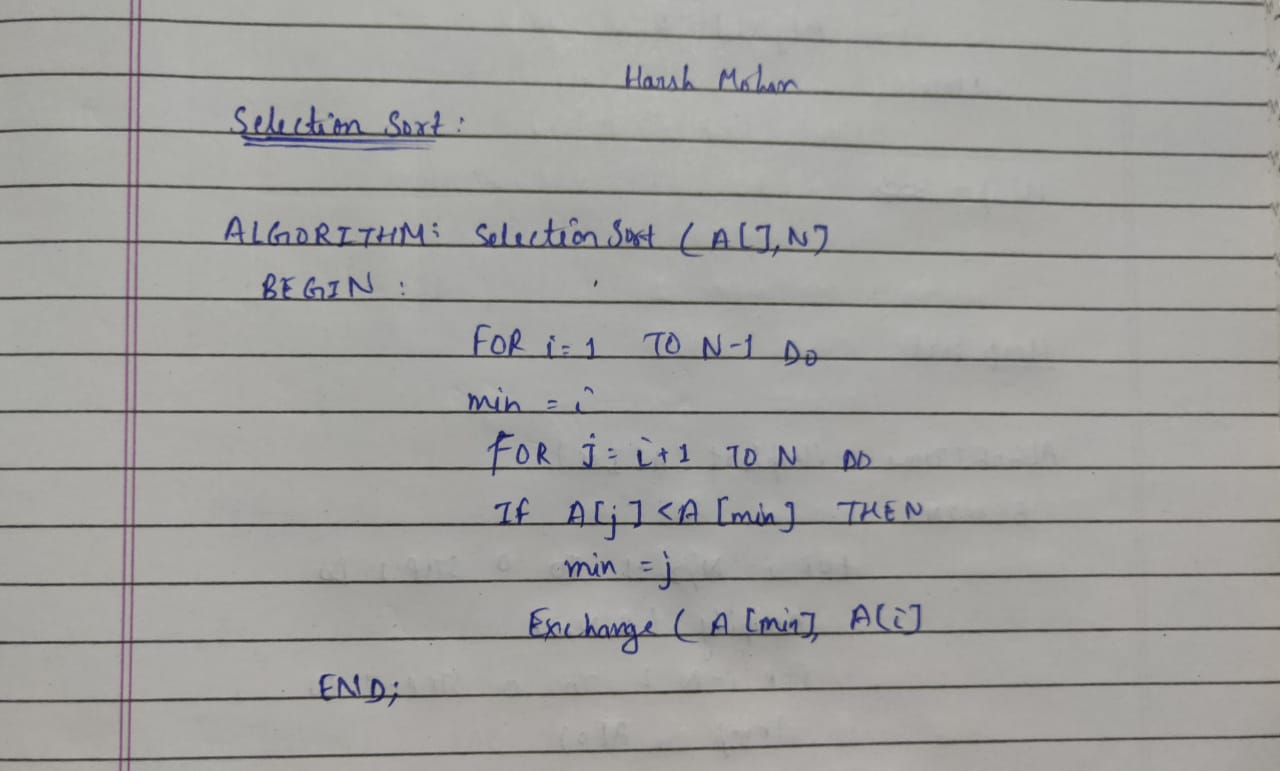
**ALGORITHM:**

**Time Complexity:** Omega(N) (Best Case)

O (N2) (Worst Case)

**Space Complexity:** Theta(1) (Best Case)

Theta(1) (Worst Case)

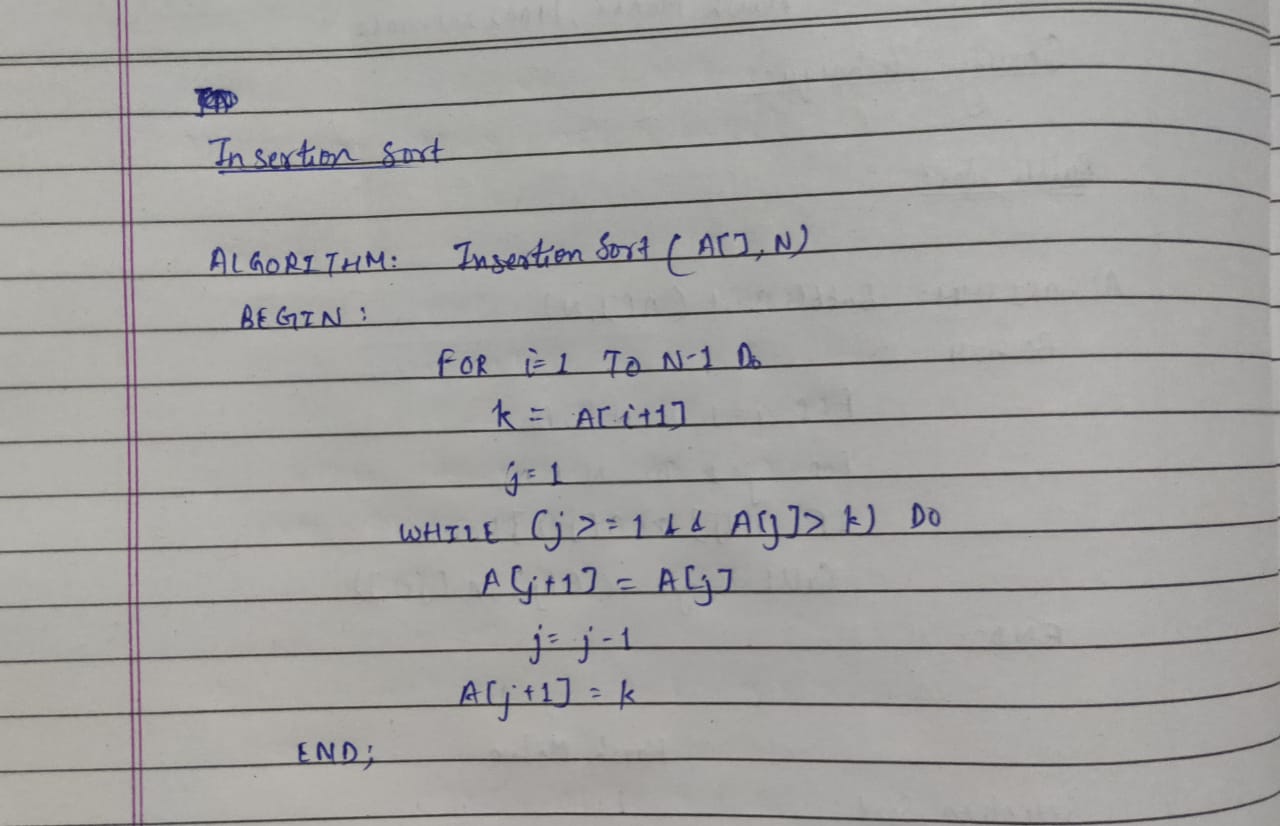


**Time Complexity:** Theta(N2) (Best Case)

Theta(N2) (Worst Case)

**Space Complexity:** Theta(1) (Best Case)

Theta(1) (Worst Case)



**Time Complexity:** Omega(N) (Best Case)

O (N2) (Worst Case)

**Space Complexity:** Theta(1) (Best Case)

Theta(1) (Worst Case)

**C CODE:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PROGRAM FOR BUBBLE, SELECTION AND INSERTION SORT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*AUTHOR: HARSH MOHAN, ADMISSION NO.:2019B101166\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

void display(int a[],int n);

void bubble\_sort(int a[],int n);

void selection\_sort(int a[],int n);

void insertion\_sort(int a[],int n);

int main()

{

int n;

printf("Enter the number of elements in array");

scanf("%d",&n);

int choice,i;

int arr[100];

printf("Enter elements of an array");

for(i=0;i<n;i++)

{

scanf("%d",&arr[i]);

}

printf("Given array is :\n");

for(i=0; i<n; i++)

{

printf(" %d",arr[i]);

}

printf("\nPlease select any option Given Below for Sorting : \n");

while(1)

{

printf("\n1. Bubble Sort\n2. Selection Sort\n3. Insertion Sort\n4. Exit the Program.\n");

printf("\nEnter your Choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

bubble\_sort(arr,n);

break;

case 2:

selection\_sort(arr,n);

break;

case 3:

insertion\_sort(arr,n);

break;

case 4:

return 0;

default:

printf("\nPlease Select only 1-4 option ----\n");

}

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void display(int arr[],int n)

{

for(int i=0;i<n;i++)

{

printf(" %d ",arr[i]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void bubble\_sort(int arr[],int n)

{

int i,j,temp;

for(i=0;i<n;i++)

{

for(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("After Bubble sort Elements are : ");

display(arr,n);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void selection\_sort(int arr[],int n)

{

int i,j,temp;

for(i=0;i<n-1;i++)

{

for(j=i+1;j<n;j++)

{

if(arr[i]>arr[j])

{

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

}

printf("After Selection sort Elements are : ");

display(arr,n);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void insertion\_sort(int arr[],int n)

{

int i,j,min;

for(i=1;i<n;i++)

{

min=arr[i];

j=i-1;

while(min<arr[j] && j>=0)

{

arr[j+1]=arr[j];

j=j-1;

}

arr[j+1]=min;

}

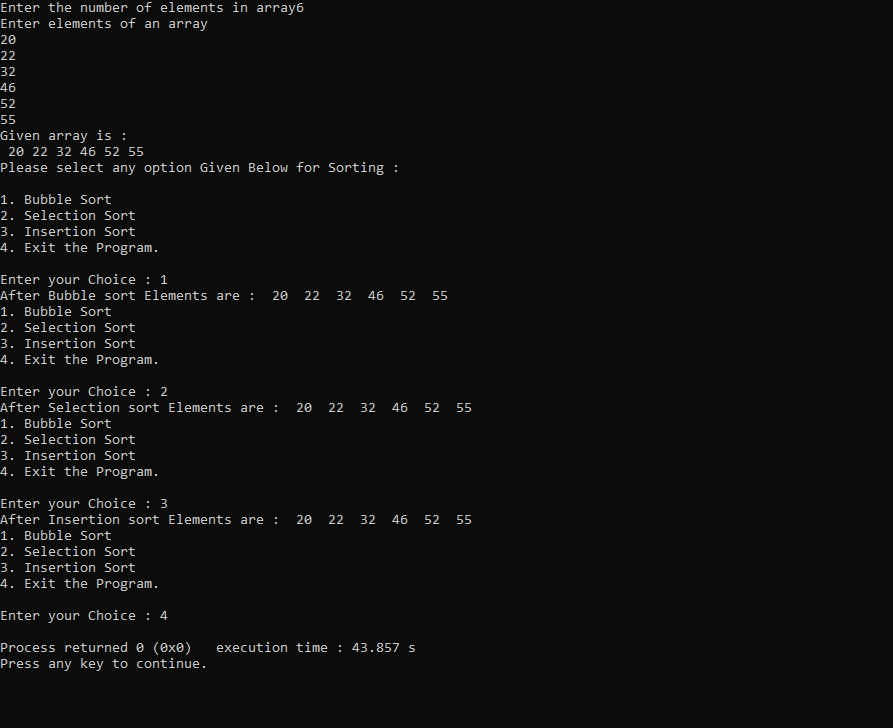
printf("After Insertion sort Elements are : ");

display(arr,n);

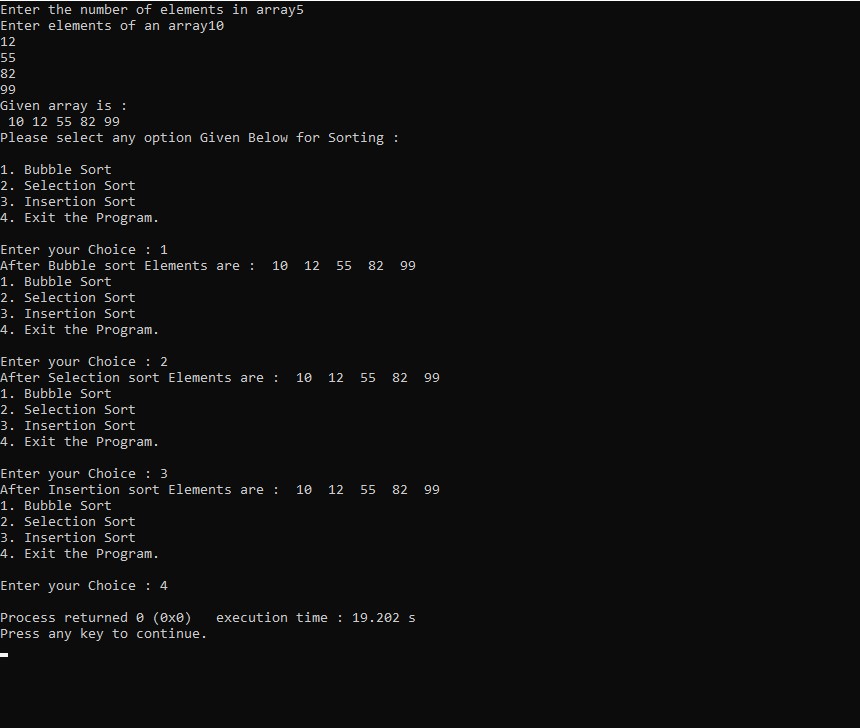
}

**OUTPUT: (3 Sample Input-Outputs)**

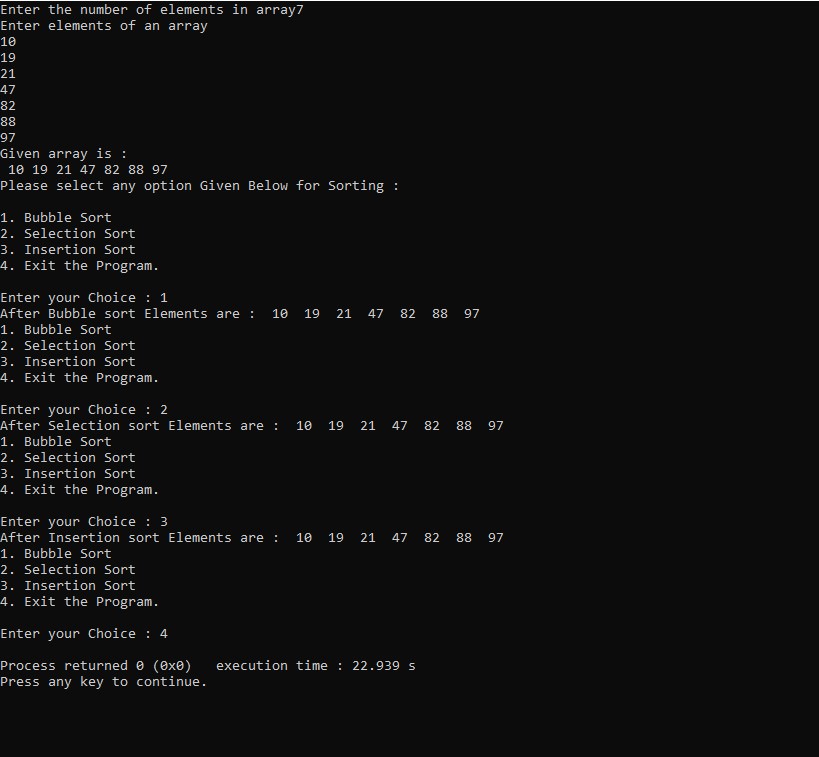
**First Sample Input-Output**



**Second Sample Input-Output**

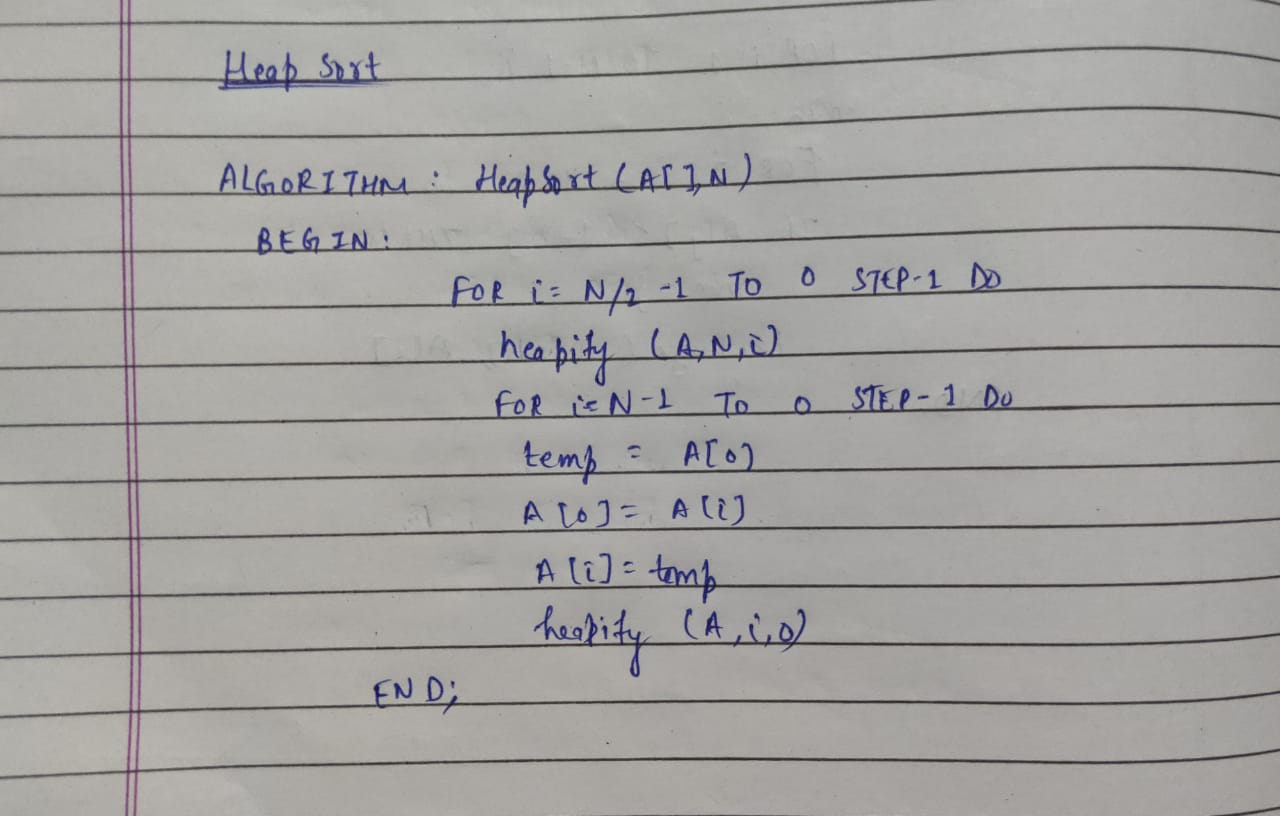


**Third Sample Input-Output**



**TITLE OF EXPERIMENT: Heap Sort**

**ALGORITHM:**



**Time Complexity:** Omega(N log (N)) (Best Case)

O(Nlog (N)) (Worst Case)

**Space Complexity:** O(1) (Best Case)

O(1) (Worst Case)

**C CODE:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PROGRAM FOR HEAP SORT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*AUTHOR- HARSH MOHAN, ADMISSION NO.2019B101166\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void heapify(int A[],int N,int i)

{

int t,l=i;

int j=2\*i+1;

int k=2\*i+2;

if (j<N && A[j]>A[l])

l=j;

if(k<N && A[k]>A[l])

l=k;

if(l!=i)

{

t=A[i];

A[i]=A[l];

A[l]=t;

heapify(A,N,l);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void heapsort(int A[],int N)

{

int i,t;

for(i=N/2-1;i>=0;i--)

heapify(A,N,i);

for(i=N-1;i>=0;i--)

{

t=A[0];

A[0]=A[i];

A[i]=t;

heapify(A,i,0);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{

int i,A[100];

int N;

printf("Enter the number of elements in an array : ");

scanf("%d",&N);

printf("Enter elements of an array : ");

for (i=0;i<N;i++)

{

scanf("%d",&A[i]);

}

printf("Given Array is :");

for(i=0;i<N;i++)

printf("%d ",A[i]);

heapsort(A,N);

printf("\nSorted Array is :");

for(i=0;i<N;i++)

{

printf("%d ",A[i]);

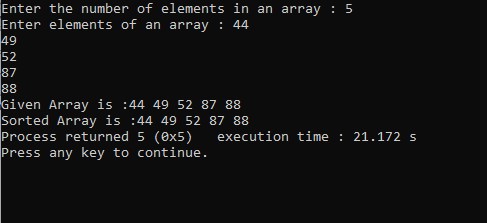
}

}

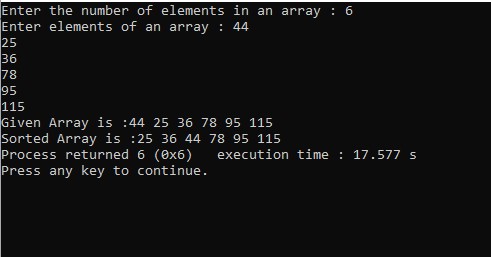
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**OUTPUT: (3 Sample Input-Outputs)**

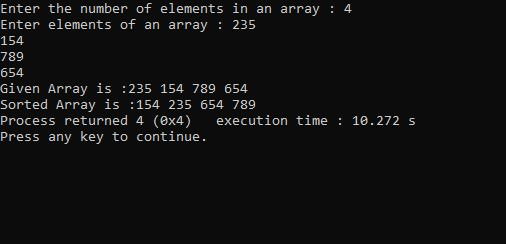
**First Sample Input-Output**



**Second Sample Input-Output**

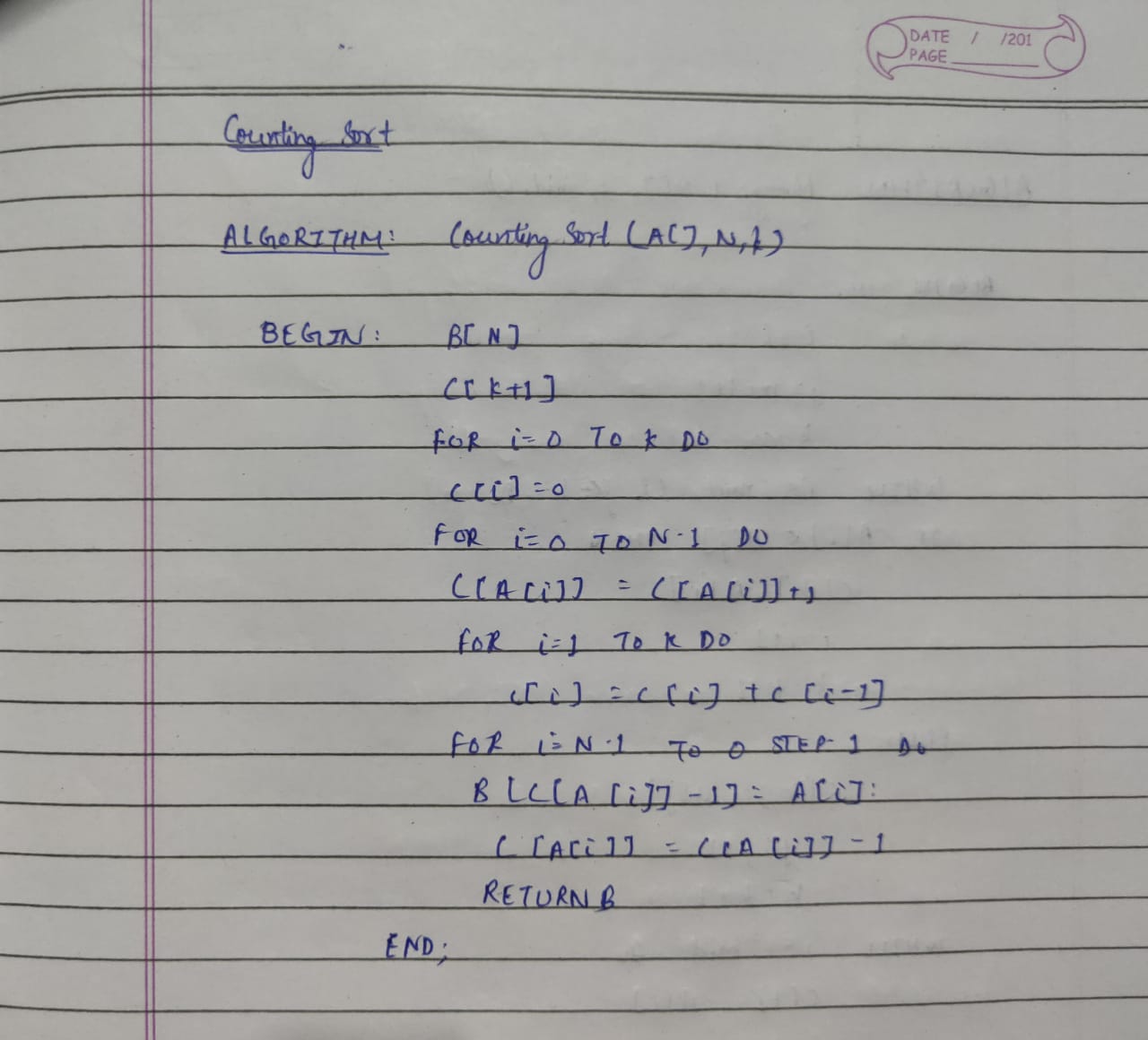


**Third Sample Input-Output**



**TITLE OF EXPERIMENT: Counting Sort**

**ALGORITHM:**



**Time Complexity:** Theta(N) (Best Case)

Theta(N) (Worst Case)

**Space Complexity:** Theta(N) (Best Case)

Theta(N) (Worst Case)

**C CODE:**

/\*\*\*\*\*\*\*\*\*\*\*PROGRAM FOR COUNTING SORT\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*AUTHOR : HARSH MOHAN , AD NO. :2019B101166\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void countingsort(int A[], int N, int k)

{

int i;

int B[N],C[k+1];

for (i=0;i<=k;i++)

{

C[i]=0;

}

for (i=0;i<=N-1;i++)

{

C[A[i]]=C[A[i]]+1;

}

for (i=1;i<=k;i++)

{

C[i]=C[i]+C[i-1];

}

for (i=N-1;i>=0;i--)

{

B[C[A[i]]-1]=A[i];

C[A[i]]=C[A[i]]-1;

}

printf("\nThe Sorted array is : ");

for (i=0;i<N;i++)

printf("%d ",B[i]);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

int A[100];

int N,k,i;

printf("Enter the number of elements in an array : ");

scanf("%d",&N);

printf("Enter elements of an array : ");

for(i=0;i<N;i++)

scanf("%d",&A[i]);

k=A[0];

for(i=1;i<=N;i++)

{

if(A[i]>k)

k=A[i];

}

printf("Given array is : ");

for(i=0;i<N;i++)

{

printf("%d ",A[i]);

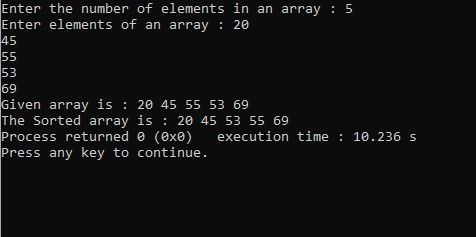
}

countingsort(A,N,k);

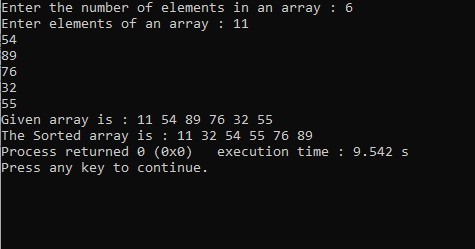
}

**OUTPUT: (3 Sample Input-Outputs)**

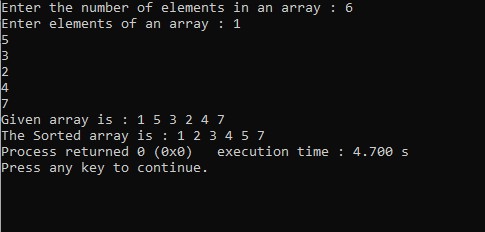
**First Sample Input-Output**



**Second Sample Input-Output**

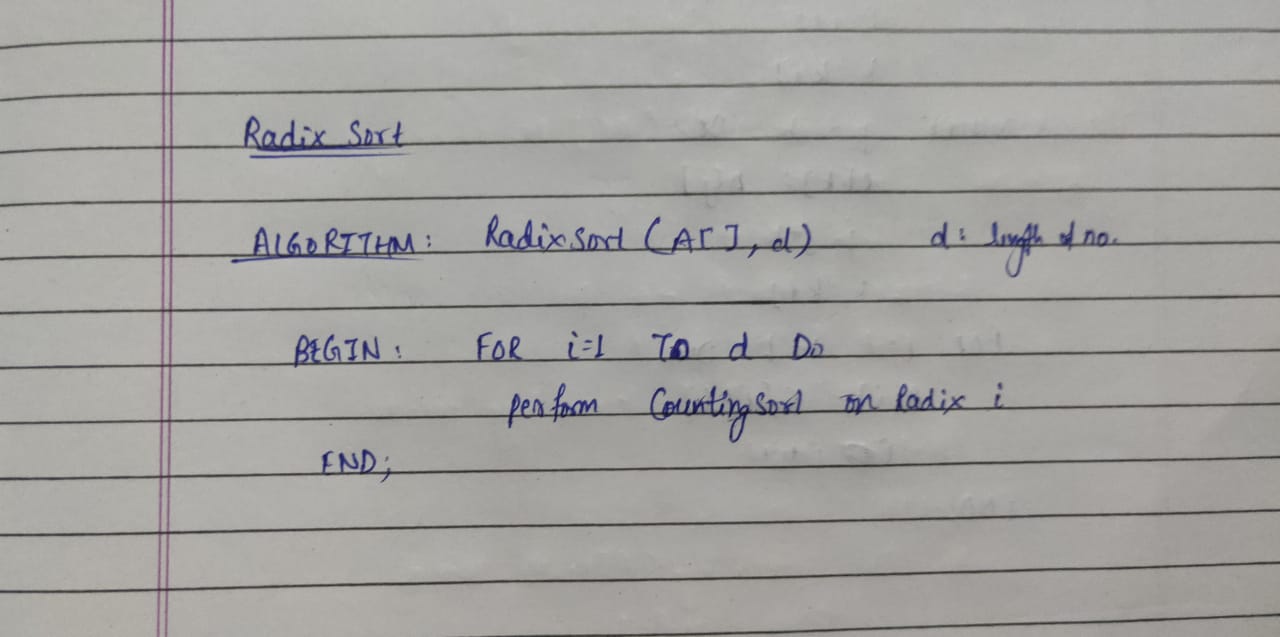


**Third Sample Input-Output**



**TITLE OF EXPERIMENT: Radix Sort**

**ALGORITHM:**



**Time Complexity:** Theta(dN) (Best Case)

Theta(dN) (Worst Case)

**Space Complexity:** Theta(N) (Best Case)

Theta(N) (Worst Case)

**C CODE:**

/\*\*\*\*\*\*\*\*\*\*\*PROGRAM FOR RADIX SORT\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*AUTHOR: HARSH MOHAN, AD NO. :2019B101166\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

void countingsort(int A[], int N, int k)

{

int i;

int B[N],C[k+1];

for (i=0;i<=k;i++)

{

C[i]=0;

}

for (i=0;i<=N-1;i++)

{

C[A[i]]=C[A[i]]+1;

}

for (i=1;i<=k;i++)

{

C[i]=C[i]+C[i-1];

}

for (i=N-1;i>=0;i--)

{

B[C[A[i]]-1]=A[i];

C[A[i]]=C[A[i]]-1;

}

for (i=0;i<N;i++)

A[i]=B[i];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void radixsort(int A[],int N,int k)

{

int i;

for(i=1; k/i>0; i\*=10)

{

countingsort(A,N,k);

}

printf("\nSorted Array is :");

for(i=0;i<N;i++)

{

printf("%d ",A[i]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

int A[100];

int N,k,i;

printf("Enter the number of elements in an array : ");

scanf("%d",&N);

printf("Enter elements of an array : ");

for(i=0;i<N;i++)

scanf("%d",&A[i]);

k=A[0];

for(i=1;i<=N;i++)

{

if(A[i]>k)

k=A[i];

}

printf("Given array is : ");

for(i=0;i<N;i++)

{

printf("%d ",A[i]);

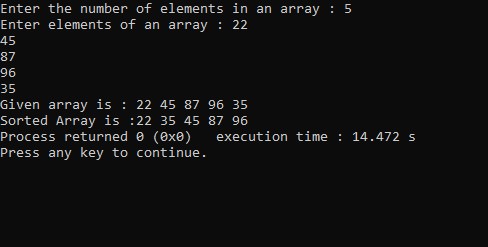
}

radixsort(A,N,k);

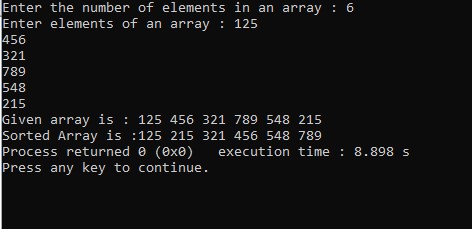
}

**OUTPUT: (3 Sample Input-Outputs)**

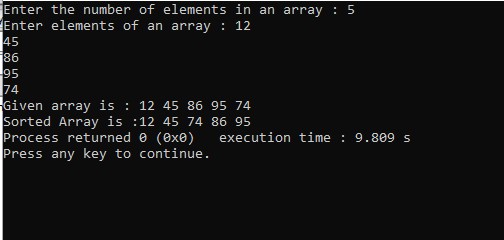
**First Sample Input-Output**



**Second Sample Input-Output**



**Third Sample Input-Output**

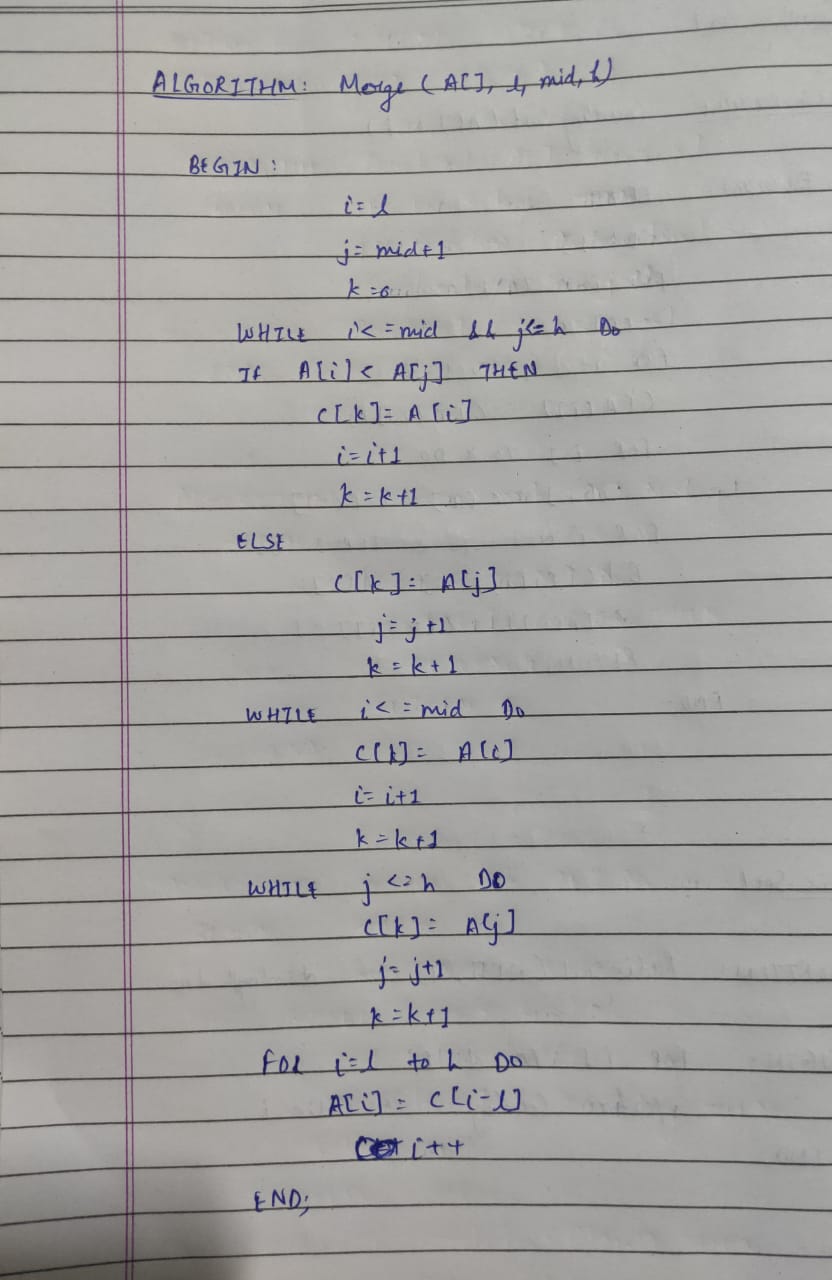


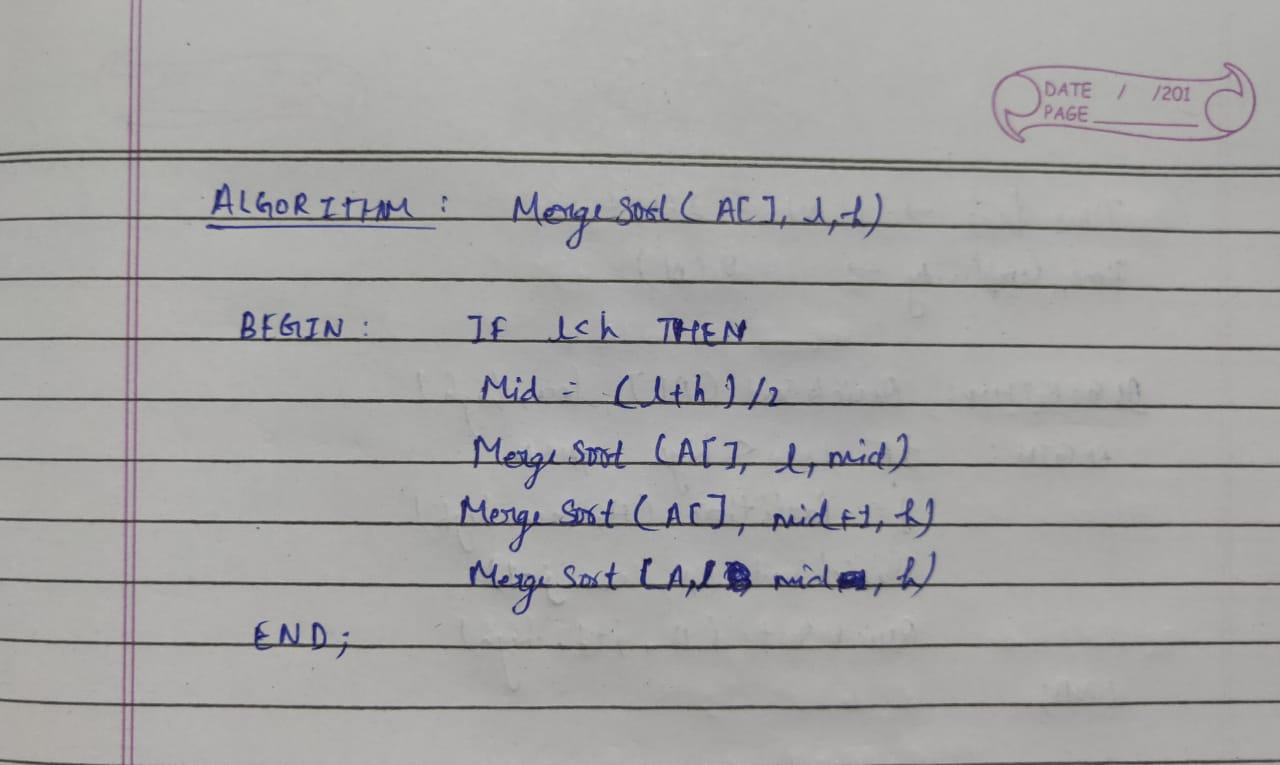
**EXPERIMENT NUMBER 2**

**Sorting Algorithms – Recursive**

**TITLE OF EXPERIMENT: Merge Sort**

**ALGORITHM:**





**Time Complexity:** Theta(N log(N)) (Best Case)

Theta(N log(N)) (Worst Case)

**Space Complexity:** Theta(N) (Best Case)

Theta(N) (Worst Case)

**C CODE:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PROGRAM FOR MERGE SORT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*AUTHOR: HARSH MOHAN, ADMISSION NO. : 2019B101166\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

int C[100];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int mergearray(int A[],int low,int mid,int high)

{

int i=low,j=mid+1,k=low;

while( i<=mid && j<=high)

{

if(A[i]<A[j])

{

C[k]=A[i];

i++;

k++;

}

else

{

C[k]=A[j];

j++;

k++;

}

}

while(i<=mid)

{

C[k]=A[i];

i++;

k++;

}

while(j<=high)

{

C[k]=A[j];

j++;

k++;

}

for(i=low;i<=high;i++)

{

A[i]=C[i];

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int mergesort(int A[],int low,int high)

{

if(low<high)

{

int i;

int mid =(low + high)/2;

mergesort(A,low,mid);

mergesort(A,mid+1,high);

mergearray(A,low,mid,high);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{

int A[]={45,12,36,4,50,15,29,90,60};

int i,low=0,high=sizeof(A)/sizeof(A[0])-1;

printf("Given Array is : ");

for(i=0;i<=high;i++)

{

printf("%d ",A[i]);

}

mergesort(A,low,high);

printf("\nSorted Array is : ");

for(i=low;i<=high;i++)

{

printf(" %d",A[i]);

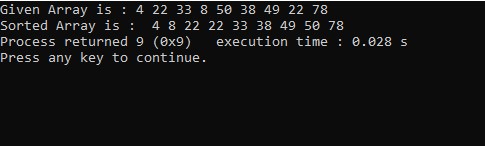
}

}

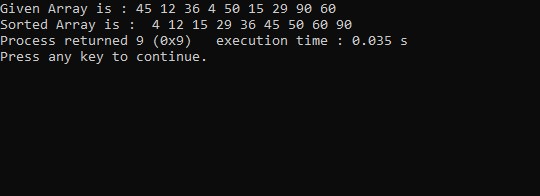
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**OUTPUT: (3 Sample Input-Outputs)**

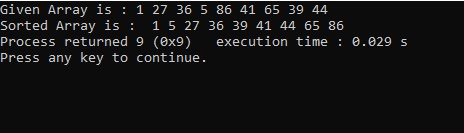
**First Sample Input-Output**



**Second Sample Input-Output**

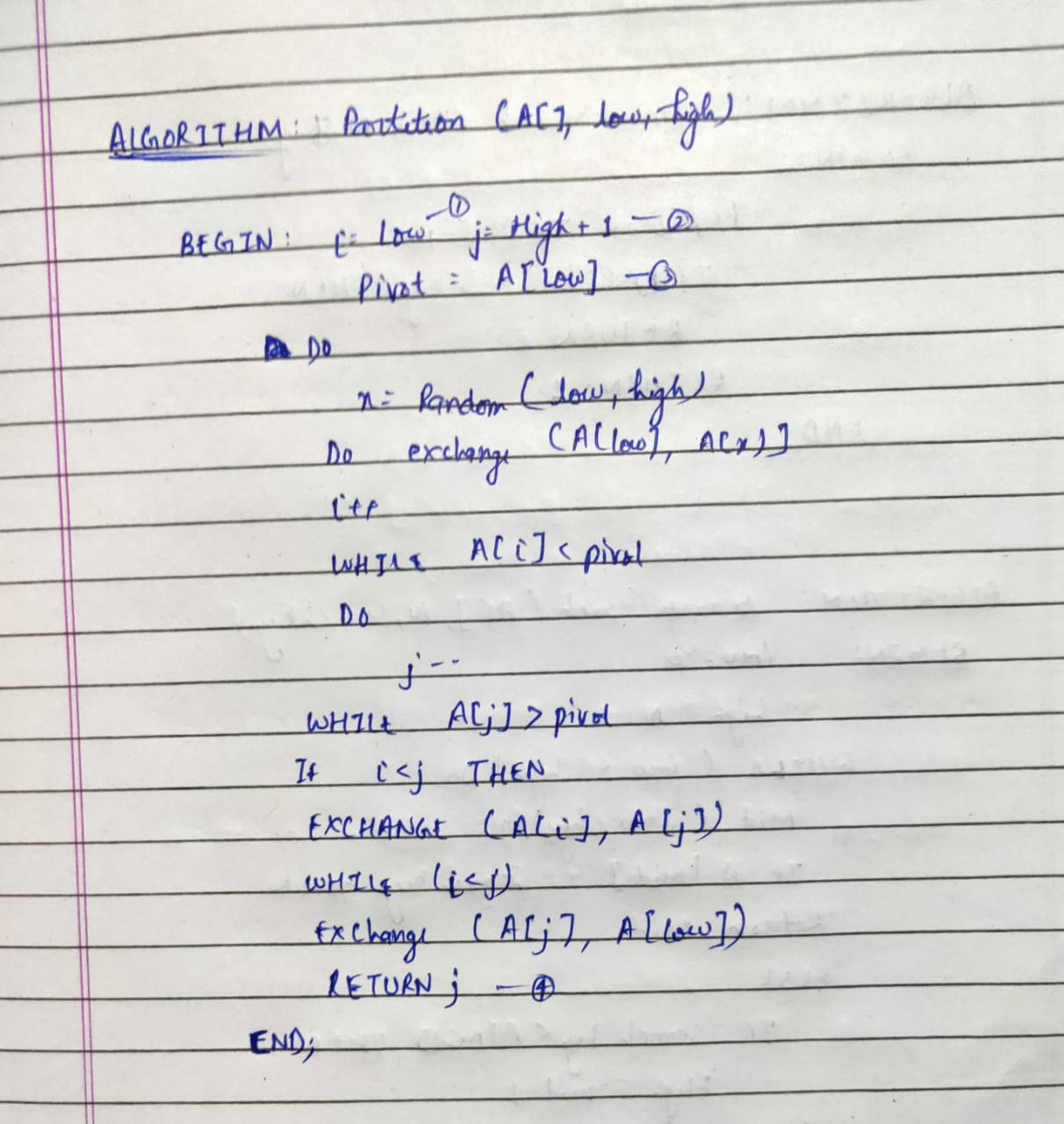


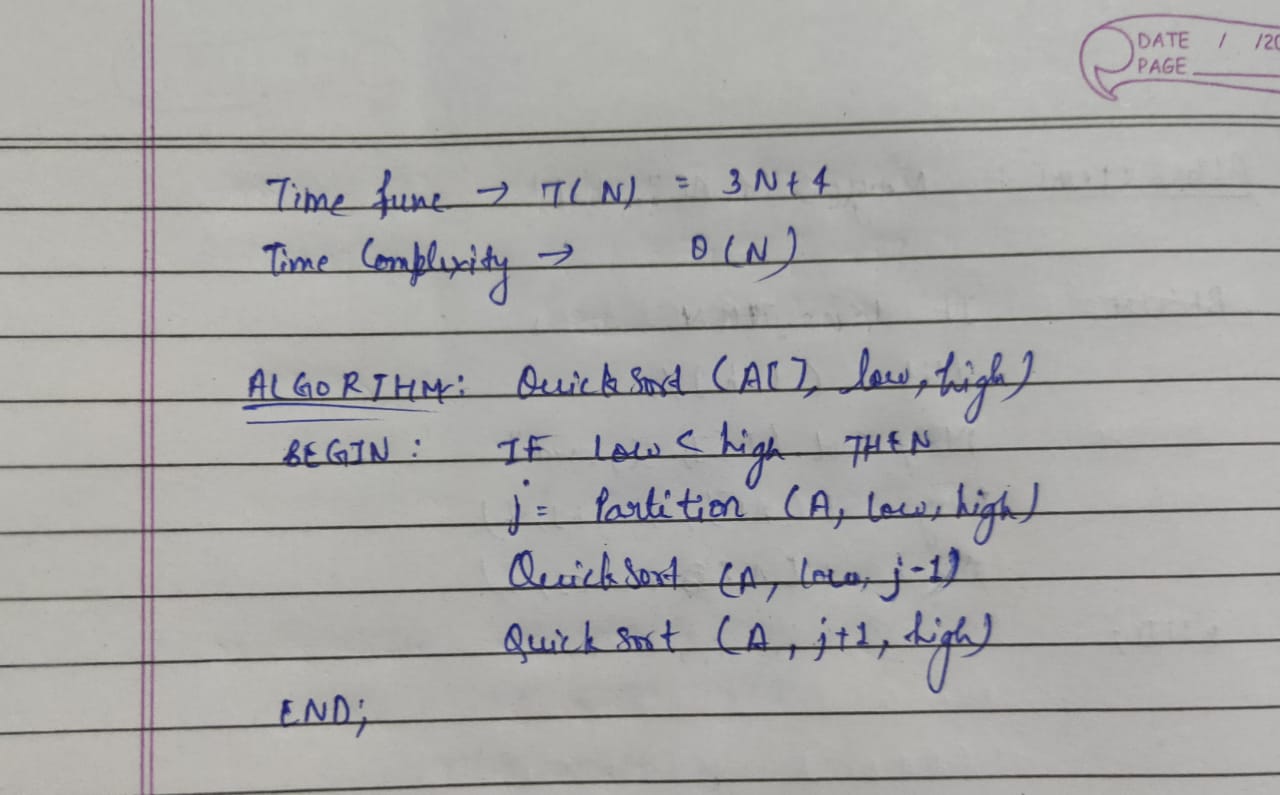
**Third Sample Input-Output**



**TITLE OF EXPERIMENT: Quick Sort**

**ALGORITHM:**





**Time Complexity:** Omega(N log(N)) (Best Case)

O(N2) (Worst Case)

**Space Complexity:** Omega(N log(N)) (Best Case)

O(N) (Worst Case)

**C CODE:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*PROGRAM FOR QUICK SORT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*AUTHOR : HARSH MOHAN, AD NO. : 2019B101166\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void swap (int\* a,int\* b)

{

int t= \*a;

\*a = \*b;

\*b = t;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int partition(int a[], int low, int high)

{

int i=low;

int j=high+1;

int pivot = a[low];

do

{

do

i++;

while (a[i] < pivot);

do

j--;

while (a[j] > pivot);

if (i<j)

{

swap(&a[i],&a[j]);

}

}

while(i<j);

{

swap(&a[j],&a[low]);

}

return j;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void quicksort(int a[],int low,int high)

{

if (low<high)

{

int j=partition(a,low,high);

quicksort(a,low,j-1);

quicksort(a,j+1,high);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void printarray(int a[],int high)

{

int i;

for(i=0;i<high;i++)

{

printf("%d ",a[i]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

int a[6]={100,90,30,40,10};

a[6]=32767;

int i,low=0;

int high=4;

printf("Given Array is: ");

for (i=0;i<5;i++)

printf("%d ",a[i]);

quicksort(a,low,high);

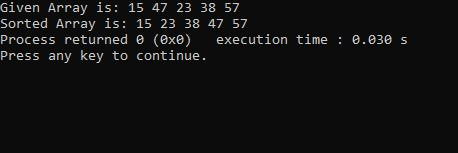
printf("\nSorted Array is: ");

printarray(a,high+1);

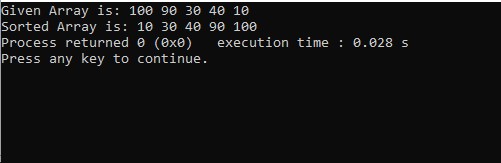
}

**OUTPUT: (3 Sample Input-Outputs)**

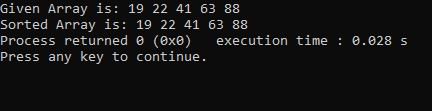
**First Sample Input-Output**



**Second Sample Input-Output**



**Third Sample Input-Output**

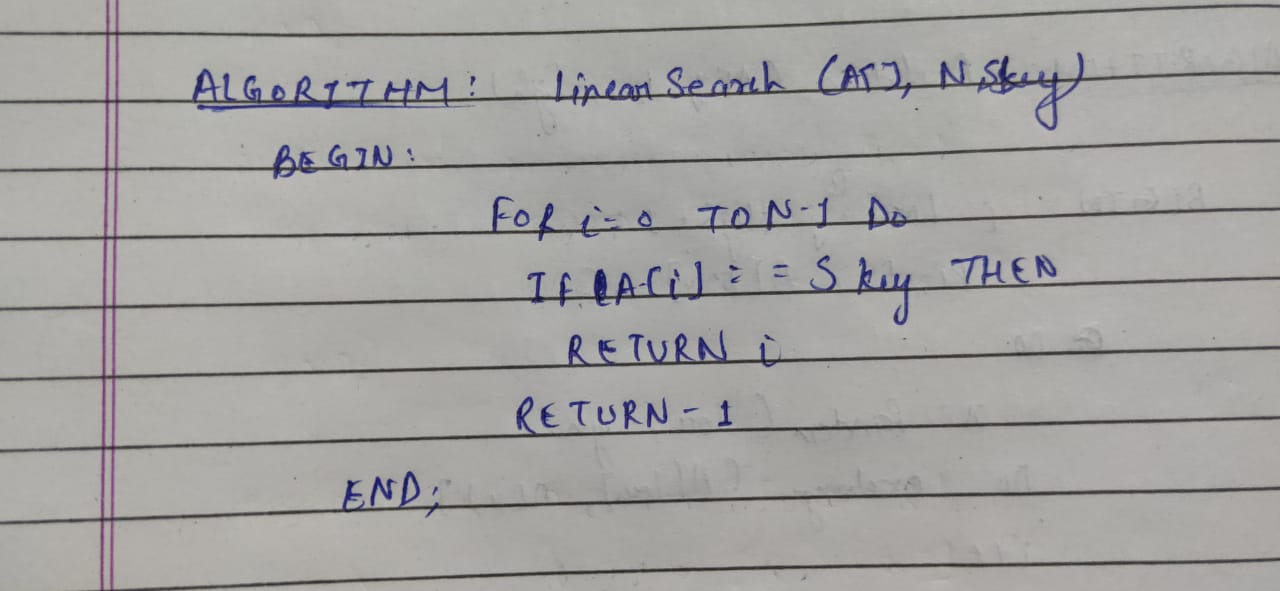


**EXPERIMENT NUMBER 3**

**Searching Algorithms**

**TITLE OF EXPERIMENT: Linear Search or Sequential Search**

**ALGORITHM:**



**Time Complexity:** Omega(1) (Best Case)

O(N) (Worst Case)

**Space Complexity:** Theta(1) (Best Case)

Theta(1) (Worst Case)

**C CODE:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PROGRAM FOR LINEAR SEARCH\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*AUTHOR: HARSH MOHAN, ADMISSION NO.2019B101166\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

int linearsearch(int a[],int n,int x)

{

int i;

for(i=0;i<n;i++)

if(a[i]==x)

return i;

return -1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

int i,n,x,a[100];

printf("Enter the size of an array : ");

scanf("%d",&n);

printf("Enter Elements of an array : ");

for(i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

for(i=0;i<n;i++)

printf("%d ",a[i]);

printf("\nEnter the element which you want to search : ");

scanf("%d",&x);

int result=linearsearch(a,n,x);

if(result==-1)

printf("Element not exist in array");

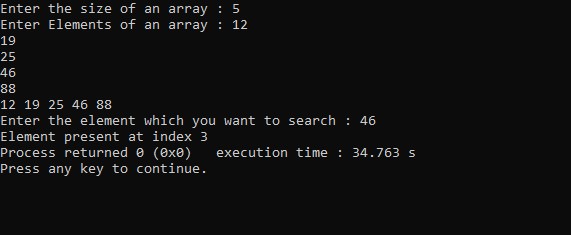
else

printf("Element present at index %d",result);

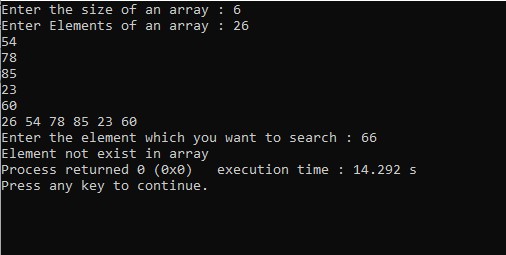
}

**OUTPUT: (3 Sample Input-Outputs)**

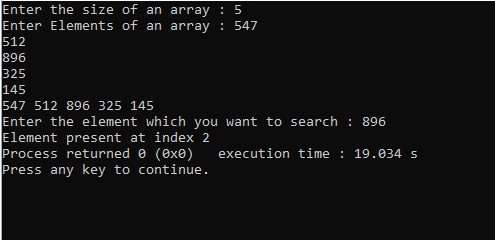
**First Sample Input-Output**



**Second Sample Input-Output**

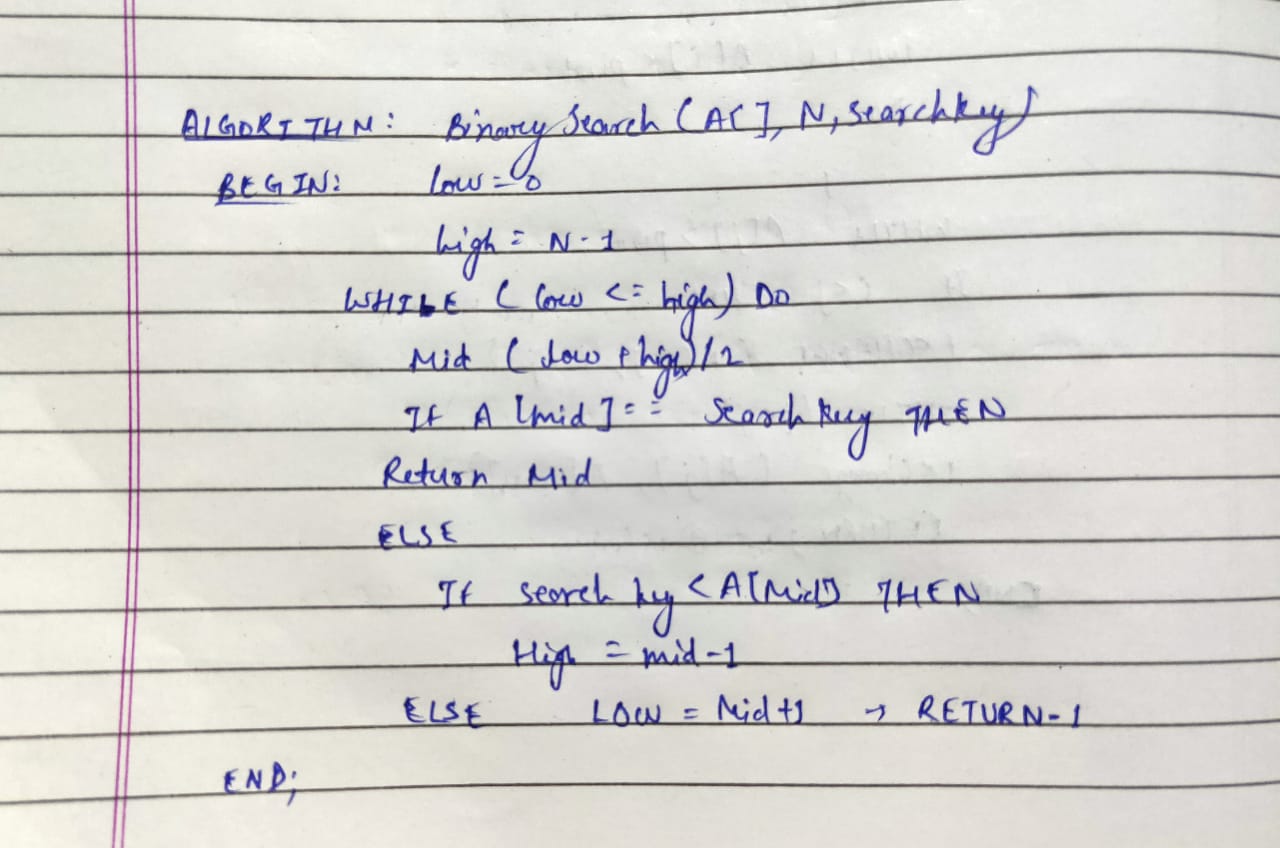


**Third Sample Input-Output**



**TITLE OF EXPERIMENT: Binary Search**

**ALGORITHM:**



**Time Complexity:** Omega(1) (Best Case)

O(N log (N)) (Worst Case)

**Space Complexity:** Theta(1) (Best Case)

Theta(1) (Worst Case)

**C CODE:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PROGRAM FOR BINARY SEARCH\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*AUTHOR: HARSH MOHAN, ADMISSION NO. :2019B101166\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

int Bsearch(int A[], int n, int x)

{

int l = 0;

int h = n-1;

while(l<=h)

{

int mid = (l+h)/2;

if( A[mid] == x)

{

return mid;

}

else if( x < A[mid] )

{

h = mid-1;

}

else

{

l = mid+1;

}

}

return -1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Bsearch(int A[], int n, int x);

int main()

{

int i,n,x,a[100];

printf("Enter the size of an array : ");

scanf("%d",&n);

printf("Enter Elements of an array : ");

for(i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

for(i=0;i<n;i++)

printf("%d ",a[i]);

printf("\nEnter the element which you want to search : ");

scanf("%d",&x);

printf("After Binary Search : ");

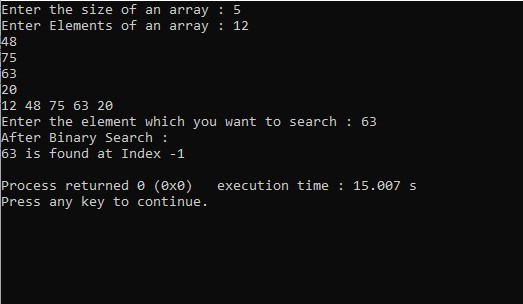
printf("\n%d is found at Index %d \n",x,Bsearch(a,n,x));

return 0;

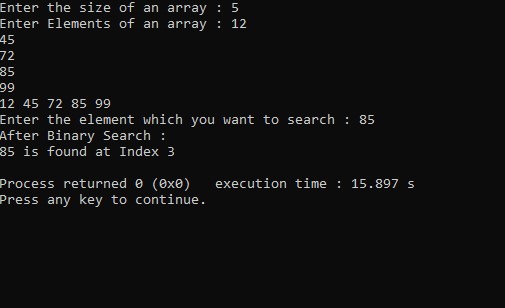
}

**OUTPUT: (3 Sample Input-Outputs)**

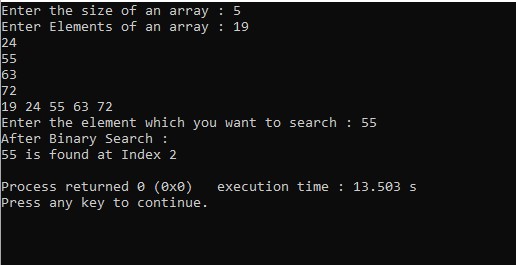
**First Sample Input-Output**



**Second Sample Input-Output**



**Third Sample Input-Output**

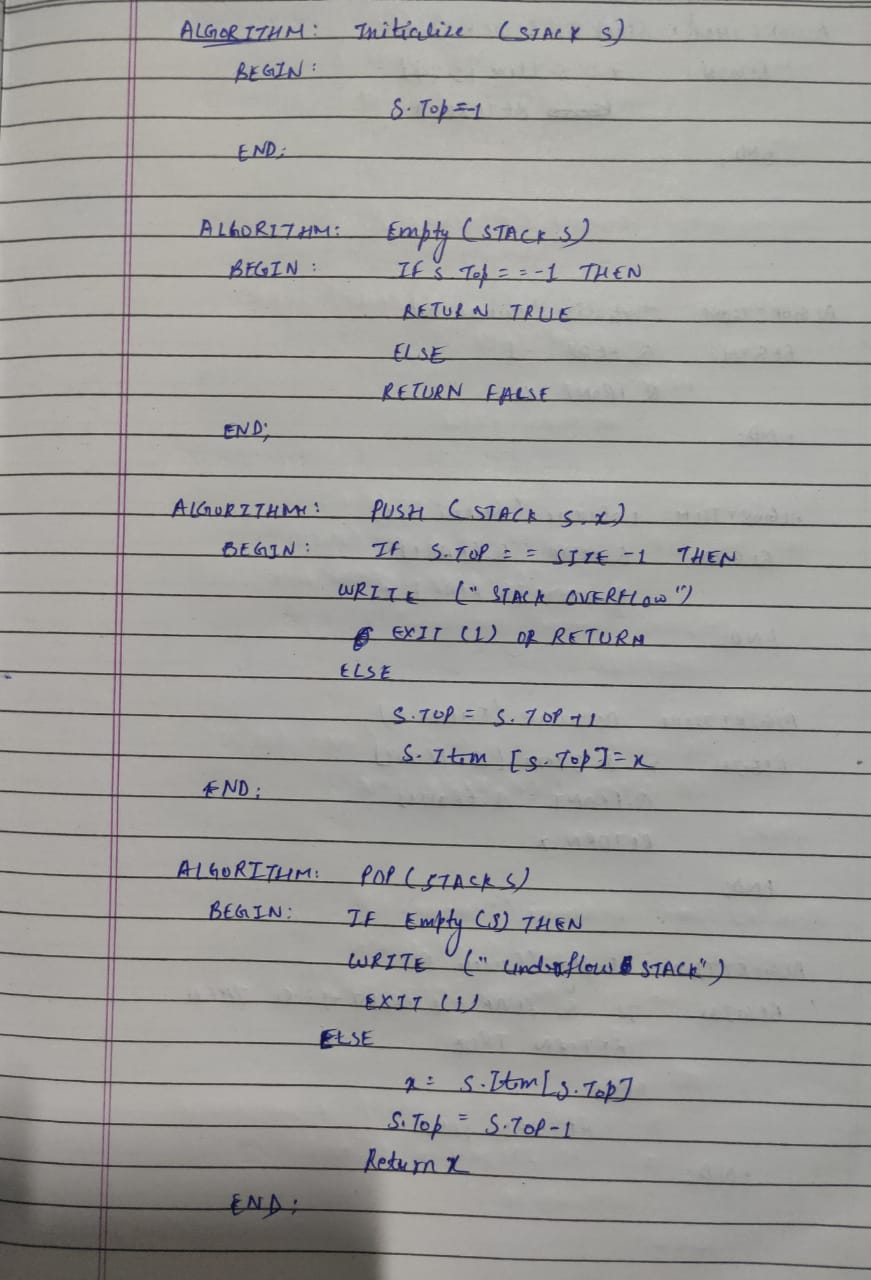


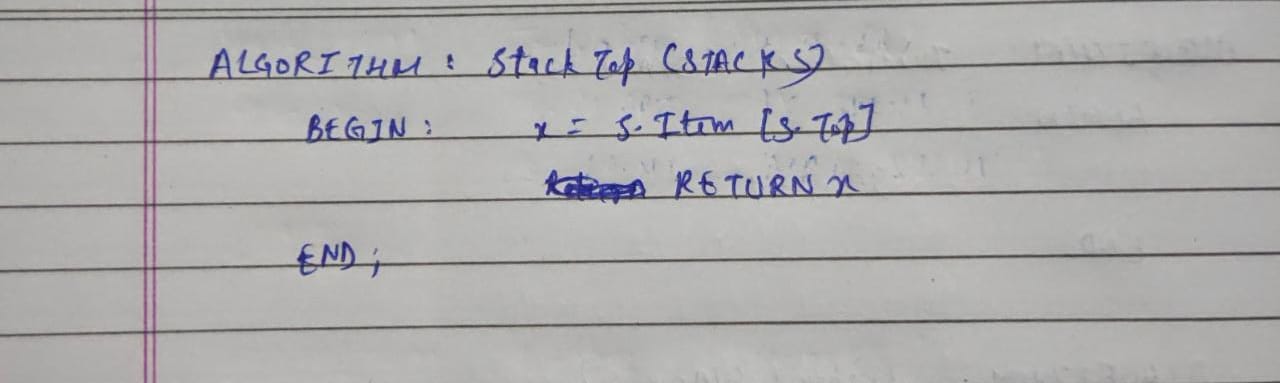
**EXPERIMENT NUMBER 4**

**Implementation of Stack using Array**

**TITLE OF EXPERIMENT: Program for Stack Primitive Operations**

**ALGORITHM:**





**For All Operation on Stack:**

**Time Complexity:**  O (1)

**Space Complexity:** O (1)

**C CODE:**

/\*\*PROGRAM FOR STACK PRIMITIVE OPERATIONS\*\*\*/

/\*AUTHOR: HARSH MOHAN AD NO. :2019B101166\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#define SIZE 20

#define TRUE 1

#define FALSE 0

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct stack

{

int item[SIZE];

int top;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct stack S;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void initialize(void)

{

S.top=-1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void push(int x)

{

if(S.top==SIZE-1)

{

printf("STACK OVERFLOW");

return;

}

else

{

S.top++;

S.item[S.top]=x;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int empty(void)

{

if (S.top==-1)

return TRUE;

else

return FALSE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int pop(void)

{

int x;

if(empty())

{

printf("STACK UNDERFLOW");

return;

}

else

{

x=S.item[S.top];

S.top--;

return x;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int stacktop(void)

{

int x;

x=S.item[S.top];

return x;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void display()

{

int i,c;

if(S.top==-1)

printf("Stack is empty!");

else

{

printf("Stack (From Top to Bottom) is : ");

for(i=S.top;i>=0;--i)

printf("%d ",S.item[i]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

initialize();

int choice;

int C;

printf("Please select any option Given Below for Sorting : \n");

while(1)

{

printf("\n1. Push\n2. Pop\n3. StackTop \n4. Empty\n5. Display\n6. Exit the Program.\n");

printf("\nEnter your Choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter item which you want to be pushed : ");

scanf("%d",&C);

push(C);

break;

case 2:

printf("Poped item is : %d",pop());

break;

case 3:

printf("Stacktop item is : %d",stacktop());

break;

case 4:

if(empty())

printf("Stack is Empty");

else

printf("Stack is Not Empty");

break;

case 5:

display();

break;

case 6:

return 0;

default:

printf("\nPlease Select only 1-4 option ----\n");

}

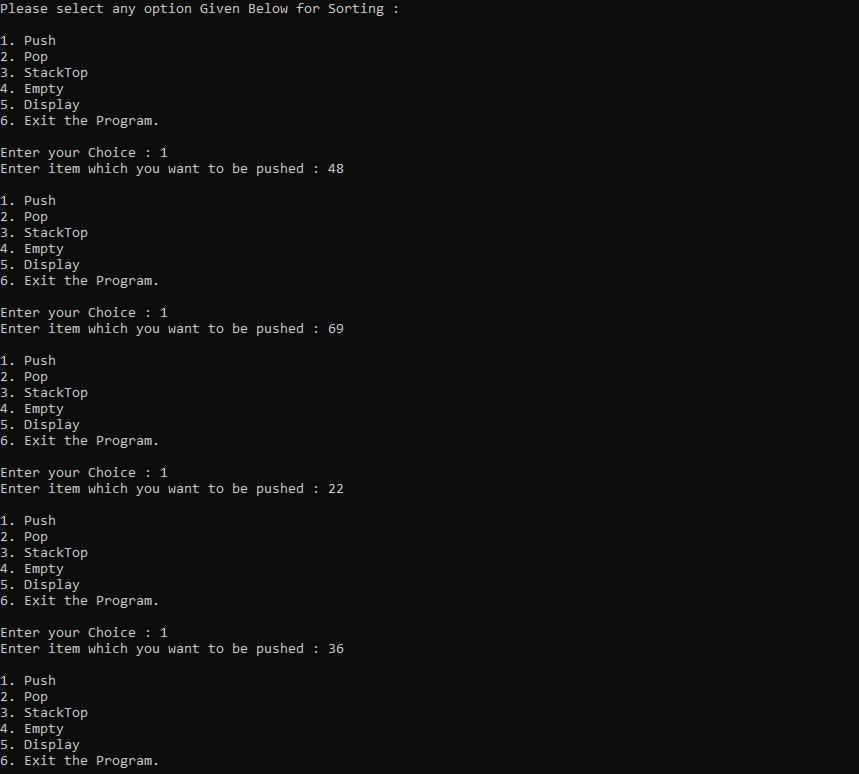
}

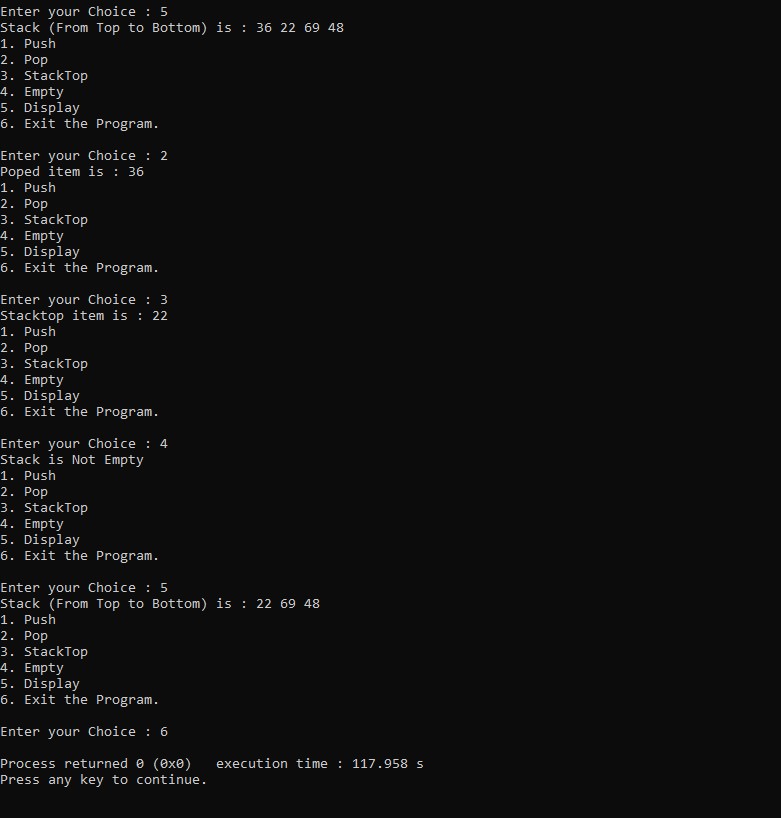
return 0;

}

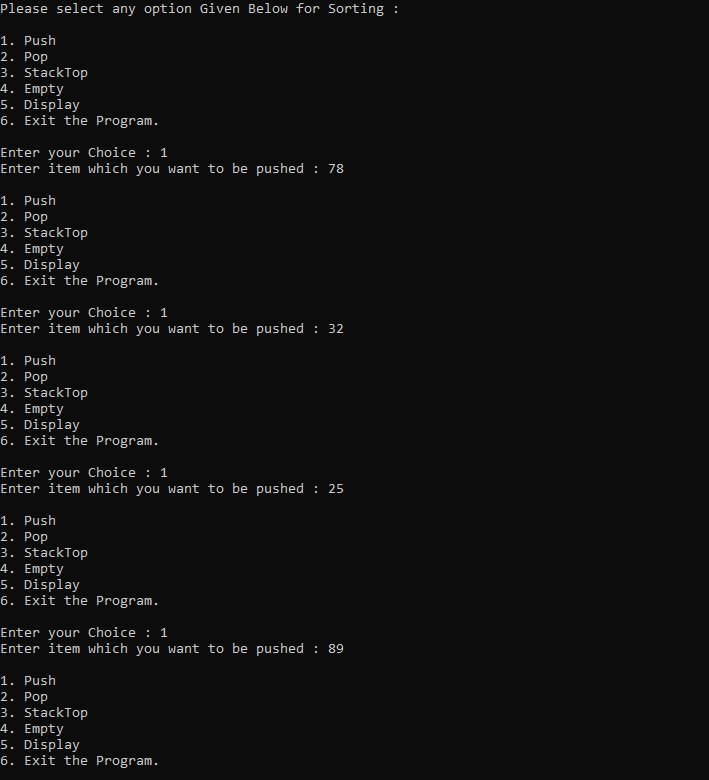
**OUTPUT: (3 Sample Input-Outputs)**

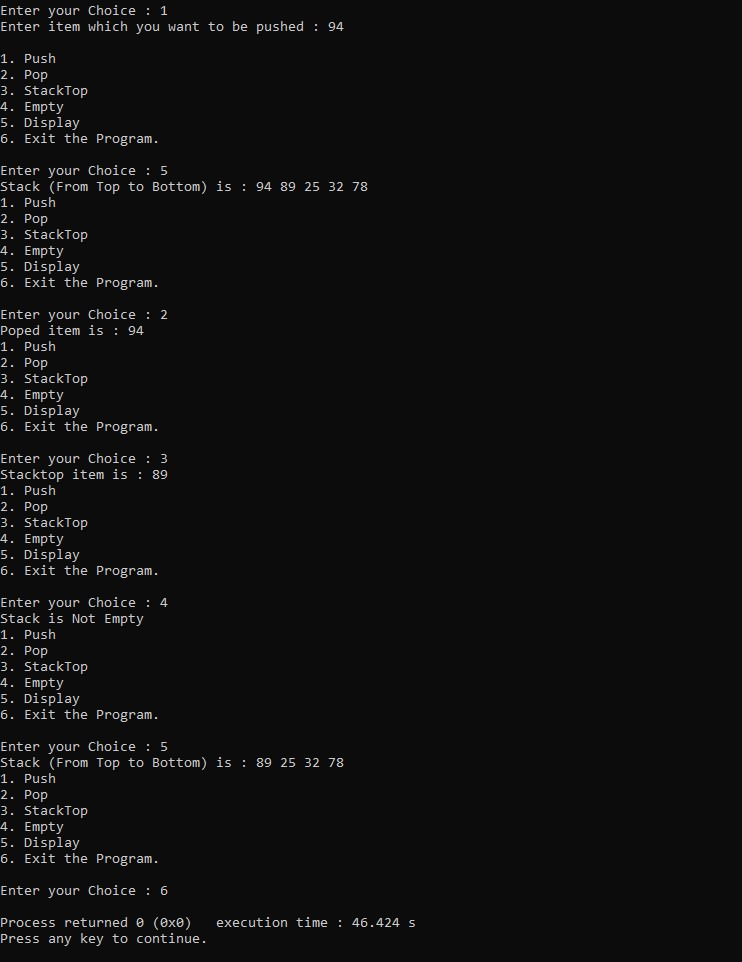
**First Sample Input-Output**



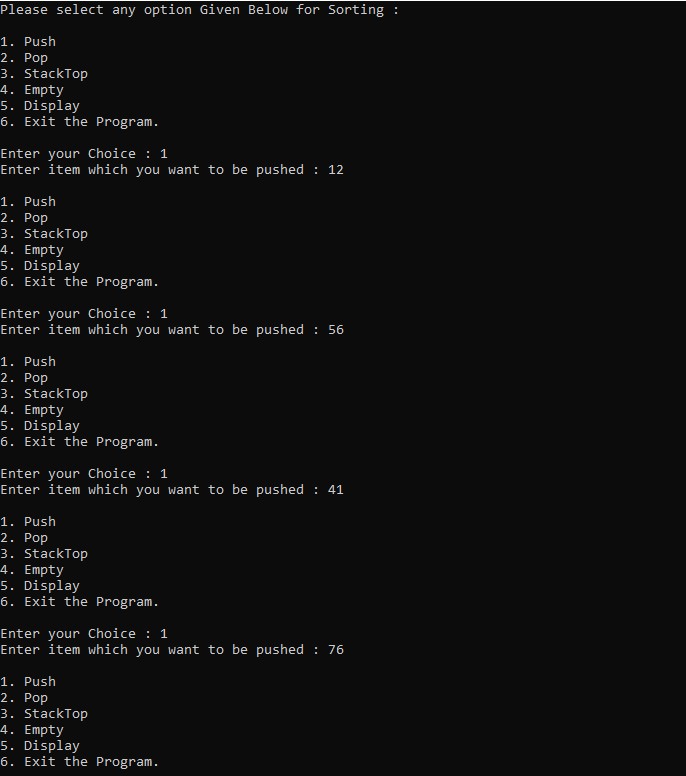


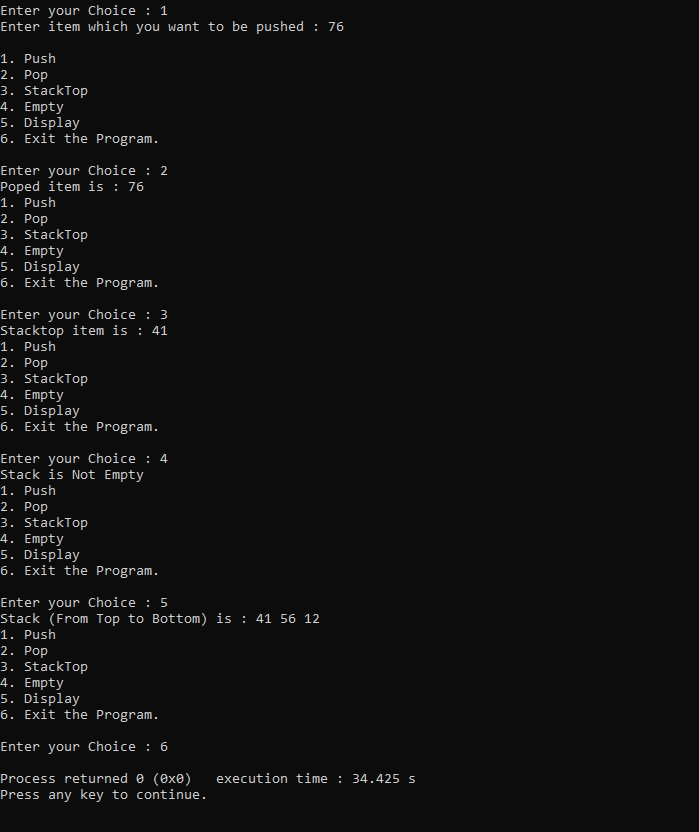
**Second Sample Input-Output**





**Third Sample Input-Output**



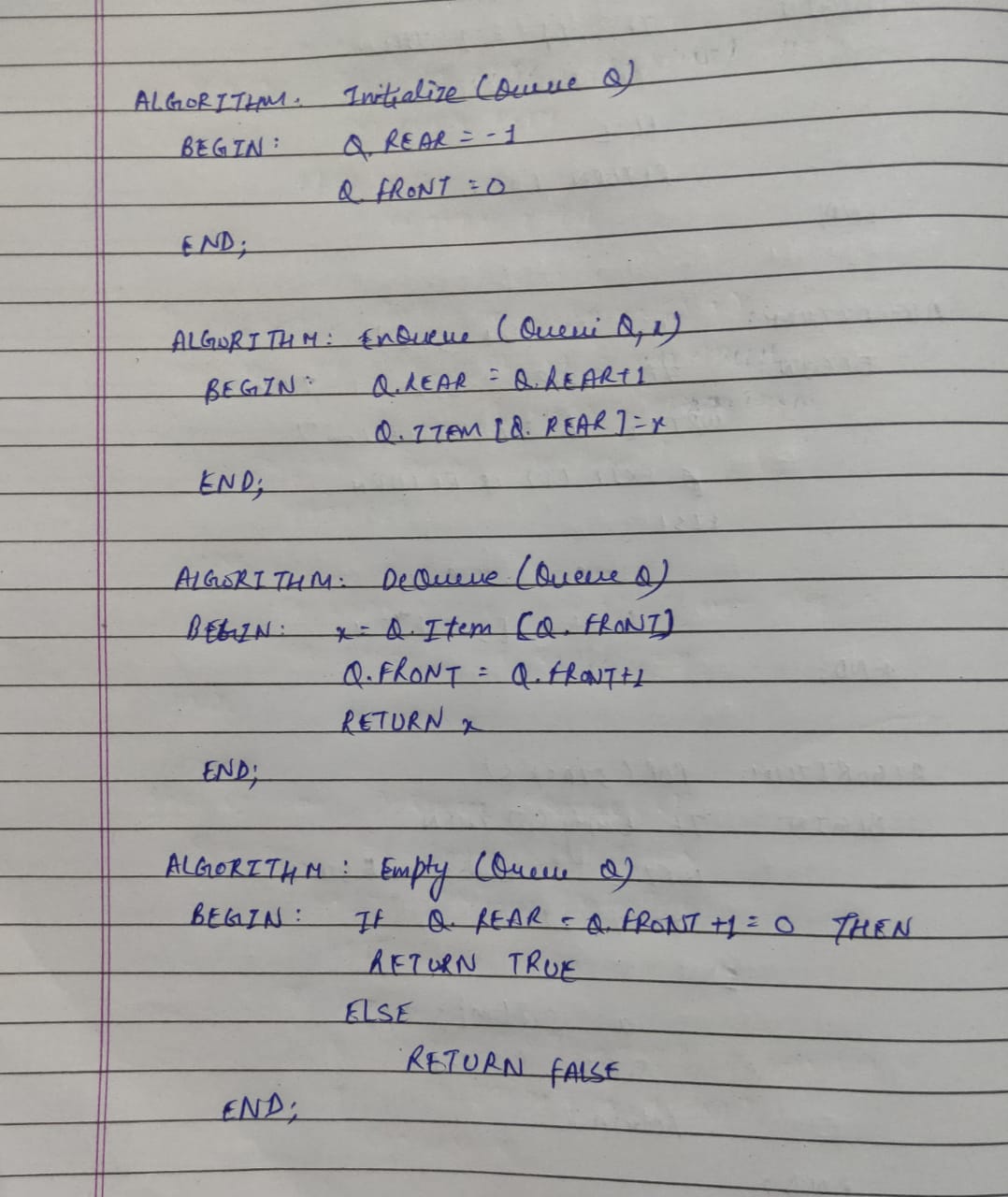


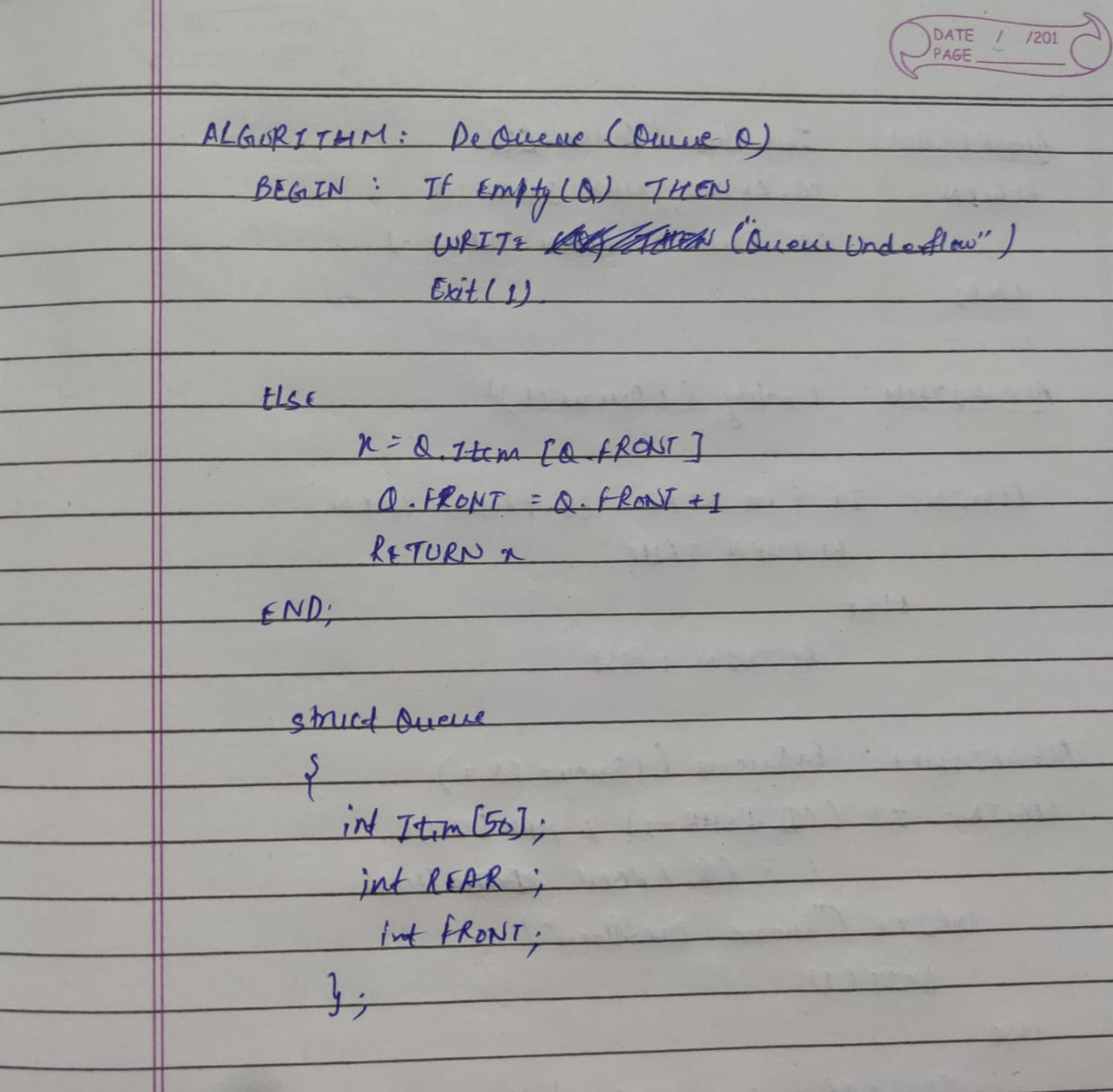
**EXPERIMENT NUMBER 5**

**Implementation of Queue using Array**

**TITLE OF EXPERIMENT: Program for Queue Primitive Operations using Array**

**ALGORITHM:**





**For All Operation on Queue:**

**Time Complexity:**  O (1)

**Space Complexity:** O (1)

**C CODE:**

/\*\*\*PROGRAM OF ARRAY IMPLEMENTATION OF LINEAR QUEUE\*\*\*\*\*/

/\*\*AUTHOR: HARSH MOHAN; ADMISSION NO. :2019B101166\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<stdlib.h>

#define MAXQUEUE 20

#define TRUE 1

#define FALSE 0

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct queue

{

int ITEM[MAXQUEUE];

int REAR;

int FRONT;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct queue Q;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void initialize()

{

Q.REAR=-1;

Q.FRONT=0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int enqueue(int x)

{

if(Q.REAR==MAXQUEUE)

{

printf("Queue Overflows");

exit(1);

}

else

{

Q.REAR=Q.REAR+1;

Q.ITEM[Q.REAR]=x;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int empty()

{

if(Q.REAR-Q.FRONT+1==0)

return TRUE;

else

return FALSE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int dequeue()

{

if(empty())

{

printf("Queue Underflows");

exit(1);

}

else

{

int x;

x=Q.ITEM[Q.FRONT];

Q.FRONT=Q.FRONT+1;

return x;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void display()

{

int i;

if(Q.FRONT==-1)

printf("Queue is Empty!!");

else

{

printf("Queue is: ");

for (i=Q.FRONT;i<=Q.REAR;i++)

printf("%d ",Q.ITEM[i]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{

initialize();

int choice,C;

printf("Please select any option Given Below for Sorting : \n");

while(1)

{

printf("\n1. Enqueue\n2. Dequeue\n3. Empty\n4. Display\n5. Exit the Program.\n");

printf("\nEnter your Choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter item which you want to be Enqueue : ");

scanf("%d",&C);

enqueue(C);

break;

case 2:

printf("Deleted item is : %d",dequeue());

break;

case 3:

if(empty())

printf("Stack is Empty");

else

printf("Stack is Not Empty");

break;

case 4:

display();

break;

case 5:

return 0;

default:

printf("\nPlease Select only 1-4 option ----\n");

}

}

return 0;

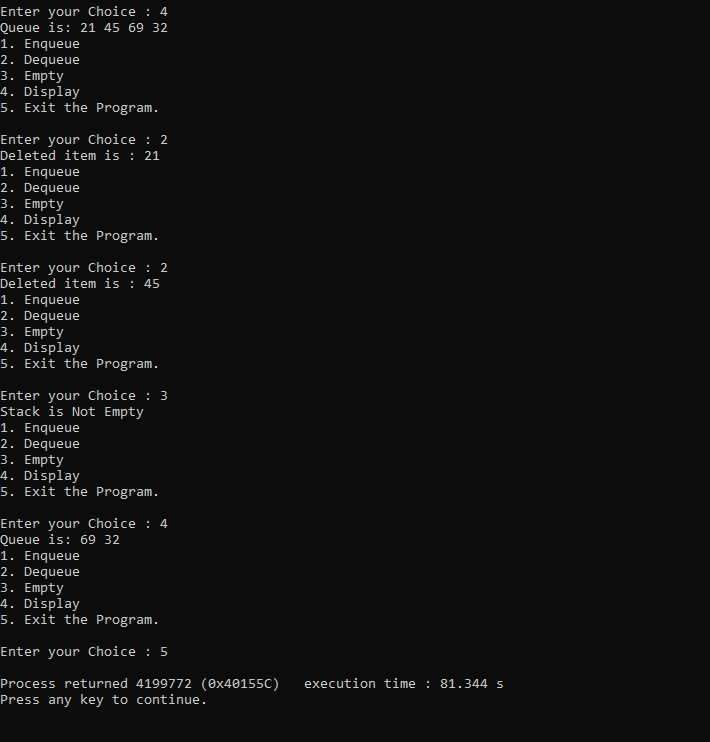
}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

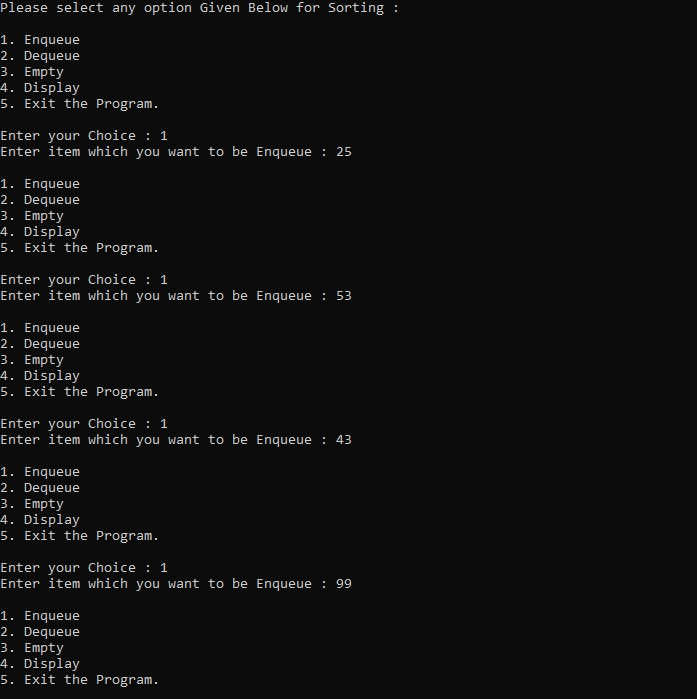
**OUTPUT: (3 Sample Input-Outputs)**

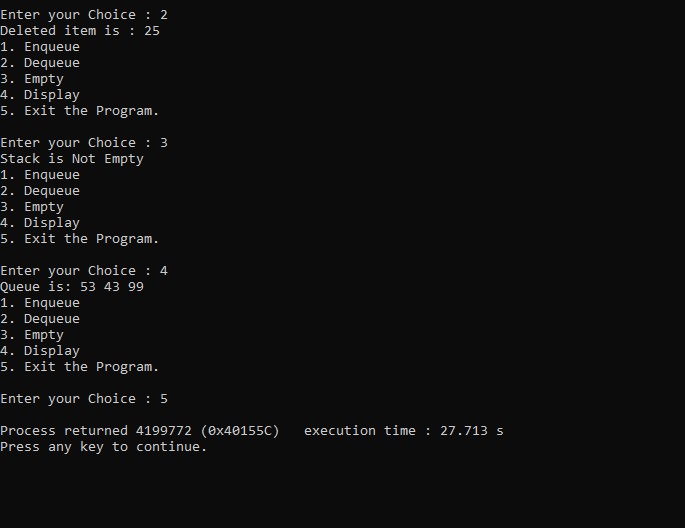
**First Sample Input-Output**



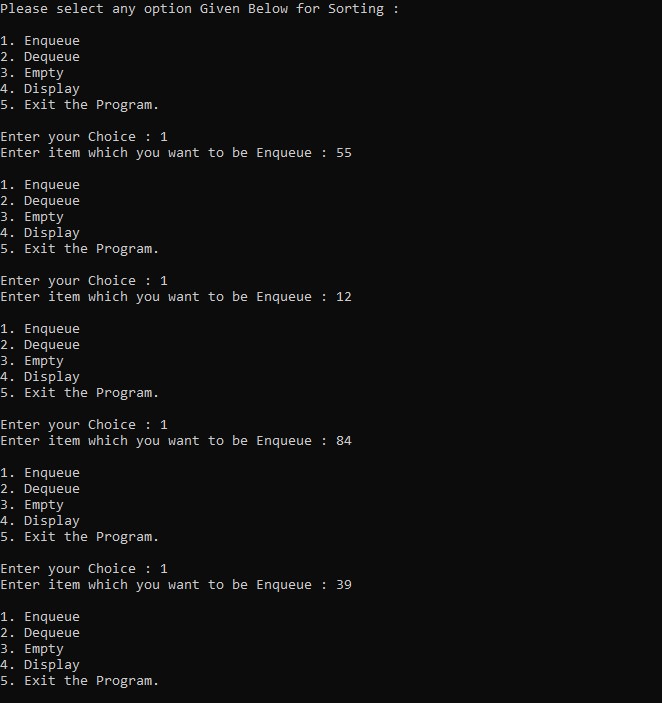


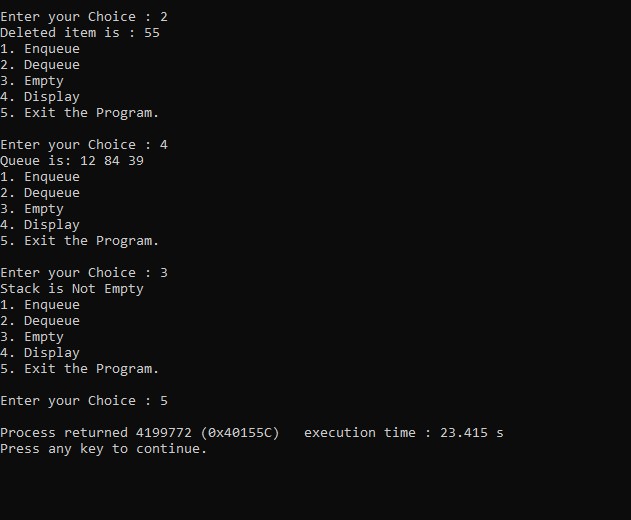
**Second Sample Input-Output**





**Third Sample Input-Output**



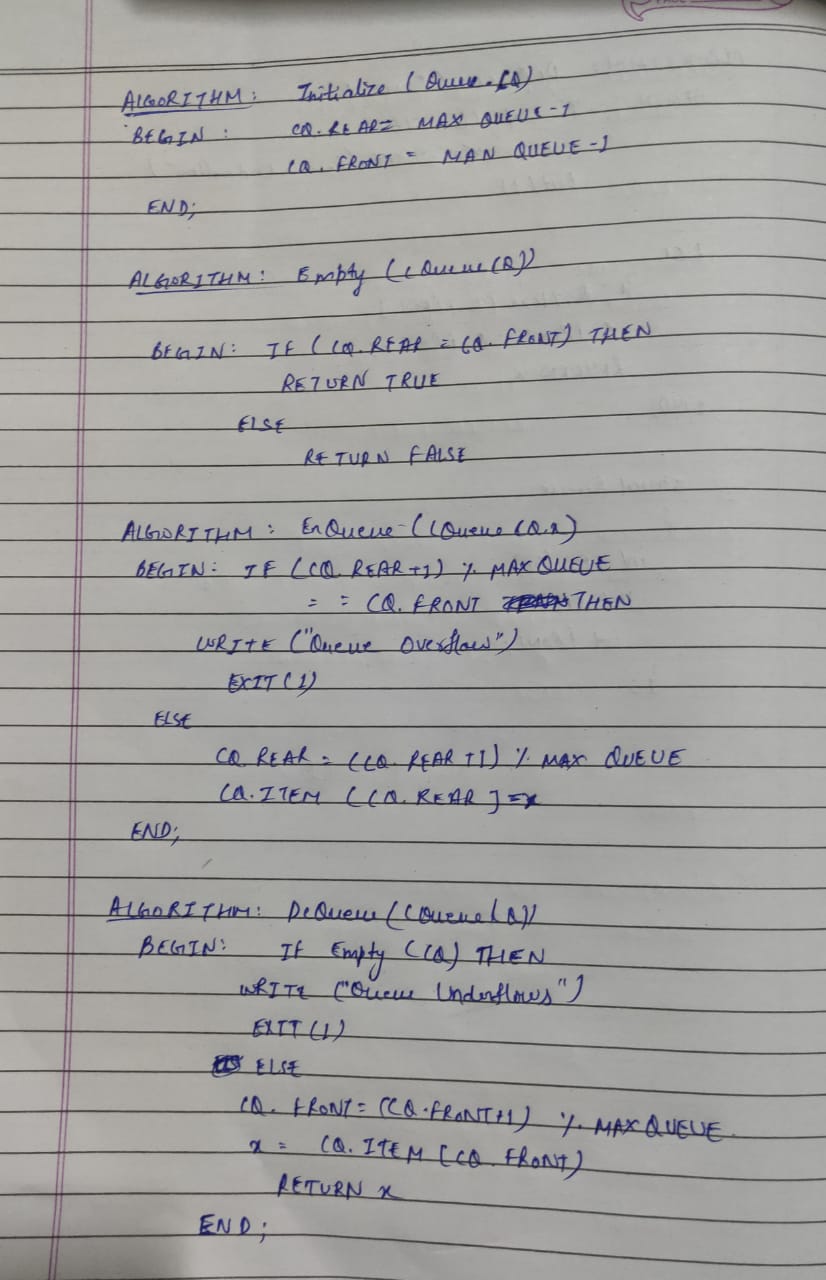


**EXPERIMENT NUMBER 6**

**Implementation of Circular Queue using Array**

**TITLE OF EXPERIMENT: Program for Queue Primitive Operations using Array**

**ALGORITHM:**



**For All Operation on Circular Queue:**

**Time Complexity:**  O (1)

**Space Complexity:** O (N)

**C CODE:**

/\*\*\*PROGRAM OF ARRAY IMPLEMENTATION OF CIRCULAR QUEUE\*\*\*/

/\*\*AUTHOR: HARSH MOHAN; ADMISSION NO. :2019B101166\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<stdlib.h>

#define MAXQUEUE 10

#define TRUE 1

#define FALSE 0

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct queue

{

int ITEM[MAXQUEUE];

int REAR;

int FRONT;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct queue CQ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void initialize()

{

CQ.REAR=MAXQUEUE-1;

CQ.FRONT=MAXQUEUE-1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int enqueue(int x)

{

if((CQ.REAR+1)%MAXQUEUE==CQ.FRONT)

{

printf("Queue Overflows");

exit(1);

}

else

{

CQ.REAR=(CQ.REAR+1)%MAXQUEUE;

CQ.ITEM[CQ.REAR]=x;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int empty()

{

if(CQ.REAR==CQ.FRONT)

return TRUE;

else

return FALSE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int dequeue()

{

if(empty())

{

printf("Queue Underflows");

exit(1);

}

else

{

int x;

CQ.FRONT=(CQ.FRONT+1)%MAXQUEUE;

x=CQ.ITEM[CQ.FRONT];

return x;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void display()

{

int i;

if(empty())

printf("Queue is Empty!!");

else

{

printf("Queue is: ");

for (i=CQ.FRONT;i!=CQ.REAR;i=(i+1)%MAXQUEUE)

printf("%d ",CQ.ITEM[i]);

printf("%d",CQ.ITEM[i]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{

initialize();

int choice,C;

printf("Please select any option Given Below for Sorting : \n");

while(1)

{

printf("\n1. Enqueue\n2. Dequeue\n3. Empty\n4. Display\n5. Exit the Program.\n");

printf("\nEnter your Choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter item which you want to be Enqueue : ");

scanf("%d",&C);

enqueue(C);

break;

case 2:

printf("Deleted item is : %d",dequeue());

break;

case 3:

if(empty())

printf("Stack is Empty");

else

printf("Stack is Not Empty");

break;

case 4:

display();

break;

case 5:

return 0;

default:

printf("\nPlease Select only 1-4 option ----\n");

}

}

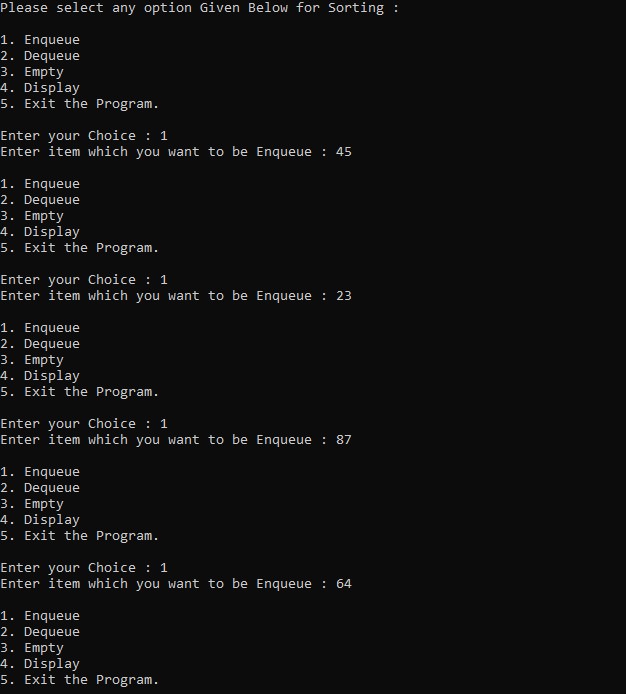
return 0;

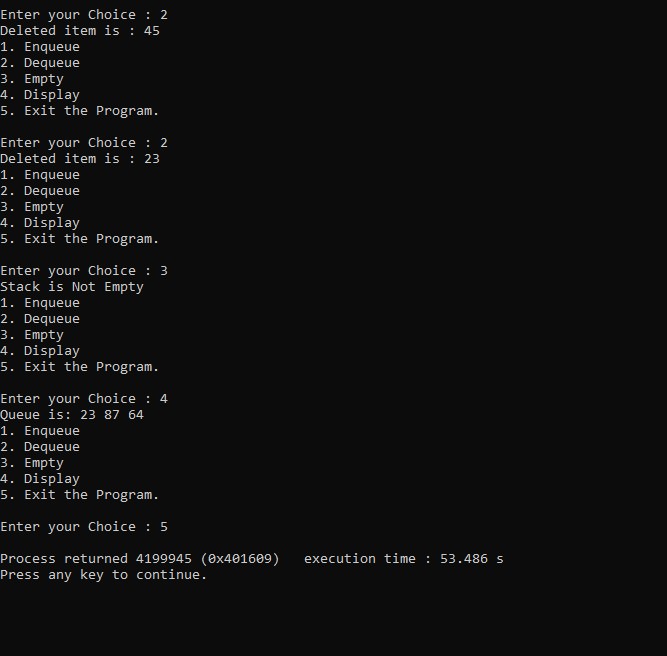
}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

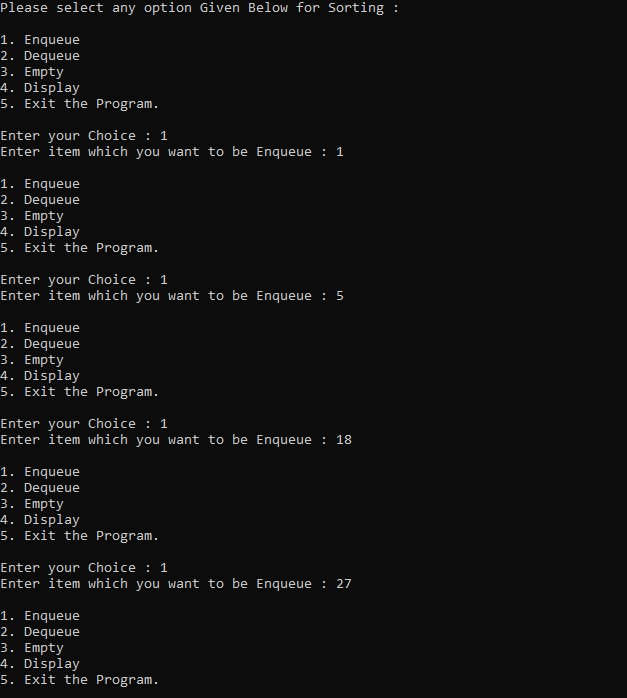
**OUTPUT: (3 Sample Input-Outputs)**

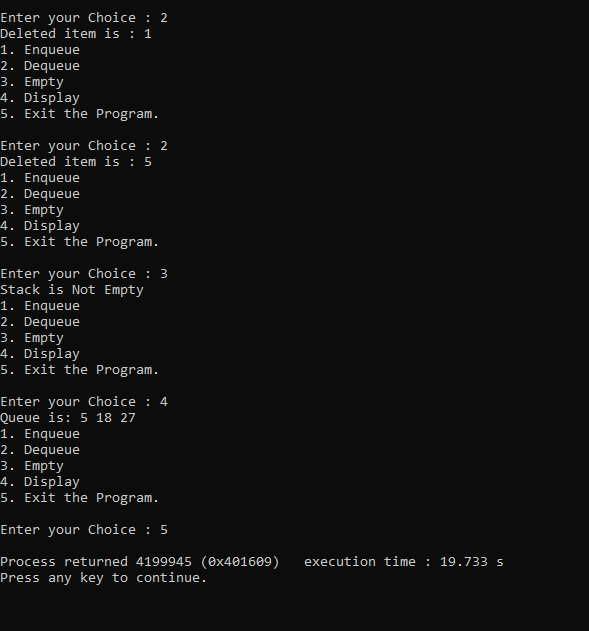
**First Sample Input-Output**



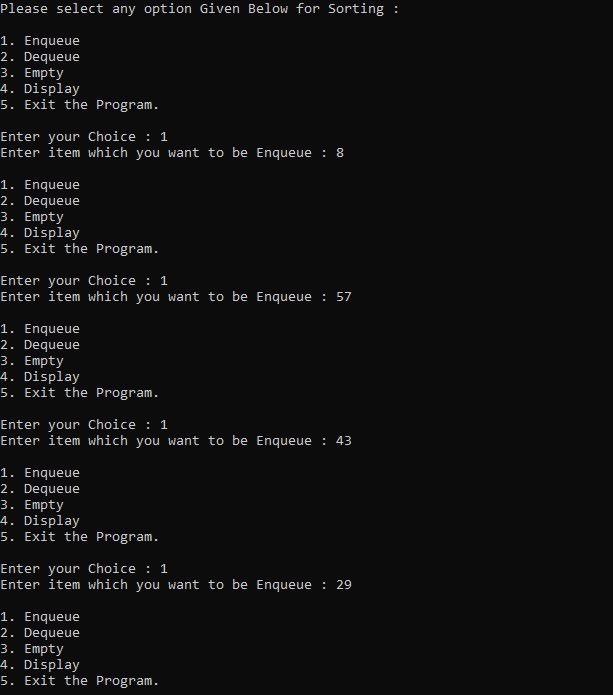


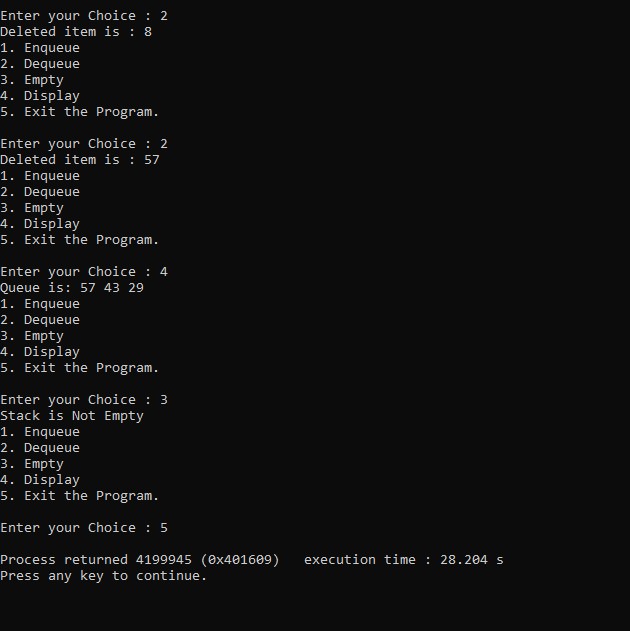
**Second Sample Input-Output**





**Third Sample Input-Output**



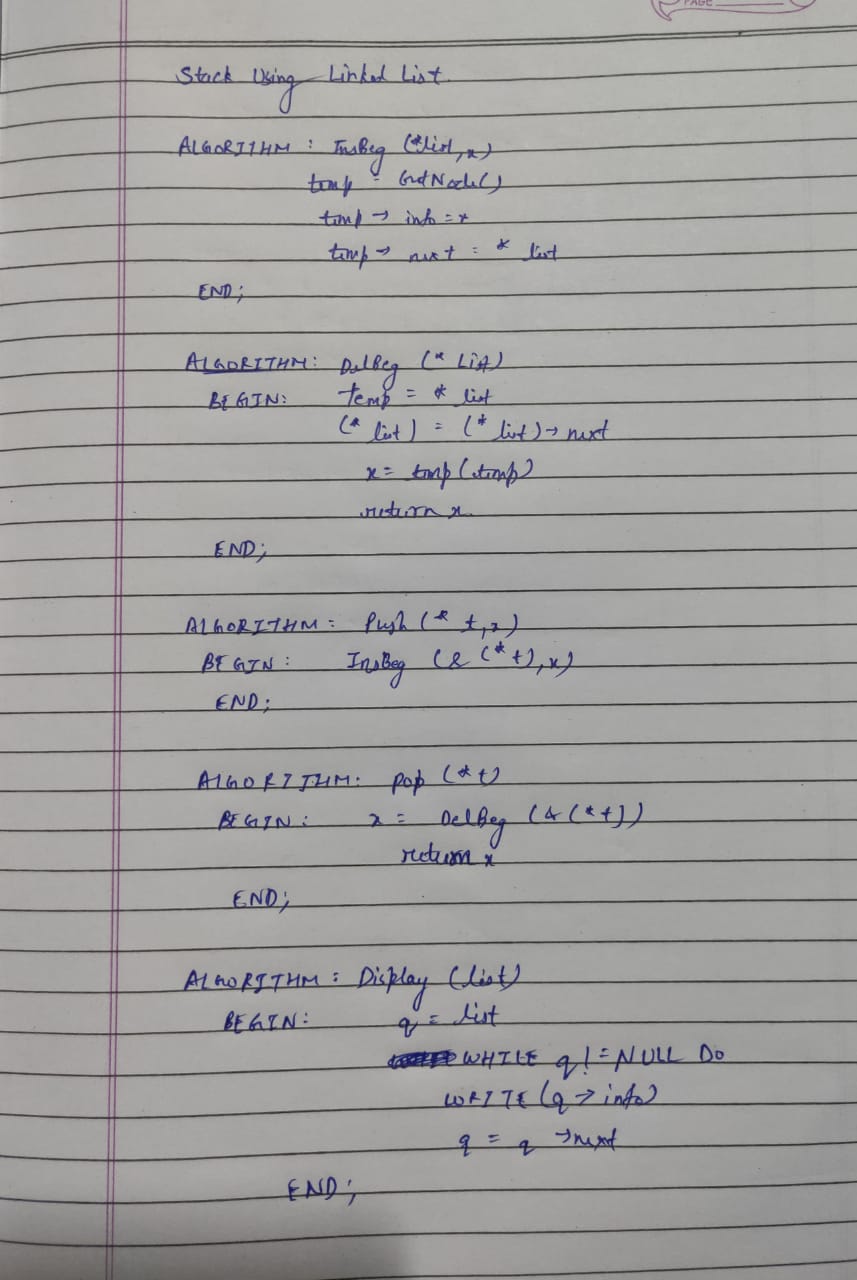


**EXPERIMENT NUMBER 7**

**Implementation of Stack using Linked List**

**TITLE OF EXPERIMENT: Program for Stack Primitive Operations using Linked List**

**ALGORITHM:**



**For insertion –**

**Time Complexity:**  O (1)

**Space Complexity:** O (1)

**For deletion –**

**Time Complexity:**  O (1)

**Space Complexity:** O (1)

**For push –**

**Time Complexity:**  O (1)

**Space Complexity:** O (1)

**For pop –**

**Time Complexity:**  O (1)

**Space Complexity:** O (1)

**For display –**

**Time Complexity:**  O (N)

**Space Complexity:** O (1)

**C CODE:**

/\*\*PROGRAM FOR STACK PRIMITIVE OPERATIONS USING LINKED LIST\*\*\*/

/\*\*\*\*\*\*\*\*\*AUTHOR : HARSH MOHAN, AD NO. :2019B101166\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<string.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node

{

int info;

struct node \*next;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node \*getnode()

{

struct node \*p;

p=(struct node\*)malloc(sizeof(struct node));

return p;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int insbeg(struct node \*\*list,int x)

{

struct node \*temp;

temp=getnode();

temp->info=x;

temp->next=\*list;

\*list=temp;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int delbeg(struct node \*\*list)

{

struct node \*temp;

int x;

temp=\*list;

(\*list)=(\*list)->next;

x=temp->info;

free(temp);

return x;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void push(struct node \*\*t,int x)

{

insbeg(&(\*t),x);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int pop(struct node \*\*t)

{

int x;

x=delbeg(&(\*t));

return x;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void display(struct node \*list)

{

struct node \*t;

t=list;

while(t!=NULL)

{

printf("%d ",t->info);

t=t->next;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

struct node \*q;

q=NULL;

int choice;

int C;

printf("Please select any option Given Below for Sorting : \n");

while(1)

{

printf("\n1. Push\n2. Pop\n3.Display\n4. Exit the Program.\n");

printf("\nEnter your Choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter item which you want to be pushed : ");

scanf("%d",&C);

push(&q,C);

break;

case 2:

printf("Poped item is : %d",pop(&q));

break;

case 3:

display(q);

break;

case 4:

return 0;

default:

printf("\nPlease Select only 1-4 option ----\n");

}

}

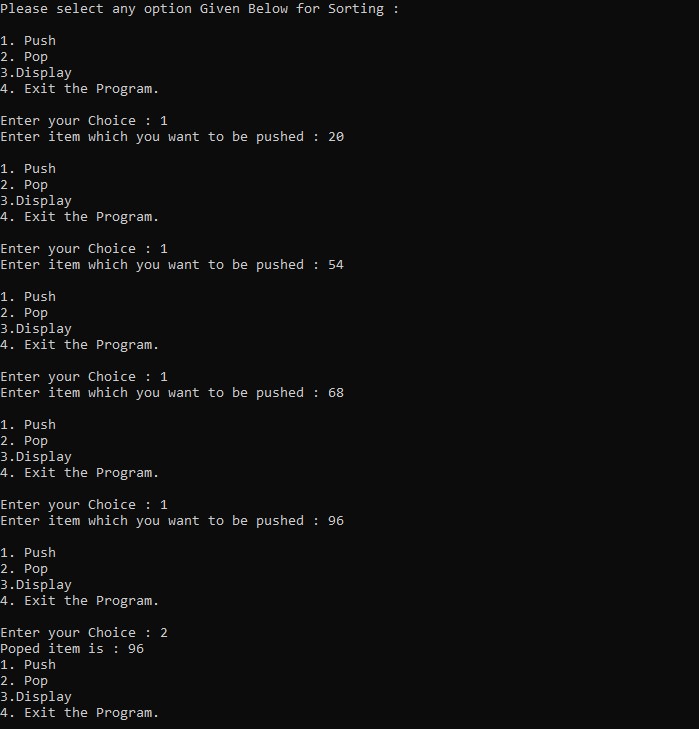
return 0;

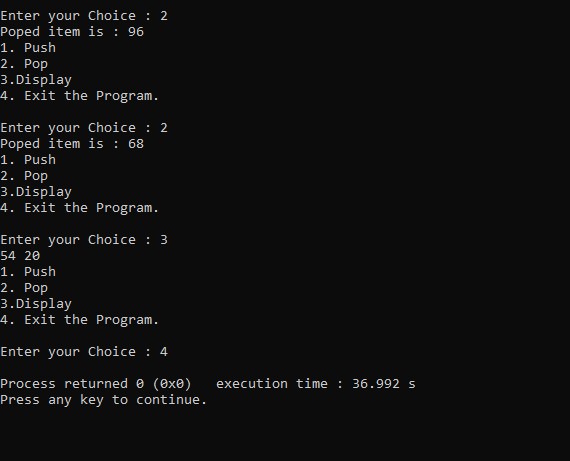
}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

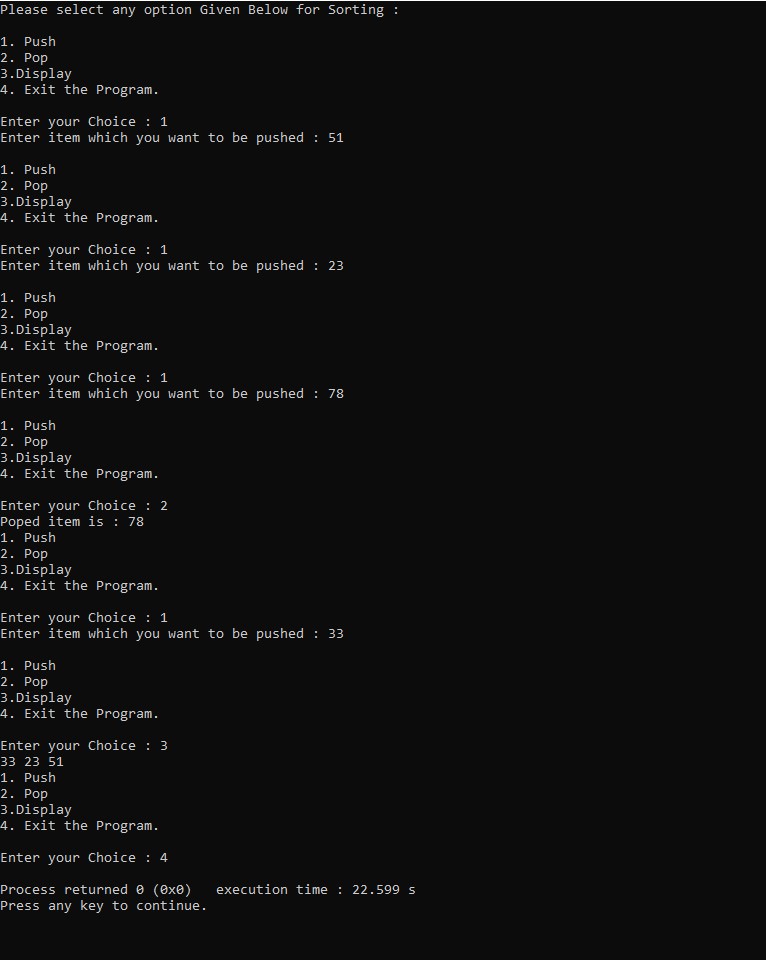
**OUTPUT: (3 Sample Input-Outputs)**

**First Sample Input-Output**

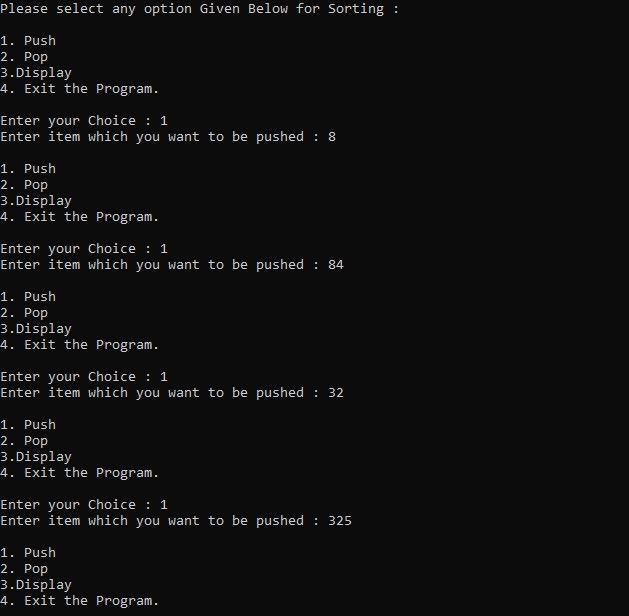


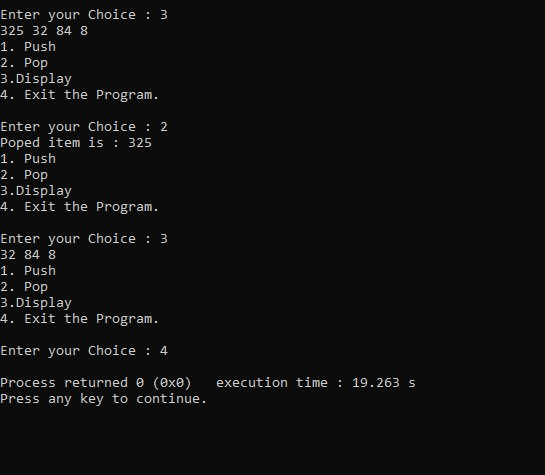


**Second Sample Input-Output**



**Third Sample Input-Output**



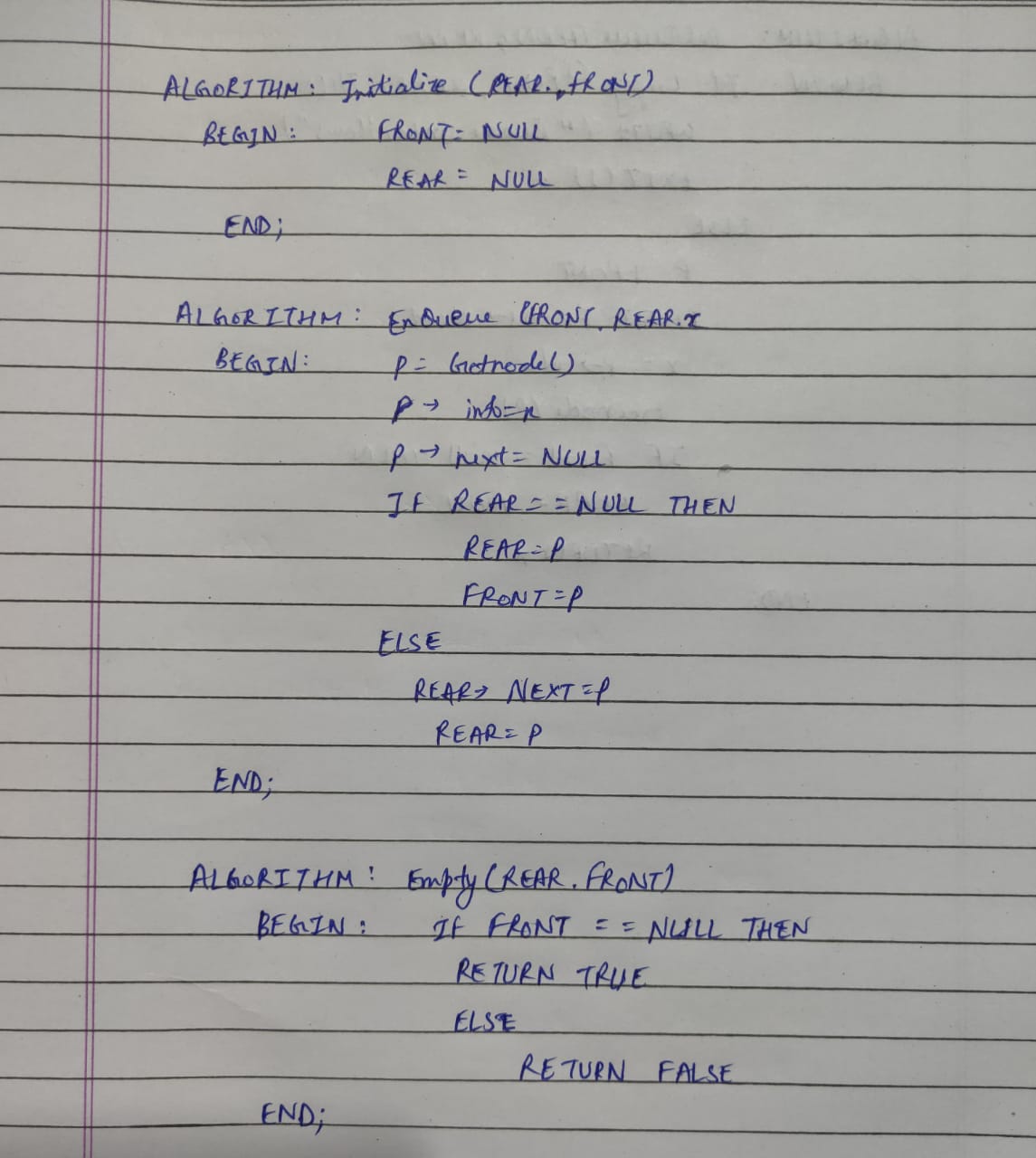


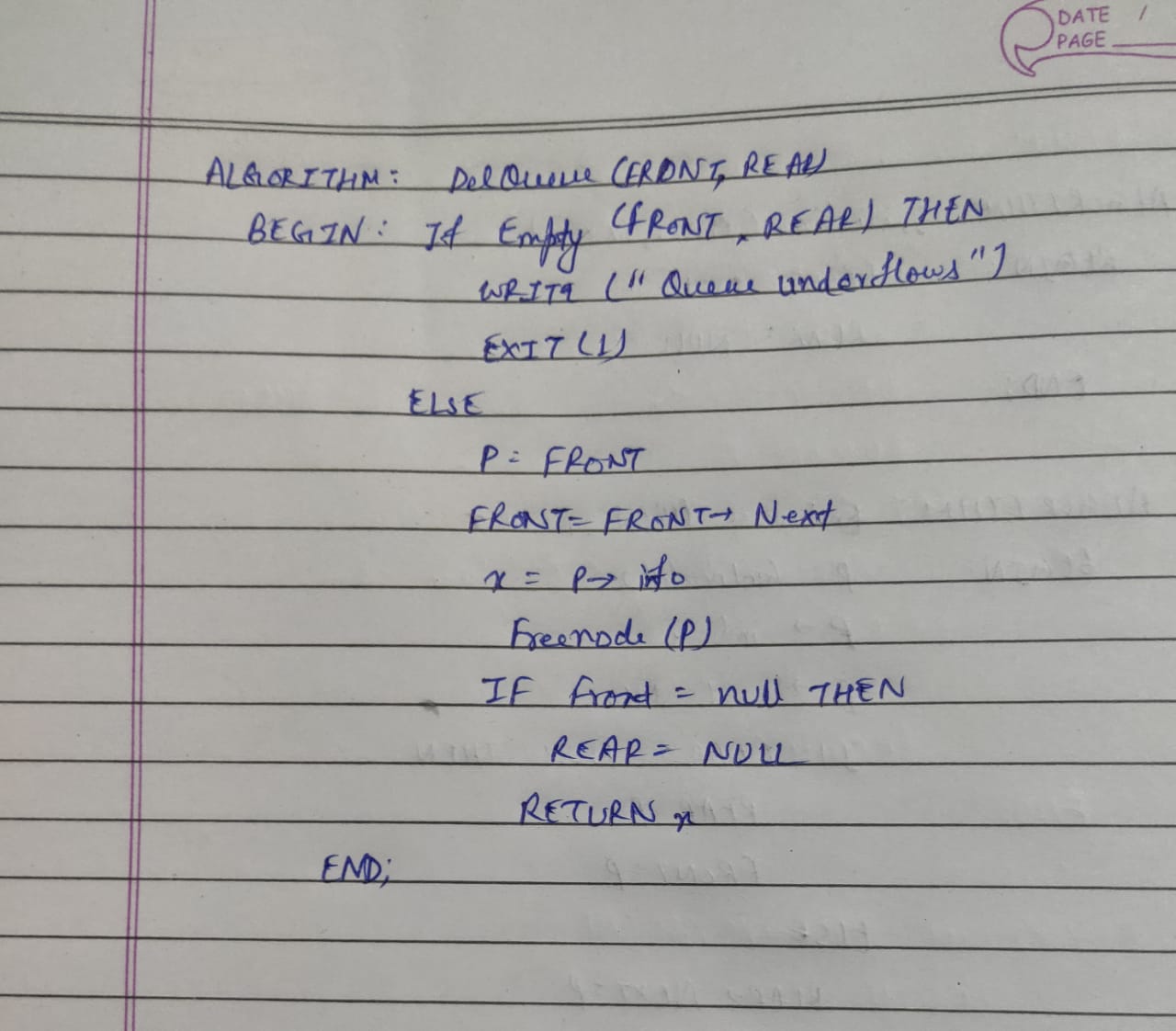
**EXPERIMENT NUMBER 8**

**Implementation of Queue using Linked List**

**TITLE OF EXPERIMENT: Program for Queue Primitive Operations using Linked List**

**ALGORITHM:**





**For All Operation on Circular Queue:**

**Time Complexity:**  O (1)

**Space Complexity:** O (N)

**C CODE:**

/\*\*\*\*PROGRAM FOR LINKED LIST IMPLEMENTATION OF LINEAR QUEUE\*\*\*\*/

/\*\*\*\*\*\*\*AUTHOR : HARSH MOHAN, ADMISSION NO.:2019B101166\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<stdlib.h>

#define TRUE 1

#define FALSE 0

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node

{

int info;

struct node \*next;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node \*FRONT, \*REAR;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Getnode()

{

struct node \*p;

p=(struct node\*)malloc(sizeof(struct node));

return p;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

initialize()

{

REAR=NULL;

FRONT=NULL;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int empty()

{

if(FRONT==NULL)

return TRUE;

else

return FALSE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

enqueue(int x)

{

struct node \*p;

p=Getnode();

p->info=x;

p->next=NULL;

if(REAR==NULL)

{

REAR=p;

FRONT=p;

}

else

{

REAR->next=p;

REAR=p;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int dequeue()

{

int x;

struct node \*p;

if(empty())

{

printf("Queue Underflows");

exit(1);

}

p=FRONT;

FRONT=FRONT->next;

x=p->info;

free(p);

if(FRONT==NULL)

REAR=NULL;

return x;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void display()

{

struct node \*p;

p=FRONT;

if(FRONT==NULL)

printf("Empty Queue!");

else

{

printf("Queue is: ");

while(p!=NULL)

{

printf("%d ",p->info);

p=p->next;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{

initialize();

int choice,C;

printf("Please select any option Given Below for Sorting : \n");

while(1)

{

printf("\n1. Enqueue\n2. Dequeue\n3. Empty\n4. Display\n5. Exit the Program.\n");

printf("\nEnter your Choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter item which you want to be Enqueue : ");

scanf("%d",&C);

enqueue(C);

break;

case 2:

printf("Deleted item is : %d",dequeue());

break;

case 3:

if(empty())

printf("Stack is Empty");

else

printf("Stack is Not Empty");

break;

case 4:

display();

break;

case 5:

return 0;

default:

printf("\nPlease Select only 1-4 option ----\n");

}

}

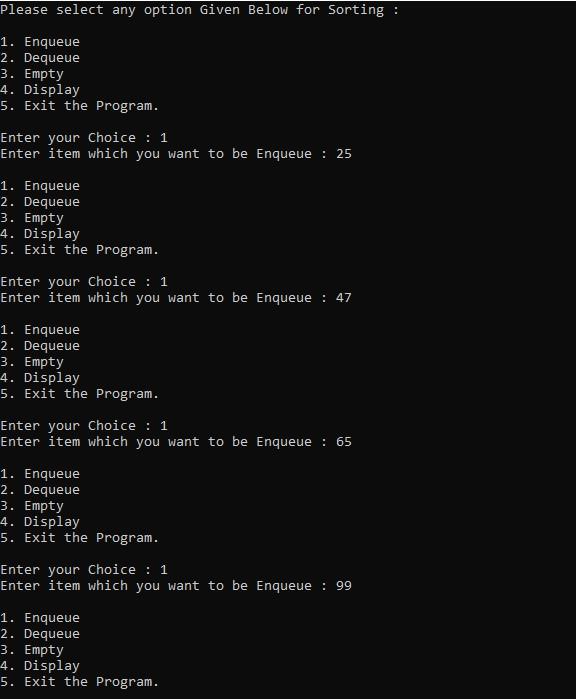
return 0;

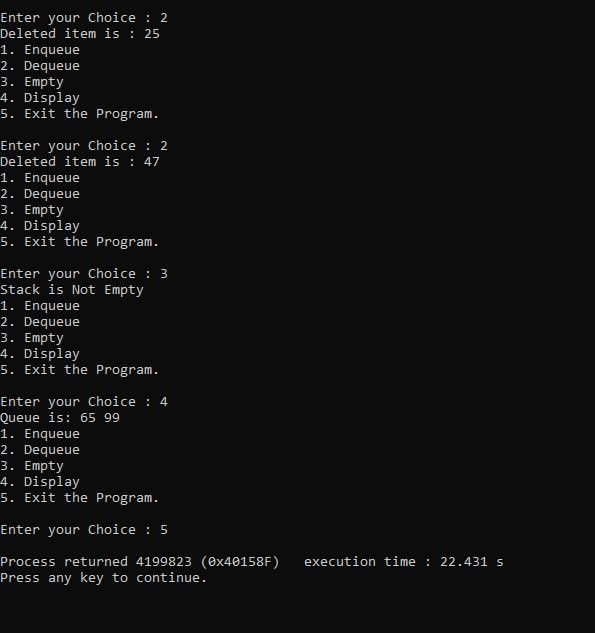
}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

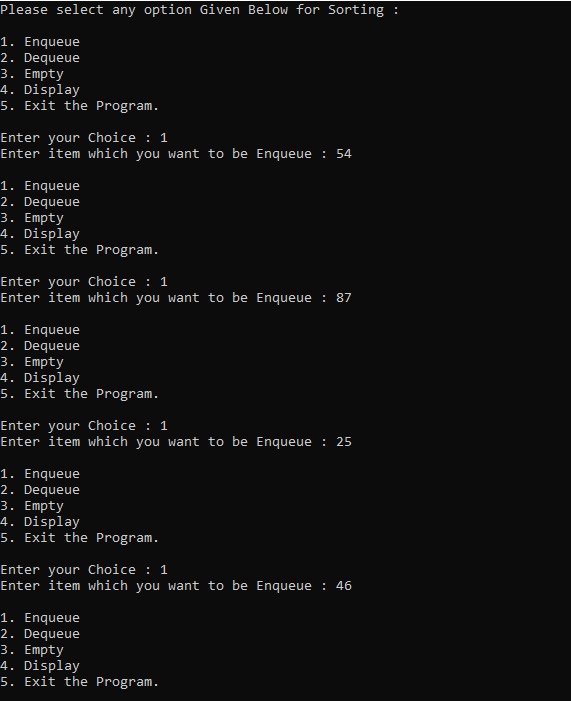
**OUTPUT: (3 Sample Input-Outputs)**

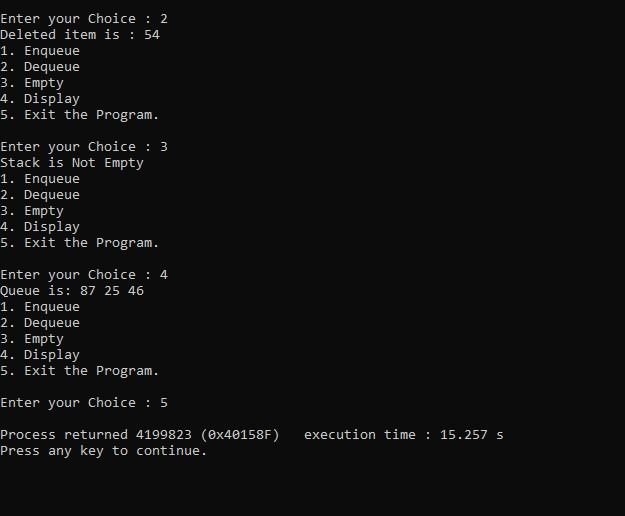
**First Sample Input-Output**



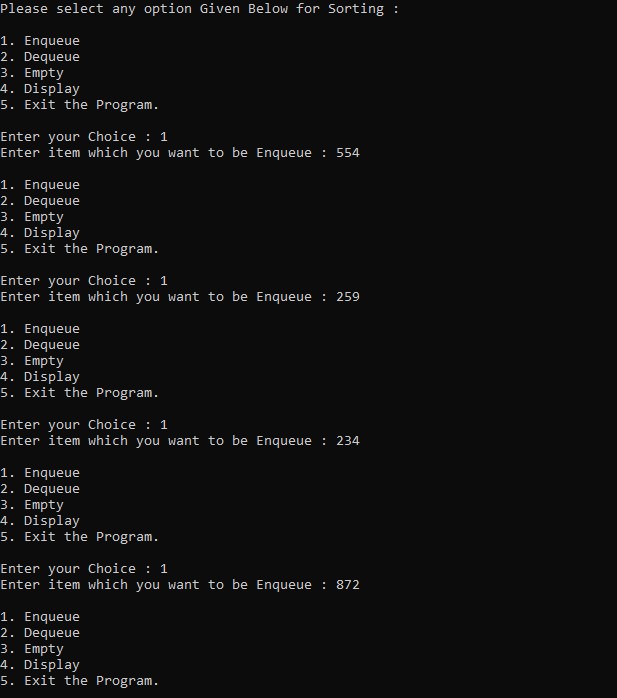


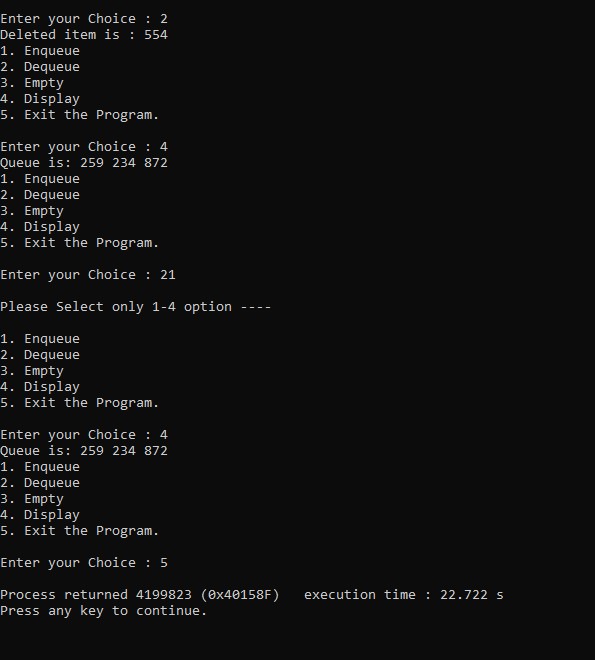
**Second Sample Input-Output**





**Third Sample Input-Output**



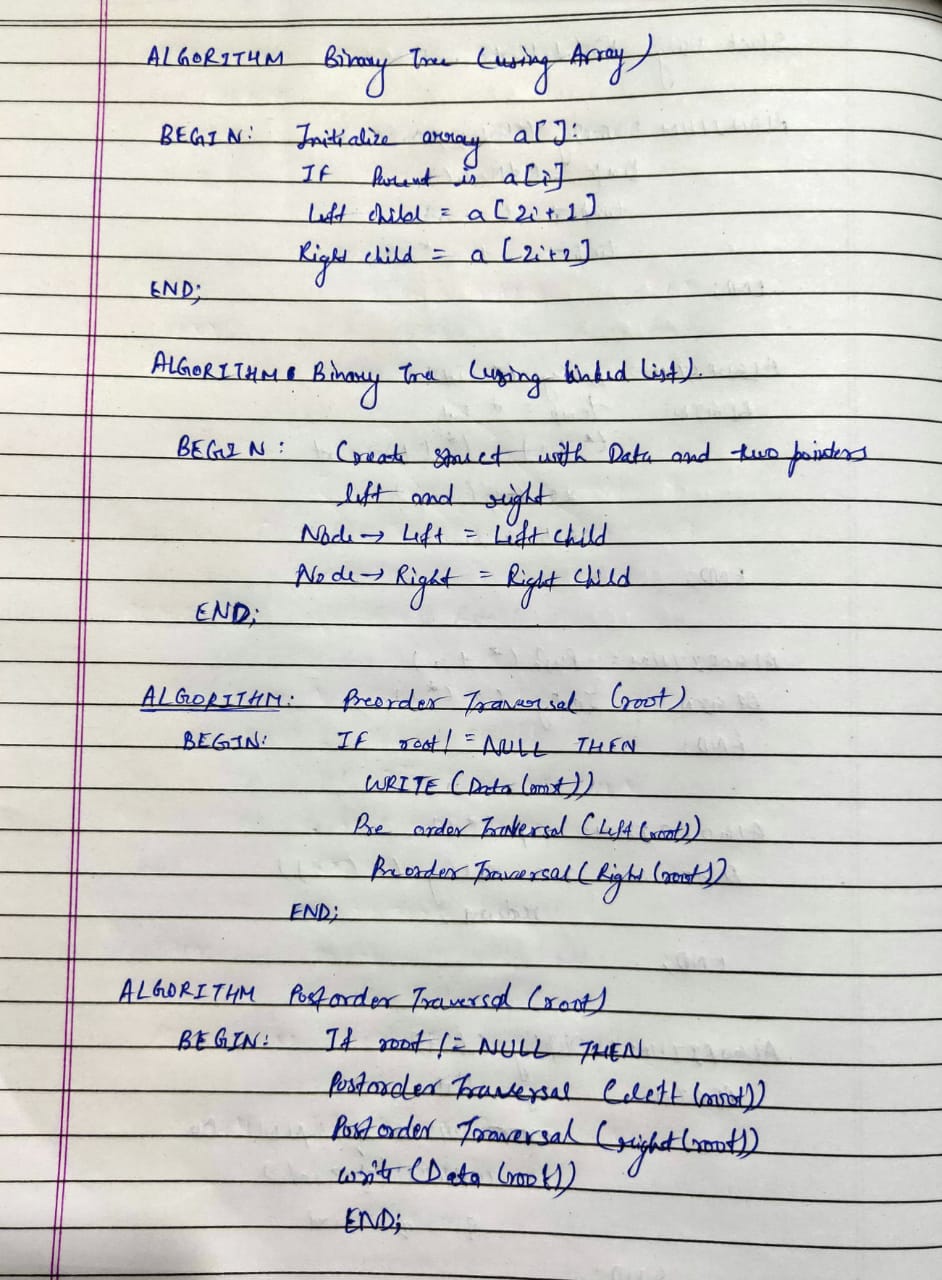


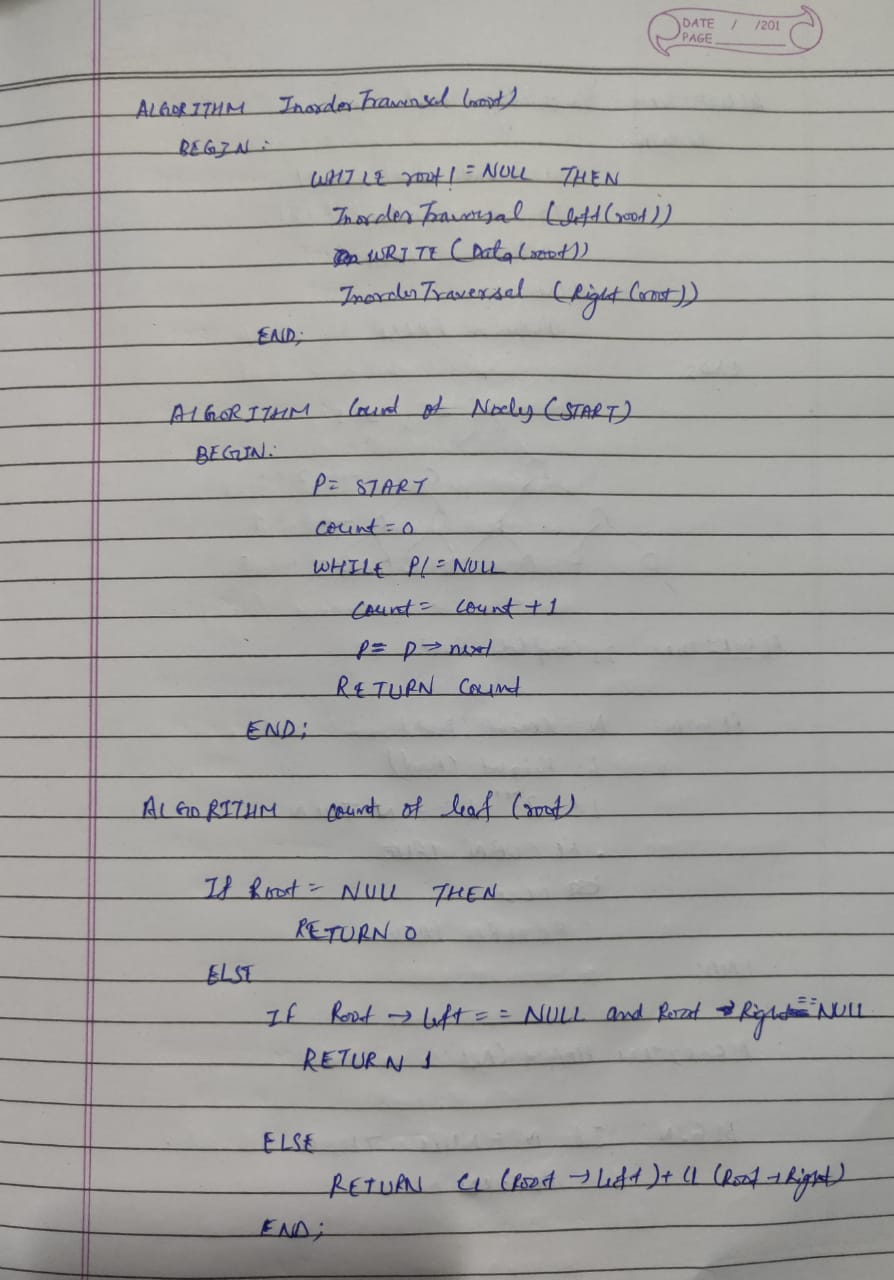
**EXPERIMENT NUMBER 10**

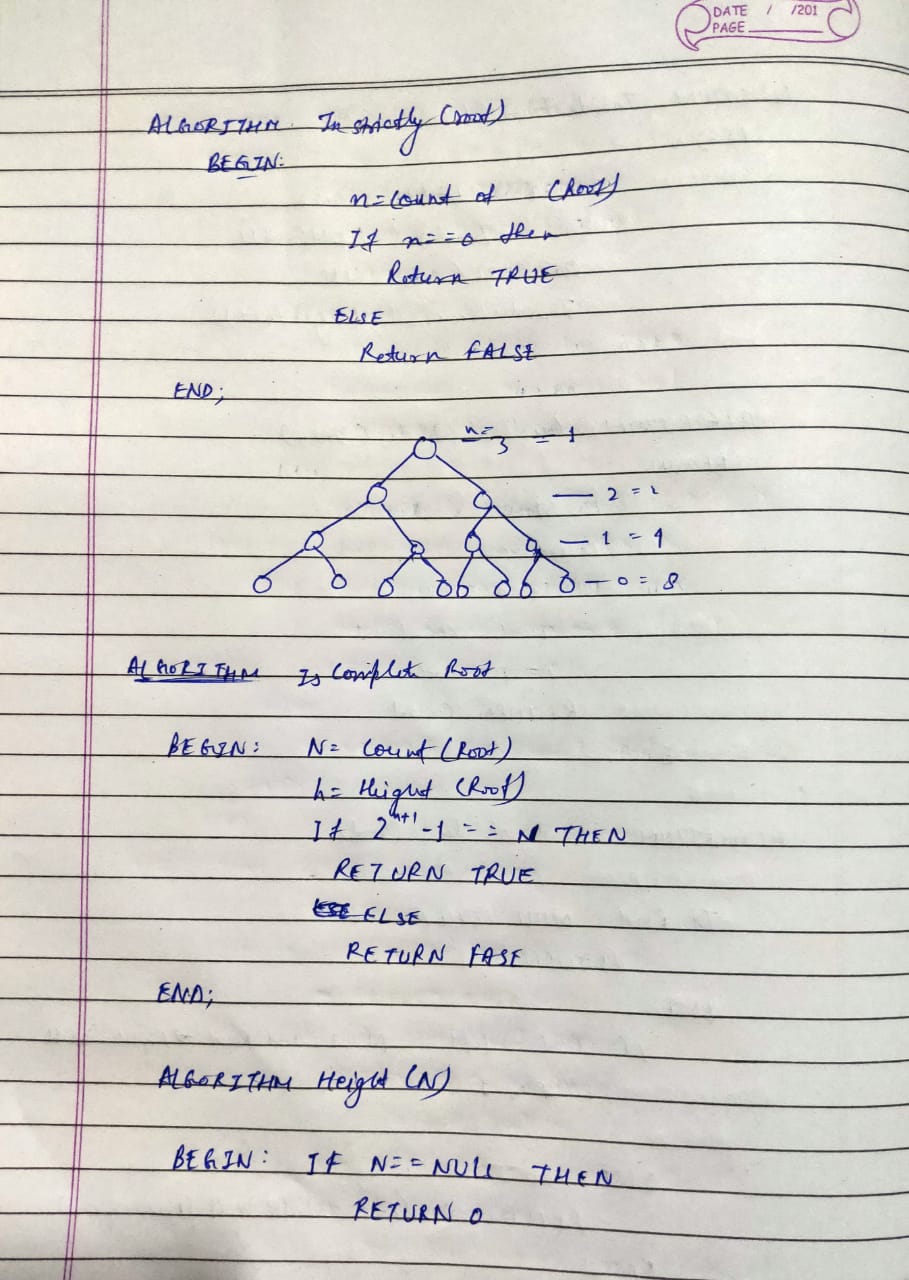
**Implementation of Tree Structures, Binary Tree, Tree Traversal, Binary Search Tree, Insertion and Deletion of BST.**

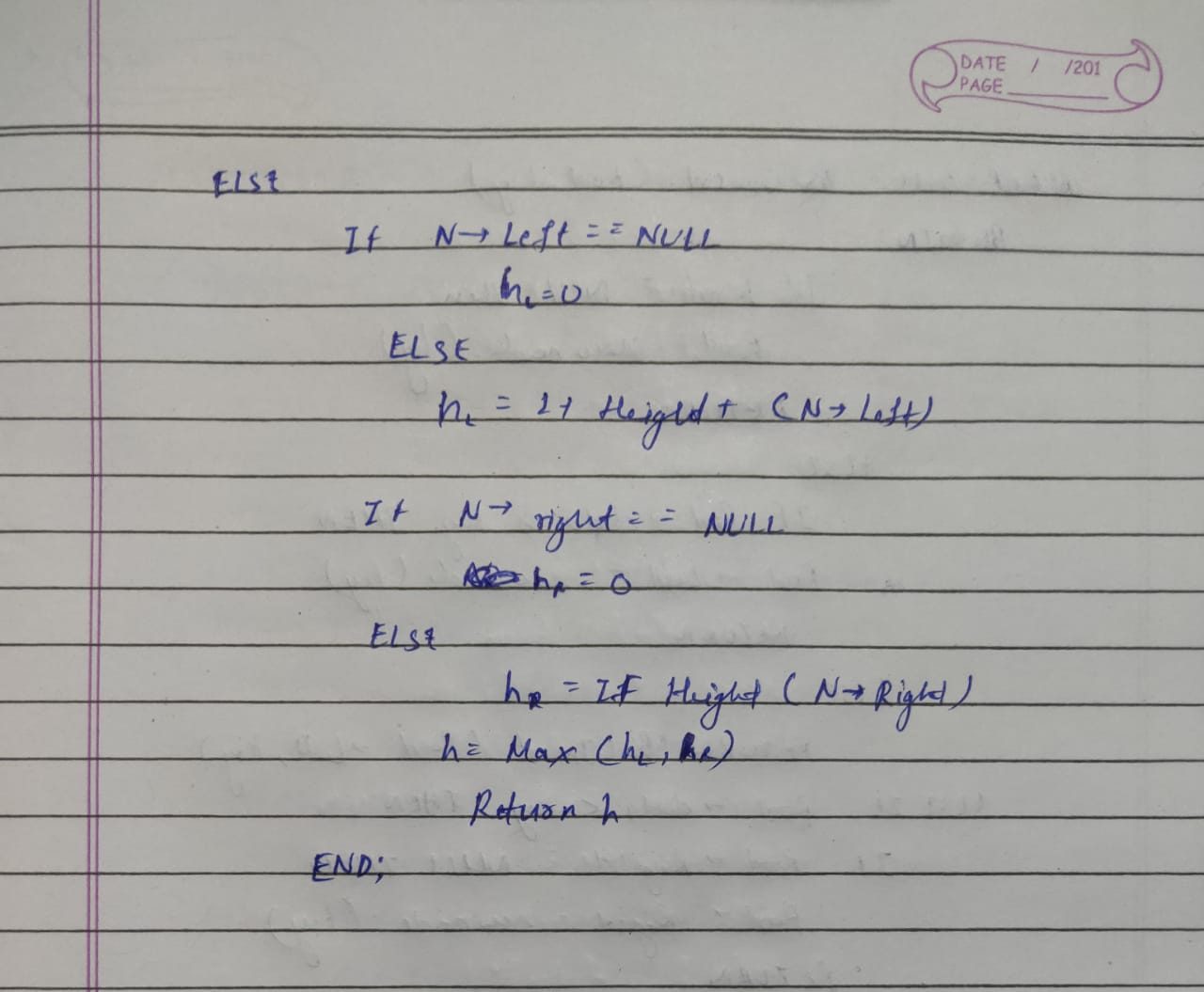
**TITLE OF EXPERIMENT: Binary Tree Primitive Operations**

**ALGORITHM:**









**For All Traversal –**

**Time Complexity:**  O (N)

**Space Complexity:** O (1)

**For count –**

**Time Complexity:**  O (N)

**Space Complexity:** O (1)

**For count of leaf –**

**Time Complexity:**  O (N)

**Space Complexity:** O (1)

**For Height –**

**Time Complexity:**  O (N)

**Space Complexity:** O (N)

**For Count of degree (1) and degree (2) –**

**Time Complexity:**  O (N)

**Space Complexity:** O (N)

**For isstrictly –**

**Time Complexity:**  O (N)

**Space Complexity:** O (1)

**For iscomplete –**

**Time Complexity:**  O (N)

**Space Complexity:** O (1)

**C CODE:**

/\*\*\*\*\*PROGRAM FOR BINARY TREE PRIMITIVE OPERATION\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*HARSH MOHAN, ADMISSION NO. 2019B101166\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node

{

int data;

struct node \*left;

struct node \*right;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node \*Makenode(int x)

{

struct node \*p;

p=(struct node\*)malloc(sizeof(struct node));

p->data=x;

p->left=NULL;

p->right=NULL;

return p;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void preorder(struct node \*root)

{

if(root!=NULL)

{

printf("%d ",root->data);

preorder(root->left);

preorder(root->right);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void inorder(struct node \*root)

{

if(root!=NULL)

{

inorder(root->left);

printf("%d ",root->data);

inorder(root->right);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void postorder(struct node \*root)

{

if(root!=NULL)

{

postorder(root->left);

postorder(root->right);

printf("%d ",root->data);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int count(struct node \*root)

{

if(root==NULL)

return 0;

else

return 1+count(root->left)+count(root->right);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int countleaf(struct node \*root)

{

if(root==NULL)

return 0;

else

{

if(root->left==NULL && root->right==NULL)

return 1;

else

return countleaf(root->left)+countleaf(root->right);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int max(int n1,int n2)

{

return (n1>n2)?n1:n2;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int height(struct node \*root)

{

if(root==NULL)

return 0;

else

{

if(root->left==NULL && root->right==NULL)

return 0;

else

return 1+max(height(root->left),height(root->right));

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int countd1(struct node \*root)

{

if (root==NULL)

return 0;

else

{

if(root->left==NULL && root->right==NULL)

return 0;

else

{

if(root->left!=NULL && root->right!=NULL)

return countd1(root->left)+countd1(root->right);

else

return 1+countd1(root->left)+countd1(root->right);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int countd2(struct node \*root)

{

if(root==NULL)

return 0;

else

{

if(root->left!=NULL && root->right!=NULL)

return 1+countd2(root->left)+countd2(root->right);

else

return countd2(root->left)+countd2(root->right);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int isstrictly(struct node \*root)

{

int n=countd1(root);

if(n==0)

return 1;

else

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int complete(struct node \*root)

{

int n=count(root);

int h=height(root);

if(pow(2,h+1)-1==n)

return 1;

else

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int CreateTree(struct node \*\*tree)

{

int x,choice;

struct node \*p;

printf("Left of %d exists (1/0)",(\*tree)->data);

scanf("%d",&choice);

if(choice==1)

{

printf("Input information of left node");

scanf("%d",&x);

p=Makenode(x);

(\*tree)->left=p;

CreateTree(&p);

}

printf("Right of %d exists (1/0)",(\*tree)->data);

scanf("%d",&choice);

if(choice==1)

{

printf("Input information of right node");

scanf("%d",&x);

p=Makenode(x);

(\*tree)->right=p;

CreateTree(&p);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{

int x;

struct node \*root;

root=NULL;

printf("Input the information of root node");

scanf("%d",&x);

root=Makenode(x);

CreateTree(&root);

printf("Pre Order Traversal is : ");

preorder(root);

printf("\nIn Order Traversal is : ");

inorder(root);

printf("\nPost Order Traversal is : ");

postorder(root);

printf("\nCount of nodes are : ");

int c=count(root);

printf("%d",c);

printf("\nCount of leaf nodes are : ");

int l=countleaf(root);

printf("%d",l);

printf("\nHeight of tree is :");

int h=height(root);

printf("%d",h);

printf("\nCount of degree(1) node is : ");

int d1=countd1(root);

printf("%d",d1);

printf("\nCount of degree(2) node is : ");

int d2=countd2(root);

printf("%d",d2);

printf("\nGiven tree is Strictly Binary Tree or not:");

int s=isstrictly(root);

if(s==1)

printf(" Strictly Binary Tree");

else

printf(" Not Strictly Binary Tree");

printf("\nGiven tree is complete or not:");

int n=complete(root);

if(n==1)

printf(" Complete Tree");

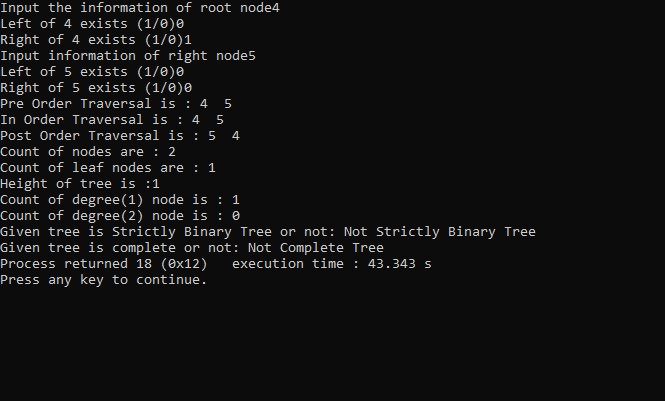
else

printf(" Not Complete Tree");

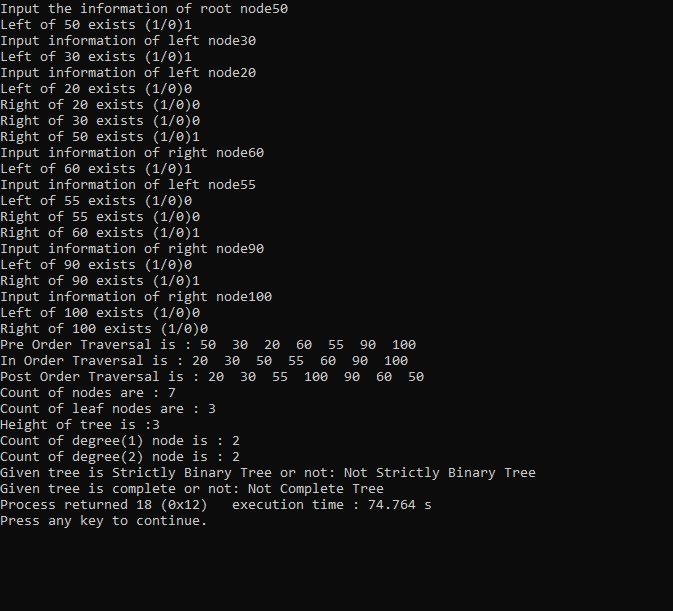
}

**OUTPUT: (3 Sample Input-Outputs)**

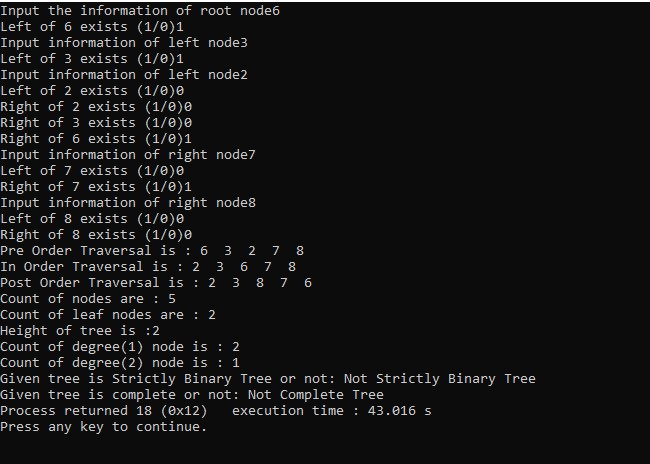
**First Sample Input-Output**



**Second Sample Input-Output**

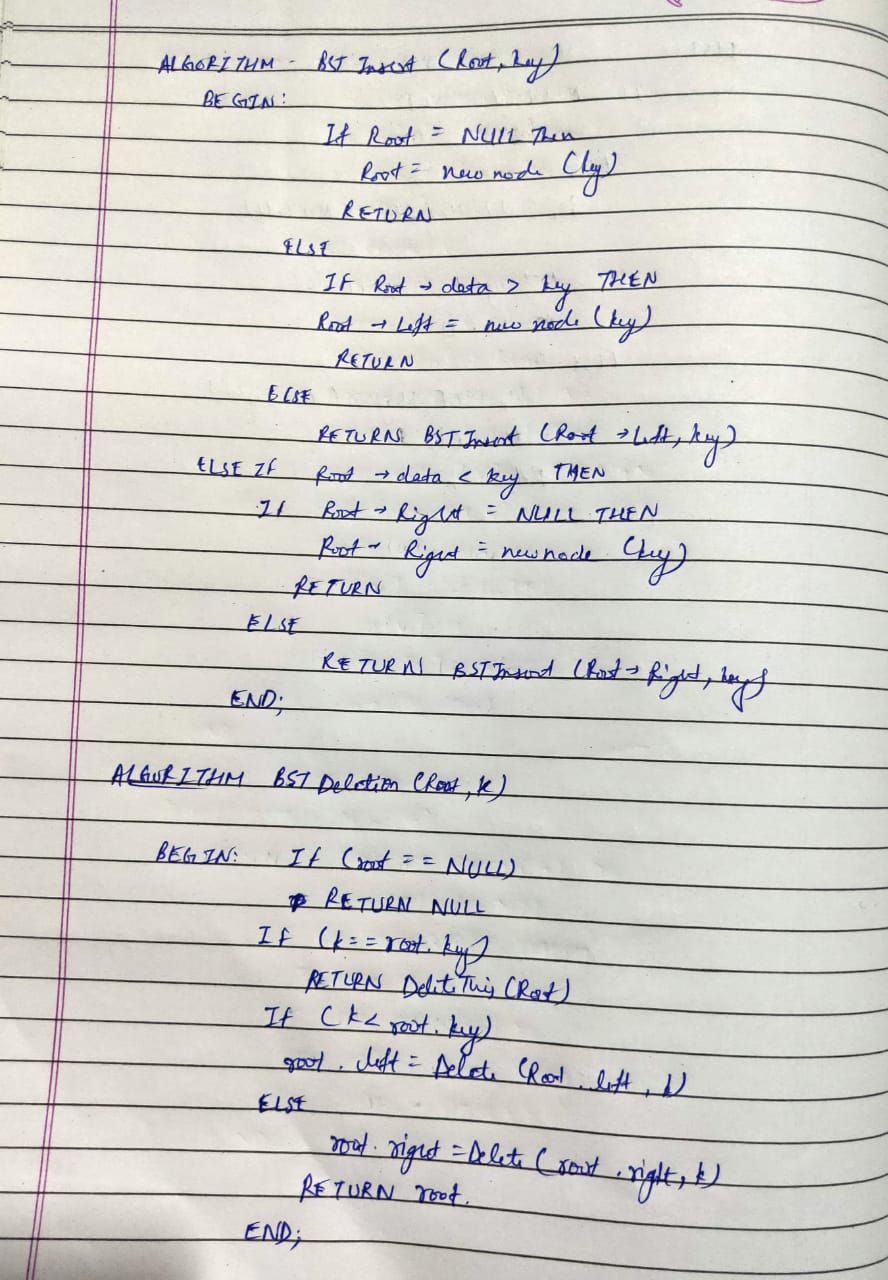


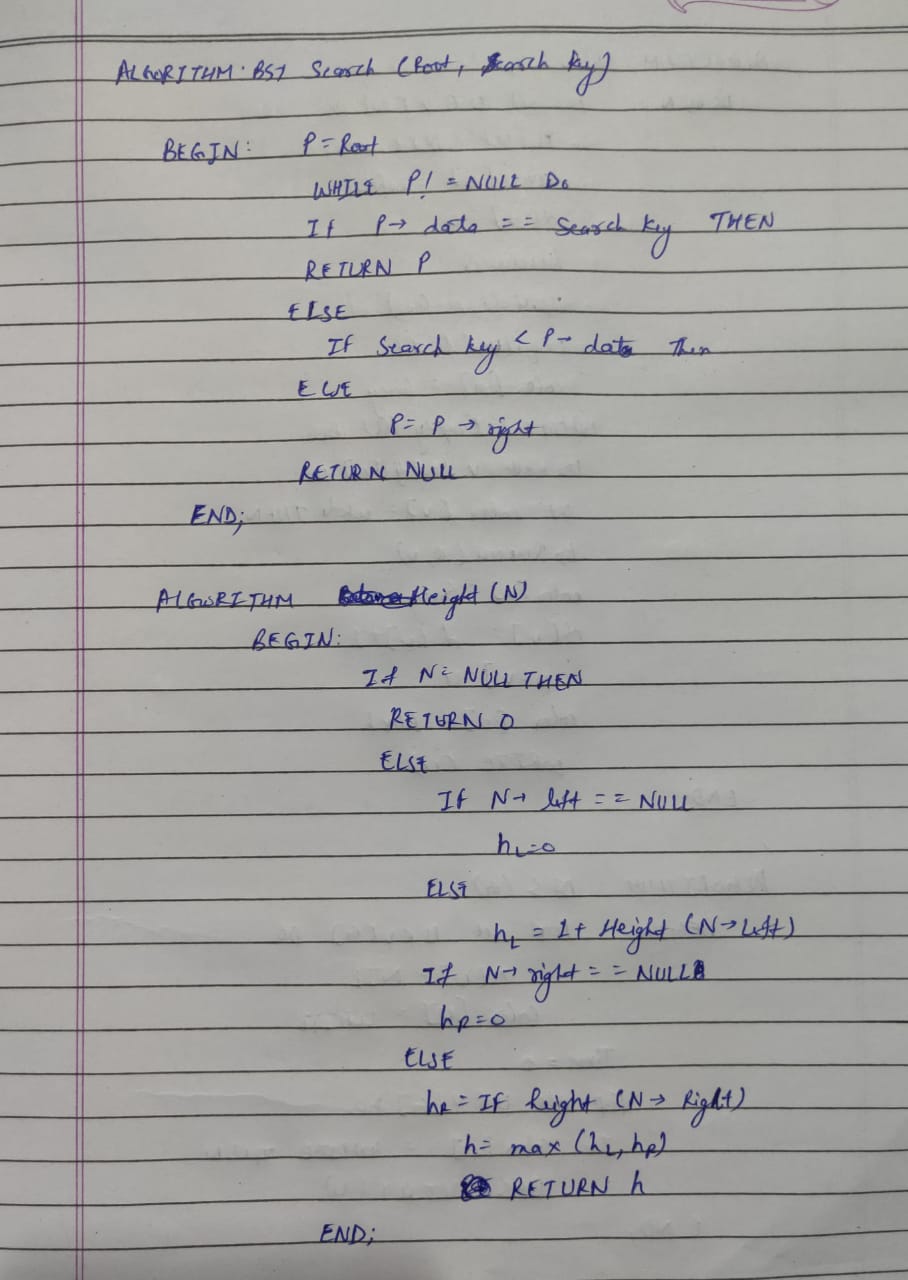
**Third Sample Input-Output**



**TITLE OF EXPERIMENT: Binary Search Tree Primitive Operations**

**ALGORITHM:**





**For All Cases –**

**Time Complexity:**  O (log N)

**Space Complexity:** O (1)

**C CODE:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PROGRAM FOR BINARY SEARCH TREE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*AUTHOR: HARSH MOHAN, ADMISSION NO.: 2019B101166\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<stdlib.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node

{

int data;

struct node \*left;

struct node \*right;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Makenode(int x)

{

struct node \*p;

p=(struct node\*)malloc(sizeof(struct node));

p->data=x;

p->left=NULL;

p->right=NULL;

return p;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node \*BSTinsert(struct node \*T,int key)

{

if(T==NULL)

T=Makenode(key);

else

{

if(key<T->data)

T->left=BSTinsert(T->left,key);

else

T->right=BSTinsert(T->right,key);

}

return T;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int height(struct node \*t)

{

int lh,rh;

if(t==NULL)

return 0;

else

{

if(t->left==NULL)

lh=0;

else

lh=1+height(t->left);

if(t->right==NULL)

rh=0;

else

rh=1+height(t->right);

return (lh>rh)?lh:rh;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int bf(struct node \*t)

{

int lh,rh;

if(t==NULL)

return 0;

else

{

if(t->left==NULL)

lh=0;

else

lh=1+height(t->left);

if(t->right==NULL)

rh=0;

else

rh=1+height(t->right);

return lh-rh;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void InorderTraversal(struct node \*t)

{

if(t!=NULL)

{

InorderTraversal(t->left);

printf("\n%d",t->data);

printf("\t%d",height(t));

printf("\t%d",bf(t));

InorderTraversal(t->right);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node \*BSTmin(struct node \*p)

{

while(p->left!=NULL)

p=p->left;

return p;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node \*BSTmax(struct node \*p)

{

while(p->right!=NULL)

p=p->right;

return p;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node \*BSTsearch(struct node \*t,int searchkey)

{

while(t!=NULL)

{

if(t->data==searchkey)

return t;

else

{

if(searchkey<t->data)

t=t->left;

else

t=t->right;

}

}

return NULL;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct node \*Delete(struct node \*t,int x)

{

if(t==NULL)

return t;

else if(x<t->data)

t->left=Delete(t->left,x);

else if(x>t->data)

t->right=Delete(t->right,x);

else

{

//CASE 1: No Child

if(t->left==NULL && t->right==NULL)

{

free(t);

t=NULL;

}

//CASE 2: One Child

else if(t->left==NULL)

{

struct node \*temp=t;

t=t->right;

free(temp);

}

else if(t->right==NULL)

{

struct node \*temp=t;

t=t->left;

free(temp);

}

//CASE 3: Two Children

else

{

struct node \*temp=BSTmin(t->right);

t->data=temp->data;

t->right=Delete(t->right,temp->data);

}

}

return t;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int CreateTree(struct node \*\*tree)

{

int x,choice;

struct node \*p;

printf("Left of %d exists (1/0)",(\*tree)->data);

scanf("%d",&choice);

if(choice==1)

{

printf("Input information of left node");

scanf("%d",&x);

p=Makenode(x);

(\*tree)->left=p;

CreateTree(&p);

}

printf("Right of %d exists (1/0)",(\*tree)->data);

scanf("%d",&choice);

if(choice==1)

{

printf("Input information of right node");

scanf("%d",&x);

p=Makenode(x);

(\*tree)->right=p;

CreateTree(&p);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{

int x,d,b;

struct node \*root;

struct node \*p;

root=NULL;

int choice;

int C;

printf("Please select any option Given Below for Binary Search Tree : \n");

while(1)

{

printf("\n\n1. Create Tree\n2. InOrder Traversal\n3. BST Minimum \n4. BST Maximum\n5. BST Search\n6. Delete A node\n7. Exit a Program.");

printf("\nEnter your Choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1: printf("Input the information of root node : ");

scanf("%d",&x);

root=Makenode(x);

CreateTree(&root);

break;

case 2: printf("\nIn order Traversal is : ");

printf("\nValue\tHeight\tBalanceFactor");

InorderTraversal(root);

break;

case 3: p=BSTmin(root);

printf("\nMinimum key is : %d",p->data);

break;

case 4: p=BSTmax(root);

printf("\nMaximum key is : %d",p->data);

break;

case 5: printf("Enter the Element which you want to Search : ");

scanf("%d",&b);

p=BSTsearch(root,b);

printf("\nAddress is :%u",p);

if(p!=NULL)

printf("\nElement found at address is : %d",p->data);

else

printf("\nElement not found");

break;

case 6: printf("Enter the Element which you want to Delete : ");

scanf("%d",&d);

printf("Deleted Element is: %d",d);

Delete(root,d);

break;

case 7: return 0;

}

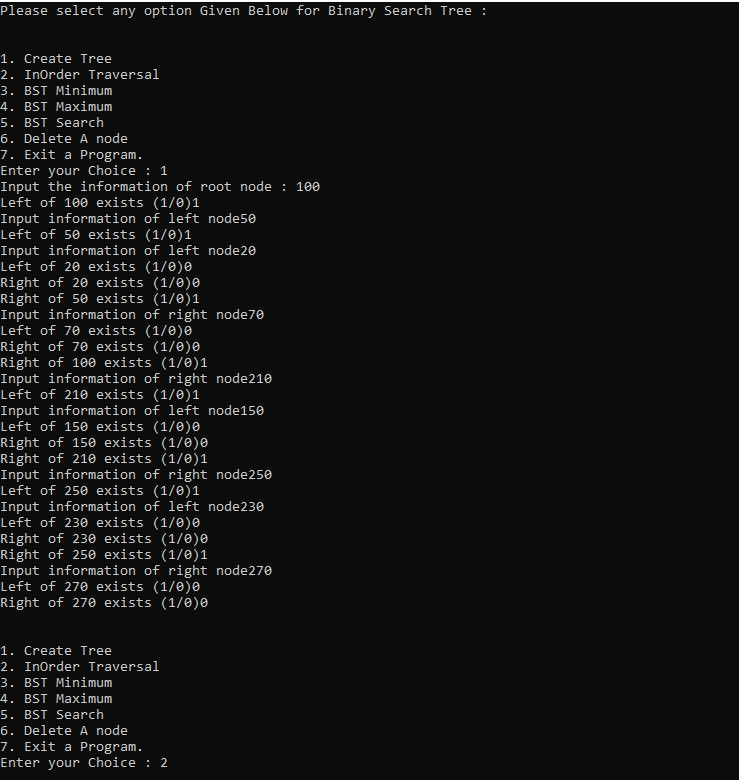
}

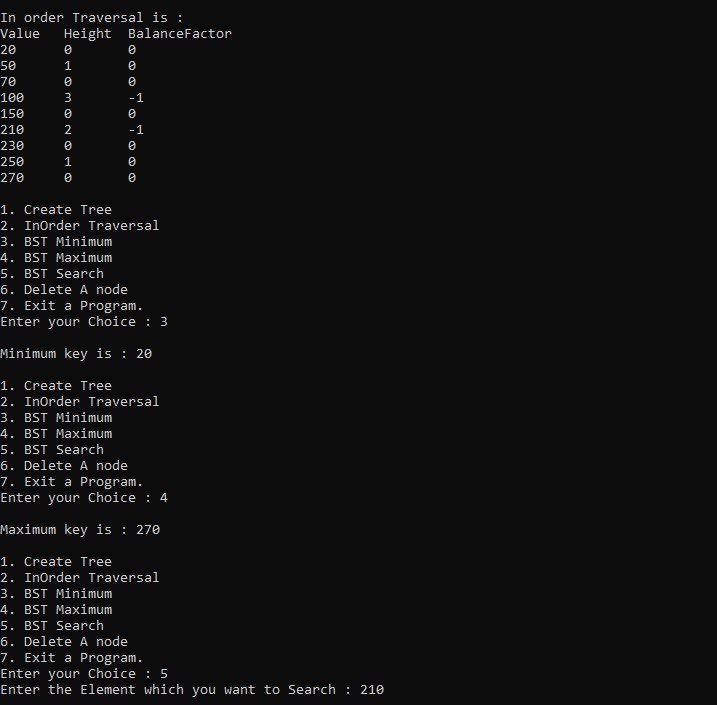
}

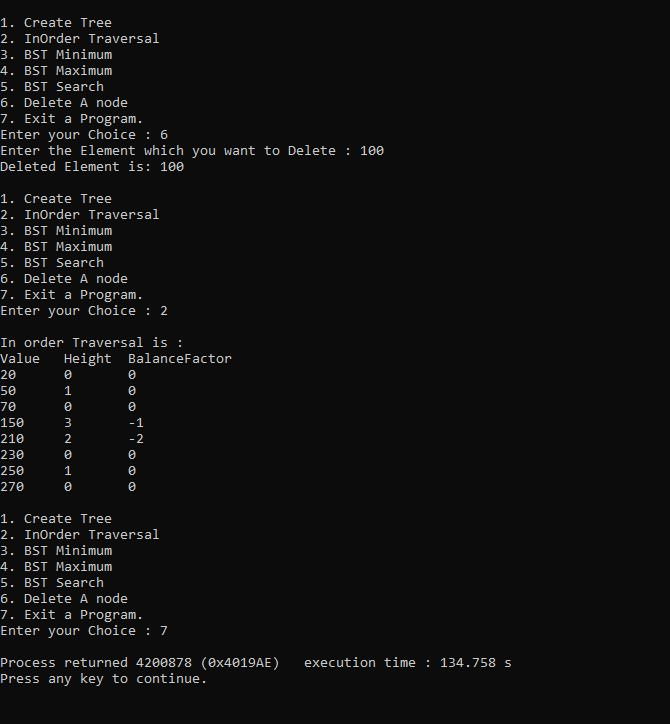
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**OUTPUT: (3 Sample Input-Outputs)**

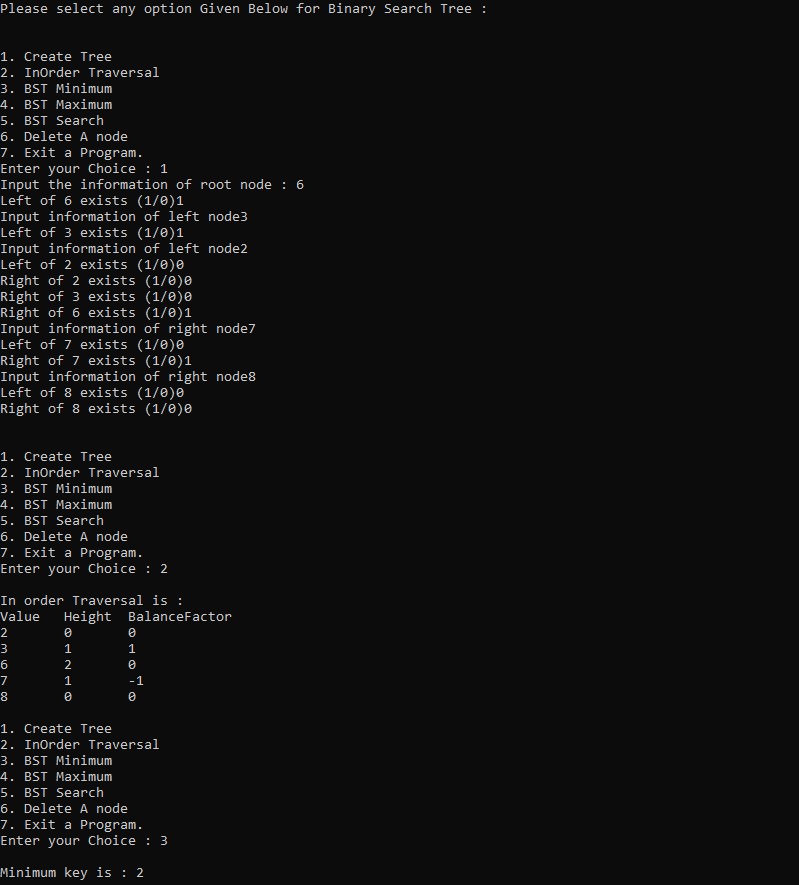
**First Sample Input-Output**

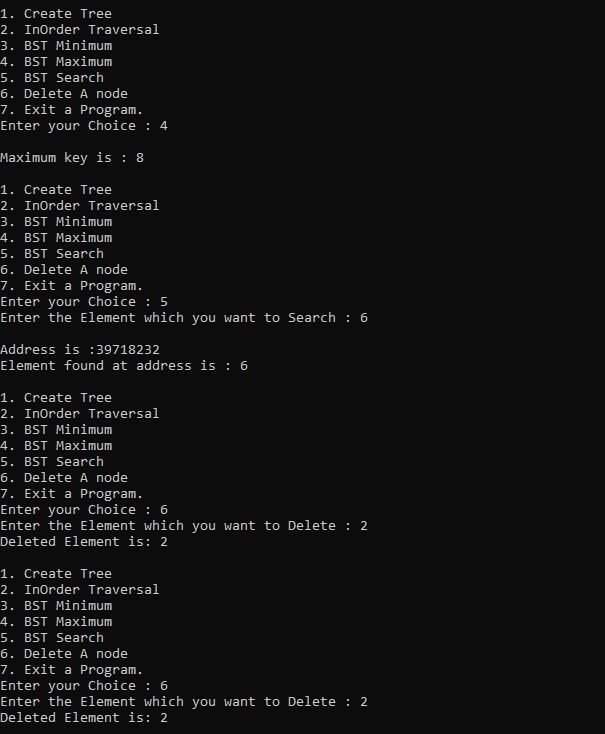


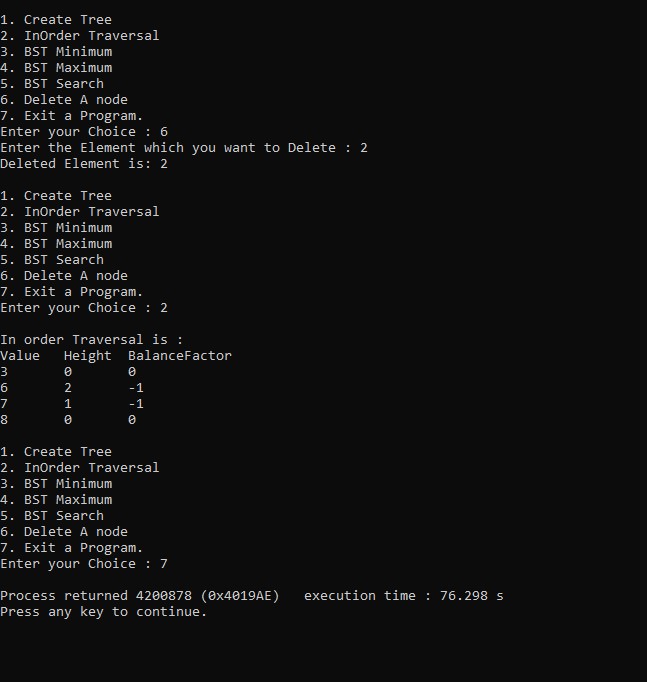




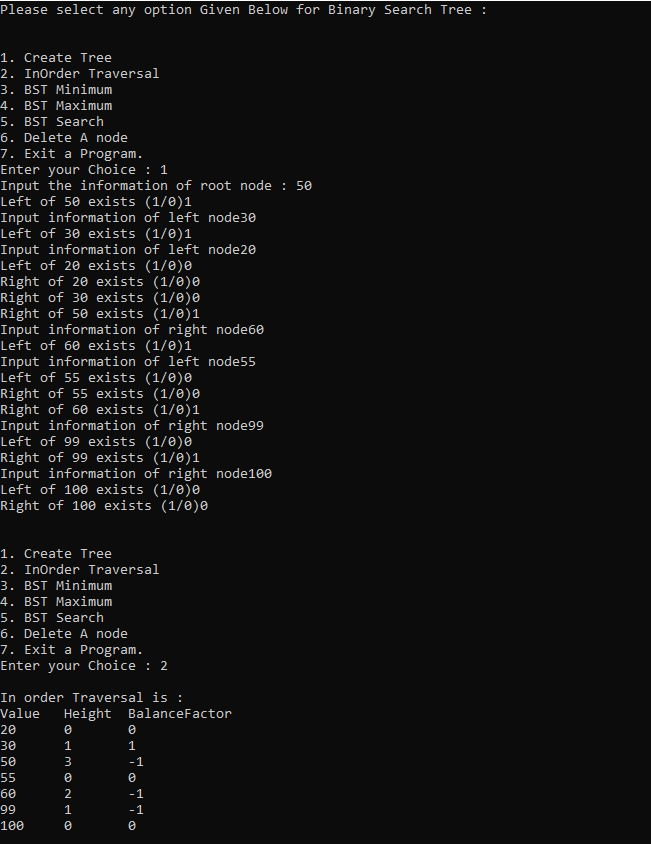
**Second Sample Input-Output**

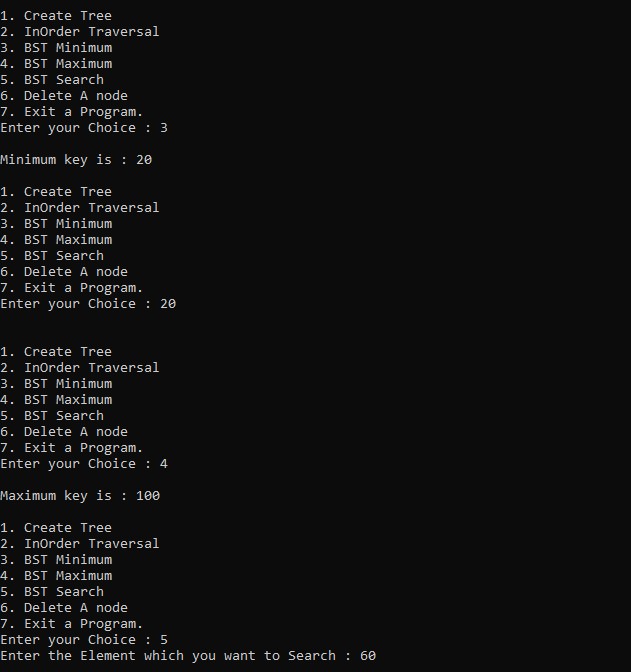


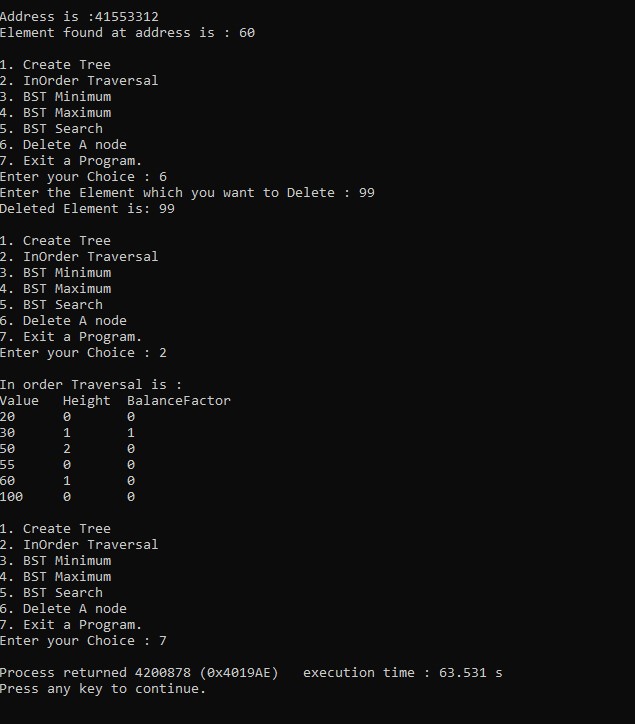




**Third Sample Input-Output**





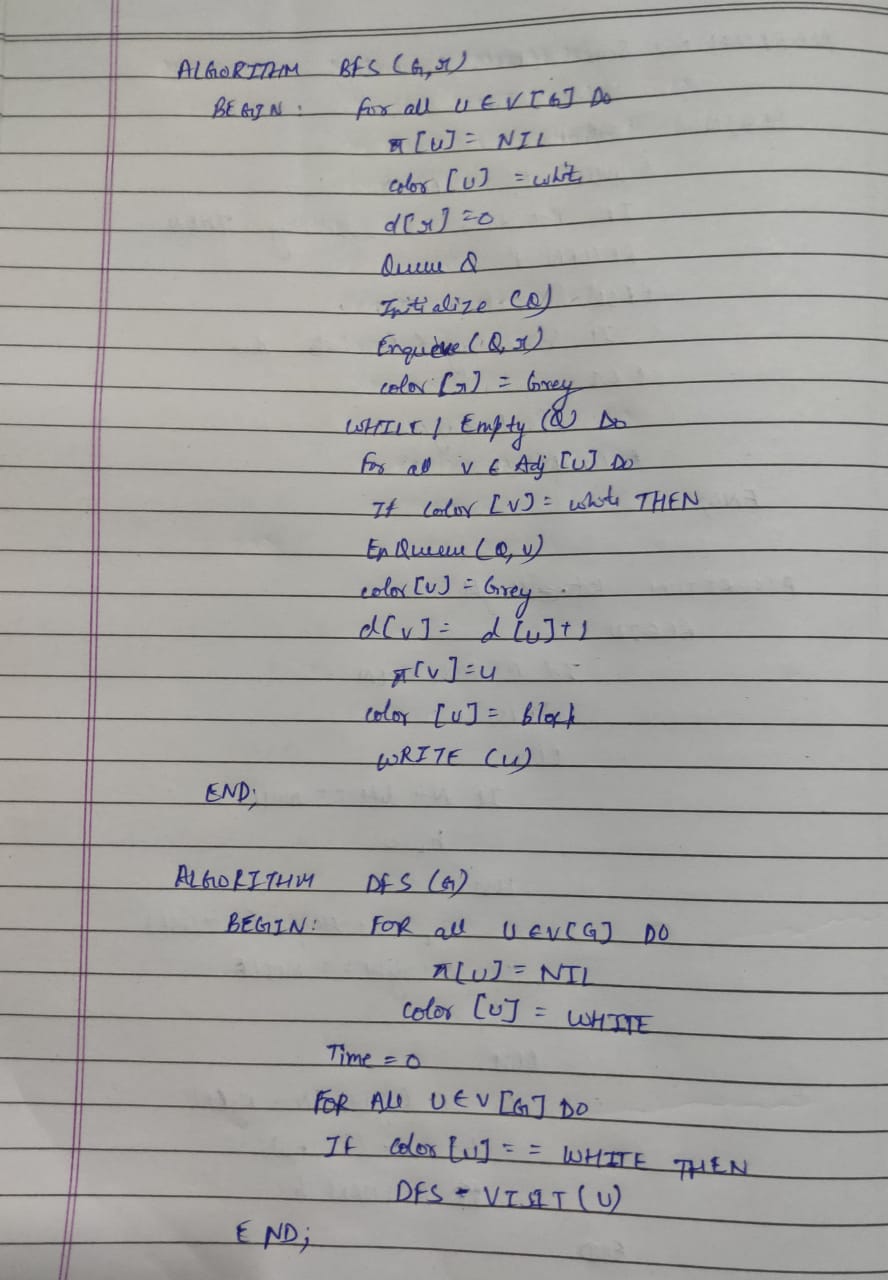


**EXPERIMENT NUMBER 11**

**Graph Implementation, BFS, DFS, Minimum cost spanning Tree, Shortest Path Algorithm.**

**TITLE OF EXPERIMENT:**

**ALGORITHM:**



**C CODE:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PROGRAM FOR BFS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* HARSH MOHAN, ADMISSION NO. : 2019B101166 \*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#define SIZE 40

struct queue {

int items[SIZE];

int front;

int rear;

};

struct queue\* createQueue();

void enqueue(struct queue\* q, int);

int dequeue(struct queue\* q);

void display(struct queue\* q);

int isEmpty(struct queue\* q);

void printQueue(struct queue\* q);

struct node {

int vertex;

struct node\* next;

};

struct node\* createNode(int);

struct Graph {

int numVertices;

struct node\*\* adjLists;

int\* visited;

};

void bfs(struct Graph\* graph, int startVertex) {

struct queue\* q = createQueue();

graph->visited[startVertex] = 1;

enqueue(q, startVertex);

while (!isEmpty(q)) {

printQueue(q);

int currentVertex = dequeue(q);

printf("Visited %d\n", currentVertex);

struct node\* temp = graph->adjLists[currentVertex];

while (temp) {

int adjVertex = temp->vertex;

if (graph->visited[adjVertex] == 0) {

graph->visited[adjVertex] = 1;

enqueue(q, adjVertex);

}

temp = temp->next;

}

}

}

struct node\* createNode(int v) {

struct node\* newNode = malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices) {

struct Graph\* graph = malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = malloc(vertices \* sizeof(struct node\*));

graph->visited = malloc(vertices \* sizeof(int));

int i;

for (i = 0; i < vertices; i++) {

graph->adjLists[i] = NULL;

graph->visited[i] = 0;

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest) {

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

struct queue\* createQueue() {

struct queue\* q = malloc(sizeof(struct queue));

q->front = -1;

q->rear = -1;

return q;

}

int isEmpty(struct queue\* q) {

if (q->rear == -1)

return 1;

else

return 0;

}

void enqueue(struct queue\* q, int value) {

if (q->rear == SIZE - 1)

printf("\nQueue is Full!!");

else {

if (q->front == -1)

q->front = 0;

q->rear++;

q->items[q->rear] = value;

}

}

int dequeue(struct queue\* q) {

int item;

if (isEmpty(q)) {

printf("Queue is empty");

item = -1;

} else {

item = q->items[q->front];

q->front++;

if (q->front > q->rear) {

printf("Resetting queue ");

q->front = q->rear = -1;

}

}

return item;

}

void printQueue(struct queue\* q) {

int i = q->front;

if (isEmpty(q)) {

printf("Queue is empty");

} else {

printf("\nQueue contains \n");

for (i = q->front; i < q->rear + 1; i++) {

printf("%d ", q->items[i]);

}

}

}

int main() {

struct Graph\* graph = createGraph(6);

addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 2);

addEdge(graph, 1, 4);

addEdge(graph, 1, 3);

addEdge(graph, 2, 4);

addEdge(graph, 3, 4);

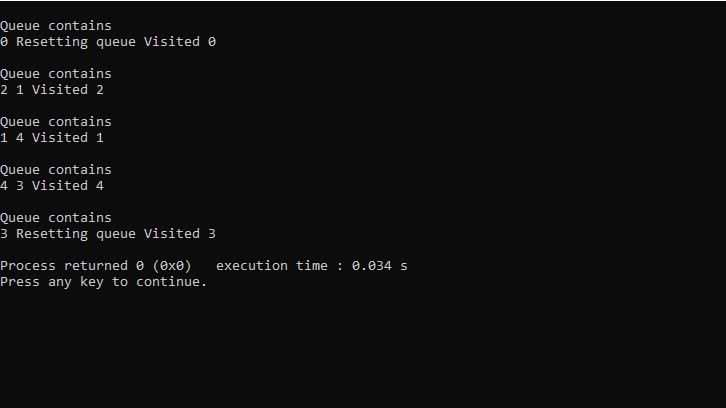
bfs(graph, 0);

return 0;

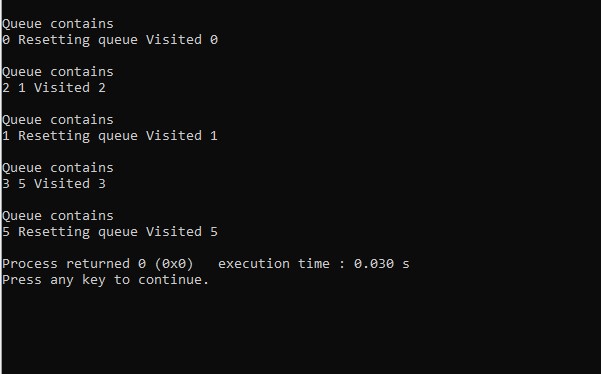
}

**OUTPUT: (3 Sample Input-Outputs)**

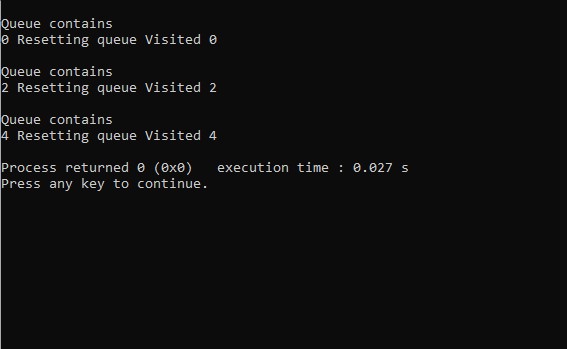
**First Sample Input-Output**



**Second Sample Input-Output**



**Third Sample Input-Output**



**C CODE:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PROGRAM FOR DFS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* HARSH MOHAN, ADMISSION NO. : 2019B101166 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

struct node {

int vertex;

struct node\* next;

};

struct node\* createNode(int v);

struct Graph {

int numVertices;

int\* visited;

struct node\*\* adjLists;

};

void DFS(struct Graph\* graph, int vertex) {

struct node\* adjList = graph->adjLists[vertex];

struct node\* temp = adjList;

graph->visited[vertex] = 1;

printf("Visited %d \n", vertex);

while (temp != NULL) {

int connectedVertex = temp->vertex;

if (graph->visited[connectedVertex] == 0) {

DFS(graph, connectedVertex);

}

temp = temp->next;

}

}

struct node\* createNode(int v) {

struct node\* newNode = malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices) {

struct Graph\* graph = malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = malloc(vertices \* sizeof(struct node\*));

graph->visited = malloc(vertices \* sizeof(int));

int i;

for (i = 0; i < vertices; i++) {

graph->adjLists[i] = NULL;

graph->visited[i] = 0;

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest) {

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

void printGraph(struct Graph\* graph) {

int v;

for (v = 0; v < graph->numVertices; v++) {

struct node\* temp = graph->adjLists[v];

printf("\n Adjacency list of vertex %d\n ", v);

while (temp) {

printf("%d -> ", temp->vertex);

temp = temp->next;

}

printf("\n");

}

}

int main() {

struct Graph\* graph = createGraph(4);

addEdge(graph, 0, 1);

addEdge(graph, 1, 1);

addEdge(graph, 1, 2);

addEdge(graph, 1, 4);

printGraph(graph);

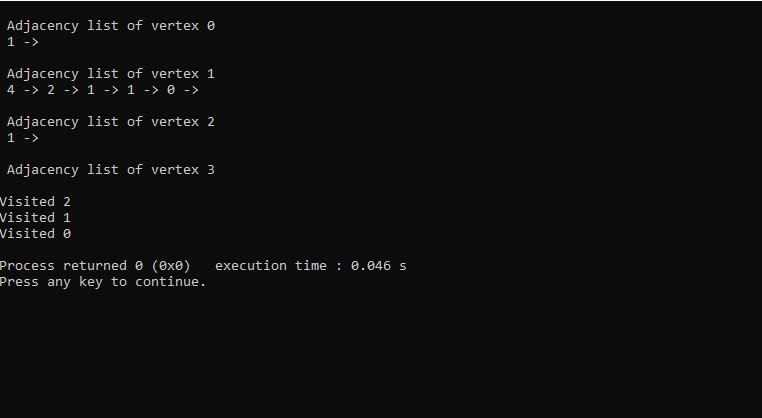
DFS(graph, 2);

return 0;

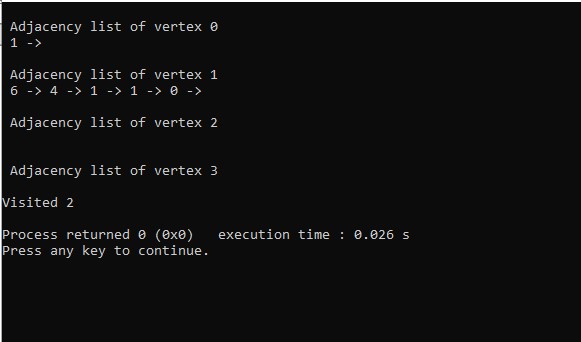
}

**OUTPUT: (3 Sample Input-Outputs)**

**First Sample Input-Output**



**Second Sample Input-Output**



**Third Sample Input-Output**

