QUESTION:

Given an array of positive integers. All numbers occur even number of times except one number which occurs odd number of times. Find the number in O(n) time & constant space.

Sample Input :

I/P = [1, 2, 3, 2, 3, 1, 3]

Sample Output :

O/P = 3

Ex. No: 1.1a

DETERMINE THE NUJMBER OCCURING ODD NUMBER OF TIMES

Date:

AIM:

To determine the number occurring od number of times in the given array.

PSEUDOCODE:

```
//Program: To determine the number occurring odd number of times.
//Input: n array elements.
//Output: The number occurs odd times.

BEGIN:

        Declare the variables i,n,j,a[n],h[n].
        Read the value for n.
        FOR(i←0 to n)
                Read a[i]
                h[i]=0
        ENDFOR
        FOR(i←0 ton)
                h[a[i]-1]+=1
        ENDFOR
        FOR(i<-0 to n)
                IF((h[i]%2!=0)then
                        Display i+1
                ENDIF
        ENDFOR

END
```

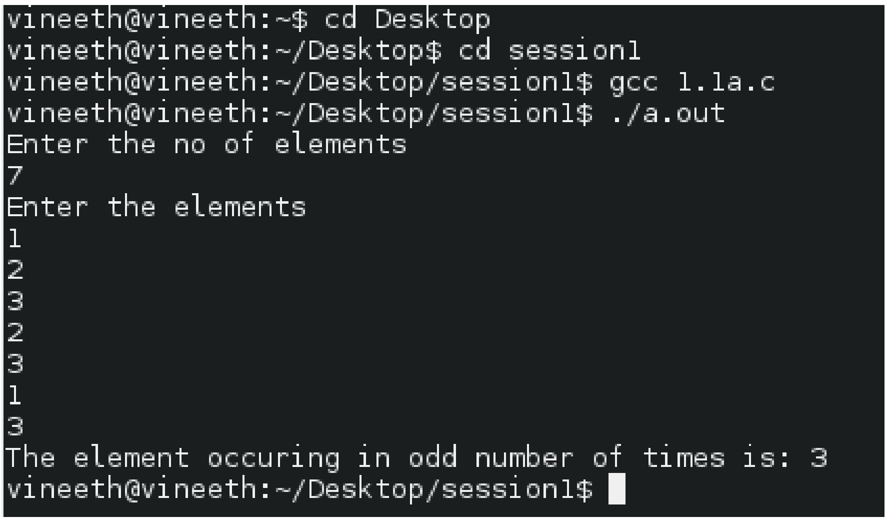
SOURCE CODE:

```
#include<stdio.h>
void main()
{
        int i,n,j;
        printf("Enter the no of elements\n");
        scanf("%d",&n);
        int arr[n],hash[n];
        printf("Enter the elements\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&arr[i]);
                hash[i]=0;
        }
```

```
		for(i=0;i<n;i++)
			hash[arr[i]-1]+=1;
		printf("The element occuring in odd number of times is:\t");
		for(i=0;i<n;i++)
		{
			if(hash[i]%2!=0)
			{
				printf("%d\n",i+1);
			}
		}
	}
```

DATA STRUCTURE USED: Array

OUTPUT:



```
vineeth@vineeth:~$ cd Desktop
vineeth@vineeth:~/Desktop$ cd session1
vineeth@vineeth:~/Desktop/session1$ gcc 1.1a.c
vineeth@vineeth:~/Desktop/session1$ ./a.out
Enter the no of elements
7
Enter the elements
1
2
3
2
3
1
3
The element occuring in odd number of times is: 3
vineeth@vineeth:~/Desktop/session1$
```

RESULT:

Thus the program that determines the number which occurs odd number of times.

QUESTION:

Write a function which takes an array and emits the majority element (if it exists), otherwise prints NONE as follows:

Sample Input 0

I/P : 3 3 4 2 4 4 2 4 4

Sample Output 0

O/P : 4

Sample Input 1

I/P : 3 3 4 2 4 4 2 4

Sample Output1

O/P : NONE

.

Ex. No: 1.1b

DETERMINE THE MAJORITY ELEMENT

Date:

AIM:

To Determine the majority element (if it exists), otherwise prints NONE .

PSEUDOCODE:

```
//Program: To Determine the majority element (if it exists), otherwise prints NONE.
//Input: Size n, Array of size n
//Output: Majority element
  BEGIN

   FOR  i←0 to  i<n
        Read a[i]
        H[i]=0
   ENDFOR
   FOR i←0 to i<n
        IF h[k]+h[i]) THEN
             K=i
        ENDIF
   ENDFOR
   PRINT k

  END
```

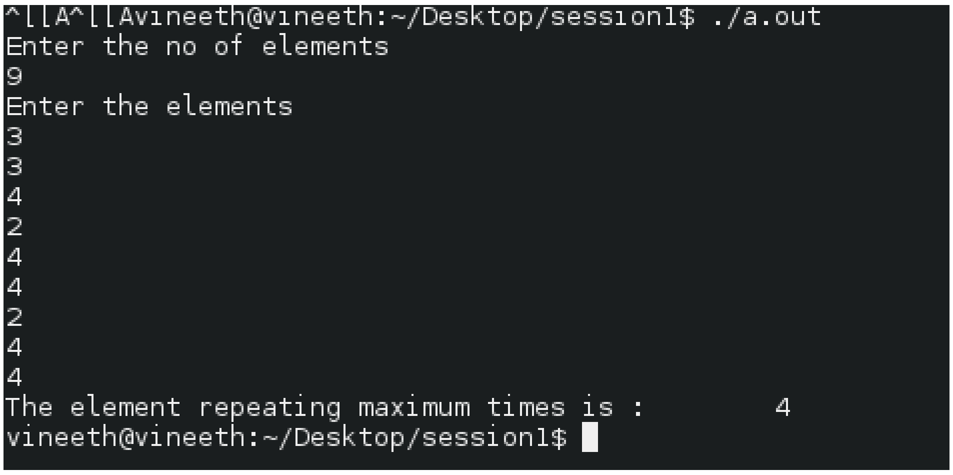
SOURCE CODE:

```
#include<stdio.h>
void main()
{
        int i,n,j,k;
        printf("Enter the no of elements\n");
        scanf("%d",&n);
        int arr[n],hash[n];
        printf("Enter the elements\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&arr[i]);
                hash[i]=0;
        }
        for(i=0;i<n;i++)
                hash[arr[i]-1]+=1;
        k=0;
        for(i=1;i<n;i++)
```

```
                {
                        if(hash[k]<hash[i])
                                k=i;
                }
                printf("The element repeating maximum times is :\t%d\n",k+1);}

        }
```

DATA STRUCTURE USED: Array

OUTPUT:



```
^[[A^[[Avineeth@vineeth:~/Desktop/session1$ ./a.out
Enter the no of elements
9
Enter the elements
3
3
4
2
4
4
2
4
4
The element repeating maximum times is :        4
vineeth@vineeth:~/Desktop/session1$ 
```

RESULT:

Thus the program to determine the majority elements and successfully executed.

QUESTION:

Provided a String of length **N**, your task is to find out whether or not the given string is a prime string. A prime string is a string in which the sum of the ASCII value of all the characters is prime.

Input:
The first line of the input contains an integer **T**, denoting the number of test cases. Then **T** test case follows. The first line of each test case contains an integer **N**denoting the length of the string, next line contains the input string str of length **N**.

Output:
For each test case print **"YES"** if the string is prime string else print **"NO"**, on a new line.

Input	**Output**
3	YES
13	NO
geeksforgeeks	NO
4	
JiiT	
5	
India	

Ex. No: 1.2a

DETERMINE GIVEN STRING IS A PRIME OR NOT

Date:

AIM:

To write a c program to determine the given string is prime or not .

PSEUDOCODE:

```
//Program: To determine the given string is prime or not
//Input: Length and String
//Output: “YES” or “NO”
  BEGIN
   FOR i←2 to add/2
        IF add%i==0 THEN
             K<-1
             break
        ENDIF
  ENDFOR
IF k==0 THEN
        Display YES
ELSE
        Display NO

END
```

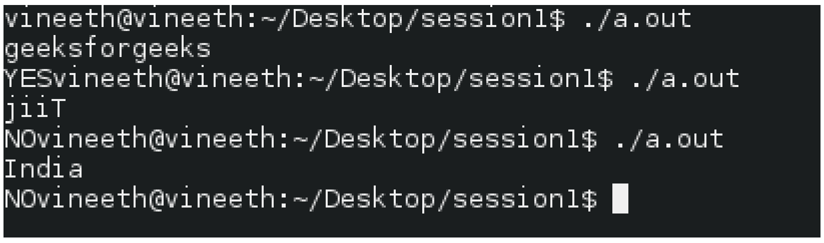
SOURCE CODE:

```
#include<stdio.h>
#include<string.h>
void main()
{
        int i,add=0,k=0;
        char arr[100];
        scanf("%s",arr);
        for(i=0;i<strlen(arr);i++)
        add=add+arr[i];
        i=2;
        for(;i<=(add/2);i++)
        {
                if(add%i==0)
                {
                        k=1;
                        break;
                }
        }
        if(k==0)
                printf("YES");
```

```
	else
		printf("NO");
	}
```

DATA STRUCTURE USED: Array

OUTPUT:



```
vineeth@vineeth:~/Desktop/session1$ ./a.out
geeksforgeeks
YESvineeth@vineeth:~/Desktop/session1$ ./a.out
jiiT
NOvineeth@vineeth:~/Desktop/session1$ ./a.out
India
NOvineeth@vineeth:~/Desktop/session1$ 
```

RESULT:

Thus the program that determines the given string is prime or not.

QUESTION:

Check for balanced parentheses in an expression

Given an expression string exp, write a program to examine whether the pairs and the orders of

“{“,”}”,”(“,”)”,”[“,”]” are correct in exp. For example, the program should print true for exp = “[()]{}{[()()]()}” and false for exp = “[(])”.

Ex. No: 1.2b

BALANCING PARANTHESIS

Date:

AIM:

To write a c program to find out whether the given brackets are balanced or not

PSEUDOCODE:

```
//Program:  find out whether the given brackets are balanced or not
    //Input: n,brackets
    //Output: TRUE or FALSE
     BEGIN
        FOR i<--0 to srlnr(exp)
            IF exp[i]=='('||exp[i]=='{'||exp[i]=='[' THEN
                Push(exp,i)
            ELSEIF top==-1
                return 0
            ENDIF
            IF exp[i]==')'||exp[i]=='}'||exp[i]==']' THEN
                IF stack[top]=='(')==(exp[i]==')'))||((stack[top]=='{')==(exp[i]=='}'))||
((stack[top]=='[')==(exp[i]==')' THEN
                    Pop()
        ENDIF
        ENDFOR

     END
```

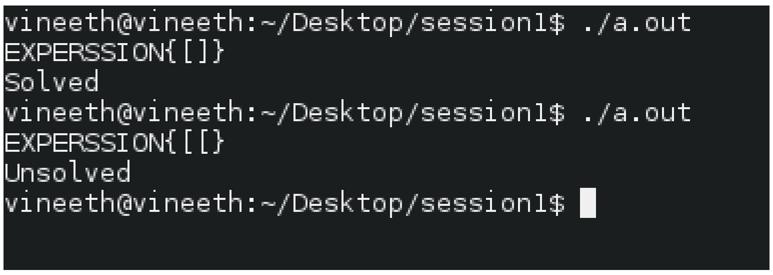
SOURCE CODE:

```
    #include<stdio.h>
    #include<string.h>
    void push(char exp[],int i);
    void pop();
    int check_paranthesis(char exp[]);
    int top=-1,size=15,res,i;
    char stack[100];
    void main()
    {
        char exp[5];
        printf("EXPERSSION");
        scanf("%s",exp);
        res=check_paranthesis(exp);
        if(res)
            printf("Solved\n");
        else
            printf("Unsolved\n");
```

```
}
int check_paranthesis(char exp[])
{
        for(i=0;i<strlen(exp);i++)
        {
                if(exp[i]=='('||exp[i]=='{'||exp[i]=='[')
                {
                        push(exp,i);
                }
                else if (top==-1)
                {
                        return 0;
                }
                if (exp[i]==')'||exp[i]=='}'||exp[i]==']')
                {
                        if(((stack[top]=='(')==(exp[i]==')'))||
                ((stack[top]=='{')==(exp[i]=='}'))||((stack[top]=='[')==(exp[i]==')')))
                        {
                                pop();
                        }
                }
}

if(top==-1)
        return 1;
else
        return 0;
}
void push(char exp[],int i)
{
        if(top!=size-1)
        {
                top++;
                stack[top]=exp[i];
        }
}
void pop()
{
        if(top!=-1)
        {
                top--;
        }
}
```

DATA STRUCTURE USED: Stack

OUTPUT:



```
vineeth@vineeth:~/Desktop/session1$ ./a.out
EXPERSSION{[]}
Solved
vineeth@vineeth:~/Desktop/session1$ ./a.out
EXPERSSION{[[}
Unsolved
vineeth@vineeth:~/Desktop/session1$ 
```

RESULT:

Thus the program that checked the given brackets are balanced or not.

QUESTION:

Write a program to evaluate the arithmetic expression in RPN as follows,
Given a vector of strings, evaluate the value of an arithmetic expression in Reverse Polish Notation.
Valid operators are +, -, *, /. Each operand may be an integer or another expression.
Examples:
["2", "1", "+", "3", "*"] -> ((2 + 1) * 3) -> 9
["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6

Ex. No: 1.3a

EVALUATION OF REVERSE POLISH NOTATION

Date:

AIM:

To write a c program to evaluvate the arithmetic expression in reverse polish notation.

PSEUDOCODE:

```
//Program: To evaluvate the arithmetic expression in reverse polish notation.
//Input: Numbers and operators
//Output: value of the expression.
BEGIN
WHILE e!=”\0”
        IF isdigit(e) THEN
                Num←=e-0
                Push(num)
        ELSE
                N1=pop()
                N2=pop()
                SWITCH(e)
                        CASE “+”:     n3=n1+n2
                        CASE “-”:     n3=n1-n2
                        CASE “*”:     n3=n1*n2
                        CASE “/”:     n3=n1/n2
                ENDSWITCH
                Push(n3)
        ENDIF
e++
ENDWHILE
Display exp,pop()

 END
```

SOURCE CODE:

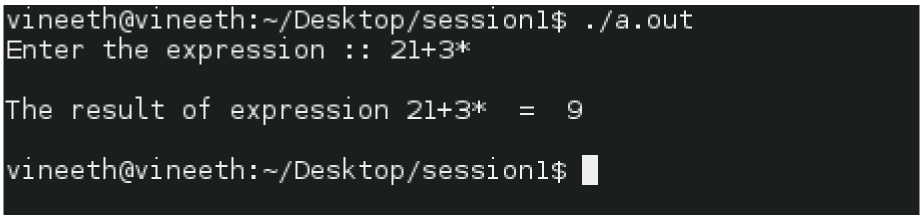
```
#include<stdio.h>
#include<ctype.h>
int stack[20];
int top = -1;
void push(int x)
{
     stack[++top] = x;
}
```

```
int pop()
{
	return stack[top--];
}
int main()
{
	char exp[20];
	char *e;
	int n1,n2,n3,num;
	printf("Enter the expression :: ");
	scanf("%s",exp);
	e = exp;
	while(*e != '\0')
	{
		if(isdigit(*e))
		{
			num = *e-'0';
			push(num);
		}
		else
		{
			n1 = pop();
			n2 = pop();
			switch(*e)
			{
				case '+':
				{
					n3 = n1 + n2;
					   break;
				}
				case '-':
				{
					n3 = n2 - n1;
					break;
				}
				case '*':
				{
					n3 = n1 * n2;
					break;
				}
				case '/':
				{
					n3 = n2 / n1;
					break;
				}
			}
			push(n3);
```

```
            }
          e++;
        }
          printf("\nThe result of expression %s  =  %d\n\n",exp,pop());
        return 0;
        }
```

DATA STRUCTURE USED: Stack

OUTPUT:



```
vineeth@vineeth:~/Desktop/session1$ ./a.out
Enter the expression :: 21+3*

The result of expression 21+3*  =  9

vineeth@vineeth:~/Desktop/session1$ 
```

RESULT:

Thus the program that the arithmetic expression is evaluated in reverse polish notation and successfully executed.

QUESTION:

Create a data structure *two Stacks* that represents two stacks. Implementation of *two Stacks* should use only one array, i.e., both stacks should use the same array for storing elements. Following functions must be supported by *two Stacks*.

push1(int x) –> pushes x to first stack

push2(int x) –> pushes x to second stack

pop1() –> pops an element from first stack and return the popped element

pop2() –> pops an element from second stack and return the popped element
Implementation of *two Stack* should be space efficient.

Ex. No: 1.3b

IMPLEMENTATION OF TWO STACK

Date:

AIM:

To implement the concept of stack data structure.

PSEUDOCODE:

```
BEGIN
    Push1(x)
        IF top1<top2-1 THEN
            Ar[++top1]←data
        ELSE
            Display Stack full
        ENDIF
    Push(x)
        IF top1<top2-1 THEN
            Ar[++top1]←data
        ELSE
            Display Stack full
        ENDIF
    Pop1()
        IF top2 <size THEN
            P=ar[top2++]
        ELSE
            Display Stack empty
        ENDIF
    Pop2()
        IF top2 <size THEN
            P=ar[top2++]
        ELSE
            Display Stack empty
        ENDIF

    Display Stack1 and Stack2


END
```

SOURCE CODE:

```
#include <stdio.h>
#include<stdlib.h>
#define SIZE 100
int ar[SIZE];
```

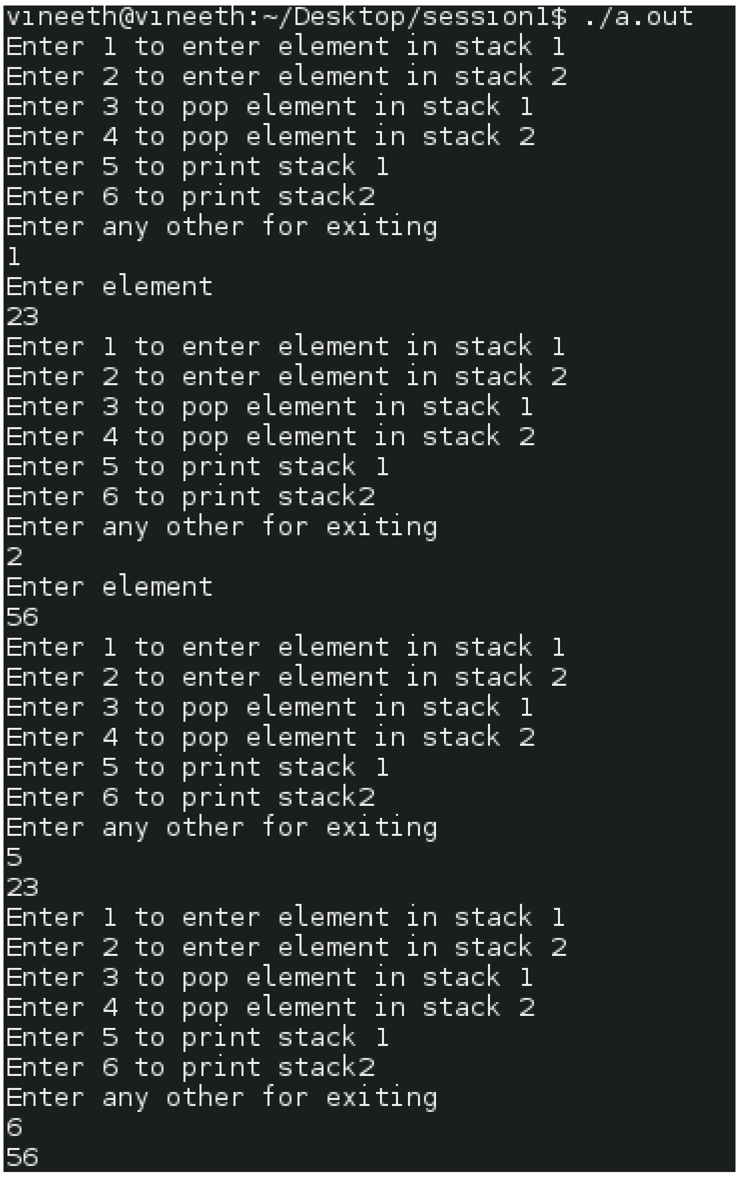
```
int top1 = -1;
int top2 = SIZE;
void push_stack1 (int data)
{
	if (top1 < top2 - 1)
	{
		ar[++top1] = data;
	}
	else
	{
		printf ("Stack Full! Cannot Push\n");
	}
}
void push_stack2 (int data)
{
	if (top1 < top2 - 1)
	{
		ar[--top2] = data;
	}
	else
	{
		printf ("Stack Full! Cannot Push\n");
	}
}
void pop_stack1 ()
{
	if (top1 >= 0)
	{
		int popped_value = ar[top1--];
		printf ("%d is being popped from Stack 1\n", popped_value);
	}
	else
	{
		printf ("Stack Empty! Cannot Pop\n");
	}
}
void pop_stack2 ()
{
	if (top2 < SIZE)
	{
		int popped_value = ar[top2++];
		printf ("%d is being popped from Stack 2\n", popped_value);
	}
	else
	{
		printf ("Stack Empty! Cannot Pop\n");
```

```
        }
}
void print_stack1 ()
{
        int i;
        for (i = top1; i >= 0; --i)
        {
                printf ("%d ", ar[i]);
        }
        printf ("\n");
}
void print_stack2 ()
{
        int i;
        for (i = top2; i < SIZE; ++i)
        {
                printf ("%d ", ar[i]);
        }
        printf ("\n");
}
int main()
{
        int ar[SIZE];
        int i,ele1,ele2;
        do
        {
              printf("Enter 1 to enter element in stack 1\nEnter 2 to enterelement\nEnter 3
              to pop element in stack 1\nEnter 4 to pop element in stack 2\nEnter 5 to print
              stack 1\nEnter 6 to print stack2\nEnter any other for exiting \n");
              scanf("%d",&i);
              switch(i)
              {
                     case 1:printf("Enter element\n");
                        scanf("%d",&ele1);
                        push_stack1(ele1);
                        break;
                     case 2:printf("Enter element\n");
                        scanf("%d",&ele2);
                        push_stack2(ele2);
                        break;
                     case 3:pop_stack1();
                        break;
                     case 4:pop_stack2();
                        break;
                     case 5:print_stack1();
                       break;
```

```
                        case 6:print_stack2();
                          break;
                        default:
                              printf("Program terminated!!!!!!!\n");
                        exit(0);
                  }
            }while(i<7 &&i>0);
      }
```

DATA STRUCTURE USED: Array, Stack

OUTPUT:



```
vineeth@vineeth:~/Desktop/session1$ ./a.out
Enter 1 to enter element in stack 1
Enter 2 to enter element in stack 2
Enter 3 to pop element in stack 1
Enter 4 to pop element in stack 2
Enter 5 to print stack 1
Enter 6 to print stack2
Enter any other for exiting
1
Enter element
23
Enter 1 to enter element in stack 1
Enter 2 to enter element in stack 2
Enter 3 to pop element in stack 1
Enter 4 to pop element in stack 2
Enter 5 to print stack 1
Enter 6 to print stack2
Enter any other for exiting
2
Enter element
56
Enter 1 to enter element in stack 1
Enter 2 to enter element in stack 2
Enter 3 to pop element in stack 1
Enter 4 to pop element in stack 2
Enter 5 to print stack 1
Enter 6 to print stack2
Enter any other for exiting
5
23
Enter 1 to enter element in stack 1
Enter 2 to enter element in stack 2
Enter 3 to pop element in stack 1
Enter 4 to pop element in stack 2
Enter 5 to print stack 1
Enter 6 to print stack2
Enter any other for exiting
6
56
```

RESULT:

Thus the implementation of stack data structure is successfully executed and displayed.

QUESTION:

Searching the elements in The array running time the program should be O(n^2)

Ex. No: 1.4a

LINEAR SEARCH

Date:

AIM:

To search the element in the array .

PSEUDOCODE:

```
//Program: To search the given element in the array.
//Input: Size n, Array of size n
//Output : Element found or not.
  BEGIN
        linearsearch(n,a[],key)
              FOR i <-- 0 to i<=n-1
                    IF(key==a[i])
                          return 1
                    ENDIF
              ENDFOR
        return 0
  END
```

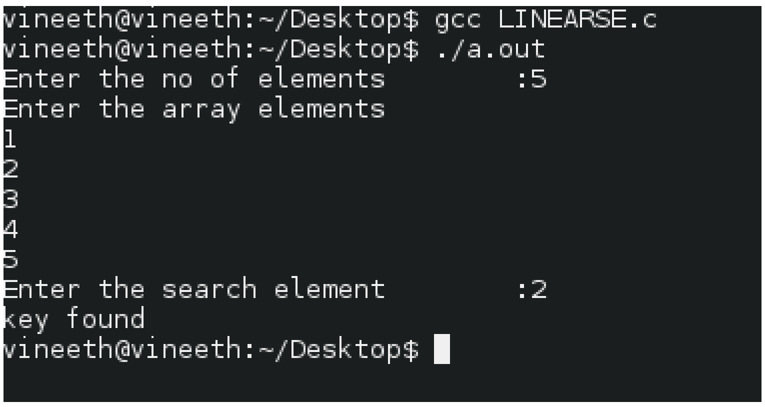
SOURCE CODE:

```
#include<stdio.h>
int linearsearch(int n,int a[],int key)
{
      int i;
      for(i=0;i<=n-1;i++)
      {
            if(key==a[i])
                  return(1);
      }
      return 0;
}
void main()
{
      int i,key,n,k,a[100];
      printf("Enter the no of elements\t:");
      scanf("%d",&n);
      printf("Enter the array elements\n");
      for(i=0;i<n;i++)
            scanf("%d",&a[i]);
      printf("Enter the search element\t:");
      scanf("%d",&key);
      if(linearsearch(n,a,key))
```

```
                        printf("key found\n");
                else
                        printf("key not found\n");
        }
```

DATA STRUCTURE USED: Array

OUTPUT:



```
vineeth@vineeth:~/Desktop$ gcc LINEARSE.c
vineeth@vineeth:~/Desktop$ ./a.out
Enter the no of elements        :5
Enter the array elements
1
2
3
4
5
Enter the search element        :2
key found
vineeth@vineeth:~/Desktop$ 
```

RESULT:

Thus the program that searches the element in running time o(n^2) was successfully executed and verified.

QUESTION:

Searching the element in the sorted array .

Ex. No: 1.4b

Date:

BINARY SEARCH

AIM:

To search the element in the sorted array using binary search.

PSEUDOCODE:

```
//Program: To search the element in the sorted array
//Input: Size n, Array of size n
//Output: Element found or not
  BEGIN
   Binarysearch(a[],low,high,key)
        IF(low<=high)
                mid=(low+high)/2
                IF(key==a[mid])
                        return(1)
                ELSE IF(key<a[mid])
                        return (binarysearch(a,low,mid-1,key))
                ELSE
                        return (binarysearch(a,mid+1,high,key))
                ENDIF
        ENDIF
        return 0
  END
```

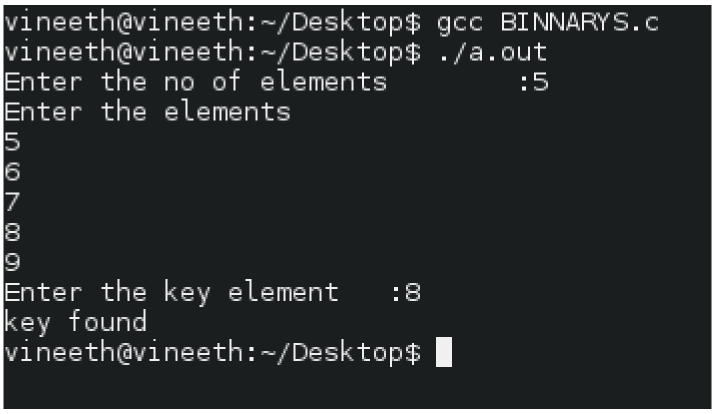
SOURCE CODE:

```
#include<stdio.h>
int binarysearch(int a[],int low,int high,int key)
{
        int mid;
        if(low<=high)
        {
                mid=(low+high)/2;
                if(key==a[mid])
                        return(1);
                else if(key<a[mid])
                        return (binarysearch(a,low,mid-1,key));
                else
                        return (binarysearch(a,mid+1,high,key));
        }
        return 0;
}
```

```
void main()
{
        int i,key,n,k,a[100];
        printf("Enter the no of elements\t:");
        scanf("%d",&n);
        printf("Enter the elements \n");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        printf("Enter the key element\t:");
        scanf("%d",&key);
        if(binarysearch(a,0,n-1,key))
                printf("key found\n");
        else
                printf("key not found\n");
}
```

DATA STRUCTURE USED: Array

OUTPUT:



```
vineeth@vineeth:~/Desktop$ gcc BINNARYS.c
vineeth@vineeth:~/Desktop$ ./a.out
Enter the no of elements        :5
Enter the elements
5
6
7
8
9
Enter the key element   :8
key found
vineeth@vineeth:~/Desktop$ 
```

RESULT:

Thus the program to search the element in the sorted array was successfully executed and verified.

QUESTION:

Given a queue of integers of even length, rearrange the elements by interleaving the first half of the queue with the second half of the queue.
Only a stack can be used as an auxiliary space.
Examples:
Input : 1 2 3 4
Output : 1 3 2 4

Input : 11 12 13 14 15 16 17 18 19 20

Ex. No: 2.1a

REARRANGING ELEMENTS

Date:

AIM:

To Rearrange the elements in the queue by interchanging them.

PSEUDOCODE:

```
//Program: To Rearrange the given elements using stack and queue.
//Input: Size n, Array of size n
//Output : Elements Arranged in different order.

  BEGIN
        Read the Elements from the user.\
        IF the length is even
              FOR i←0 to i->n-1
                    Enqueue the elements into the list
              ENDFOR
              FOR i←0 to (n/2)-1
                    Dequeue the elements
              ENDFOR
              FOR i←0 to (n/2)-1
                    Push and pop the elements
              ENDFOR
              FOR i←1 to (n/2)-1
                    Display pop1() and dequeue1()
              ENDFOR
        ENDIF
  END
```

SOURCE CODE:

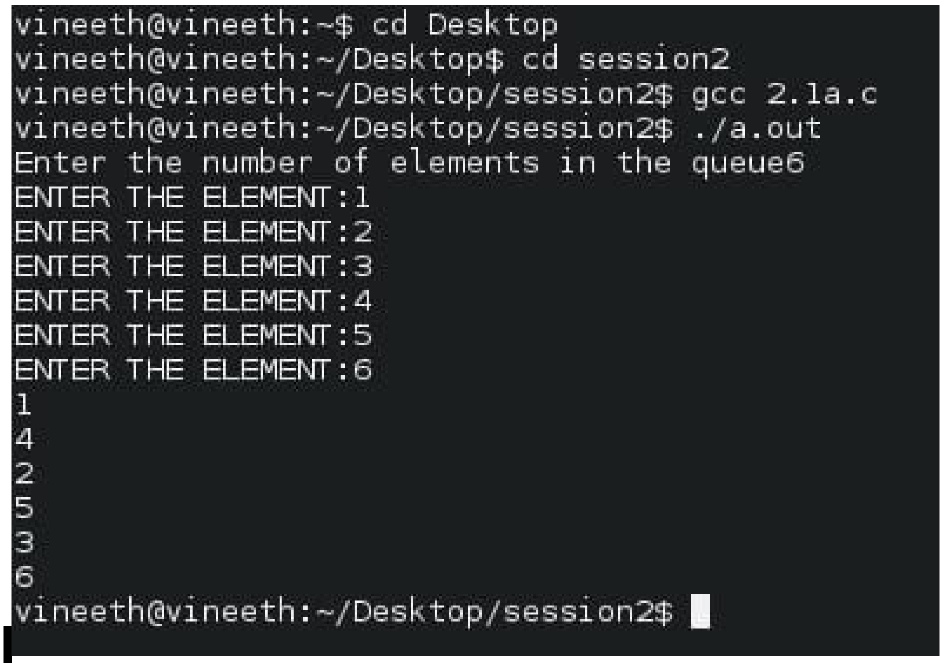
```c
#include<stdio.h>
#include<stdlib.h>
void enqueue(int ele);
int queue[100],stack[100],stk[100],max,top,top1;
int front=-1,rear=-1;
int pop()
{
        int x;
        x=stack[top];
        top--;
        return(x);
}
int pop1()
```

```
{
        int x;
        x=stk[top1];
        top1--;
        return(x);
}

void push(int element)
{
         top++;
         stack[top]=element;
}
void push1(int element)
{
         top1++;
         stk[top1]=element;
}
void dequeue()
{
        push(queue[front]);
        if(front==rear)
                front=rear=-1;
        else
                front++;
}
int dequeue1()
{
        int x;
        x=queue[front];
        if(front==rear)
                front=rear=-1;
        else
                front++;
        return(x);
}
void enqueue(int ele)
{
        if(rear==max-1)
                printf("Queue overflow\n");
        else
        {
                if((rear==-1)&&(front==-1))
                        front=rear=0;
                else
                        rear=rear+1;
                queue[rear]=ele;
        }
```

```
}
int main()
{
        int n,i,ele;
        top=top1=-1;
        printf("Enter the number of elements in the queue");
        scanf("%d",&n);
        max=n;
        if(n%2==0)
        {
                for(i=0;i<n;i++)
                {
                        printf("ENTER THE ELEMENT:");
                        scanf("%d",&ele);
                        enqueue(ele);
                }

                for(i=0;i<n/2;i++)
                        dequeue();
                for(i=0;i<n/2;i++)
                        push1(pop());
                for(i=0;i<n/2;i++)
                {
                        printf("%d\n",pop1());
                        printf("%d\n",dequeue1());
                }
        }
        else
        {
                printf("Queue of integers doesn't have even length\n");
        }
   return 0;
}
```

DATA STRUCTURE USED: Stack,Queue

OUTPUT:



```
vineeth@vineeth:~$ cd Desktop
vineeth@vineeth:~/Desktop$ cd session2
vineeth@vineeth:~/Desktop/session2$ gcc 2.1a.c
vineeth@vineeth:~/Desktop/session2$ ./a.out
Enter the number of elements in the queue6
ENTER THE ELEMENT:1
ENTER THE ELEMENT:2
ENTER THE ELEMENT:3
ENTER THE ELEMENT:4
ENTER THE ELEMENT:5
ENTER THE ELEMENT:6
1
4
2
5
3
6
vineeth@vineeth:~/Desktop/session2$
```

RESULT:

Thus the program that Rearranges the given elements in the queue has been determined and executed.

QUESTION:

Create a data structure kQueues that represents k queues. Implementation of kQueues should use only one array, i.e., k queues should use the same array for storing elements. Following functions must be supported by kQueues.

enqueue(int x, int qn) –> adds x to queue number 'qn' where qn is from 0 to k-1
dequeue(int qn) –> deletes an element from queue number 'qn' where qn is from 0 to k-1

Ex. No: 2.1b

CREATING K QUEUES

Date:

AIM:

To Create a data structure kQueues that represents k queues. Implementation of kQueues should use only one array, i.e., k queues should use the same array for storing elements. Following functions must be supported by kQueues.

PSEUDOCODE:

```
//Program: To create a data structure kQueues that represents KQueues.
//Input: Size n, Array of size n
//Output: Sorted  Queue
  BEGIN

   FOR  i←0 to  i<n
          Read the elements in the queue.
   Read the option
              case 1 :  qno   =getQueueNumber(n);
                        Read data
                        reply = insq(queue, qno-1, r, limit, &data)
                        IF( reply == -1)
                                    Dispaly  Queue is Full
                        ELSE
                                    Dispaly Element is inserted in a Queue
                        END IF
              case 2 : qno    = getQueueNumber(n)
                        reply = delq(queue, qno-1, f, r, &data);
                        IF( reply == -1)
                              Dispaly Queue is Empty
                        ELSE
                              Diaplay Element is deleted from Queue
`                       END IF
              case 3: qno= getQueueNumber(n)
                              IF(f[qno-1]==r[qno-1])
                                    Print  Queue is empty
                              END IF
                              FOR (i=f[qno-1]+1; i<=r[qno-1]; i++)
                                     Dispaly queue[i]
              default:
                        Dispaly Invalid input. Please try again

  END
```

SOURCE CODE:

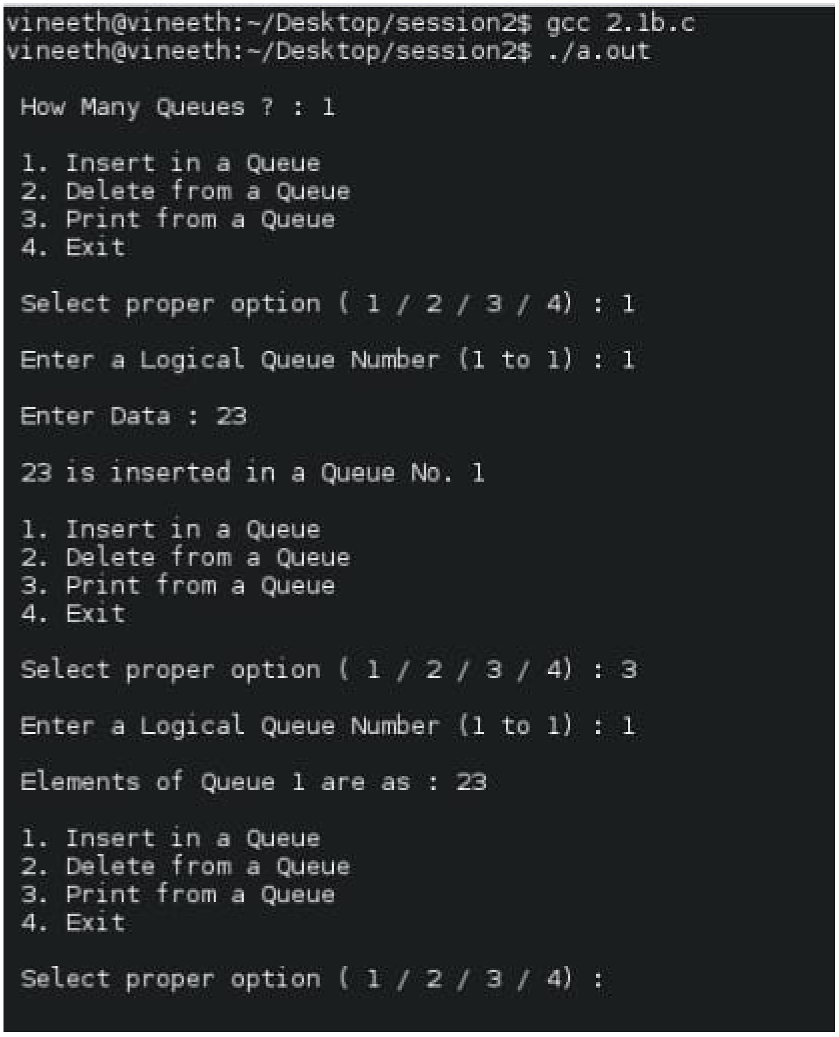
```
#include<stdio.h>
# define max 200
 int insq (int queue[max], int qno, int rear[], int limit[], int *data)
{
        if (rear[qno] == limit[qno])
                return(-1);
        else
         {
                rear[qno]++;
                queue[ rear[qno] ] = *data;
                return(1);
        }
}
int delq (int queue[max], int qno, int front[], int rear[], int *data)
{
        if( front[qno] == rear[qno] )
                return(-1);
        else
        {
                front[qno]++;
                *data = queue[ front[qno] ];
                return(1);
        }
}
int getQueueNumber(int n)
{
        int qNo=0;
        Inva:
        printf("\n Enter a Logical Queue Number (1 to %d) : ", n);
        scanf("%d", &qNo);
        if (qNo<1 || qNo >n)
        {
                printf(" Invalid Queue Number. Please try again.\n");
                goto Inva;
        }
        return qNo;
}
void main()
{
        int queue[max],  data;
        int bott[10], limit[10], f[10], r[10];
        int i, n, qno, size, option, reply;
        printf("\n How Many Queues ? : ");
        scanf("%d", &n);
```

```
size = max / n;
bott[0] = -1;
for(i = 1; i < n; i++)
        bott[i] = bott[i-1] + size;
for(i = 0; i < n; i++)
        limit[i] = bott[i] + size;
for(i = 0; i < n; i++)
        f[i] = r[i] = bott[i];
do
{
        printf("\n 1. Insert in a Queue");
        printf("\n 2. Delete from a Queue");
        printf("\n 3. Print from a Queue");
        printf("\n 4. Exit \n");
        printf("\n Select proper option ( 1 / 2 / 3 / 4) : ");
        scanf("%d", &option);
        switch(option)
        {
                case 1 :  qno    = getQueueNumber(n);
                        printf("\n Enter Data : ");
                        scanf("%d", &data);
                        reply = insq(queue, qno-1, r, limit, &data);
                        if( reply == -1)
                                printf("\n Queue %d is Full \n", qno);
                        else
                                printf("\n %d is inserted in a Queue No. %d \n", data,
                        qno);
                        break;
                case 2 : qno    = getQueueNumber(n);
                        reply = delq(queue, qno-1, f, r, &data);
                        if( reply == -1)
                                printf("\n Queue %d is Empty \n", qno);
                        else
                                printf("\n %d is deleted from Queue No. %d \n", data,
                        qno);
                        break;
                case 3: qno     = getQueueNumber(n);
                        printf("\n Elements of Queue %d are as : ", qno);
                        if (f[qno-1]==r[qno-1])
                        {
                                printf("\n Queue is empty");
                                break;
                        }
                        for (i=f[qno-1]+1; i<=r[qno-1]; i++)
                                printf("%d\t", queue[i]);
                        printf("\n");
                        break;
```

```
                        case 4 :break;
                        default: printf("\n Invalid input. Please try again.");
                }
        }while(option!=4);
}
```

DATA STRUCTURE USED: Array,Queue

OUTPUT:



```
vineeth@vineeth:~/Desktop/session2$ gcc 2.1b.c
vineeth@vineeth:~/Desktop/session2$ ./a.out

 How Many Queues ? : 1

 1. Insert in a Queue
 2. Delete from a Queue
 3. Print from a Queue
 4. Exit

 Select proper option ( 1 / 2 / 3 / 4) : 1

 Enter a Logical Queue Number (1 to 1) : 1

 Enter Data : 23

 23 is inserted in a Queue No. 1

 1. Insert in a Queue
 2. Delete from a Queue
 3. Print from a Queue
 4. Exit

 Select proper option ( 1 / 2 / 3 / 4) : 3

 Enter a Logical Queue Number (1 to 1) : 1

 Elements of Queue 1 are as : 23

 1. Insert in a Queue
 2. Delete from a Queue
 3. Print from a Queue
 4. Exit

 Select proper option ( 1 / 2 / 3 / 4) :
```

RESULT:

Thus the program to sort the elements in the queue has been determined and verified.

QUESTION:

Given a singly linked list, group all odd nodes together followed by the even nodes. Please note here we are talking about the node number and not the value in the nodes.
You should try to do it in place. The program should run in O(1) space complexity and O(nodes) time complexity.
Example:
Given 1->2->3->4->5->NULL,
return 1->3->5->2->4->NULL.
Note:
The relative order inside both the even and odd groups should remain as it was in the input.
The first node is considered odd, the second node even and so on.

Ex. No: 2.2a

REARRANGE THE ELEMENTS IN SINGLY LINKED LIST

Date:

AIM:

To write a c program to group the odd th elements in a singly linked list followed by the elements in the even position.

PSEUDOCODE:

```
//Program: To rearrange the elements in a singly linked list.
//Input  :Elements for the linked list
//Output: Resultant linked list
  BEGIN
        Read the no of elements n
        FOR i←0 to n-1
                Read the element
                Insend(element)
        ENDFOR
        Arrange(n)
        Display(n)
  END
```

SOURCE CODE:

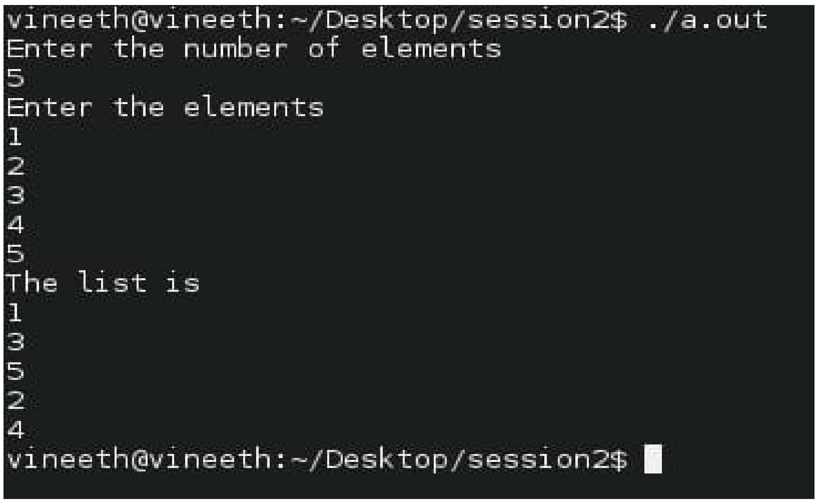
```
#include<stdio.h>
#include<stdlib.h>
int stack[100],top=-1,stk[100],top1=-1;
struct node
{
        int data;
        struct node *next;
}*head=NULL,*tail=NULL,*x=NULL;
int pop()
{
        int x;
        x=stack[top];
        top--;
        return(x);
}
int pop1()
{
        int x;
        x=stk[top1];
        top1--;
        return(x);
}
```

```
void push(int element)
{
            top++;
            stack[top]=element;
}
void push1(int element)
{
            top1++;
            stk[top1]=element;
}
void insend(int ele)
{
           struct node *nn;
           nn=(struct node*)malloc(sizeof(struct node));
           nn->data=ele;
           nn->next=NULL;
           if(head==NULL)
           {
                      head=nn;
                      tail=nn;
           }
           else
           {
                      tail->next=nn;
                      tail=nn;
           }
}
void display(int n)
{
           struct node *i;
           if(head==NULL)
           {
                      printf("NO LIST\n");
           }
           else
           {
                      printf("The list is\n");
                      for(i=head;i!=tail;i=i->next)
                                 printf("%d\n",i->data);
                      printf("%d\n",tail->data);
                      if(n%2==0)
                                 printf("%d\n",x->data);
           }
}
void arrange(int n)
{
           int i;
```

```
	struct node *temp;
	temp=head;
	if(n%2!=0)
	{
		for(i=0;i<n/2;i++)
		{
			push(temp->next->data);
			temp->next=temp->next->next;
			temp=temp->next;
		}
	}
	else
	{
		while(temp->next->next!=NULL)
		{
			push(temp->next->data);
			temp->next=temp->next->next;
			temp=temp->next;
		}
	}
	x=tail;
	tail=temp;
	while(top>-1)
		push1(pop());
	while(top1>-1)
		insend(pop1());
	}
	void main()
	{
		int i,n;
		printf("Enter the number of elements\n");
		scanf("%d",&n);
		int k;
		printf("Enter the elements\n");
		for(i=0;i<n;i++)
		{
			scanf("%d",&k);
			insend(k);
		}
		arrange(n);
		display(n);
	}
```

DATA STRUCTURE USED: Stack, Singly Linked list

OUTPUT:



```
vineeth@vineeth:~/Desktop/session2$ ./a.out
Enter the number of elements
5
Enter the elements
1
2
3
4
5
The list is
1
3
5
2
4
vineeth@vineeth:~/Desktop/session2$
```

RESULT:

Thus the program that to find the missing number in the given two lists was is successfully executed and verified.

QUESTION:

Given two linked lists, merge their nodes together into first list by taking nodes alternately between the two lists. If first list runs out, remaining nodes of second list should not be moved.

For example, consider lists `{1, 2, 3}` and `{4, 5, 6, 7, 8}`. Merging them should result in `{1, 4, 2, 5, 3, 6}` and `{7, 8}` respectively.

Ex. No: 2.2b

MERGING THE GIVEN LINKED LIST IN AN ORDER

Date:

AIM:

To write a c program that merges the nodes of two linked list by taking the nodes alternatively.

PSEUDOCODE:

```
//Program: To merge the nodes of two linked lists alternatively.
//Input: Elements for the two linked list.
//Output: Merged alternative list.
  BEGIN

  Read u
  FOR i->0 to i-> u-1
          Read  data
          head1=insert_end(head1,data)
ENDFOR
 Read u
 FOR i->0 to i-> u-1
          Read  data
          Head2=insert_end(head2,data)
ENDFOR
Display(head1)
DisplayList(head2)
temp1=head1
temp2=head2
WHILE(temp1!=NULL && temp2!=NULL)
          head3=insert_end(head3,temp1->data)
          head3=insert_end(head3,temp2->data)
          temp1=temp1->next
          temp2=temp2->nexT
          head1=delete_begin(head1)
          head2=delete_begin(head2)
ENDWHILE
displayList(head3)
displayList(head1)
displayList(head2)

END
```

SOURCE CODE:

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

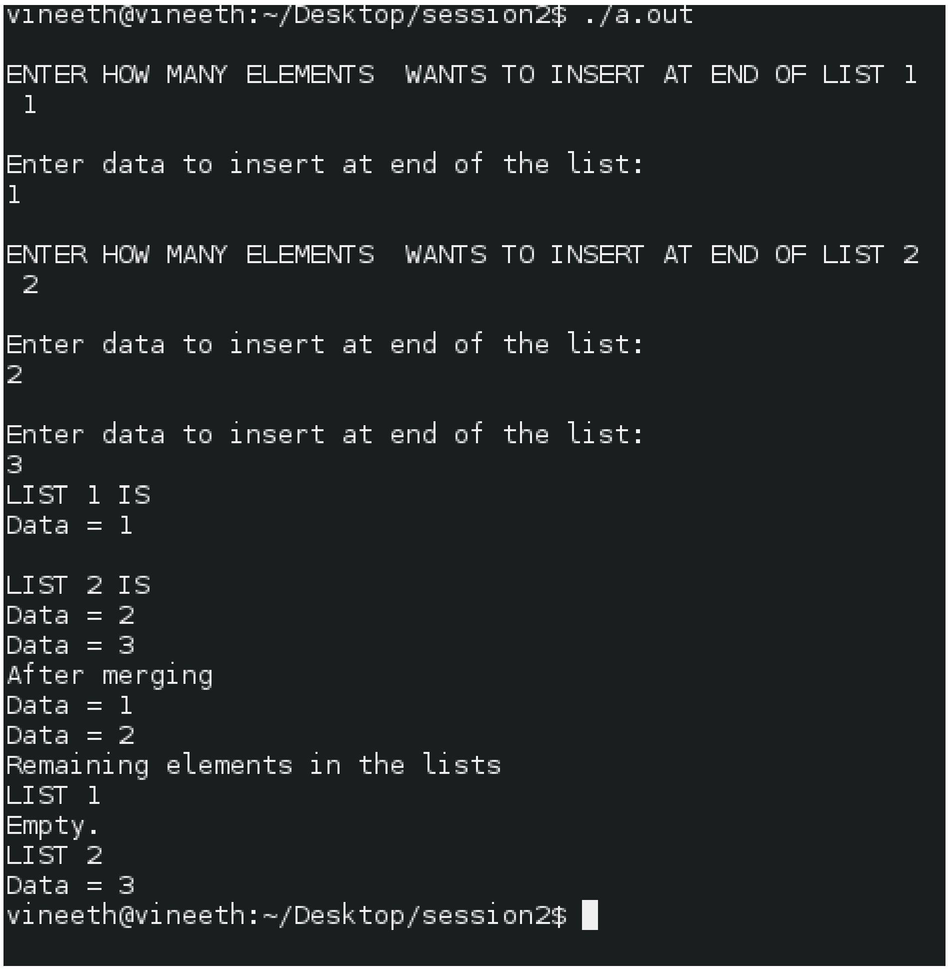
```
struct node
{
        int data;
        struct node *next;
}*head1=NULL,*head2=NULL,*head3=NULL;
struct node* delete_begin(struct node *hed)
{
        if(hed==NULL)
                printf("EMPTY LINKED LIST");
        else
        {
                struct node *delnode;
                delnode=hed;
                hed=hed->next;
                free(delnode);
        }
        return hed;
}
void displayList(struct node *hed)
{
   struct node *temp;
   if(hed== NULL)
   {
           printf("Empty.\n");
   }
   else
   {
      temp = hed;
      while(temp != NULL)
      {
              printf("Data = %d\n", temp->data);
              temp = temp->next;
      }
   }
}
struct node* insert_end(struct node *hed,int data)
{
   struct node *newNode,*temp;
   newNode = (struct node*)malloc(sizeof(struct node));
   newNode->data=data;
   newNode->next=NULL;
   if(hed == NULL)
   {
           hed=newNode;
   }
   else
   {
```

```
            temp=hed;
            while(temp->next!=NULL)
                temp=temp->next;
            temp->next=newNode;
    }
    return hed;
}
void main()
{
        struct node *temp1,*temp2;
         int n, data,u,i,a,pos;
        printf("\nENTER HOW MANY ELEMENTS  WANTS TO INSERT AT END OF
        LIST 1\n ");
        scanf("%d", &u);
        for (i=0;i<u;i++)
        {
                printf("\nEnter data to insert at end of the list: \n");
                scanf("%d", &data);
                head1=insert_end(head1,data);
        }
        printf("\nENTER HOW MANY ELEMENTS  WANTS TO INSERT AT END OF
        LIST 2\n ");
        scanf("%d", &u);
        for (i=0;i<u;i++)
        {
                printf("\nEnter data to insert at end of the list: \n");
                scanf("%d", &data);
                head2=insert_end(head2,data);
        }
        printf("LIST 1 IS \n");
        displayList(head1);
        printf("\nLIST 2 IS \n");
        displayList(head2);
        printf("After merging\n");
        temp1=head1;
        temp2=head2;
        while(temp1!=NULL && temp2!=NULL)
        {
                head3=insert_end(head3,temp1->data);
                head3=insert_end(head3,temp2->data);
                temp1=temp1->next;
                temp2=temp2->next;
                head1=delete_begin(head1);
                head2=delete_begin(head2);
        }
        displayList(head3);
        printf("Remaining elements in the lists\n");
```

```
            printf("LIST 1\n");
            displayList(head1);
            printf("LIST 2\n");
            displayList(head2);
    }
```

DATA STRUCTURE USED: Linked list

OUTPUT:



```
vineeth@vineeth:~/Desktop/session2$ ./a.out

ENTER HOW MANY ELEMENTS  WANTS TO INSERT AT END OF LIST 1
 1

Enter data to insert at end of the list:
1

ENTER HOW MANY ELEMENTS  WANTS TO INSERT AT END OF LIST 2
 2

Enter data to insert at end of the list:
2

Enter data to insert at end of the list:
3
LIST 1 IS
Data = 1

LIST 2 IS
Data = 2
Data = 3
After merging
Data = 1
Data = 2
Remaining elements in the lists
LIST 1
Empty.
LIST 2
Data = 3
vineeth@vineeth:~/Desktop/session2$
```

RESULT:

Thus the program that to find out whether any two different elements of the array , sum to the number was successfully executed and verified

QUESTION:

Given an linked list of integers, rearrange it such that every second node of the linked list is greater than its left and right nodes. In other words, rearrange linked list node in alternating high-low.

Assume no duplicate nodes are present in the linked list. Several lists might satisfies the constraints, we need to print any one of them. For example,

```
Input:  1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7
Output: 1 -> 3 -> 2 -> 5 -> 4 -> 7 -> 6
```

```
Input:  9 -> 6 -> 8 -> 3 -> 7
Output: 6 -> 9 -> 3 -> 8 -> 7
```

```
Input:  6 -> 9 -> 2 -> 5 -> 1 -> 4
Output: 6 -> 9 -> 2 -> 5 -> 1 -> 4
```

Ex. No: 2.3a

ARRANGING LINKED LIST IN ORDER HIGH-LOW

Date:

AIM:

To write a c program that rearranges the linked list of integers such that every second node of a linked list is greater than its left and right nodes.

PSEUDOCODE:

```
//Program: Arranging the linked list in alternating high-low.
//Input: Elements for the linked list.
//Output: Arranged linked list in an alternating high-low order.
  BEGIN
        Read  u
        FOR i←0 to i<u
              FOR j←i+1 to j<u
                    IF(arr[i]>arr[j])
                          a=arr[i]
                          arr[i]=arr[j]
                          arr[j]=a
                    ENDIF
              ENDFOR
        ENDFOR
        FOR i←1 to i<u&&i+1<u
              a=arr[i]
              arr[i]=arr[i+1]
              arr[i+1]=a
        FOR i←0 to i<u
              head2=insert_end(head2,arr[i]);
        displayList(head2);
END
```

SOURCE CODE:

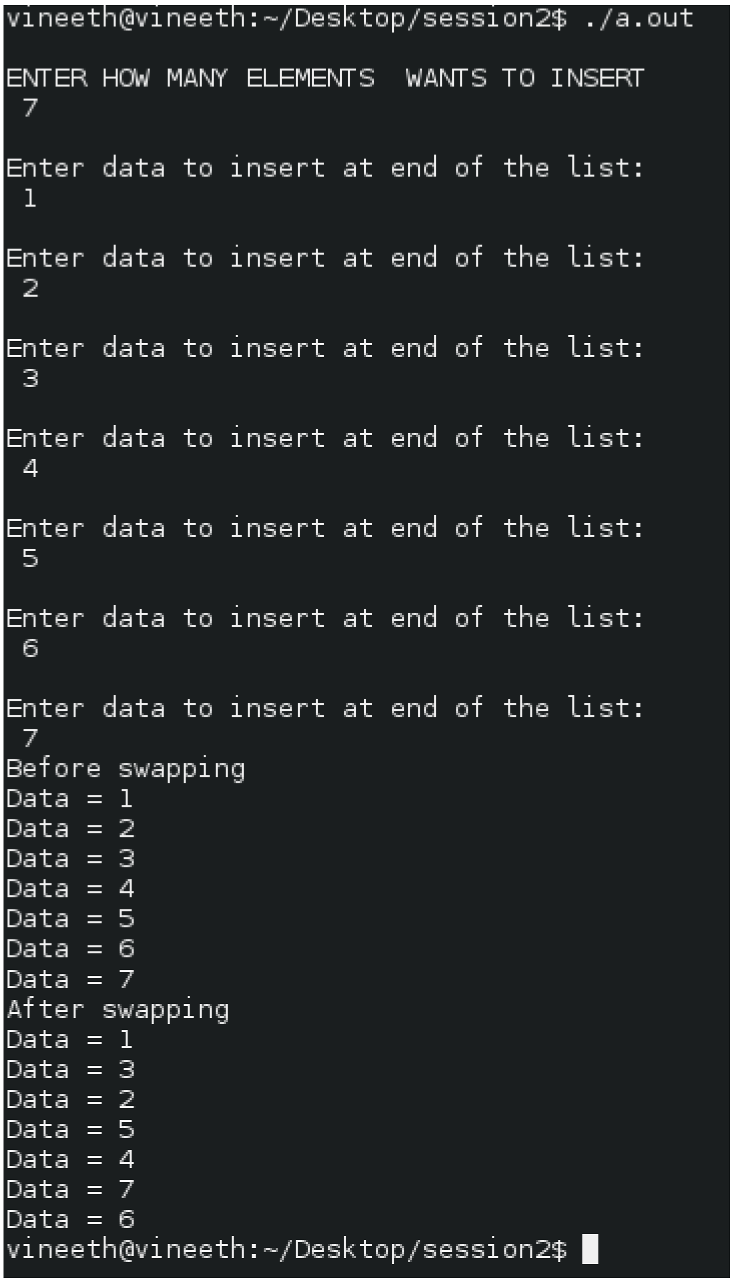
```
#include <stdio.h>
#include <stdlib.h>
int arr[100],z=0;
struct node
{
   int data;
   struct node *next;
}*head1=NULL,*head2=NULL;
void array(struct node *hed)
{
```

```
    struct node *temp;
    if(hed == NULL)
    {
       printf("List is empty.");
    }
    else
    {
          temp = hed->next;
          arr[z++]=hed->data;
       while(temp != NULL)
       {
          arr[z++]=temp->data;
          temp = temp->next;
       }
    }
}

void displayList(struct node *hed)
{
    struct node *temp;
    if(hed== NULL)
    {
       printf("Empty.\n");
    }
    else
    {
       temp = hed;
       while(temp != NULL)
       {
          printf("Data = %d\n", temp->data);
          temp = temp->next;
       }
    }
}
struct node* insert_end(struct node *hed,int data)
{
    struct node *newNode,*temp;
    newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data=data;
    newNode->next=NULL;
    if(hed == NULL)
    {
          hed=newNode;
    }
    else
    {
      temp=hed;
```

```
    while(temp->next!=NULL)
        temp=temp->next;
    temp->next=newNode;
   }
return hed;
}

void main()
{
        int n, data,u,i,j,a,pos;
        printf("\nENTER HOW MANY ELEMENTS  WANTS TO INSERT\n ");
        scanf("%d", &u);
        for (i=0;i<u;i++)
        {
                printf("\nEnter data to insert at end of the list:\n ");
                scanf("%d", &data);
                head1=insert_end(head1,data);
        }
        printf("Before swapping\n");
        displayList(head1);
        printf("After swapping\n");
        array(head1);
        u=z;
        for(i=0;i<u;i++)
        {
                for(j=i+1;j<u;j++)
                {
                        if(arr[i]>arr[j])
                        {
                                a=arr[i];
                                arr[i]=arr[j];
                                arr[j]=a;
                        }
                }
        }
        for(i=1;i<u&&i+1<u;i=i+2)
        {
                a=arr[i];
                arr[i]=arr[i+1];
                arr[i+1]=a;
        }
        for(i=0;i<u;i++)
                head2=insert_end(head2,arr[i]);
        displayList(head2);
}
```

DATA STRUCTURE USED: Array , Linked list

OUTPUT:



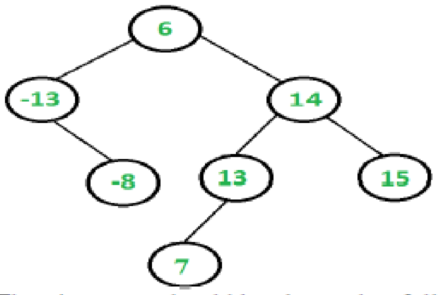
```
vineeth@vineeth:~/Desktop/session2$ ./a.out

ENTER HOW MANY ELEMENTS  WANTS TO INSERT
 7

Enter data to insert at end of the list:
 1

Enter data to insert at end of the list:
 2

Enter data to insert at end of the list:
 3

Enter data to insert at end of the list:
 4

Enter data to insert at end of the list:
 5

Enter data to insert at end of the list:
 6

Enter data to insert at end of the list:
 7
Before swapping
Data = 1
Data = 2
Data = 3
Data = 4
Data = 5
Data = 6
Data = 7
After swapping
Data = 1
Data = 3
Data = 2
Data = 5
Data = 4
Data = 7
Data = 6
vineeth@vineeth:~/Desktop/session2$ 
```

RESULT:

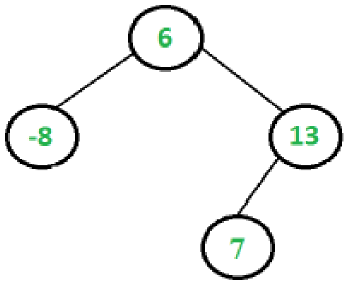
Thus the program to alternate the nodes in the given linked list in an alternative high-low order has been determined and verified.

QUESTION:

Given a Binary Search Tree (BST) and a range [min, max], remove all keys which are outside the given range. The modified tree should also be BST. For example, consider the following BST and range [-10, 13].



The given tree should be changed to following. Note that all keys outside the range [-10, 13] are removed and modified tree is BST.



Ex. No: 2.3b

REMOVING ELEMENTS IN THE TREE

Date:

AIM:

To write a c program that removes elements in a tree which are out of the given range.

PSEUDOCODE:

```
//Program: To remove elements which are out of given range.
//Input: Elements of the tree,Range.
//Output: Resultant BST .
  BEGIN
        Read the no of elements n
        FOR i←0 to i<n
         Read ele[i]
     root=insert(root,ele[i])
        ENDFOR
        FOR i←0 to i<n
                IF(ele[i]<-10 || ele[i] > 13)
                        root=delete(root,ele[i]);
                ENDIF
        ENDFOR
  END
```

SOURCE CODE:

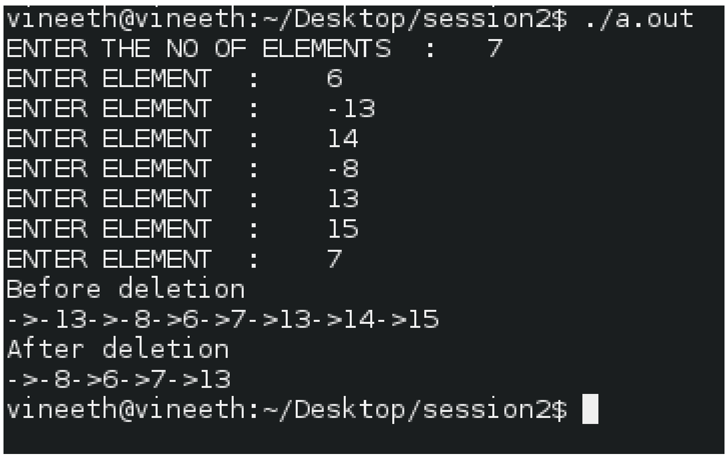
```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node *left,*right;
}*root=NULL;
struct node* minimum(struct node *temp)
{
         if(temp==NULL)
                printf("the tree is empty");
        else
        {
                while(temp->left!=NULL)
                {
                        temp=temp->left;
                }
        }
```

```
        return temp;
}
struct node* delete(struct node *root,int ele)
{
        if(root==NULL)
                return NULL;
        if(ele<root->data)
                root->left=delete(root->left,ele);
        else if(ele>root->data)
                root->right=delete(root->right,ele);
        else
        {
                if(root->left==NULL && root->right==NULL)
                {
                        free(root);
                        root=NULL;
                }
                else if(root->left==NULL)
                {
                        struct node *temp=root;
                        root=root->right;
                        free(temp);
                }
                else if(root->right==NULL)
                {
                        struct node *temp=root;
                        root=root->left;
                        free(temp);
                }
                else
                {
                        struct node *temp=minimum(root->right);
                        root->data=temp->data;
                        root->right=delete(root->right,temp->data);
                }
        }
        return root;
}
struct node *insert(struct node *root,int ele)
{
         if(root==NULL)
         {
           root=(struct node *)malloc(sizeof(struct node));
           root->data=ele;
           root->left=NULL;
           root->right=NULL;
          }
```

```
            else if(ele<root->data)
                  root->left=insert(root->left,ele);
            else if(ele>root->data)
                 root->right=insert(root->right,ele);
           return root;
    }
    void inorder(struct node *root)
     {
           if(root!=NULL)
           {
                  inorder(root->left);
                  printf("->%d",root->data);
                  inorder(root->right);
           }
     }
    void main()
    {
           int n,i,num,elem;
           printf("ENTER THE NO OF ELEMENTS");
           scanf("\n%d",&n);
           int ele[n];
           for(i=0;i<n;i++)
              {
                    printf("ENTER ELEMENT          :          ");
                    scanf("%d",&ele[i]);
                    root=insert(root,ele[i]);
              }
           printf("Before deletion\n");
           inorder(root);
           for(i=0;i<n;i++)
           {
              if(ele[i]<-10 || ele[i] > 13)
                    root=delete(root,ele[i]);
            }

           printf("\nAfter deletion \t\n");
           inorder(root);
    }
```

DATA STRUCTURE USED: Array , Tree

OUTPUT:



```
vineeth@vineeth:~/Desktop/session2$ ./a.out
ENTER THE NO OF ELEMENTS  :   7
ENTER ELEMENT  :    6
ENTER ELEMENT  :    -13
ENTER ELEMENT  :    14
ENTER ELEMENT  :    -8
ENTER ELEMENT  :    13
ENTER ELEMENT  :    15
ENTER ELEMENT  :    7
Before deletion
->-13->-8->6->7->13->14->15
After deletion
->-8->6->7->13
vineeth@vineeth:~/Desktop/session2$
```

RESULT:

Thus the program that gives the tree within the given range has been determined and executed.

QUESTION:

Given a Binary Tree, find the maximum sum path from a leaf to root. For example, in the following tree, there are three leaf to root paths 8->-2->10, -4->-2->10 and 7->10. The sums of these three paths are 16, 4 and 17 respectively. The maximum of them is 17 and the path for maximum is 7->10.

```
          10
        /    \
     -2        7
    /  \
  8     -4
```

Ex. No: 2.4a

Finding the maximum sum of the path

Date:

AIM:

To write a c program that gives the maximum sum of the path from root to leaf.

PSEUDOCODE:

```
//Program: To find the sum of maximum sum of path from root to leaf.
//Input: The Elements of the tree.
//Output: Maximum  sum  from  root  to  leaf.
 BEGIN
        Read  the  size  n
        FOR i←0 to i<n
                Read element
                root=insert(root,element)
        ENDFOR
        pathleaf(root,0);
        k=arr[0];
        FOR i←1 to arr[i]!='\0'
                IF(arr[i]>k)
                        k=arr[i]
                ENDIF
        ENDFOR
        DISPALY k
  END
```

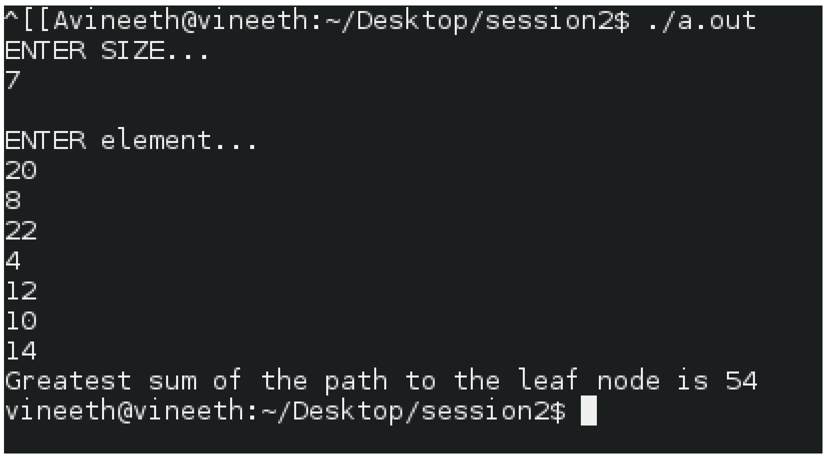
SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
int arr[100],x=0;
struct node
{
        int data;
        struct node *left,*right;
}*root=NULL;
struct node *insert(struct node *root,int element)
{
         if(root==NULL)
           {
                   root=(struct node*)malloc(sizeof(struct node));
                   root->data=element;
                   root->left=NULL;
                   root->right=NULL;
```

```
               }
              else if(element<root->data)
                     root->left=insert(root->left,element);
              else if(element>root->data)
                     root->right=insert(root->right,element);
              return root;
       }
       void pathleaf(struct node *root,int val)
       {
              if(root!=NULL)
               {
                     if((root->left==NULL)&&(root->right==NULL))
                     {
                            arr[x]=val+root->data;
                            x++;
                     }
                     else
                            val=val+root->data;
                     pathleaf(root->left,val);
                     pathleaf(root->right,val);
               }
       }
       void main()
       {
             printf("ENTER SIZE...\n");
             int n,k=0;
             scanf("%d",&n);
             int i,element;
             printf("\nENTER element...\n");
             for(i=0;i<n;i++)
             {
                    scanf("%d",&element);
                    root=insert(root,element);
             }
             pathleaf(root,0);
             k=arr[0];
             for(i=1;arr[i]!='\0';i++)
             {
                    if(arr[i]>k)
                           k=arr[i];
             }
             printf("Greatest sum of the path to the leaf node is %d\n",k);
       }
```

DATA STRUCTURE USED: Tree

OUTPUT:



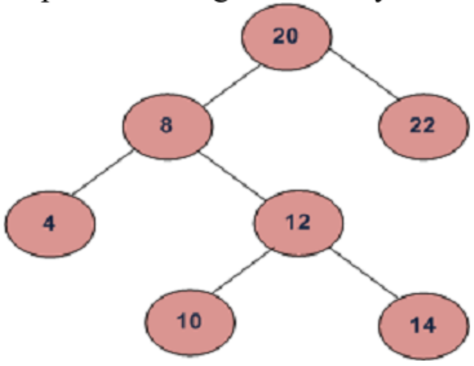
```
^[[Avineeth@vineeth:~/Desktop/session2$ ./a.out
ENTER SIZE...
7

ENTER element...
20
8
22
4
12
10
14
Greatest sum of the path to the leaf node is 54
vineeth@vineeth:~/Desktop/session2$ 
```

RESULT:

Thus the program to find the greatest sum from root to leaf node has been displayed and verified.

QUESTION:

Implement the given Binary search tree and Find the node with minimum value.



Ex. No: 2.4b

Finding the node with minimum value

Date:

AIM:

To write a c program to find the node with the minimum value.

PSEUDOCODE:

```
    //Program: To find the node with minimum value.
    //Input: Elements of the tree.
    //Output: Node with minimum value.
  BEGIN
         Read the size N
      FOR i←0 to i<n
                  Read ele[i]
                  root=insert(root,ele[i]);
      ENDFOR
            Display minimum(root)

  END
```

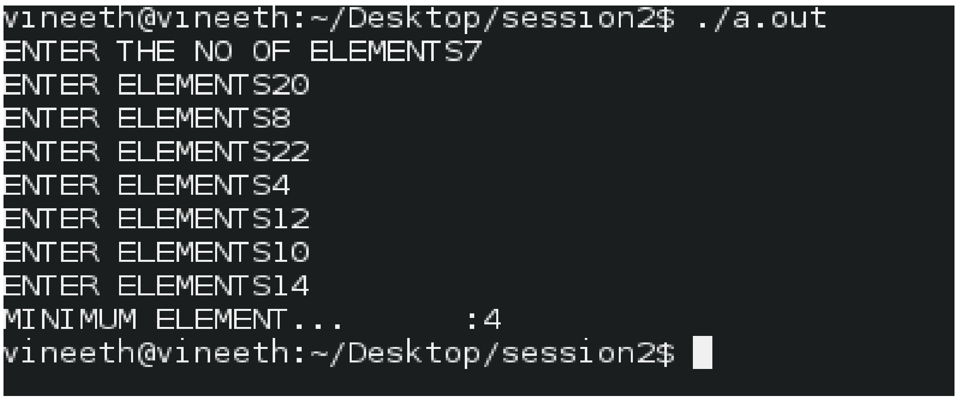
SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node *left,*right;
}*root=NULL;
int minimum(struct node *temp)
{
         if(temp==NULL)
                printf("the tree is empty");
        else
        {
                while(temp->left!=NULL)
                {
                        temp=temp->left;
                }
        }
        return temp->data;
}
struct node *insert(struct node *root,int ele)
{
         if(root==NULL)
```

```
    {
            root=(struct node *)malloc(sizeof(struct node));
            root->data=ele;
            root->left=NULL;
            root->right=NULL;
    }
 else if(ele<root->data)
        root->left=insert(root->left,ele);
 else if(ele>root->data)
         root->right=insert(root->right,ele);
return root;
}
void main()
{
        int n,i,num,elem;
        printf("ENTER THE NO OF ELEMENTS");
        scanf("\n%d",&n);
        int ele[n];
        for(i=0;i<n;i++)
           {
                printf("ENTER ELEMENTS");
                scanf("%d",&ele[i]);
                root=insert(root,ele[i]);
           }
        printf("MINIMUM ELEMENT...\t:%d\n",minimum(root));
}
```

DATA STRUCTURE USED: Tree

OUTPUT:

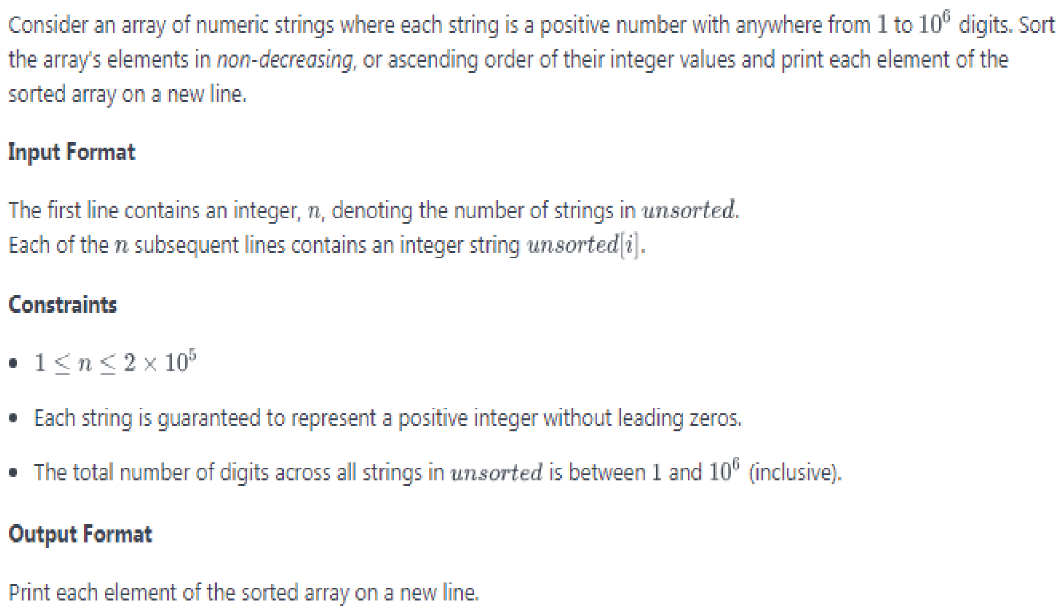


```
vineeth@vineeth:~/Desktop/session2$ ./a.out
ENTER THE NO OF ELEMENTS7
ENTER ELEMENTS20
ENTER ELEMENTS8
ENTER ELEMENTS22
ENTER ELEMENTS4
ENTER ELEMENTS12
ENTER ELEMENTS10
ENTER ELEMENTS14
MINIMUM ELEMENT...      :4
vineeth@vineeth:~/Desktop/session2$
```

RESULT:

Thus the program to find the node with minimum value has been determined and executed.

QUESTION:



Consider an array of numeric strings where each string is a positive number with anywhere from 1 to 10^6 digits. Sort the array's elements in *non-decreasing*, or ascending order of their integer values and print each element of the sorted array on a new line.

Input Format

The first line contains an integer, n, denoting the number of strings in $unsorted$.
Each of the n subsequent lines contains an integer string $unsorted[i]$.

Constraints

- $1 \leq n \leq 2 \times 10^5$
- Each string is guaranteed to represent a positive integer without leading zeros.
- The total number of digits across all strings in $unsorted$ is between 1 and 10^6 (inclusive).

Output Format

Print each element of the sorted array on a new line.

Ex. No: 3.1a

SORTING THE STRINGS

Date:

AIM:

To sort the array elements in non-decreasing and ascending order of their integer values and print each of the sorted array.

PSEUDOCODE:

```
//Program: To sort the given array
//Input: The unsorted array
//Output: The sorted array
BEGIN
 FOR i = num / 2 to i > 0
      FOR j = i to j < num
          FOR k <- j – i to k >= 0
              IF (atoi(arr[k+i]) >= atoi(arr[k]))
                  break
              ELSE
                  strcpy(tmp,arr[k])
                  strcpy(arr[k],arr[k+i])
                  strcpy(arr[k+i],tmp)
            ENDIF
          ENDFOR
      ENDFOR
  ENDFOR
 END
```

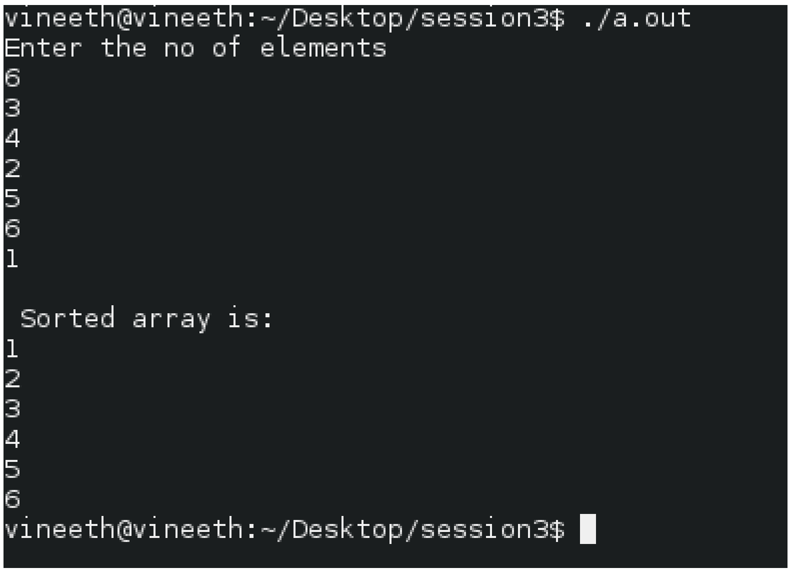
SOURCE CODE:

```
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
void shellsort(char arr[][7], int num)
{
        int i, j, k;
        char tmp[7];
        for (i = num / 2; i > 0; i = i / 2)
        {
                for (j = i; j < num; j++)
                {
                        for(k = j - i; k >= 0; k = k - i)
                        {
                                if (atoi(arr[k+i]) >= atoi(arr[k]))
                                        break;
```

```
                                else
                                {
                                    strcpy(tmp,arr[k]);
                                    strcpy(arr[k],arr[k+i]);
                                    strcpy(arr[k+i],tmp);
                                }
                        }
                    }
            }
    }
    int main()
    {
        int n;
        printf("Enter the no of elements\n");
        scanf("%d",&n);
        char arr[n][7];
        int k;
        for (k =  0 ; k < n; k++)
        {
            scanf("%s",arr[k]);
        }
        shellsort(arr, n);
        printf("\n Sorted array is: \n");
        for (k = 0; k < n; k++)
            printf("%s\n", arr[k]);
        return 0;
    }
```

DATA STRUCTURE USED: Array

OUTPUT:

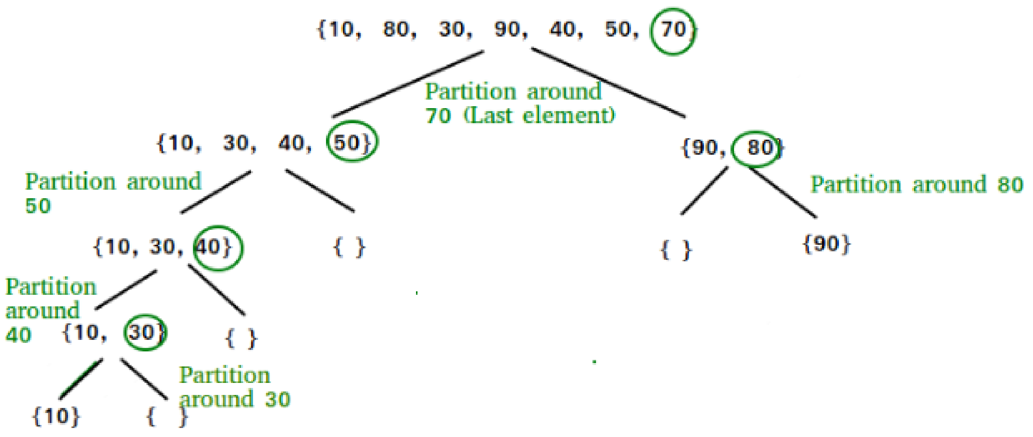


```
vineeth@vineeth:~/Desktop/session3$ ./a.out
Enter the no of elements
6
3
4
2
5
6
1

 Sorted array is:
1
2
3
4
5
6
vineeth@vineeth:~/Desktop/session3$ 
```

RESULT:

Thus the program that sorts the given array is sorted.

QUESTION:


{10, 80, 30, 90, 40, 50, 70}
Partition around 70 (Last element)
{10, 30, 40, 50}
{90, 80}
Partition around 50
Partition around 80
{10, 30, 40}
{ }
{ }
{90}
Partition around 40
{10, 30}
{ }
Partition around 30
{10}
{ }

Ex. No: 3.1b

Date:

Sorting the elements using Quick sort

AIM:

To sort the elements using the quick sort.

PSEUDOCODE:

```
//Program: To sort the elements using the quick sort.
//Input: Size n, Array of size n
//Output: Sorted array
BEGIN
  IF (low < high)
      pivot = low
      i = low
      j = high
      WHILE (i < j)
          WHILE (list[i] <= list[pivot] && i <= high)
              i++
          ENDWHILE
          WHILE (list[j] > list[pivot] && j >= low)
                j--
          ENDWHILE
          IF (i < j)
              temp = list[i]
              list[i] = list[j]
              list[j] = temp
          ENDIF
      ENDWHILE
      temp = list[j]
      list[j] = list[pivot]
      list[pivot] = temp
      quicksort(list, low, j - 1)
      quicksort(list, j + 1, high)
  ENDIF
END
```

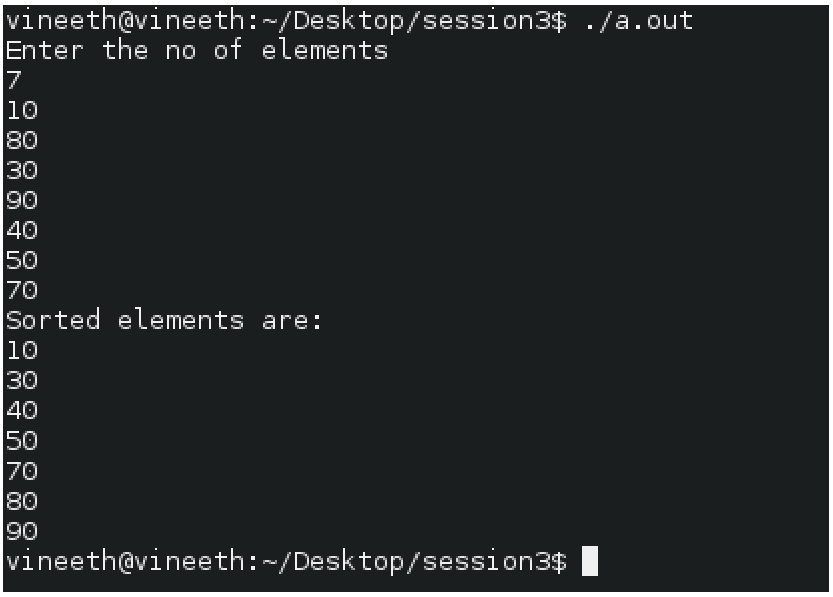
SOURCE CODE:

```
#include<stdio.h>
void quicksort(int list[], int low, int high)
{
   int pivot, i, j, temp;
   if (low < high)
   {
      pivot = low;
```

```
            i = low;
            j = high;
            while (i < j)
            {
               while (list[i] <= list[pivot] && i <= high)
               {
                  i++;
               }
               while (list[j] > list[pivot] && j >= low)
               {
                  j--;
               }
               if (i < j)
               {
                  temp = list[i];
                  list[i] = list[j];
                  list[j] = temp;
               }
            }
            temp = list[j];
            list[j] = list[pivot];
            list[pivot] = temp;
            quicksort(list, low, j - 1);
            quicksort(list, j + 1, high);
         }
      }
      void main()
      {
               int i,n;
               printf("Enter the no of elements\n");
               scanf("%d",&n);
               int arr[n];
               for(i=0;i<n;i++)
                        scanf("%d",&arr[i]);
               quicksort(arr,0,n-1);
               printf("Sorted elements are:\n");
               for(i=0;i<n;i++)
                        printf("%d\n",arr[i]);
      }
```

DATA STRUCTURE USED: Array

OUTPUT:

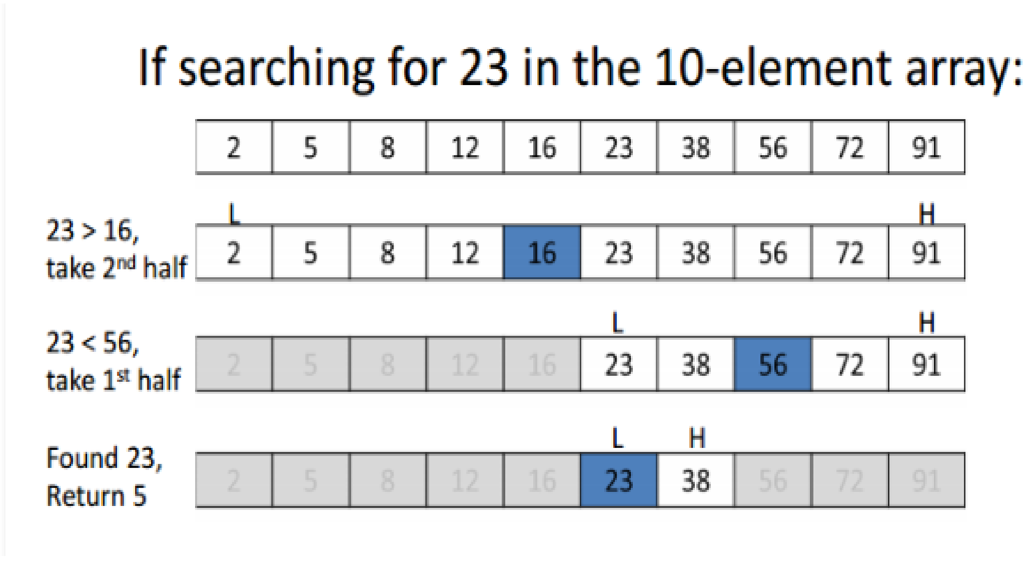


```
vineeth@vineeth:~/Desktop/session3$ ./a.out
Enter the no of elements
7
10
80
30
90
40
50
70
Sorted elements are:
10
30
40
50
70
80
90
vineeth@vineeth:~/Desktop/session3$ 
```

RESULT:

Thus the program to sort the given elements using Quick sort was sorted and displayed.

QUESTION:


If searching for 23 in the 10-element array:
2 5 8 12 16 23 38 56 72 91
23 > 16, take 2nd half
L
H
23 < 56, take 1st half
Found 23, Return 5

Ex. No: 3.2a

SEARCH THE GIVEN NUMBER IN ARRAY

Date:

AIM:

To write a c program search the given number in the array.

PSEUDOCODE:

```
  //Program: To search the given number in the array.
  //Input: An array.
  //Output:  The index of the element to be searched.
BEGIN

  IF(low<=high)
        mid=(low+high)/2
        IF(key==a[mid])
              return mid+1
        ELSE IF(key<a[mid])
              return binarysearch(a,low,mid-1,key)
        ELSE
              return binarysearch(a,mid+1,high,key)
        ENDIF
  ENDIF

 END
```

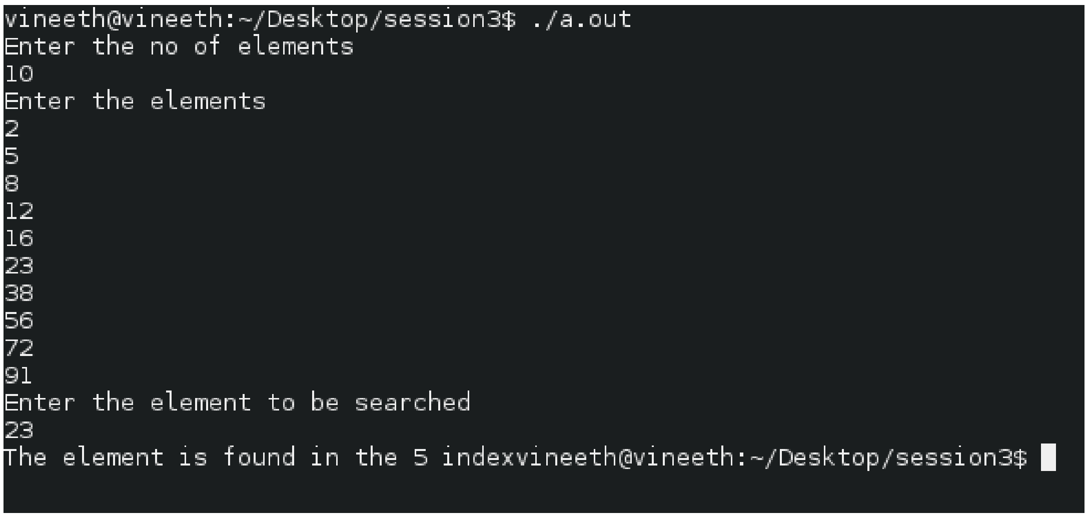
SOURCE CODE:

```
#include<stdio.h>
int binarysearch(int a[],int low,int high,int key)
{
      int mid;
      if(low<=high)
      {
            mid=(low+high)/2;
            if(key==a[mid])
                  return mid+1;
            else if(key<a[mid])
                  return binarysearch(a,low,mid-1,key);
            else
                  return binarysearch(a,mid+1,high,key);
      }
      return 0;
}
void main()
```

```
{
        int n,i,key,y;
        printf("Enter the no of elements\n");
        scanf("%d",&n);
        int arr[n];
        printf("Enter the elements\n");
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        printf("Enter the element to be searched\n");
        scanf("%d",&key);
        y=binarysearch(arr,0,n,key);
        if(y)
                printf("The element is found in the %d index",y-1);
        else
                printf("The element is not found");
}
```

DATA STRUCTURE USED: Array

OUTPUT:



```
vineeth@vineeth:~/Desktop/session3$ ./a.out
Enter the no of elements
10
Enter the elements
2
5
8
12
16
23
38
56
72
91
Enter the element to be searched
23
The element is found in the 5 indexvineeth@vineeth:~/Desktop/session3$ 
```

RESULT:

Thus the program that to search the element in the given array was found and it’s index is printed.

QUESTION:

Write a program for implementing the sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Ex. No: 3.2b

BUBBLE SORT

Date:

AIM:

To write a c program to implement the sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

PSEUDOCODE:

```
    //Program: To sort the given array.
    //Input: Array size, array
    //Output: Sorted array
  BEGIN
     FOR i=0 to i<=n-2
         FOR(j=0 to j<=n-i-2
             IF(a[j]>a[j+1])
                 temp=a[j]
                 a[j]=a[j+1]
                 a[j+1]=temp
             ENDIF
         ENDFOR
    ENDFOR
   END
```

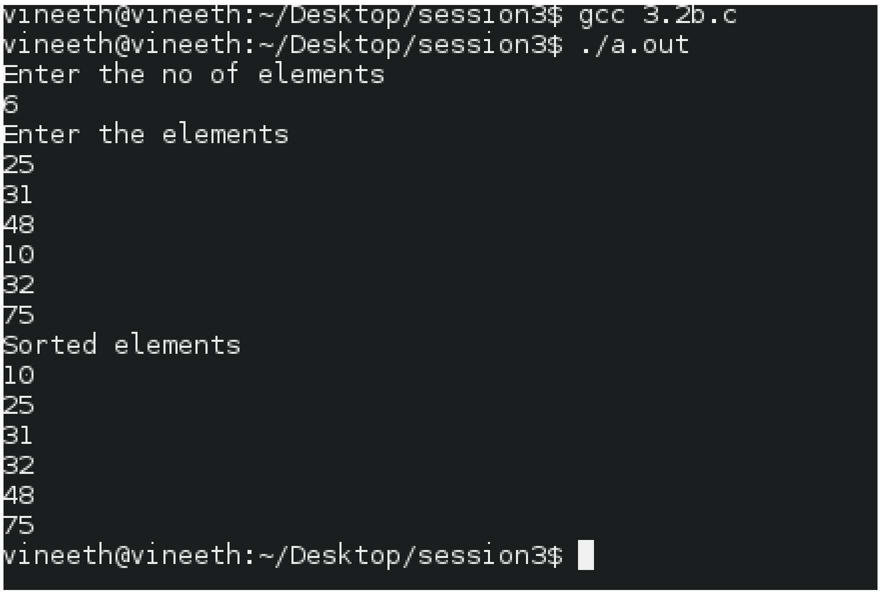
SOURCE CODE:

```
#include<stdio.h>
void bubble(int n,int a[])
{
    int i,j,temp;
    for(i=0;i<=n-2;i++)
    {
        for(j=0;j<=n-i-2;j++)
        {
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
}
```

```
void main()
{
        int n,i;
        printf("Enter the no of elements\n");
        scanf("%d",&n);
        int arr[n];
        printf("Enter the elements\n");
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        bubble(n,arr);
        printf("Sorted elements\n");
        for(i=0;i<n;i++)
                printf("%d\n",arr[i]);
}
```

DATA STRUCTURE USED: Array

OUTPUT:



```
vineeth@vineeth:~/Desktop/session3$ gcc 3.2b.c
vineeth@vineeth:~/Desktop/session3$ ./a.out
Enter the no of elements
6
Enter the elements
25
31
48
10
32
75
Sorted elements
10
25
31
32
48
75
vineeth@vineeth:~/Desktop/session3$
```

RESULT:

Thus the program that to find out whether any two different elements of the array , sum to the number was successfully executed and verified.

QUESTION:

Given a collection of integers, develop an algorithm to find the index of maximum occurring element with equal probability.

For example, consider the input: `{4, 3, 6, 8, 4, 6, 2, 4, 5, 9, 7, 4}`. The maximum occurring element, which happens to be `4`, occurs at index `0, 4, 7` and `11`. The solution should return any one of these indices with the equal probability.

Ex. No: 3.3a

INDEX OF THE MAXIMUM OCCURING ELEMENT

Date:

AIM:

To write a c program to find the index of the maximum occurring element.

PSEUDOCODE:

```
  //Program: To find the index of the maximum occurring element..
  //Input: Array
  //Output: Index of the maximum occurring element.
BEGIN

   FOR i=0 to i<n
          scanf("%d",&arr[i])
          hash[i]=0
  ENDFOR
  FOR i=0 to i<n
          hash[arr[i]-1]+=1
  ENDFOR
  k=0
  FOR i=1 to i<n
          IF(hash[k]<hash[i])
                  k=i
          ENDIF
   ENDFOR
 END
```

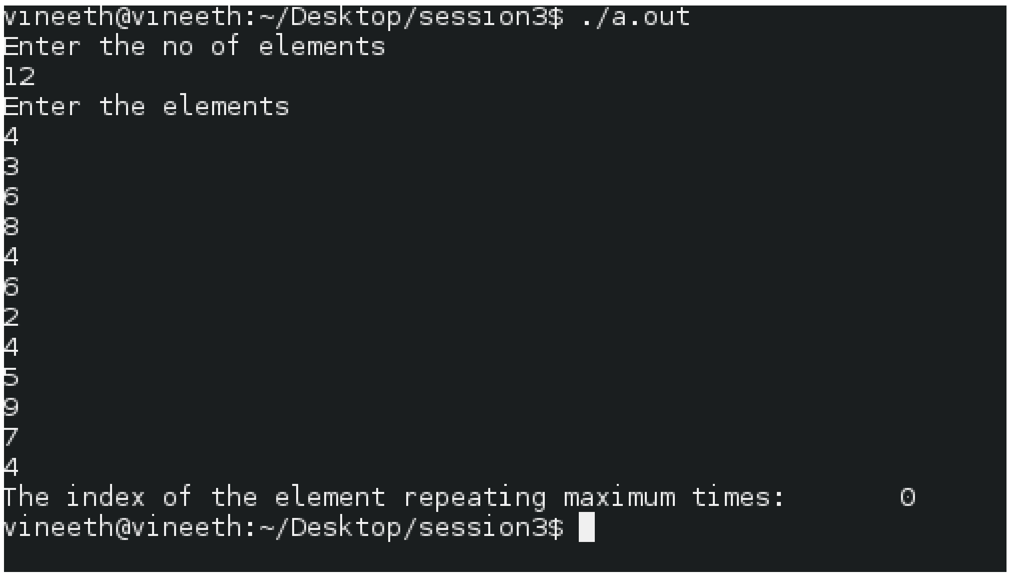
SOURCE CODE:

```
#include<stdio.h>
void main()
{
        int i,n,j,k;
        printf("Enter the no of elements\n");
        scanf("%d",&n);
        int arr[n],hash[n];
        printf("Enter the elements\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&arr[i]);
                hash[i]=0;
        }
        for(i=0;i<n;i++)
```

```
                        hash[arr[i]-1]+=1;
                k=0;
                for(i=1;i<n;i++)
                {
                        if(hash[k]<hash[i])
                        k=i;
                }
                printf("The index of the element repeating maximum times:\t");
                for(i=0;i<n;i++)
                {
                        if(arr[i]==k+1)
                        {
                                printf("%d\n",i);
                                break;
                        }
                }
        }
```

DATA STRUCTURE USED: Array

OUTPUT:



```
vineeth@vineeth:~/Desktop/session3$ ./a.out
Enter the no of elements
12
Enter the elements
4
3
6
8
4
6
2
4
5
9
7
4
The index of the element repeating maximum times:       0
vineeth@vineeth:~/Desktop/session3$ 
```

RESULT:

Thus the program that to print the names in lexicographical order is was successfully executed and verified.

QUESTION:

Given an array of integers, find minimum index of a repeating element in linear time and doing just one traversal of the array.

For example,

```
Input:  { 5, 6, 3, 4, 3, 6, 4 }
Output: Minimum index of repeating element is 1
```

```
Input:  { 1, 2, 3, 4, 5, 6 }
Output: Invalid Input
```

Ex. No: 3.3b

MINIMUM INDEX OF THE REPEATING ELEMENT

Date:

AIM:

To write a c program to find the minimum index of a repeating element in linear time.

PSEUDOCODE:

```
  //Program: To find the minimum index of a repeating element in linear time.
  //Input: Array
  //Output: Minimum index of a repeating element.
BEGIN

  FOR i=0 to i<n
        scanf("%d",&arr[i])
        hash[i]=0
  ENDFOR
  FOR i=0 to i<n
        hash[arr[i]-1]+=1
  ENDFOR
  k=0
  FOR i=1 to i<n
        IF(hash[i]>=2)
              k=i
              break
        ENDIF
  ENDFOR

END
```

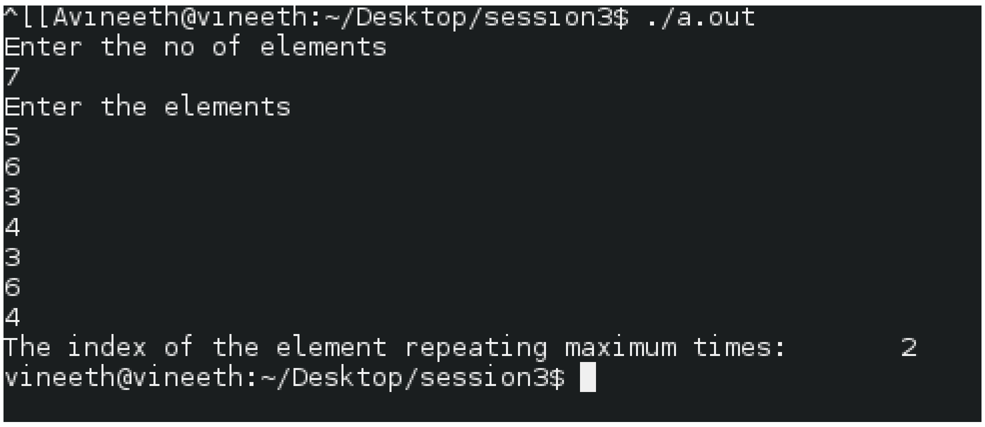
SOURCE CODE:

```
#include<stdio.h>
void main()
{
      int i,n,j,k;
      printf("Enter the no of elements\n");
      scanf("%d",&n);
      int arr[n],hash[n];
      printf("Enter the elements\n");
      for(i=0;i<n;i++)
      {
            scanf("%d",&arr[i]);
            hash[i]=0;
      }
```

```
		for(i=0;i<n;i++)
			hash[arr[i]-1]+=1;
		k=0;
		for(i=1;i<n;i++)
		{
			if(hash[i]>=2)
			{
				k=i;
				break;
			}
		}
		printf("The index of the element repeating maximum times:\t");
		for(i=0;i<n;i++)
		{
			if(arr[i]==k+1)
			{
				printf("%d\n",i);
				break;
			}
		}
	}
```

DATA STRUCTURE USED: Array

OUTPUT:



```
^[[Avineeth@vineeth:~/Desktop/session3$ ./a.out
Enter the no of elements
7
Enter the elements
5
6
3
4
3
6
4
The index of the element repeating maximum times:       2
vineeth@vineeth:~/Desktop/session3$ 
```

RESULT:

Thus the program for the scenario that to find whether Rick died or not was successfully executed and verified.

QUESTION:

Write a program to implement linear probing.

Ex. No: 3.4a

LINEAR PROBING

Date:

AIM:

To write a c program to implement linear probing.

PSEUDOCODE:

```
    //Program: To mplement linear probing.
    //Input: Array
    //Output: Hash table
  BEGIN

    FOR i←0 to i<=n-1
        Read x
        j←0
        WHILE(1)
            hashkey←(((x%tablesize)+j)%tablesize)
            IF(hash[hashkey]==-1)
                hash[hashkey]=x
                break
            ELSE
                j++
            ENDIF
        ENDWHILE
  END
```

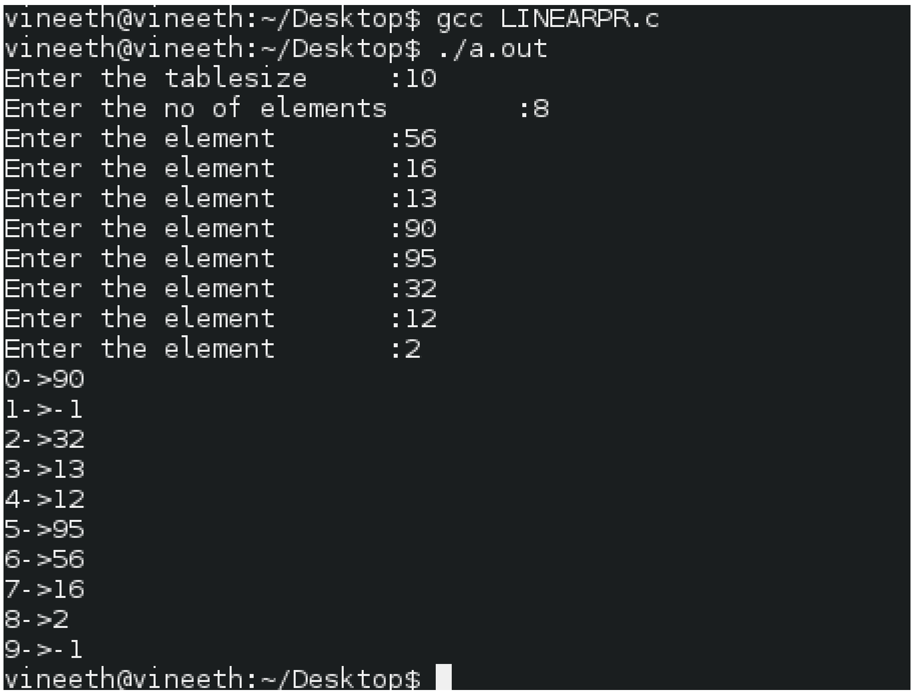
SOURCE CODE:

```
#include<stdio.h>
void linearprobing(int tablesize,int hash[])
{
    int i,n,x,j,hashkey;
    printf("Enter the no of elements\t:");
    scanf("%d",&n);
    for(i=0;i<=n-1;i++)
    {
        printf("Enter the element\t:");
        scanf("%d",&x);
        j=0;
        while(1)
        {
            hashkey=(((x%tablesize)+j)%tablesize);
            if(hash[hashkey]==-1)
            {
                hash[hashkey]=x;
                break;
```

```
                                }
                                else
                                        j++;
                        }
                }
                for(i=0;i<=tablesize-1;i++)
                        printf("%d->%d\n",i,hash[i]);
        }
        void main()
        {
                int i,tablesize;
                printf("Enter the tablesize\t:");
                scanf("%d",&tablesize);
                int hash[tablesize];
                for(i=0;i<=tablesize-1;i++)
                        hash[i]=-1;
                linearprobing(tablesize,hash);
        }
```

DATA STRUCTURE USED: Array

OUTPUT:



```
vineeth@vineeth:~/Desktop$ gcc LINEARPR.c
vineeth@vineeth:~/Desktop$ ./a.out
Enter the tablesize     :10
Enter the no of elements        :8
Enter the element       :56
Enter the element       :16
Enter the element       :13
Enter the element       :90
Enter the element       :95
Enter the element       :32
Enter the element       :12
Enter the element       :2
0->90
1->-1
2->32
3->13
4->12
5->95
6->56
7->16
8->2
9->-1
vineeth@vineeth:~/Desktop$
```

RESULT:

Thus the program for the implementation of linear probing was successfully executed and verified.

QUESTION:

Write a program to implement Quadratic probing.

Ex. No: 3.4b

QUADRATRIC PROBING

Date:

AIM:

To write a c program to implement quadratic probing.

PSEUDOCODE:

```
  //Program: To mplement quadratic probing.
  //Input: Array
  //Output: Hash table
BEGIN

  FOR i←0 to i<=n-1
      Read x
      j←0
      WHILE(1)
          hashkey←(((x%tablesize)+j*j)%tablesize)
          IF(hash[hashkey]==-1)
              hash[hashkey]=x
              break
          ELSE
              j++
          ENDIF
      ENDWHILE
END
```

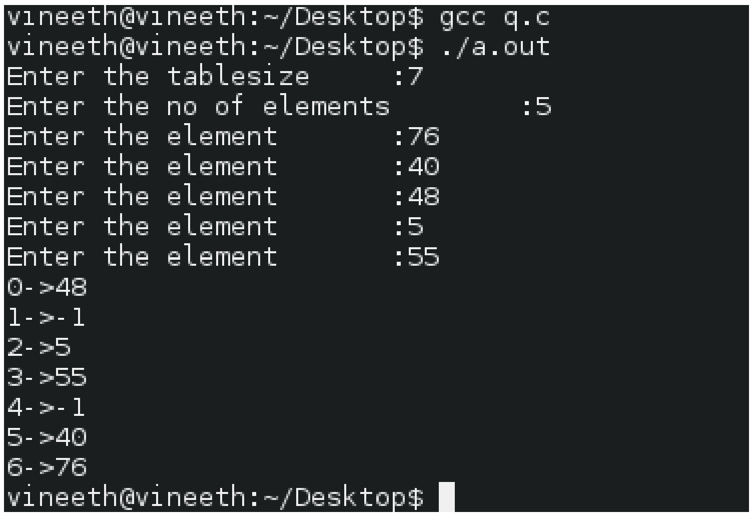
SOURCE CODE:

```
#include<stdio.h>
void linearprobing(int tablesize,int hash[])
{
    int i,n,x,j,hashkey;
    printf("Enter the no of elements\t:");
    scanf("%d",&n);
    for(i=0;i<=n-1;i++)
    {
        printf("Enter the element\t:");
        scanf("%d",&x);
        j=0;
        while(1)
        {
            hashkey=(((x%tablesize)+j*j)%tablesize);
            if(hash[hashkey]==-1)
            {
                hash[hashkey]=x;
                break;
```

```
                        }
                        else
                                j++;
                }
        }
        for(i=0;i<=tablesize-1;i++)
                printf("%d->%d\n",i,hash[i]);
}
void main()
{
        int i,tablesize;
        printf("Enter the tablesize\t:");
        scanf("%d",&tablesize);
        int hash[tablesize];
        for(i=0;i<=tablesize-1;i++)
                hash[i]=-1;
        linearprobing(tablesize,hash);
}
```

DATA STRUCTURE USED: Array

OUTPUT:



```
vineeth@vineeth:~/Desktop$ gcc q.c
vineeth@vineeth:~/Desktop$ ./a.out
Enter the tablesize     :7
Enter the no of elements        :5
Enter the element       :76
Enter the element       :40
Enter the element       :48
Enter the element       :5
Enter the element       :55
0->48
1->-1
2->5
3->55
4->-1
5->40
6->76
vineeth@vineeth:~/Desktop$
```

RESULT:

Thus the program for the implementation of Quadratic probing was successfully executed and verified.