BINARY TREE

PROGRAM CODE:

```
//SANIN MOHAMMED N
//B21CSB55
#include<stdio.h>
#include<stdlib.h>
struct NODE * loc;
int found=0;
struct NODE{
int DATA;
struct NODE *l;
struct NODE *r;
};
struct NODE * GetNode(){
struct NODE* new = (struct NODE*) malloc (sizeof(struct NODE));
return new;
}
void BuildTree(struct NODE* ptr){
if(ptr!=NULL){
int c;
printf("\nENTER ELEMENT TO INSERT : ");
scanf("%d",&(ptr->DATA));
printf("\n %d AS LEFT CHILD (1/0)? ",ptr->DATA);
scanf("%d",&c);
if(c==1){
struct NODE * left = GetNode();
ptr->l = left;
BuildTree(left);
}
else{
ptr->l=NULL;
}
printf("\n%d HAS RIGHT CHILD (1/0)?",ptr->DATA);
scanf("%d",&c);
if(c==1){
struct NODE * right = GetNode();
ptr->r = right;
BuildTree(right);
```

```
}
else{
ptr->r=NULL;
}
}
}
void Inorder(struct NODE* ptr){
if (ptr!=NULL){
Inorder(ptr->l);
printf("%d\t",ptr->DATA);
Inorder(ptr->r);
}
}
void Preorder(struct NODE* ptr){
if (ptr!=NULL){
printf("%d\t",ptr->DATA);
Preorder(ptr->l);
Preorder(ptr->r);
}
}
void Postorder(struct NODE* ptr){
if (ptr!=NULL){
Postorder(ptr->l);
Postorder(ptr->r);
printf("%d\t",ptr->DATA);
}
}
struct NODE* Search(struct NODE * ptr,int item){
if(ptr->DATA != item){
if ((ptr->l!= NULL)&&(found==0)){
loc = Search(ptr->l,item);
if (loc!=NULL){
found=1;
return loc;
}
}
if ((ptr->r != NULL)&&(found==0)){
   loc = Search(ptr->r,item);
```

```
if (loc!=NULL){
found=1;
return loc;
}
}
else {
return NULL;
}
}
else
return ptr;
}
void Insert(struct NODE * root,int key, int item){
found=0;
loc=NULL;
char ch;
struct NODE * ptr = Search(root,key);
if (ptr == NULL){
printf("KEY NOT FOUND\n");
}
else{
printf("INSERT AS LEFT NODE OR RIGHT NODE (L/R) ?");
ch=getchar();
scanf("%c",&ch);
if (ch=='L'){
if (ptr->l==NULL){
struct NODE * new = GetNode();
new->DATA = item;
new->l=NULL;
new->r = NULL;
ptr->l=new;
printf("ELEMENT ADDED");
}
else{
printf("CANNOT BE INSERTED AS IT IS LEAF NODE\n");
}
}
else{
```

```
if (ptr->r==NULL){
struct NODE * new = GetNode();
new->DATA = item;
new->l=NULL;
new->r = NULL;
ptr->r=new;
printf("ELEMENT ADDED");
}
else{
printf("CANNOT INSERT AS LEAF NODE\n");
}
}
}
}
struct NODE * SearchParent(struct NODE * ptr, int key)
{
if (ptr->DATA!=key){
if((ptr->l!=NULL)&&(found==0)){
if(ptr->l->DATA == key){
found=1;
return ptr;
}
else{
loc=SearchParent(ptr->l,key);
if (loc!=NULL){
found=1;
return loc;
}
}
}
if((ptr->r!=NULL)&&(found==0))
{
if(ptr->r->DATA == key){
found=1;
return ptr;
}
else{
loc=SearchParent(ptr->r,key);
```

```
if (loc!=NULL){
found=1;
return loc;
}
}
}
else
return NULL;
}
else{
return ptr;
}
}
void Delete(struct NODE * ptr,int key){
loc=NULL;
found=0;
struct NODE * locp = SearchParent(ptr,key);
if (locp==NULL)
{
printf("ELEMENT NOT FOUND\n");
}
else{
if(locp->l!=NULL){
if(locp->l->DATA == key){
if((locp->l->l==NULL)&&(locp->l->r == NULL)){
locp->l=NULL;
printf("ELEMENT DELETED");
}
else{
printf("CANNOT DELETE ELEMENT\n");
}
}
}
if(locp->r!=NULL){
if (locp->r->DATA == key){
if((locp->r->l==NULL)&&(locp->r->r == NULL)){
locp->r=NULL;
printf("ELEMENT DELETED");
```

```
}
else{
printf("CANNOT DELETE ELEMENT\n");
}
}
}
}
}
void main(){
struct NODE * ROOT = GetNode();
BuildTree(ROOT);
int ch ;
do{
printf("\n\nMENU\n****\n1.INSERT\n2.INORDER TRAVERSAL\n3.PREORDER TRAVERSAL\n4.POSTORDER TRAVERSAL\n5.DELETE\n6.EXIT\n\nENTER YOUR CHOICE : ");
scanf("%d",&ch);
if(ch==1){
int key,item;
printf("ENTER PARENT NODE TO SEARCH AND INSERT : ");
scanf("%d",&key);
printf("ENTER DATA IN NODE TO INSERT : ");
scanf("%d",&item);
Insert(ROOT,key,item);
}
else if(ch==2){
Inorder(ROOT);
}
else if(ch==3){
Preorder(ROOT);
}
else if(ch==4){
Postorder(ROOT);
}
else if(ch==5){
int key;
printf("ENTER ELEMENT TO DELETE : ");
scanf("%d",&key);
Delete(ROOT,key);
```

```
}
else if (ch==6){
printf("EXIT\n");
break;
}
else{
printf("ENTER VALID OPTION\n");
}
}while(ch!=6);
}
```

OUTPUT:

```
ENTER ELEMENT TO INSERT : 2
2 AS LEFT CHILD (1/0)? 1
ENTER ELEMENT TO INSERT : 4
4 AS LEFT CHILD (1/0)? 1
ENTER ELEMENT TO INSERT : 6
6 AS LEFT CHILD (1/0)? 0
6 HAS RIGHT CHILD (1/0)?0
4 HAS RIGHT CHILD (1/0)?3
2 HAS RIGHT CHILD (1/0)?0
MENU
****
1.INSERT
2.INORDER TRAVERSAL
3.PREORDER TRAVERSAL
4.POSTORDER TRAVERSAL
5.DELETE
6.EXIT
ENTER YOUR CHOICE : 1
ENTER PARENT NODE TO SEARCH AND INSERT : 2
ENTER DATA IN NODE TO INSERT : 7
INSERT AS LEFT NODE OR RIGHT NODE (L/R) ?R
ELEMENT ADDED
MENU
****
1.INSERT
```

2.INORDER TRAVERSAL
3.PREORDER TRAVERSAL
4.POSTORDER TRAVERSAL
5.DELETE
6.EXIT
ENTER YOUR CHOICE : 2
6 4 2 7
MENU
****
1.INSERT
2.INORDER TRAVERSAL
3.PREORDER TRAVERSAL
4.POSTORDER TRAVERSAL
5.DELETE
6.EXIT
ENTER YOUR CHOICE : 3
2 4 6 7
MENU
****
1.INSERT
2.INORDER TRAVERSAL
3.PREORDER TRAVERSAL
4.POSTORDER TRAVERSAL
5.DELETE
6.EXIT
ENTER YOUR CHOICE : 4
6 4 7 2
MENU
****
1.INSERT
2.INORDER TRAVERSAL
3.PREORDER TRAVERSAL
4.POSTORDER TRAVERSAL
5.DELETE
6.EXIT
ENTER YOUR CHOICE : 5
ENTER ELEMENT TO DELETE : 6
ELEMENT DELETED

MENU

****

1.INSERT

2.INORDER TRAVERSAL

3.PREORDER TRAVERSAL

4.POSTORDER TRAVERSAL

5.DELETE

6.EXIT

ENTER YOUR CHOICE : 6

EXIT

BINARYSEARCH TREE

PROGRAM CODE:

```
//SANIN MOHAMMED N
//B21CSB55
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
struct node
{
int DATA;
struct node* LC;
struct node* RC;
};
void preOrder(struct node* ptr)
{
if(ptr == NULL)
printf(" Tree is empty!");
else
{
printf(" %d", ptr->DATA);
if(ptr->LC != NULL)
preOrder(ptr->LC);
if(ptr->RC != NULL)
preOrder(ptr->RC);
}
}
void inOrder(struct node* ptr)
{
if(ptr == NULL)
printf(" Tree is empty!");
else
{
if(ptr->LC != NULL)
inOrder(ptr->LC);
printf(" %d", ptr->DATA);
if(ptr->RC != NULL)
inOrder(ptr->RC);
```

```
}
}
void postOrder(struct node* ptr)
{
if(ptr == NULL)
printf(" Tree is empty!");
else
{
if(ptr->LC != NULL)
postOrder(ptr->LC);
if(ptr->RC != NULL)
postOrder(ptr->RC);
printf(" %d", ptr->DATA);
}
}
int leafNum(struct node* ptr)
{
int count = 0;
if(ptr == NULL)
return 0;
else
{
if(ptr->LC != NULL)
count += leafNum(ptr->LC);
if(ptr->RC != NULL)
count += leafNum(ptr->RC);
if(ptr->LC == NULL && ptr->RC == NULL)
count++;
}
return count;
}
struct node* succ(struct node* ptr)
{
struct node* ptr1 = ptr->RC;
if(ptr1 != NULL) //No need to check in this program
while(ptr1->LC != NULL)
ptr1 = ptr1->LC;
return(ptr1);
```

```
}
void insertBST(struct node* ptr, int ITEM)
{
struct node* ptr1;
bool flag = false;
while(ptr != NULL && flag == false)
{
if(ITEM < ptr->DATA)
{
ptr1 = ptr;
ptr = ptr->LC;
}
else if(ITEM > ptr->DATA)
{
ptr1 = ptr;
ptr = ptr->RC;
}
else
{
flag = true;
printf("\n%d already exists!\n", ITEM);
}
}
if(ptr == NULL)
{
if(ptr1->DATA < ITEM)
{
ptr1->RC = (struct node*) malloc(sizeof(struct node));
ptr1->RC->LC = NULL;
ptr1->RC->RC = NULL;
ptr1->RC->DATA = ITEM;
printf("\n%d inserted successfully!\n", ITEM);
}
else
{
ptr1->LC = (struct node*) malloc(sizeof(struct node));
ptr1->LC->LC = NULL;
ptr1->LC->RC = NULL;
```

```
ptr1->LC->DATA = ITEM;
printf("\n%d inserted successfully!\n", ITEM);
}
}
}
bool deleteBST(struct node* ROOT, int ITEM)
{
struct node* ptr = ROOT;
bool flag = false;
struct node* parent;
int CASE;
while(ptr != NULL && flag == false)
{
   if(ITEM < ptr->DATA)
{
parent = ptr;
ptr = ptr->LC;
}
else if(ITEM > ptr->DATA)
{
parent = ptr;
ptr = ptr->RC;
}
else
flag = true;
}
if(flag == false)
{
return flag;
}
if(ptr->LC == NULL && ptr->RC == NULL)
CASE = 1;
else
if(ptr->LC != NULL && ptr->RC != NULL)
CASE = 3;
else
CASE = 2;
if(CASE == 1)
```

```
{
if(parent->LC == ptr)
parent->LC = NULL;
else
parent->RC = NULL;
free(ptr);
}
else if(CASE == 2)
{
if(parent->LC == ptr)
if(ptr->LC == NULL)
parent->LC = ptr->RC;
else
parent->LC = ptr->LC;
else
if(ptr->LC == NULL)
parent->RC = ptr->RC;
else
parent->RC = ptr->LC;
free(ptr);
}
else
{
parent = succ(ptr);
ITEM = parent->DATA;
deleteBST(ROOT, parent->DATA);
ptr->DATA = ITEM;
}
return flag;
}
void main()
{
struct node* ROOT = NULL;
struct node* ptr1;
int n;
L:
printf("\nChoose the operation\n\n");
printf("1. Insert a node\n");
```

```
printf("2. Delete a node\n");
printf("3. Inorder traversal\n");
printf("4. Preorder traversal\n");
printf("5. Postorder traversal\n");
printf("6. Count no. of leaf nodes\n");
printf("7. Exit\n");
scanf("%d", &n);
switch(n)
{
case 1:
if(ROOT == NULL)
{
ROOT = (struct node*) malloc(sizeof(struct node));
ROOT->LC = NULL;
ROOT->RC = NULL;
printf("\nEnter data\n");
scanf("%d", &ROOT->DATA);
printf("\n%d inserted successfully!\n", ROOT->DATA);
}
else
{
printf("\nEnter data\n");
scanf("%d", &n);
insertBST(ROOT, n);
}
goto L;
case 2:
if(ROOT == NULL)
printf("\nTree is empty!\n");
else
{
printf("\nEnter the data to be deleted\n");
scanf("%d", &n);
if(n == ROOT->DATA)
{
if(ROOT->LC == NULL && ROOT->RC == NULL)
{
free(ROOT);
```

```
ROOT = NULL;
printf("\n%d deleted successfully!\n", n);
}
else
{
if(ROOT->LC != NULL && ROOT->RC != NULL)
{
ptr1 = succ(ROOT);
int temp = ptr1->DATA;
deleteBST(ROOT, ptr1->DATA);
ROOT->DATA = temp;
printf("\n%d deleted successfully!\n", n);
}
else
{
if(ROOT->LC == NULL)
{
ptr1 = ROOT->RC;
free(ROOT);
ROOT = ptr1;
}
else
{
ptr1 = ROOT->LC;
free(ROOT);
ROOT = ptr1;
}
printf("\n%d deleted successfully!\n", n);
}
}
}
else
{
if(deleteBST(ROOT, n))
printf("\n%d deleted successfully!\n", n);
else
printf("\n%d not found!\n", n);
}
```

```
}
goto L;
case 3:
printf("\nInorder :");
inOrder(ROOT);
printf("\n");
goto L;
case 4:
printf("\nPreorder :");
preOrder(ROOT);
printf("\n");
goto L;
case 5:
printf("\nPostorder :");
postOrder(ROOT);
printf("\n");
goto L;
case 6:
printf("\nNo of leaf nodes : %d\n", leafNum(ROOT));
goto L;
case 7:
exit(0);
default:
printf("Invalid entry!\n");
goto L;
}
}
```

OUTPUT:

```
Choose the operation
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
Enter your choice
1
```

Enter data
3
3 inserted successfully!
Choose the operation
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
Enter your choice
1
Enter data
4
4 inserted successfully!
Choose the operation
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
Enter your choice
1
Enter data
5
5 inserted successfully!
Choose the operation
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
Enter your choice

1
Enter data
12
12 inserted successfully!
Choose the operation
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
Enter your choice
1
Enter data
43
43 inserted successfully!
Choose the operation
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
Enter your choice
1
Enter data
6
6 inserted successfully!
Choose the operation
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit

Enter your choice
1
Enter data
17
17 inserted successfully!
Choose the operation
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
Enter your choice
3
Inorder : 3 4 5 6 12 17 43
Choose the operation
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
Enter your choice
4
Preorder : 3 4 5 12 6 43 17
Choose the operation
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
Enter your choice
5
Postorder : 6 17 43 12 5 4 3

Choose the operation
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
Enter your choice
6
No of leaf nodes : 2
Choose the operation
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
Enter your choice
2
Enter the data to be deleted
17
17 deleted successfully!
Choose the operation
1. Insert a node
2. Delete a node
3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
Enter your choice
3
Inorder : 3 4 5 6 12 43
Choose the operation
1. Insert a node
2. Delete a node

3. Inorder traversal
4. Preorder traversal
5. Postorder traversal
6. Count no. of leaf nodes
7. Exit
Enter your choice
7

LEAFNODES IN BINARYSEARCH TREE

PROGRAM CODE:

```
//SANIN MOHAMMED N
//B21CSB55
#include<stdio.h>
#include<stdlib.h>
int count=0;
struct NODE{
int DATA;
struct NODE *l;
struct NODE *r;
};
struct NODE * GetNode(){
struct NODE* new = (struct NODE*) malloc (sizeof(struct NODE));
return new;
}
void Insert(struct NODE * root, int item){
struct NODE * ptr=root;
struct NODE * ptr1;
int flag = 0;
while((ptr!=NULL)&&(flag==0)){
if(ptr->DATA > item){
ptr1=ptr;
ptr=ptr->l;
}
else if(ptr->DATA ==item){
flag=1;
printf("ELEMENT ALREADY EXISTS");
break;
}
else{
ptr1=ptr;
ptr=ptr->r;
}
}
if(ptr==NULL){
struct NODE * new = GetNode();
```

```
new->DATA = item;
new->l=NULL;
new->r = NULL;
if(ptr1->DATA<item){
ptr1->r = new;
}
else{
ptr1->l = new;
}
printf("ELEMENT ADDED");
}
}
void Count(struct NODE * ptr){
if(ptr!=NULL){
if((ptr->l==NULL)&&(ptr->r==NULL)){
count++;
}
else{
if (ptr->l!=NULL){
Count(ptr->l);
}
if(ptr->r!=NULL){
Count(ptr->r);
}
}
}
}
void main(){
struct NODE * ROOT = GetNode();
printf("ENTER ROOT NODE OF BST : ");
scanf("%d",&(ROOT->DATA));
ROOT->l=NULL;
ROOT->r=NULL;
int ch;
do{
printf("\n\nMENU\n1.INSERT\n2.COUNT LEAF NODE\n3.EXIT\n\nENTER
YOUR CHOICE : ");
scanf("%d",&ch);
```

```
if(ch==1){
int key,item;
printf("ENTER DATA IN NODE TO INSERT : ");
scanf("%d",&item);
Insert(ROOT,item);
}
else if(ch==2){
count=0;
Count(ROOT);
printf("NUMBER OF LEAF NODES : %d\n",count);
}
else if (ch==3){
printf("EXIT.\n");
break;
}
else{
printf("ENTER VALID OPTION\n");
}
}while(ch!=3);
}
```

OUTPUT:

```
ENTER ROOT NODE OF BST : 2
MENU
1.INSERT
2.COUNT LEAF NODE
3.EXIT
ENTER YOUR CHOICE : 1
ENTER DATA IN NODE TO INSERT : 3
ELEMENT ADDED
MENU
1.INSERT
2.COUNT LEAF NODE
3.EXIT
ENTER YOUR CHOICE : 1
ENTER DATA IN NODE TO INSERT : 4
ELEMENT ADDED
MENU
1.INSERT
```

2.COUNT LEAF NODE
3.EXIT
ENTER YOUR CHOICE : 1
ENTER DATA IN NODE TO INSERT : 12
ELEMENT ADDED
MENU
1.INSERT
2.COUNT LEAF NODE
3.EXIT
ENTER YOUR CHOICE : 1
ENTER DATA IN NODE TO INSERT : 45
ELEMENT ADDED
MENU
1.INSERT
2.COUNT LEAF NODE
3.EXIT
ENTER YOUR CHOICE : 2
NUMBER OF LEAF NODES : 1
MENU
1.INSERT
2.COUNT LEAF NODE
3.EXIT
ENTER YOUR CHOICE : 3
EXIT

SORTING USING BINARY TREE

PROGRAM CODE:

```
//SANIN MOHAMMED N
//B21CSB55
#include<stdio.h>
#include<stdlib.h>
int i = 0;
struct node
{
int DATA;
struct node* LC;
struct node* RC;
};
void sortBST(struct node* ptr, int* arr)
{
if(ptr->LC != NULL)
sortBST(ptr->LC, arr);
arr[i] = ptr->DATA;
i++;
if(ptr->RC != NULL)
sortBST(ptr->RC, arr);
}
void insertBST(struct node* ptr, int ITEM)
{
struct node* ptr1;
while(ptr != NULL)
{
if(ITEM <= ptr->DATA)
{
ptr1 = ptr;
ptr = ptr->LC;
}
else if(ITEM > ptr->DATA)
{
ptr1 = ptr;
ptr = ptr->RC;
}
```

```
}
if(ptr == NULL)
{
if(ptr1->DATA < ITEM)
{
ptr1->RC = (struct node*) malloc(sizeof(struct node));
ptr1->RC->LC = NULL;
ptr1->RC->RC = NULL;
ptr1->RC->DATA = ITEM;
}
else
{
ptr1->LC = (struct node*) malloc(sizeof(struct node));
ptr1->LC->LC = NULL;
ptr1->LC->RC = NULL;
ptr1->LC->DATA = ITEM;
}
}
}
void main(){
int* arr;
int n;
printf("Enter the array size\n");
scanf("%d", &n);
arr = malloc(n*sizeof(int));
printf("Enter the numbers\n");
for(int i=0; i<n; i++)
scanf("%d", &arr[i]);
struct node* ROOT = (struct node*) malloc(sizeof(struct node));
ROOT->LC = NULL;
ROOT->RC = NULL;
ROOT->DATA = arr[0];
for(int i=1; i<n; i++)
insertBST(ROOT, arr[i]);
sortBST(ROOT, arr);
printf("Sorted array: ");
for(int i=0; i<n; i++)
printf("%d ", arr[i]);
```

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

OUTPUT:

```
Enter the array size
4
Enter the numbers
2
4
1
5
Sorted array: 1 2 4 5
```

GRAPH TRAVERSALS (DFS AND BFS)

PROGRAM CODE:

```
//SANIN MOHAMMED N
//B21CSB55
#include<stdio.h>
#include<stdlib.h>
struct stack
{
int size;
int TOP;
int *arr;
};
struct queue
{
int FRONT;
int REAR;
int *arr;
int SIZE;
};
int isFull(struct stack *st)
{
if(st->TOP >= st->size-1)
return 1;
return 0;
}
int isEmpty(struct stack *st)
{
if(st->TOP == -1)
return 1;
return 0;
}
void push(struct stack *st, char x)
{
if(!isFull(st))
{
st->arr[++st->TOP] = x;
}
}
```

```
char pop(struct stack *st)
{
if(!isEmpty(st))
{
char x = st->arr[st->TOP];
st->TOP--;
return x;
}
}
void createStack(struct stack *st, int n)
{
st->size = n;
st->arr = (int*) malloc (st->size * sizeof(int));
st->TOP = -1;
}
void enqueue(struct queue *q, char X)
{
if(q->REAR != q->SIZE-1)
{
if(q->FRONT == -1)
q->FRONT = 0;
q->REAR += 1;
        q->arr[q->REAR] = X;
}
}
char dequeue(struct queue *q)
{
if(q->FRONT != -1)
{
char X = q->arr[q->FRONT];
if(q->FRONT == q->REAR)
{
q->FRONT = -1;
q->REAR = -1;
}
else
q->FRONT += 1;
return X;
```

```
}
}
void createQueue(struct queue *q, int n)
{
q->SIZE = n;
q->arr = malloc(q->SIZE * sizeof(int));
q->FRONT = -1;
q->REAR = -1;
}
void dfs(int n, char arr[][n+1])
{
   struct stack *st = malloc(sizeof(struct stack));
int count = 0;
int i = 0;
char v;
char visit[n];
createStack(st, n*n);
push(st, arr[0][1]);
while(!isEmpty(st))
{
v = pop(st);
for(int j=0; j<n; j++)
if(visit[j] == v)
count++;
if(count == 0)
{
printf("%c ", v);
visit[i] = v;
i++;
for(int j=1; i<=n; j++)
if(arr[0][j] == v)
{
for(int k=1; k<=n; k++)
if(arr[k][j] == '1')
push(st, arr[k][0]);
break;
}
}
```

```
count = 0;
}
}
void bfs(int n, char arr[][n+1])
{
struct queue *q = malloc(sizeof(struct queue));
int i = 1;
int count = 0;
char visit[n];
char v;
createQueue(q, n*n);
enqueue(q, arr[0][1]);
printf("%c ", arr[0][1]);
visit[0] = arr[0][1];
while(q->FRONT != -1)
{
v = dequeue(q);
for(int j=1; i<=n; j++)
if(arr[0][j] == v)
{
for(int k=1; k<=n; k++)
if(arr[k][j] == '1')
{
for(int l=0; l<n; l++)
if(visit[l] == arr[k][0])
count++;
if(count == 0)
{
enqueue(q, arr[k][0]);
printf("%c ", arr[k][0]);
visit[i] = arr[k][0];
i++;
}
count = 0;
}
break;
}
}
```

```
}
void main()
{
int n;
char c;
int m;
printf("Enter the no. of vertices\n");
scanf("%d", &n);
char arr[n+1][n+1];
arr[0][0] = ' ';
printf("Enter the vertices\n");
for(int i=1; i<=n; i++)
{
scanf("\n%c", &arr[i][0]);
arr[0][i] = arr[i][0];
}
for(int i=1;i<=n;i++)
for(int j=i;j<=n;j++)
{
   if(arr[i][0] == arr[0][j])
{
printf("Is %c a self loop ? (Y/N)\n", arr[i][0]);
L1:
scanf("\n%c", &c);
if(c == 'y' || c == 'Y')
arr[i][j] = '1';
else if(c == 'n' || c == 'N')
arr[i][j] = '0';
else
{
printf("Enter Y/N!\n");
goto L1;
}
continue;
}
printf("Are %c and %c adjacent ? (Y/N)\n", arr[i][0], arr[0][j]);
L2:
scanf("\n%c", &c);
```

```
if(c == 'y' || c == 'Y')
{
arr[i][j] = '1';
arr[j][i] = '1';
}
else if(c == 'n' || c == 'N')
{
arr[i][j] = '0';
arr[j][i] = '0';
}
else
{
printf("Enter Y/N!\n");
goto L2;
}
}
printf("\nAdjacency matrix of the graph:\n");
for(int i=0;i<=n;i++)
{
for(int j=0;j<=n;j++)
printf("%c ", arr[i][j]);
printf("\n");
}
L3:
printf("\nChoose the operation\n");
printf("1. DFS Traversal\n2. BFS Traversal\n3. Quit\n");
printf(“Enter your choice\n”);
scanf("%d", &m);
if(m == 1)
{
printf("\nDFS Traversal : ");
dfs(n, arr);
printf("\n");
goto L3;
}
else if(m == 2)
{
printf("\nBFS Traversal : ");
```

```
bfs(n, arr);
printf("\n");
goto L3;
}
else if(m == 3)
exit(0);
else
{
printf("Invalid entry\n");
goto L3;
}
}
```

OUTPUT:

```
Enter the no. of vertices
4
Enter the vertices
2
5
3
7
Is 2 a self loop ? (Y/N)
Y
Are 2 and 5 adjacent ? (Y/N)
Y
Are 2 and 3 adjacent ? (Y/N)
Y
Are 2 and 7 adjacent ? (Y/N)
N
Is 5 a self loop ? (Y/N)
Y
Are 5 and 3 adjacent ? (Y/N)
N
Are 5 and 7 adjacent ? (Y/N)
Y
Is 3 a self loop ? (Y/N)
N
Are 3 and 7 adjacent ? (Y/N)
Y
```

Is 7 a self loop ? (Y/N)
Y
Adjacency matrix of the graph:
2 5 3 7
2 1 1 1 0
5 1 1 0 1
3 1 0 0 1
7 0 1 1 1
Choose the operation
1. DFS Traversal
2. BFS Traversal
3. Quit
Enter your choice
1
DFS Traversal : 2 3 7 5
Choose the operation
1. DFS Traversal
2. BFS Traversal
3. Quit
Enter your choice
2
BFS Traversal : 2 5 3 7
Choose the operation
1. DFS Traversal
2. BFS Traversal
3. Quit
Enter your choice
3

QUICK SORT AND MERGE SORT

PROGRAM CODE:

```
//SANIN MOHAMMED N
//B21CSB55
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
#include<math.h>
struct student
{
char name[20];
float height;
float weight;
};
int partition(struct student* st, int p, int r)
{
struct student temp;
float x = st[r].height;
int i = p-1;
for(int j = p; j < r; j++)
if(st[j].height <= x)
{
i++;
if(i != j)
{
temp = st[i];
st[i] = st[j];
st[j] = temp;
}
}
if(r != i+1)
{
temp = st[i+1];
st[i+1] = st[r];
st[r] = temp;
}
```

```
return i+1;
}
void quicksort(struct student* st, int p, int r)
{
if(p < r)
{
int q = partition(st, p ,r);
quicksort(st, p, q-1);
quicksort(st, q+1, r);
}
}
void merge(struct student* st1, int p, int q, int r)
{
int n1 = q - p + 1;
int n2 = r - q;
struct student L[n1], R[n2];
for(int i = 0; i < n1; i++)
L[i] = st1[p+i];
for(int j = 0; j < n2; j++)
R[j] = st1[q+j+1];
int i = 0, j = 0;
int k;
for(k = p; k <= r; k++)
{
if(L[i].height <= R[j].height)
{
st1[k] = L[i];
i++;
if(i == n1)
{
k++;
break;
}
}
else
{
st1[k] = R[j];
j++;
```

```
if(j == n2)
{
k++;
break;
}
}
}
while(i < n1)
{
st1[k] = L[i];
i++;
k++;
}
while(j < n2)
{
st1[k] = R[j];
j++;
k++;
}
}
void mergesort(struct student* st1, int p, int r)
{
if(p < r)
{
int q = floor((p+r)/2);
mergesort(st1, p, q);
mergesort(st1, q+1, r);
merge(st1, p, q, r);
}
}
void main()
{
int n;
char c;
printf("Enter the number of students\n");
scanf("%d", &n);
FILE *fp = fopen("Student details.txt", "w");
FILE *fpq = fopen("Quick student details.txt", "w");
```

```
FILE *fpm = fopen("Merge student details.txt", "w");
fprintf(fp, "NAME\t\tHEIGHT\tWEIGHT\n");
fprintf(fpq, "NAME\t\tHEIGHT\tWEIGHT\n");
fprintf(fpm, "NAME\t\tHEIGHT\tWEIGHT\n");
struct student* st = malloc(n * sizeof(struct student));
for(int i=0; i<n; i++)
{
printf("\nEnter the student details\n");
printf("Name = ");
scanf("%c", &c);
fgets(st[i].name, 20, stdin);
st[i].name[strlen(st[i].name) - 1] = '\0';
printf("Height = ");
scanf("%f", &st[i].height);
printf("Weight = ");
scanf("%f", &st[i].weight);
}
printf("\nWriting to file...\n");
for(int i = 0; i < n; i++)
fprintf(fp, "%s\t\t%.2f\t%.2f\n", st[i].name, st[i].height, st[i].weight);
printf("\nPerforming quick sort...\n");
clock_t t = clock();
quicksort(st, 0, n-1);
t = clock() - t;
for(int i = 0; i < n; i++)
fprintf(fpq, "%s\t\t%.2f\t%.2f\n", st[i].name, st[i].height, st[i].weight);
fprintf(fpq, "\nTime taken = %lf", (double) t / CLOCKS_PER_SEC);
for(int i = 0; i < n; i++)
fscanf(fp, "%s\t\t%f\t%f\n", st[i].name, &st[i].height, &st[i].weight);
printf("\nPerforming merge sort...\n");
t = clock();
mergesort(st, 0, n-1);
t = clock() - t;
for(int i = 0; i < n; i++)
fprintf(fpm, "%s\t\t%.2f\t%.2f\n", st[i].name, st[i].height, st[i].weight);
fprintf(fpm, "\nTime taken = %lf", (double) t / CLOCKS_PER_SEC);
printf("\nWrite successful.\n\n");
}
```

OUTPUT:

```
Enter the number of students
2
Enter the student details
Name = aparna
Height = 155
Weight = 50
Enter the student details
Name = tania
Height = 162
Weight = 45
Writing to file...
Performing quick sort...
Performing merge sort...
Write successful.
MERGE SORT .txt
NAME HEIGHT WEIGHT
aparna 155.00 50.00
tania 162.00 45.00
Time taken = 0.000002
QUICK SORT .txt
NAME HEIGHT WEIGHT
aparna 155.00 50.00
tania 162.00 45.00
Time taken = 0.000001
```

HEAP SORT

PROGRAM CODE:

```
//SANIN MOHAMMED N
//B21CSB55

#include<stdio.h>
#include<stdlib.h>
void createheap(int* arr, int n)
{
int i = 0, temp, j;
while(i < n)
{
j = i;
while(j > 0)
{
if(arr[j] > arr[(j-1)/2])
{
temp = arr[j];
arr[j] = arr[(j-1)/2];
arr[(j-1)/2] = temp;
j = (j-1)/2;
}
else
break;
}
i++;
}
}
void removemax(int* arr, int i)
{
int temp = arr[i];
arr[i] = arr[0];
arr[0] = temp;
}
void rebuildheap(int* arr, int i)
{
if(i == 0)
```

```
return;
int j = 0, temp, lc, rc;
while(1)
{
lc = 2 * j + 1;
rc = 2 * (j + 1);
if(rc <= i)
{
if(arr[j] <= arr[lc] && arr[lc] >= arr[rc])
{
temp = arr[j];
arr[j] = arr[lc];
arr[lc] = temp;
j = lc;
}
else if(arr[j] <= arr[rc] && arr[rc] >= arr[lc])
{
temp = arr[j];
arr[j] = arr[rc];
arr[rc] = temp;
j = rc;
}
else
break;
}
else if(lc <= i)
{
if(arr[j] <= arr[lc])
{
temp = arr[j];
arr[j] = arr[lc];
arr[lc] = temp;
break;
}
else
break;
}
else
```

```
break;
}
}
void heapsort(int* arr, int n)
{
createheap(arr, n);
for(int i = n-1; i > 0; i--)
{
removemax(arr, i);
rebuildheap(arr, i-1);
}
}
int binarysearch(int* arr, int num, int l, int r)
{
while(l <= r)
{
int m = l + (r - l) / 2; //For small size, (l + r) / 2
if(arr[m] == num)
return m;
else if(arr[m] < num)
l = m + 1;
else
r = m - 1;
}
return -1;
}
void main()
{
int n, num;
printf("Enter the array size\n");
scanf("%d", &n);
int* arr = malloc(n * sizeof(int));
printf("Enter the elements\n");
for(int i = 0; i < n; i++)
scanf("%d", &arr[i]);
heapsort(arr, n);
printf("\nThe sorted array: ");
for(int i = 0; i < n; i++)
```

```
printf("%d ", arr[i]);
printf("\n");
while(1)
{
printf("\nEnter the number to search (Enter -1 to exit)\n");
scanf("%d", &num);
if(num == -1)
break;
int index = binarysearch(arr, num, 0, n);
if(index != -1)
printf("%d found at index %d\n", num, index);
else
printf("Search unsuccessful!\n");
}
}
```

OUTPUT:

```
Enter the array size
4
Enter the elements
3
2
1
4
The sorted array: 1 2 3 4
Enter the number to search (Enter -1 to exit)
2
2 found at index 1
Enter the number to search (Enter -1 to exit)
3
3 found at index 2
Enter the number to search (Enter -1 to exit)
1
1 found at index 0
Enter the number to search (Enter -1 to exit)
-1
```

HASHTABLE USING CHAINING METHOD

PROGRAM CODE

```
//SANIN MOHAMMED N
//B21CSB55
#include<stdio.h>
#include<stdlib.h>
struct node
{
int DATA;
struct node* LINK;
};
void display(struct node** hash)
{
struct node* ptr;
for(int i = 0; i < 10; i++)
{
ptr = hash[i]->LINK;
printf("\n%d - ", i);
while(ptr != NULL)
{
printf("%d ", ptr->DATA);
ptr = ptr->LINK;
}
}
printf("\n");
}
void new_entry(struct node** hash)
{
int key;
printf("Enter the element\n");
scanf("%d", &key);
int h = key % 10;
struct node *ptr = hash[h];
while(ptr->LINK != NULL)
ptr = ptr->LINK;
ptr->LINK = malloc(sizeof(struct node));
```

```
ptr->LINK->DATA = key;
ptr->LINK->LINK = NULL;
display(hash);
}
void main()
{
int flag;
struct node** hash = malloc(10 * sizeof(struct node*));
for(int i = 0; i < 10; i++)
{
hash[i] = malloc(sizeof(struct node));
hash[i]->LINK = NULL;
}
while(1)
{
printf("\nEnter\n1. New entry\n2. Display Hash table\n3. Exit\n");
scanf("%d", &flag);
switch(flag)
{
case 1:
new_entry(hash);
break;
case 2:
display(hash);
break;
case 3:
exit(0);
default:
printf("\nInvalid entry!\n");
}
}
}
```

OUTPUT:

```
Enter
1. New entry
2. Display Hash table
3. Exit
1
```

Enter the element
43
0 -
1 -
2 -
3 - 43
4 -
5 -
6 -
7 -
8 -
9 -
Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
11
0 -
1 - 11
2 -
3 - 43
4 -
5 -
6 -
7 -
8 -
9 -
Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
24
0 -
1 - 11

2 -
3 - 43
4 - 24
5 -
6 -
7 -
8 -
9 -
Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
13
0 -
1 - 11
2 -
3 - 43 13
4 - 24
5 -
6 -
7 -
8 -
9 -
Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
35
0 -
1 - 11
2 -
3 - 43 13
4 - 24
5 - 35

6 -
7 -
8 -
9 -
Enter
1. New entry
2. Display Hash table
3. Exit
2
0 -
1 - 11
2 -
3 - 43 13
4 - 24
5 - 35
6 -
7 -
8 -
9 -
Enter
1. New entry
2. Display Hash table
3. Exit
3

HASHTABLE USING LINEAR PROBING

PROGRAM CODE:

```
//SANIN MOHAMMED N
//B21CSB55
#include<stdio.h>
#include<stdlib.h>
void display(int hash[], int n)
{
printf("\n");
for(int i = 0; i < n; i++)
printf("%d\n", hash[i]);
}
void new_entry(int hash[], int n)
{
int key;
printf("\nEnter the element\n");
scanf("%d", &key);
int h = key % n;
if(hash[h] == 0)
{
hash[h] = key;
display(hash, n);
}
else
{
for(int i = h+1; i < n; i++)
if(hash[i] == 0)
{
hash[i] = key;
display(hash, n);
return;
}
for(int i = 0; i < h; i++)
if(hash[i] == 0)
{
hash[i] = key;
display(hash, n);
```

```
return;
}
printf("\nHash table is full!\n");
}
}
void main()
{
int size, flag;
printf("\nEnter size of hash table\n");
scanf("%d", &size);
int* hash = calloc(size, sizeof(int));
while(1)
{
printf("\nEnter\n1. New entry\n2. Display Hash table\n3. Exit\n");
scanf("%d", &flag);
switch(flag)
{
case 1:
new_entry(hash, size);
break;
case 2:
display(hash, size);
break;
case 3:
exit(0);
default:
printf("\nInvalid entry!\n");
break;
}
}
}
```

OUTPUT:

```
Enter size of hash table
5
Enter
1. New entry
2. Display Hash table
3. Exit
```

1
Enter the element
5
5
0
0
0
0
Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
2
5
0
2
0
0
Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
8
5
0
2
8
0
Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element

3
5
0
2
8
3
Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
1
5
1
2
8
3
Enter
1. New entry
2. Display Hash table
3. Exit
1
Enter the element
3
Hash table is full!
Enter
1. New entry
2. Display Hash table
3. Exit
3