EX.NO.1a. ARRAY IMPLEMENTATION OF LIST

Aim:

To create a list using array and perform operations such as display, insertions and deletions.

Algorithm:

1. Start
2. Define list using an array of size n.
3. First item with subscript or position = 0
4. Display menu on list operation
5. Accept user choice
6. If choice = 1 then
7. Locate node after which insertion is to be done
8. Create an empty location in array after the node by moving the existing elements one position ahead.
9. Insert the new element at appropriate position
10. Else if choice = 2
11. Get element to be deleted.
12. Locate the position of the element replace the element with the element in the following position.
13. Repeat the step for successive elements until end of the array.
14. Else
15. Traverse the list from position or subscript 0 to n.
16. Stop

Program:

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
void create();
void insert();
void deletion();
void search();
void display();
int a,b[20], n, p, e, f, i, pos;
void main()
{
//clrscr();
int ch;
char g='y';
do
{printf("\n main Menu");
printf("\n 1.Create \n 2.Delete \n 3.Search \n 4.Insert \n 5.Display\n 6.Exit \n");
printf("\n Enter your Choice");
scanf("%d", &ch);
switch(ch)
{
case 1:
create();
break;
case 2:
deletion();
break;
case 3:
search();
break;
case 4:
insert();
break;
case 5:
display();
break;
case 6:
exit();
break;
default:
printf("\n Enter the correct choice:");
}
printf("\n Do u want to continue:::");
scanf("\n%c", &g);
}
while(g=='y'||g=='Y');
getch();
}
void create()
{
printf("\n Enter the number of nodes");
scanf("%d", &n);
for(i=0;i<n;i++)
{
```

```
printf("\n Enter the Element:",i+1);
scanf("%d", &b[i]);
}
}
void deletion()
{
printf("\n Enter the position u want to delete::");
scanf("%d", &pos);
if(pos>=n)
{
printf("\n Invalid Location::");
}
else
{
for(i=pos+1;i<n;i++)
{
b[i-1]=b[i];
}
n--;
}
printf("\n The Elements after deletion");
for(i=0;i<n;i++)
{
printf("\t%d", b[i]);
}
}
void search()
{
printf("\n Enter the Element to be searched:");
scanf("%d", &e);
for(i=0;i<n;i++)
{
if(b[i]==e)
{
printf("Value is in the %d Position", i);
}
else
{
printf("Value %d is not in the list::", e);
continue;
}
}
}
void insert()
{
printf("\n Enter the position u need to insert::");
scanf("%d", &pos);
if(pos>=n)
{
printf("\n invalid Location::");
}
else
{
for(i=MAX-1;i>=pos-1;i--)
{
b[i+1]=b[i];
}
printf("\n Enter the element to insert::\n");
scanf("%d",&p);
b[pos]=p;
n++;
}
printf("\n The list after insertion::\n");
display();
}
void display()
{
printf("\n The Elements of The list ADT are:");
for(i=0;i<n;i++)
{
printf("\n\n%d", b[i]);
}
}
```

Output :

```
Array of 10 numbers:
Input elements:
1
2
4
5
6
7
8
9
10
11
Current array: 1 2 4 5 6 7 8 9 10 11
Would you like to enter another element? [1/0]: 1
Input element: 3
Input position: 2
Current array: 1 2 3 4 5 6 7 8 9 10
```

Result:

Thus the array implementation of list was demonstrated.

EX.NO 1b. IMPLEMENTATION OF SINGLY LINKED LIST

Aim:

Write the program to implement Single linked list.

Algorithm:

Step1.Create the new node using create () function.
Step2.insert () used to add the element to the list at the given position.
Step3.Display () will display the element in the list.
Step4. Find () used to find the location of the given element.
Step5.Delete () will delete the element in the list for given position.
Step6.End

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void create();
void display();
void insert();
void find();
void delete();
typedef struct node *position;
position L,p,newnode;
struct node
{
int data;
position next;
};
void main()
{
int choice; clrscr(); do
{
printf("1.create\n2.display\n3.insert\n4.find\n5.delete\n\n\n"); printf("Enter your choice\n\n");
scanf("%d",&choice); switch(choice)
{
case 1:
create();
break;
case 2:
display();
break;
case 3:
insert();
break;
case 4:
find();
break;
case 5:
delete();
break;
case 6:
exit(0);
}
}
while(choice<7);
getch();
}
void create()
{
int i,n;
L=NULL;
newnode=(struct node*)malloc(sizeof(struct node));
printf("\n Enter the number of nodes to be inserted\n");
scanf("%d",&n);
printf("\n Enter the data\n");
```

```
scanf("%d",&newnode->data);
newnode->next=NULL;
L=newnode; p=L;
for(i=2;i<=n;i++)
{
newnode=(struct node *)malloc(sizeof(struct node));
scanf("%d",&newnode->data);
newnode->next=NULL;
p->next=newnode;
p=newnode;
}
}
void display()
{ p=L;
while(p!=NULL)
{
printf("%d -> ",p->data);
p=p->next;
}
printf("Null\n");
}
void insert()
{
int ch;
printf("\nEnter ur choice\n");
printf("\n1.first\n2.middle\n3.end\n");
scanf("%d",&ch);
switch(ch)
{
case 2:
{
int pos,i=1; p=L;
newnode=(struct node*)malloc(sizeof(struct node));
printf("\nEnter the data to be inserted\n");
scanf("%d",&newnode->data);
printf("\nEnter the position to be inserted\n");
scanf("%d",&pos);
newnode->next=NULL;
while(i<pos-1)
{
p=p->next; i++;
}
newnode->next=p->next;
p->next=newnode;
p=newnode;
display();
break;
}
case 1:
{ p=L;
newnode=(struct node*)malloc(sizeof(struct node));
printf("\nEnter the data to be inserted\n");
scanf("%d",&newnode->data);
newnode->next=L;
L=newnode;
display();
break;
}
```

```
case 3:
{ p=L;
newnode=(struct node*)malloc(sizeof(struct node));
printf("\nEnter the data to be inserted\n");
scanf("%d",&newnode->data);
while(p->next!=NULL)
p=p->next;
newnode->next=NULL;
p->next=newnode;
p=newnode;
display();
break;
}
}
}
void find()
{
int search,count=0;
printf("\n Enter the element to be found:\n");
scanf("%d",&search);
p=L;
while(p!=NULL)
{
if(p->data==search)
{
count++; break;
}
p=p->next;
}
if(count==0)
printf("\n Element Not present\n");
else
printf("\n Element present in the list \n\n");
}
void delete()
{
position p,temp;
int x; p=L;
if(p==NULL)
{
printf("empty list\n");
}
else
{
printf("\nEnter the data to be deleted\n");
scanf("%d",&x);
if(x==p->data)
{ temp=p; L=p->next;
free(temp);
display();
}
else
{
while(p->next!=NULL && p->next->data!=x)
{
p=p->next;
}
temp=p->next;
p->next=p->next->next;
free(temp);
display();
}
}
}
```

Output:

```
1.create
2.display
3.insert
4.find
5.delete
Enter your choice
1
Enter the number of nodes to be inserted
5
Enter the data 1
2
3
4
5
--------------------------------------------
1.create
2.display
3.insert
4.find
5.delete
Enter your choice 2
1 -> 2 -> 3 -> 4 -> 5 ->
Null
--------------------------------------------------
1.create
2.display
3.insert
4.find
5.delete
Enter your choice 3
Enter ur choice
1.first
2.middle
3.end
1
Enter the data to be inserted
7
7 -> 1 -> 2 -> 3 -> 4 ->5-.
Null
```

Result:

Thus the program single linked list was executed and output was verified.

EX.NO:1C. **DOUBLY LINKED LIST**

Aim: Write the program to implement Doubly linked list.

Algorithm:
Step1. Create() **function used to create the element for duble linked list**
Step2. Insert() will insert the element in the list and change the previous and next address of the node based on the insert node.
Step3. Delete() used to delete the element in the list
Step4. Find() will print the position of the element.
Step5. Display() will print the element in the list.
Step6. End.

Program:
```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void insert();
void deletion();
void display();
void find();
typedef struct node *position;
position newnode,temp,L=NULL,P;
struct node
{
int data;
position next;
position prev;
};
void main()
{
int choice;
clrscr();
do
{ printf("\n1.INSERT");
printf("\n2.DELETE");
printf("\n3.DISPLAY");
printf("\n4.FIND");
printf("\n5.EXIT")
; printf("\nEnter ur option");
scanf("%d",&choice);
switch(choice)
{
case 1: insert();
break;
case 2:
deletion();
break;
case 3:
display();
break;
case 4:
find();
break;
case 5:
exit(1);
}
}while(choice!=5)
; getch();
}
void insert()
{
int pos,I;
newnode=(struct node*)malloc(sizeof(struct node));
printf("\nEnter the data to be inserted");
scanf("%d",&newnode->data);
if(L==NULL)
{
L=newnode;
L->next=NULL;
L->prev=NULL;
```

```
}
else
{
printf("\nEnter the position where the data is to be inserted");
scanf("%d",&pos);
if(pos==1)
{
newnode->next=L;
newnode->prev=NULL;
L->prev=newnode;
L=newnode;
}
else
{ P=L;
for(i=1;i<pos-1&&P->next!=NULL;i++)
{
P=P->next;
}
newnode->next=P->next;
P->next=newnode;
newnode->prev=P;
P->next->prev=newnode;
}
}
void deletion()
{
int pos,I; if(L==NULL)
printf("\nThe list is empty");
else
{
printf("\nEnter the position of the data to be deleted");
scanf("%d",&pos);
if(pos==1)
{ temp=L;
L=temp->next;
L->prev=NULL;
printf("\nThe deleted element is %d",temp->data);
free(temp);
}
else
{ P=L;
for(i=1;i<pos-1;i++)
P=P->next;
temp=P->next;
printf("\nThe deleted element is %d",temp->data);
P->next=temp->next;
temp->next->prev=P;
free(temp);
}
}
}
void display()
{ if(L==NULL)
printf("\nNo of elements in the list");
else
{
printf("\nThe elements in the listare\n");
for(P=L;P!=NULL;P=P->next)
printf("%d",P->data);
}
}
void find()
{
int a,flag=0,count=0;
if(L==NULL)
printf("\nThe list is empty");
else
{
printf("\nEnter the elements to be searched");
scanf("%d",&a);
for(P=L;P!=NULL;P=P->next)
{
count++;
if(P->data==a)
{
flag=1;
printf("\nThe element is found");
printf("\nThe position is %d",count);
break;
}
}
if(flag==0)
printf("\nThe element is not found");
}
}
```

Output:

```
Enter the data to be inserted10
1. INSERT
2. DELETE
3. DISPLAY
4. FIND
5. EXIT
Enter your option 1
Enter the data to be inserted 20
Enter the position where the data is to be inserted 2
1. INSERT
2. DELETE
3. DISPLAY
4. FIND
5. EXIT
Enter ur option 1
Enter the data to be inserted 30
Enter the position where the data is to be inserted 3
1. INSERT
2. DELETE
3. DISPLAY
4. FIND
5. EXIT
Enter ur option 3
The elements in the list are
10 20 30
1. INSERT
2. DELETE
3. DISPLAY
4. FIND
5. EXIT
Enter ur option 2
Enter the position of the data to be deleted 2
The deleted element is 20
1. INSERT
2. DELETE
3. DISPLAY
4. FIND
5. EXIT
Enter ur option 3
The elements in the list are
10 30
1. INSERT
2. DELETE
3. DISPLAY
4. FIND
5. EXIT
Enter ur option 4
Enter the elements to be searched 20
The element is not found
1. INSERT
2. DELETE
3. DISPLAY
4. FIND
5. EXIT
Enter ur option 4
Enter the elements to be searched 30
The element is found The position is 2 1.INSERT
```

```
2. DELETE
3. DISPLAY
4. FIND
5. EXIT
Enter ur option5
Press any key to continue …
Enter the elements to be searched 30
The element is found The position is 2
```

Result:

Thus the program was written and executed and the output was verified.

Ex. No. 2a ARRAY IMPLEMENTATION OF STACK

Aim:

To implement stack operations using array.

Algorithm :

```
1. Start
2. Define a array stack of size max = 5
3. Initialize top = -1
4. Display a menu listing stack operations
5. Accept choice
6. If choice = 1 then
7. If top < max -1
8. Increment top
9. Store element at current position of top
10. Else
11. Print Stack overflow
12. If choice = 2 then
13. If top < 0 then
14. Print Stack underflow
15. Else
16. Display current top element
17. Decrement top
18. If choice = 3 then
19. Display stack elements starting from top
20. Stop
```

Program

```
/* 1a - Stack Operation using Arrays */
#include <stdio.h>
#include <conio.h>
#define max 5
static int stack[max];
int top = -1;
void push(int x)
{
stack[++top] = x;
}
int pop()
{
return (stack[top--]);
}
void view()
{
Int i;

if (top < 0)
printf("\n Stack Empty \n");
else
{
printf("\n Top-->");
for(i=top; i>=0; i--)
{
printf("%4d", stack[i]);
}
printf("\n");
}
}
main()
{
int ch=0, val;
clrscr();
while(ch != 4)
{
printf("\n STACK OPERATION \n");
printf("1.PUSH ");
printf("2.POP ");
printf("3.VIEW ");
printf("4.QUIT \n");
printf("Enter Choice : ");
scanf("%d", &ch);
switch(ch)
{
case 1:
if(top < max-1)
{
printf("\nEnter Stack element : ");
scanf("%d", &val);
push(val);
}
else
printf("\n Stack Overflow \n");
break;
case 2:
if(top < 0)
printf("\n Stack Underflow \n");
else
{
val = pop();
printf("\n Popped element is %d\n", val);
}
break;
case 3:
view();
break;
case 4:

exit(0);
default:
printf("\n Invalid Choice \n");
}
}
}
```

Output

```
STACK OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 1
Enter Stack element : 12
STACK OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 1
Enter Stack element : 23
STACK OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 1
Enter Stack element : 34
STACK OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 1
Enter Stack element : 45
STACK OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 3
Top--> 45 34 23 12
STACK OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 2
Popped element is 45
STACK OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 3
Top--> 34 23 12
STACK OPERATION
1.PUSH 2.POP 3.VIEW 4.QUIT
Enter Choice : 4
```

Result:

Thus push and pop operations of a stack was demonstrated using arrays.

Ex. No. 2b ARRAY IMPLEMENTATION OF QUEUE

Aim:

To implement queue operations using array.

Algorithm

```
1. Start
2. Define a array queue of size max = 5
3. Initialize front = rear = –1
4. Display a menu listing queue operations
5. Accept choice
6. If choice = 1 then
7. If rear < max -1
8. Increment rear
9. Store element at current position of rear
10. Else
11. Print Queue Full
12. If choice = 2 then
13. If front = –1 then
14. Print Queue empty
15. Else
16. Display current front element
17. Increment front
18. If choice = 3 then
19. Display queue elements starting from front to rear.
20. Stop
```

Program

```
/* 1b - Queue Operation using Arrays */
#include <stdio.h>
#include <conio.h>
#define max 5
static int queue[max];
int front = -1;
int rear = -1;
void insert(int x)
{
queue[++rear] = x;
if (front == -1)
front = 0;
}
int remove()
{

int val;
val = queue[front];
if (front==rear && rear==max-1)
front = rear = -1;
else
front ++;
return (val);
}
void view()
{
int i;
if (front == -1)
printf("\n Queue Empty \n");
else
{
printf("\n Front-->");
for(i=front; i<=rear; i++)
printf("%4d", queue[i]);
printf(" <--Rear\n");
}
}
main()
{
int ch= 0,val;
clrscr();
while(ch != 4)
{
printf("\n QUEUE OPERATION \n");
printf("1.INSERT ");
printf("2.DELETE ");
printf("3.VIEW ");
printf("4.QUIT\n");
printf("Enter Choice : ");
scanf("%d", &ch);
switch(ch)
{
case 1:
if(rear < max-1)
{
printf("\n Enter element to be inserted : ");
scanf("%d", &val);
insert(val);
}
else
printf("\n Queue Full \n");
break;
case 2:
if(front == -1)
printf("\n Queue Empty \n");
else
```

```
{
val = remove();
printf("\n Element deleted : %d \n", val);
}
break;
case 3:
view();
break;
case 4:
exit(0);
default:
printf("\n Invalid Choice \n");
}
}
}
```

Output

```
QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT
Enter Choice : 1
Enter element to be inserted : 12
QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT
Enter Choice : 1
Enter element to be inserted : 23
QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT
Enter Choice : 1
Enter element to be inserted : 34
QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT
Enter Choice : 1
Enter element to be inserted : 45
QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT
Enter Choice : 1
Enter element to be inserted : 56
QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT
Enter Choice : 1
Queue Full
QUEUE OPERATION
1.INSERT 2.DELETE 3.VIEW 4.QUIT
Enter Choice : 3
Front--> 12 23 34 45 56 <--Rear
```

Result:

Thus insert and delete operations of a queue was demonstrated using arrays.

Ex. No. 3 **INFIX TO POSTFIX**

Aim:

To convert infix expression to its postfix form using stack operations.

Algorithm

1. Start
2. Define a array *stack* of size *max* = 20
3. Initialize *top* = -1
4. Read the infix expression character-by-character
5. If character is an operand print it
6. If character is an operator
7. Compare the operator‟s priority with the stack[top] operator.
8. If the stack [top] operator has higher or equal priority than the inputoperator, Pop it from the stack and print it.
9. Else
10. Push the input operator onto the stack
11. If character is a left parenthesis, then push it onto the stack.
12. If the character is a right parenthesis, pop all the operators from the stack andprint it until a left parenthesis is encountered. Do not print the parenthesis.

Program

```
/*  Conversion of infix to postfix expression */
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define MAX 20
int top = -1;
char stack[MAX];
char pop();
void push(char item);
int prcd(char symbol)
{
switch(symbol)
{
case '+':
case '-':
return 2;
break;
case '*':
case '/':
return 4;
break;
case '^':
case '$':
return 6;
break;
case '(':
case ')':

case '#':
return 1;
break;
}
}
int isoperator(char symbol)
{
switch(symbol)
{
case '+':
case '-':
case '*':
case '/':
case '^':
case '$':
case '(':
case ')':
return 1;
break;
default:
return 0;
}
}
void convertip(char infix[],char postfix[])
{
int i,symbol,j = 0;
stack[++top] = '#';
for(i=0;i<strlen(infix);i++)
```

```
{
symbol = infix[i];
if(isoperator(symbol) == 0)
{
postfix[j] = symbol;
j++;
}
else
{
if(symbol == '(')
push(symbol);
else if(symbol == ')')
{
while(stack[top] != '(')
{
postfix[j] = pop();
j++;
}
pop(); //pop out (.
}
Else

{

if(prcd(symbol) > prcd(stack[top]))
push(symbol);

else
{
while(prcd(symbol) <= prcd(stack[top]))
{
postfix[j] = pop();
j++;
}
push(symbol);
}}}}
while(stack[top] != '#')
{
postfix[j] = pop();
j++;
}
postfix[j] = '\0';
}
main()
{
char infix[20],postfix[20];
clrscr();
printf("Enter the valid infix string: ");
gets(infix);
convertip(infix, postfix);
printf("The corresponding postfix string is: ");
puts(postfix);
getch();
}
void push(char item)
{
top++;
stack[top] = item;
}
```

```
char pop()
{
char a;
a = stack[top];
top--;
return a;
}
```

Output

```
Enter the valid infix string: (a+b*c)/(d$e)
The corresponding postfix string is: abc*+de$/
Enter the valid infix string: a*b+c*d/e
The corresponding postfix string is: ab*cd*e/+
Enter the valid infix string: a+b*c+(d*e+f)*g
The corresponding postfix string is: abc*+de*f+g*+
```

Result:

Thus the given infix expression was converted into postfix form using stack.

Ex No. 4 IMPLEMENTATION BINARY SEARCH TREE

Aim:

To construct a binary search tree and perform search.

Algorithm

1. Start
2. Call insert to insert an element into binary search tree.
3. Call delete to delete an element from binary search tree.
4. Call findmax to find the element with maximum value in binary search tree
5. Call findmin to find the element with minimum value in binary search tree
6. Call find to check the presence of the element in the binary search tree
7. Call display to display the elements of the binary search tree
8. Call makeempty to delete the entire tree.
9. Stop

Program

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct searchtree
{
int element;
struct searchtree *left,*right;
}*root;
typedef struct searchtree *node;
typedef int ElementType;
node insert(ElementType, node);
node delete(ElementType, node);
void makeempty();
node findmin(node);
node findmax(node);
node find(ElementType, node);
void display(node, int);
void main()
{
int ch;
ElementType a;
node temp;
makeempty();
while(1)
{
printf("\n1. Insert\n2. Delete\n3. Find\n4. Find min\n5. Find max\n6.Display\n7. Exit\nEnter Your
Choice : ");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("Enter an element : ");
scanf("%d", &a);
root = insert(a, root);
break;
case 2:
printf("\nEnter the element to delete : ");
scanf("%d",&a);
root = delete(a, root);
break;
case 3:
printf("\nEnter the element to search : ");
scanf("%d",&a);
temp = find(a, root);
if (temp != NULL)
printf("Element found");

else

printf("Element not found");

break;
case 4:
temp = findmin(root);
if(temp==NULL)
printf("\nEmpty tree");
else
printf("\nMinimum element : %d", temp->element);
```

```
break;
case 5:
temp = findmax(root);
if(temp==NULL)
printf("\nEmpty tree");
else
printf("\nMaximum element : %d", temp->element);
break;
case 6:
if(root==NULL)printf("\nEmpty tree");
else
display(root, 1);
break;
case 7:
exit(0);
default:
printf("Invalid Choice");
}
}
}
node insert(ElementType x,node t)
{
if(t==NULL)
{
t = (node)malloc(sizeof(node));
t->element = x;
t->left = t->right = NULL;
}
else
{
if(x < t->element)
t->left = insert(x, t->left);
else
if(x > t->element)t->right = insert(x, t->right);
}
return t;
}
node delete(ElementType x,node t)
{

node temp;

if(t == NULL)

printf("\nElement not found");
else
{
if(x < t->element)
t->left = delete(x, t->left);
else if(x > t->element)
t->right = delete(x, t->right);
else
{
if(t->left && t->right)
{
temp = findmin(t->right);
t->element = temp->element;
t->right = delete(t->element,t->right);
}
```

```
else if(t->left == NULL)
{
temp = t;
t=t->right;
free (temp);
}
else
{
temp = t;
t=t->left;
free (temp);
}
}
}
return t;
}
void makeempty()
{
root = NULL;
}
node findmin(node temp)
{
if(temp == NULL || temp->left == NULL)
return temp;
return findmin(temp->left);
}
node findmax(node temp)
{
if(temp==NULL || temp->right==NULL)
return temp;

return findmax(temp->right);
}
node find(ElementType x, node t)
{
if(t==NULL)
return NULL;
if(x<t->element)
return find(x,t->left);
if(x>t->element)
return find(x,t->right);
return t;
}
void display(node t,int level)
{
int i;
if(t)
{
display(t->right, level+1);
printf(" \n");
for(i=0;i<level;i++)
printf(" ");
printf("%d", t->element);
display(t->left, level+1);
}
}
```

```
Output:
1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 1
Enter an element : 10
1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 1
Enter an element : 20
1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 1
Enter an element : 5
1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 4
The smallest Number is 5
1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 3
Enter an element : 100
Element not Found
1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 2
Enter an element : 20
1. Insert
```

```
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 6
20
10
1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 7
```

Result:

The program for binary search tree is executed and verified.

Ex. No. 5 **TREE TRAVERSAL**

Aim :

To implement tree traversal (In order, preorder, post order traversals)

Algorithm.:

```
Step1.Start
Step2.declare tree roort,left and right
Step3.In inorder function ()
     Traverse the left subtree, i.e., call Inorder(left->subtree)
     Visit the root.
     Traverse the right subtree, i.e., call Inorder(right->subtree)
Step4.In Preorder()
    Visit the root.
     Traverse the left subtree, i.e., call Preorder(left->subtree)
    Traverse the right subtree, i.e., call Preorder(right->subtree)
Step5.In postorder()
   Traverse the left subtree, i.e., call Postorder(left->subtree)
   Traverse the right subtree, i.e., call Postorder(right->subtree)
   Visit the root
Step6.Stop
```

Program

```
// Tree traversal in C
#include <stdio.h>
#include <stdlib.h>
struct node {
int item;
struct node* left;
struct node* right;
};
// Inorder traversal
void inorderTraversal(struct node* root) {
  if (root == NULL) return;
inorderTraversal(root->left);
printf("%d ->", root->item);
inorderTraversal(root->right);
}
// preorderTraversal traversal
void preorderTraversal(struct node* root) {
  if (root == NULL) return;
printf("%d ->", root->item);
preorderTraversal(root->left);
preorderTraversal(root->right);
}
// postorderTraversal traversal
void postorderTraversal(struct node* root) {
  if (root == NULL) return;
postorderTraversal(root->left);
postorderTraversal(root->right);
printf("%d ->", root->item);
}
// Create a new Node
struct node* createNode(value) {
struct node* newNode = malloc(sizeof(struct node));
newNode->item = value;
newNode->left = NULL;
newNode->right = NULL;
```

```
  return newNode;
}
// Insert on the left of the node
struct node* insertLeft(struct node* root, int value) {
  root->left = createNode(value);
  return root->left;
}
// Insert on the right of the node
struct node* insertRight(struct node* root, int value) {
  root->right = createNode(value);
  return root->right;
}
int main() {
struct node* root = createNode(1);
insertLeft(root, 12);
insertRight(root, 9);
insertLeft(root->left, 5);
insertRight(root->left, 6);
printf("Inorder traversal \n");
inorderTraversal(root);
printf("\nPreorder traversal \n");
preorderTraversal(root);
printf("\nPostorder traversal \n");
postorderTraversal(root);
}
```

Output:

```
Inorder:
5->12->6->1->9->
Preorder:
1->12->5->6->9->
Postorder:
5->6->12->9->1->
```

Result:

The program for binary search tree is executed and verified.

Ex. No.6 IMPLEMENTATION OF HEAP USING PRIORITY QUEUES

AIM

To implement the program for Heap Using Priority Queues.

ALGORITHM

1. Priority queue is a type of queue in which every element has a key associated to it and the queue returns the element according to these keys, unlike the traditional queue which works on first come first serve basis.
2. Thus, a max-priority queue returns the element with maximum key first whereas, a min-priority queue returns the element with the smallest key first.
3. max-priority queue and min-priority queue
4. Priority queues are used in many algorithms like Huffman Codes, Prim's algorithm, etc. It is also used in scheduling processes for a computer, etc.
5. Heaps are great for implementing a priority queue because of the largest and smallest element at the root of the tree for a max-heap and a min-heap respectively. We use a max-heap for a max-priority queue and a min-heap for a min-priority queue.
6. There are mainly 4 operations we want from a priority queue:
7. **Insert** → To insert a new element in the queue.
8. **Maximum/Minimum** → To get the maximum and the minimum element from the max-priority queue and min-priority queue respectively.
9. **Extract Maximum/Minimum** → To remove and return the maximum and the minimum element from the max-priority queue and min-priority queue respectively.
10. **Increase/Decrease key** → To increase or decrease key of any element in the queue.
11. A priority queue stores its data in a specific order according to the keys of the elements. So, inserting a new data must go in a place according to the specified order. This is what the insert operation does.
12. The entire point of the priority queue is to get the data according to the key of the data and the Maximum/Minimum and Extract Maximum/Minimum does this for us.
13. We know that the maximum (or minimum) element of a priority queue is at the root of the max-heap (or min-heap). So, we just need to return the element at the root of the heap.
14. This is like the pop of a queue, we return the element as well as **delete** it from the heap. So, we have to return and delete the root of a heap. Firstly, we store the value of the root in a variable to return it later from the function and then we just make the root equal to the last element of the heap. Now the root is equal to the last element of the heap, we delete the last element easily by reducing the size of the heap by 1.
15. Doing this, we have disturbed the heap property of the root but we have not touched any of its children, so they are still heaps. So, we can call *Heapify* on the root to make the tree a heap again.

```
EXTRACT-MAXIMUM(A)
max = A[1] // stroing maximum value
A[1] = A[heap_size] // making root equal to the last element
heap_size = heap_size-1 // delete last element
MAX-HEAPIFY(A, 1) // root is not following max-heap property
return max //returning the maximum value
```

PROGRAM

```
#include <stdio.h>
int tree_array_size = 20;
int heap_size = 0;
const int INF = 100000;
void swap( int *a, int *b )
{
int t;
t = *a;
*a = *b;
*b = t;
}
//function to get right child of a node of a tree
int get_right_child(int A[], int index)
{
if((((2*index)+1) < tree_array_size) && (index >= 1))
return (2*index)+1;
return -1;
}
//function to get left child of a node of a tree
int get_left_child(int A[], int index)
{
if(((2*index) < tree_array_size) && (index >= 1))
return 2*index;
return -1;
}
//function to get the parent of a node of a tree
int get_parent(int A[], int index)
{
if ((index > 1) && (index < tree_array_size))
{
return index/2;
}
return -1;
}
void max_heapify(int A[], int index)
{
int left_child_index = get_left_child(A, index);
int right_child_index = get_right_child(A, index);
// finding largest among index, left child and right child
int largest = index;
if ((left_child_index <= heap_size) && (left_child_index>0))
{
if (A[left_child_index] > A[largest]) {
largest = left_child_index;
}
}

if ((right_child_index <= heap_size && (right_child_index>0)))
{
if (A[right_child_index] > A[largest])
{
largest = right_child_index;
}
void build_max_heap(int A[])
```

```
{
int i;
for(i=heap_size/2; i>=1; i--)
{
max_heapify(A, i);
}
}
int maximum(int A[])
{
return A[1];
}
int extract_max(int A[])
{
int maxm = A[1];
A[1] = A[heap_size];
heap_size--;
max_heapify(A, 1);
return maxm;
}
void increase_key(int A[], int index, int key)
{
A[index] = key;
while((index>1) && (A[get_parent(A, index)] < A[index]))
{
swap(&A[index], &A[get_parent(A, index)]);
index = get_parent(A, index);
}
}
void decrease_key(int A[], int index, int key)
{
A[index] = key;
max_heapify(A, index);
}
void insert(int A[], int key)
{
heap_size++;
A[heap_size] = -1*INF;
increase_key(A, heap_size, key);
}
void print_heap(int A[])
{
int i;
for(i=1; i<=heap_size; i++)
{
printf("%d\n",A[i]);
}
printf("\n");
}
int main()
{
int A[tree_array_size];
insert(A, 20);
insert(A, 15);
insert(A, 8);
insert(A, 10);
insert(A, 5);
insert(A, 7);
insert(A, 6);
insert(A, 2);
insert(A, 9);
insert(A, 1);
print_heap(A);
increase_key(A, 5, 22);
print_heap(A);
decrease_key(A, 1, 13);
print_heap(A);
printf("%d\n\n", maximum(A));
printf("%d\n\n", extract_max(A));
print_heap(A);
printf("%d\n", extract_max(A));
printf("%d\n", extract_max(A));
printf("%d\n", extract_max(A));
printf("%d\n", extract_max(A));
printf("%d\n", extract_max(A));
printf("%d\n", extract_max(A));
printf("%d\n", extract_max(A));
 printf("%d\n", extract_max(A));
printf("%d\n", extract_max(A));
return 0;
}
```

OUTPUT

```
Priority Queue using Heap:
Enter the number of elements: 7
Enter elements: 234
6543
12234
64
53
6
876
77
PQueue: 64 77 234 536 876 6543 12234
```

RESULT:

Thus the implementation heap using priority queue has been executed successfully..

EX.NO: 7(a) IMPLEMENTATION OF DIJKSTRA'S ALGORITHM

AIM :

To Implementation of Dijkstra's Algorithm

ALGORITHM:

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

PROGRAM:

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 9
int minDistance(int dist[], bool sptSet[])
{
int min = INT_MAX, min_index;
for (int v = 0; v < V; v++)
if (sptSet[v] == false && dist[v] <= min)
min = dist[v], min_index = v;
return min_index;
}
void printSolution(int dist[])
{
printf("Vertex \t\t Distance from Source\n");
for (int i = 0; i < V; i++)
printf("%d \t\t\t\t %d\n", i, dist[i]);
}
void dijkstra(int graph[V][V], int src)
{
int dist[V];
bool sptSet[V];
for (int i = 0; i < V; i++)
dist[i] = INT_MAX, sptSet[i] = false;
dist[src] = 0;
for (int count = 0; count < V - 1; count++) {
int u = minDistance(dist, sptSet);
sptSet[u] = true;
for (int v = 0; v < V; v++)
if (!sptSet[v] && graph[u][v]&& dist[u] != INT_MAX&& dist[u] + graph[u][v] < dist[v])
dist[v] = dist[u] + graph[u][v];
}
printSolution(dist);
}
int main()
{
/* Let us create the example graph discussed above */
int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                    { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                    { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                    { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                    { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                    { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                    { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                    { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                    { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
dijkstra(graph, 0);
return 0;
}
```

OUTPUT

```
Vertex      Distance from Source
0           0
1           4
2           12
3           19
4           21
5           11
6           9
7           8
8           14
```

RESULT:

Thus the Implementation of Dijkstra algorithm is executed successfully.

EX.NO: 7(b) IMPLEMENTATION OF PRIM’S ALGORITHM

AIM:

To implement Prim’s algorithm.

ALGORITHM:

Step 1: Determine an arbitrary vertex as the starting vertex of the MST.
Step 2: Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).
Step 3: Find edges connecting any tree vertex with the fringe vertices.
Step 4: Find the minimum among these edges.
Step 5: Add the chosen edge to the MST if it does not form any cycle.
Step 6: Return the MST and exit

PROGRAM:

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 5
int minKey(int key[], bool mstSet[])
{
int min = INT_MAX, min_index;
for (int v = 0; v < V; v++)
if (mstSet[v] == false && key[v] < min)
min = key[v], min_index = v;
return min_index;
}
int printMST(int parent[], int graph[V][V])
{
printf("Edge \tWeight\n");
for (int i = 1; i < V; i++)
printf("%d - %d \t%d \n", parent[i], i,
graph[i][parent[i]]);
}

void primMST(int graph[V][V])
{
int parent[V];
int key[V];
bool mstSet[V];
for (int i = 0; i < V; i++)
key[i] = INT_MAX, mstSet[i] = false;
key[0] = 0;
parent[0] = -1;
for (int count = 0; count < V - 1; count++) {
int u = minKey(key, mstSet);
mstSet[u] = true;
for (int v = 0; v < V; v++)
if (graph[u][v] && mstSet[v] == false&& graph[u][v] < key[v])
parent[v] = u, key[v] = graph[u][v];
}
printMST(parent, graph);
}
int main()
{
int graph[V][V] = { { 0, 2, 0, 6, 0 },
                    { 2, 0, 3, 8, 5 },
                    { 0, 3, 0, 0, 7 },
                    { 6, 8, 0, 0, 9 },
                    { 0, 5, 7, 9, 0 } };
primMST(graph);
return 0;
}
```

OUTPUT

```
Edge    Weight

0 - 1    2

1 - 2    3

0 - 3    6

1 - 4    5
```

RESULT:

Thus the implementation of prim’s algorithm is executed successfully.

.

EX.NO: 7(c) IMPLEMENTATION OF KRUSHKAL ALGORITHM

AIM:

To implement the program for Krushkal algorithm.

ALGORITHM:

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
int comparator(const void* p1, const void* p2)
{
const int(*x)[3] = p1;
const int(*y)[3] = p2;
return (*x)[2] - (*y)[2];
}
void makeSet(int parent[], int rank[], int n)
{
for (int i = 0; i < n; i++) {
parent[i] = i;
rank[i] = 0;
}
}
int findParent(int parent[], int component)
{
if (parent[component] == component)
return component;
return parent[component] = findParent(parent, parent[component]);
}
void unionSet(int u, int v, int parent[], int rank[], int n)
{
u = findParent(parent, u);
v = findParent(parent, v);
if (rank[u] < rank[v])
 {
parent[u] = v;
}
else if (rank[u] > rank[v]) {
parent[v] = u;
}
else {
parent[v] = u;
rank[u]++;
}
}
void kruskalAlgo(int n, int edge[n][3])
{
qsort(edge, n, sizeof(edge[0]), comparator);
int parent[n];
int rank[n];
makeSet(parent, rank, n);
int minCost = 0;
printf( "Following are the edges in the constructed MST\n");
for (int i = 0; i < n; i++)
 {
int v1 = findParent(parent, edge[i][0]);
int v2 = findParent(parent, edge[i][1]);
int wt = edge[i][2];
if (v1 != v2)
 {
unionSet(v1, v2, parent, rank, n);
minCost += wt;
printf("%d -- %d == %d\n", edge[i][0],
edge[i][1], wt);
}
}
printf("Minimum Cost Spanning Tree: %d\n", minCost);
}
int main()
{
int edge[5][3] = { { 0, 1, 10 },
                   { 0, 2, 6 },
                   { 0, 3, 5 },
                   { 1, 3, 15 },
                   { 2, 3, 4 } };
kruskalAlgo(5, edge);
return 0;
}
```

OUTPUT:

```
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost Spanning Tree: 19
```

RESULT:

Thus the implementation of Kruskal algorithm is executed successfully.

Ex. No. 8a LINEAR SEARCH

AIM: To perform linear search of an element on the given array.

ALGORITHM :

```
1. Start
2. Read number of array elements n
3. Read array elements Ai, i = 0,1,2,…n–1
4. Read search value
5. Assign 0 to found
6. Check each array element against search
7. If Ai = search then
        found = 1
        Print "Element found"
        Print position i
        Stop
8. If found = 0 then
        print "Element not found"
9.Stop
```

PROGRAM

```
/* Linear search on a sorted array */
#include <stdio.h>
#include <conio.h>
main()
{
int a[50],i, n, val, found;
clrscr();
printf("Enter number of elements : ");
scanf("%d", &n);
printf("Enter Array Elements : \n");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
printf("Enter element to locate : ");
scanf("%d", &val);
found = 0;
for(i=0; i<n; i++)
{
if (a[i] == val)
{
printf("Element found at position %d", i);
found = 1;
break;
}
}
if (found == 0)
printf("\n Element not found");
getch();
}
```

OUTPUT

```
Enter number of elements : 7
Enter Array Elements :
23 6 12 5 0 32 10
Enter element to locate : 5
Element found at position 3
```

RESULT

Thus an array was linearly searched for an element's existence.

Ex. No. 8b BINARY SEARCH

AIM

To locate an element in a sorted array using Binary search method

ALGORITHM

```
1. Start
2. Read number of array elements, say n
3. Create an array arr consisting n sorted elements
4. Get element, say key to be located
5. Assign 0 to lower and n to upper
6. While (lower < upper)
a. Determine middle element mid = (upper+lower)/2
b. If key = arr[mid] then
i. Print mid
ii. Stop
c. Else if key > arr[mid] then
i. lower = mid + 1
d. else
i. upper = mid – 1
7. Print "Element not found"
8. Stop
```

PROGRAM

```
/* Binary Search on a sorted array */
#include <stdio.h>
void main()
{
int a[50],i, n, upper, lower, mid, val, found, att=0;
printf("Enter array size : ");
scanf("%d", &n);
for(i=0; i<n; i++)
a[i] = 2 * i;
printf("\n Elements in Sorted Order \n");
for(i=0; i<n; i++)
printf("%4d", a[i]);
printf("\n Enter element to locate : ");
scanf("%d", &val);
upper = n;
lower = 0;
found = -1;
while (lower <= upper)
{
mid = (upper + lower)/2;
att++;
if (a[mid] == val)
{
Printf("found at index %d in %d attempts",mid,att);
found = 1;
break;
}
else if(a[mid] > val)
upper = mid - 1;
else
lower = mid + 1;
}
if (found == -1)
printf("Element not found");
}
```

OUTPUT

```
Enter array size : 10
Elements in Sorted Order
0 2 4 6 8 10 12 14 16 18
Enter element to locate : 16
Found at index 8 in 2 attempts
```

RESULT

Thus an element is located quickly using binary search method.

Ex. No. 9a INSERTION SORT

AIM: To sort an array of N numbers using Insertion sort.

ALGORITHM

1. Start
2. Read number of array elements n
3. Read array elements Ai
4. Outer index i varies from second element to last element
5. Inner index j is used to compare elements to left of outer index
6. Insert the element into the appropriate position.
7. Display the array elements after each pass
8. Display the sorted array elements.
9. Stop

PROGRAM

```
#include <stdio.h>
void main()
{
int i, j, k, n, temp, a[20], p=0;
printf("Enter total elements: ");
scanf("%d",&n);
printf("Enter array elements: ");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
for(i=1; i<n; i++)
{
temp = a[i];
j = i - 1;
while((temp<a[j]) && (j>=0))
{
a[j+1] = a[j];
j = j - 1;
}
a[j+1] = temp;
p++;
printf("\n After Pass %d: ", p);
for(k=0; k<n; k++)
printf(" %d", a[k]);
}
printf("\n Sorted List : ");
for(i=0; i<n; i++)
printf(" %d", a[i]);
}
```

OUTPUT

```
Enter total elements: 6
Enter array elements: 34 8 64 51 32 21
After Pass 1: 8 34 64 51 32 21
After Pass 2: 8 34 64 51 32 21
After Pass 3: 8 34 51 64 32 21
After Pass 4: 8 32 34 51 64 21
After Pass 5: 8 21 32 34 51 64
Sorted List : 8 21 32 34 51 64
```

RESULT:

Thus array elements was sorted using insertion sort.

Ex. No. 9b SELECTION SORT

AIM: To sort an array of N numbers using Selection sort

ALGORITHM:

1. Find the minimum element in the array and swap it with the element in the 1st position.
2. Find the minimum element again in the remaining array[2, n] and swap it with the element at 2nd position, two elements at their correct positions.
3. We have to do this n-1 times to sort the array.

PROGRAM

```
#include <stdio.h>
void main( )
{
int a [ 100 ] , n , i , j , position , temp ;
printf ( "Enter number of elements \n" ) ;
scanf ( "%d", &n ) ;
printf ( " Enter %d integers \n ", n ) ;
for ( i = 0 ; i < n ; i ++ )
scanf ( "%d", & a[ i ] ) ;
for ( i = 0 ; i < ( n - 1 ) ; i ++ )
{
position = i ;
for ( j = i + 1 ; j < n ; j ++ )
{
if ( a [ position ] > a [ j ] ) position = j ;
}
if ( position != i )
{
temp = a [ i ] ;
a [ i ] = a [ position ] ;
a [ position ] = temp ;
}
}
printf ( "Sorted list in ascending order: \n ") ;
for ( i = 0 ; i < n ; i ++ )
printf ( " %d \n ", a[ i ] ) ;
}
```

OUTPUT:

```
Enter number of elements 5
Enter 5 integers
8 3 9 5 1
Sorted list in ascending order:
1
3
5
8
9
```

RESULT:

Thus the program to sort an array of N numbers using Insertion sort has been implemented successfully.

Ex. No. 9c QUICK SORT

AIM:

To sort an array of N numbers using Quick sort.

ALGORITHM

1. Start
2. Read number of array elements n
3. Read array elements Ai
4. Select an pivot element x from Ai
5. Divide the array into 3 sequences: elements < x, x, elements > x
6. Recursively quick sort both sets (Ai < x and Ai > x)
7. Display the sorted array elements
8. Stop

PROGRAM

```
/* 11c - Quick Sort */
#include<stdio.h>
#include<conio.h>
void qsort(int arr[20], int fst, int last);
main()
{
int arr[30];
int i, size;
printf("Enter total no. of the elements : ");
scanf("%d", &size);
printf("Enter total %d elements : \n", size);
for(i=0; i<size; i++)
scanf("%d", &arr[i]);
qsort(arr,0,size-1);
printf("\n Quick sorted elements \n");
for(i=0; i<size; i++)
printf("%d\t", arr[i]);
getch();
}
void qsort(int arr[20], int fst, int last)
{
int i, j, pivot, tmp;
if(fst < last)
{
pivot = fst;
i = fst;
j = last;
while(i < j)
{
while(arr[i] <=arr[pivot] && i<last)
i++;
while(arr[j] > arr[pivot])
j--;
if(i <j )
{
tmp = arr[i];
arr[i] = arr[j];
arr[j] = tmp;
}
}
tmp = arr[pivot];
arr[pivot] = arr[j];
arr[j] = tmp;
qsort(arr, fst, j-1);
qsort(arr,j+1, last); } }
```

OUTPUT

```
Enter total no. of the elements : 8
Enter total 8 elements :
127
-1
04
-2
```

```
3
Quick sorted elements
-2 -1 0 1 2 3 4 7
```

RESULT

Thus the array was sorted using quick sort’s divide and conquers method.

Ex. No. 9d MERGE SORT

AIM:

To sort an array of N numbers using Merge sort.

ALGORITHM

1. Start
2. Read number of array elements n
3. Read array elements Ai
4. Divide the array into sub-arrays with a set of elements
5. Recursively sort the sub-arrays
6. Display both sorted sub-arrays
7. Merge the sorted sub-arrays onto a single sorted array.
8. Display the merge sorted array elements
9. Stop

PROGRAM

```
/* 11d – Merge sort */
#include <stdio.h>
#include <conio.h>
void merge(int [],int ,int ,int );
void part(int [],int ,int );
int size;
main()
{
int i, arr[30];
printf("Enter total no. of elements : ");
scanf("%d", &size);
printf("Enter array elements : ");
for(i=0; i<size; i++)
scanf("%d", &arr[i]);
part(arr, 0, size-1);
printf("\n Merge sorted list : ");
for(i=0; i<size; i++)
printf("%d ",arr[i]);
getch();
}
void part(int arr[], int min, int max)
{
int mid;
if(min < max)
{
mid = (min + max) / 2;
part(arr, min, mid);
part(arr, mid+1, max);
merge(arr, min, mid, max);
}
if (max-min == (size/2)-1)
{
printf("\n Half sorted list : ");
for(i=min; i<=max; i++)
printf("%d ", arr[i]);
}
}
```

```
void merge(int arr[],int min,int mid,int max)
{
int tmp[30];
int i, j, k, m;
j = min;
m = mid + 1;
for(i=min; j<=mid && m<=max; i++)
{
if(arr[j] <= arr[m])
{
tmp[i] = arr[j];
j++;
}
else
{
tmp[i] = arr[m];
m++;
}
}
if(j > mid)
{
for(k=m; k<=max; k++)
{
tmp[i] = arr[k];
i++;
}
}
else
{
for(k=j; k<=mid; k++)
{
tmp[i] = arr[k];
i++;
}
}
for(k=min; k<=max; k++)
arr[k] = tmp[k];
}
```

OUTPUT

```
Enter total no. of elements : 8
Enter array elements : 24 13 26 1 2 27 38 15
Half sorted list : 1 13 24 26
Half sorted list : 2 15 27 38
Merge sorted list : 1 2 13 15 24 26 27 38
```

RESULT

Thus array elements was sorted using merge sort's divide and conquer method

Ex.NO:10 OPEN ADDRESSING

AIM:

To implement program for open Addressing

ALGORITHM:

1. Declare an array of a linked list with the hash table size.
2. Initialize an array of a linked list to NULL.
3. Find hash key.
4. If chain[key] == NULL
5. Make chain[key] points to the key node.
6. Otherwise(collision),
7. Insert the key node at the end of the chain[key].
8. Get the value
9. Compute the hash key.
10. Search the value in the entire chain. i.e. chain[key].
11. If found, print "Search Found"
12. Otherwise, print "Search Not Found"
13. Get the value
14. Compute the key.
15. Using linked list deletion algorithm, delete the element from the chain[key].
16. Deleting a node in the linked list
17. If unable to delete, print "Value Not Found"

PROGRAM

```
#include <stdio.h>
```

```
#include<stdlib.h>
#define TABLE_SIZE 10
int h[TABLE_SIZE]={NULL};
void insert()
{
int key,index,i,flag=0,hkey;
printf("\nenter a value to insert into hash table\n");
scanf("%d",&key);
hkey=key%TABLE_SIZE;
for(i=0;i<TABLE_SIZE;i++)
{
index=(hkey+i)%TABLE_SIZE;
if(h[index] == NULL)
{
h[index]=key;
break;
}
}
if(i == TABLE_SIZE)
printf("\nelement cannot be inserted\n");
}
void search()
{
int key,index,i,flag=0,hkey;
printf("\nenter search element\n");
scanf("%d",&key);
hkey=key%TABLE_SIZE;
for(i=0;i<TABLE_SIZE; i++)
{
index=(hkey+i)%TABLE_SIZE;
if(h[index]==key)
{
printf("value is found at index %d",index);
break;
}
}
if(i == TABLE_SIZE)
printf("\n value is not found\n");
}
void display()
{
int i;
printf("\nelements in the hash table are \n");
for(i=0;i< TABLE_SIZE; i++)
printf("\nat index %d \t value = %d",i,h[i]);
}
main()
{
int opt,i;
while(1)
{
printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
scanf("%d",&opt);
switch(opt)
{
case 1:
insert();
break;
case 2:
display();
break;
case 3:
search();
break;
case 4:exit(0);
}
}
}
```

OUTPUT

```
Press 1. Insert 2. Display 3. Search 4.Exit
1
enter a value to insert into hash table
12
Press 1. Insert 2. Display 3. Search 4.Exit
```

```
1
enter a value to insert into hash table
13
Press 1. Insert 2. Display 3. Search 4.Exit
1
enter a value to insert into hash table
22
Press 1. Insert 2. Display 3. Search 4.Exit
2
elements in the hash table are
at index 0 value = 0
at index 1 value = 0
at index 2 value = 12
at index 3 value = 13
at index 4 value = 22
at index 5 value = 0
at index 6 value = 0
at index 7 value = 0
at index 8 value = 0
at index 9 value = 0
Press 1. Insert 2. Display 3. Search 4.Exit
3
enter search element
12
value is found at index 2
Press 1. Insert 2. Display 3. Search 4.Exit
2 3
enter search element
23
value is not found
Press 1. Insert 2. Display 3. Search 4.Exit
          4
```

RESULT:

Thus the program for open Addressing using Hash table has been implemented successfully.