Dr. Ohidujjaman Tuhin
Assistant Professor
Dept. of CSE , UIU

Odd even number checking:

Step1: read input number from keyboard
Step2: if (the number is divisible by 2)
Print the number is even
Step3: if (the number is not divisible by 2)
Print the number is odd
Step4: Stop

Pseudo –code :

```
read input number
if (n mod 2= =0)
        print even number;
else
        print odd number;
end if
```

```
//C code
#include<stdio.h>
void main()
{
        int n;
        printf("Enter the value of n:=");
        scanf("%d",&n);
        if(n%2==0)
                {
                        printf("The number is even:=%d\n",n);
                }
        else
                {
                        printf("The number is odd:=%d\n",n);
                }
}
```

Basic algorithm using array: Find the data value of location 3 from 10 input data.
Find the data value of location 4 to 7 from 10 input data.
Find the average value of n input data.

Linear Searching: Linear search or sequential search is a method for finding a particular value in a list that checks each element in sequence until the desired element is found or the list is exhausted. The list need not be ordered.

Mechanism of Linear Search(successful case)

Let , the array 'A', number of elements 'n' , searching item (key) k and location of elements 'i'.

Suppose, n=6, k=13

A	10	15	12	13	18	16
	1	2	3	4	5	6

i=1 A[i]= =k
=>10= =13 ; False
i=2 A[i]= =k
=>15==13; False
i=3 A[i]= =k
=>12= =13; False
i=4 A[i]= =k
=>13==13; True "Found'; Location, i = 4
i=5 ×
i=6 ×

(Unsuccessful case)

If the key (k) is not in the array, the mechanism will continue as shown in below:.

Suppose, n=6, k=14

A	10	15	12	13	18	16
	1	2	3	4	5	6

i=1 A[i]= =k
=>10= =14 ; False
i=2 A[i]= =k
=>15==14; False
i=3 A[i]= =k
=>12= =14 ; False
i=4 A[i]= =k
=>13==14; False
i=5 A[i]= =k
=>18==14; False
i=6 A[i]= =k
=>16==14; False
i=7 ; => since i>n, "Not Found"

Iterative Linear Search Algorithm

```
for  i=1 to n
    if(A[i]=k)
    printf(" Found Location:=%d",i);
    exit loop
    end if
end for
if (i>n)
printf("Not Found");
end if
```

Write a program in C for Iterative Linear Search Algorithm

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
        int A[6],i,k,n;
        printf("Enter the number of elemnts n:=");
        scanf("%d",&n);
for(i=1;i<=n;i++)
{
   printf("Enter the elemnts of Array A[%d]:=",i);
   scanf("%d",&A[i]);
}
printf("Enter the key item K:=");
scanf("%d",&k);
for(i=1;i<=n;i++)
{
        if(A[i]==k)
        {
                printf("Found Key Location:= %d\n",i);
                exit(0);
        }
}
if(i>n)
{
        printf("Key Item is not in Array\n");
}
}
```

Output on monitor:

```
Enter the number of elemnts n:=6
Enter the elemnts of Array A[1]:=10
Enter the elemnts of Array A[2]:=15
Enter the elemnts of Array A[3]:=12
Enter the elemnts of Array A[4]:=13
Enter the elemnts of Array A[5]:=18
Enter the elemnts of Array A[6]:=16
Enter the key item K:=13
Found Key Location:= 4
```

Recursive Linear Search Algorithm

Recursive algorithm for Linear search.

```
void main()
{
          //read number of elements n
         //read arrary element
          //read key item
         int j=0;
          flag=Linear(A,j,n,key);
          if(flag==-1)
          printf("\nNot Found\n");
          else
          printf("\nFound Location=%d\n",flag);

}
int Linear(int A[], int i, int n, int key)
{
          if(i>n-1)
             return -1;
          else if(A[i]==key)
          {
             printf("\nFound value=%d\n",A[i]);
             return i;
          }
          else
          return Linear(A, i+1, n, k);
}
```

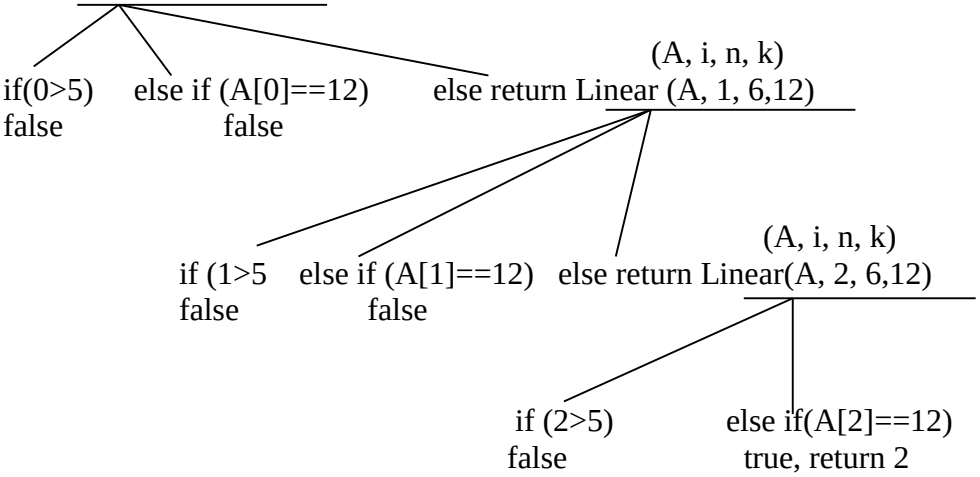
Recursive tree for Linear search.

Array

A	10	9	12	15	16	14
	0	1	2	3	4	5

Number of elements n=6
Key item k=12
Array elements location= i
First element location i=0

(A, i, n, k)
flag=Linear(A, 0, 6, 15)



Write a C program for linear search recursive algorithm.

```
#include<stdio.h>
int Linear (int A[], int i, int n, int k);
void main()
{
        int i,n,k,A[10];
        int flag;
        int j=0;
        printf("Enter the value of n:=");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
            printf("Enter the value of array A[%d]:=",i);
           scanf("%d",&A[i]);
        }
        printf("Enter the value of key:=");
       scanf("%d",&k);
        flag=Linear(A,j,n,k);
        if(flag==-1)
        printf("\nNot Found\n");
        else
        printf("\nFound Location=%d\n",flag);
}
int Linear(int A[],int i,int n,int k)
{
        if(i>n-1)
        return -1;
        else if(A[i]==k)
        {
        printf("\nFound value=%d\n",A[i]);
        return i;
        }
        else
        return Linear(A,i+1,n,k);
}
```

Output on monitor:

```
Enter the value of n:=6
Enter the value of array A[0]:=10
Enter the value of array A[1]:=9
Enter the value of array A[2]:=12
Enter the value of array A[3]:=15
Enter the value of array A[4]:=16
Enter the value of array A[5]:=14
Enter the value of key:=12
Found value=12
Found Location=2
```

Element Comparison of Linear Search Algorithm

Best case:

A	10	9	12	15	16	14
	0	1	2	3	4	5

n=6; Key =10

Element comparison =1

Therefore, for best case of linear search, the key is in the first place

Worst case:

A	10	9	12	15	16	14
	0	1	2	3	4	5

n=6; Key =20

Element comparison =n

Therefore, for worst case of linear search, the key is not in the array list.

Average case:

Average case= (best case + worst case)/2

= (1+n)/2

= (n+1)/2

Binary Searching

A binary search algorithm finds the position of a specified input value (the search "key") within a sorted array. For binary search, the array should be arranged in ascending or descending order. In each step, the algorithm compares the search key value with the key value of the middle element of the array. If the keys match, then a matching element has been found and its index, or position, is returned. Otherwise, if the search key is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the search key is greater, on the sub-array to the right. If the remaining array to be searched is empty, then the key cannot be found in the array and a special "not found" indication is returned.

Iterative Binary Search Algorithm
For Ascending Order:

A	10	20	30	40	50	60

```
low =1
high =n
while (low<=high)
   mid=⌊ ( low + high)/2 ⌋
   if(A[mid]==key)
        print "Found", print mid value location
        exit loop
    else if  (A[mid]>key)
         high=mid-1
     else
         low=mid+1
end while
if (low>high)
print "Not Found"
end if
```

Mechanism for Successful Case
In the successful case, the searching data value (key item) is found within the sorted array.
Ascending Order
Numbers are said to be in ascending order when they are arranged from the smallest to the largest number (e.g. 5, 9, 13, 17, and 21 are arranged in ascending order).

A	10	20	30	40	50	60	70	80	90	100
	0	1	2	3	4	5	6	7	8	9

Let number of elements n=10, key =70, First element's location low=0, Last element's location high=9.

low	high	mid= ⌊(low + high)/2⌋	A[mid]==key	A[mid]>key	A[mid]< key
0	9	4	50==70; F	50>70; F	50<70; T
mid+1=5	9	7	80==70; F	80>70; T	×
5	mid-1=6	5	60==70;F	60>70;F	60<70;T
mid+1=6	6	6	70==70;T	×	×

↓
Found Location=6

Complexity analysis:

Element Comparison of Binary Search Algorithm

Best case complexity:

If first mid value is equal to the key item as follows the array

10	20	30	40	**50**	60	70	80	90	100

1 **5** 10

Mid=(1+10)/2=5

Key item is 50, location is 5. 1 element comparison and O(n)=1.

Worst case complexity:

If the key item is found by dividing the array several times.

10	20	30	40	50	60	70	80	90	100

1 n=10

1. n
2. $n/2=n/2^1$
3. $n/4 = n/2^2$
4. $n/8= n/2^3$

Therefore, $n/2^k=1$

⇨ $2^k=n$

⇨ $k\ log_2^2 = \log_2^n$

⇨ k=logn => O(logn) (solved)

Average case complexity:

Average case and worst are same, for average case n elements are also divided into two sub-lists successively.

Average case: O(logn)

Write a program in C for iterative binary search algorithm (in case of ascending order)

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
        int i,k,n,A[10],low,high,mid;
        printf("Enter number of value n:=");
        scanf("%d",&n);
        for (i=0;i<n;i++)
        {
                printf("Enter elenments of array A[%d]:=",i);
                scanf("%d",&A[i]);
        }
```

```
printf("Enter the value of Key:=");
scanf("%d",&k);
low =1;
high =n;
while (low<=high)
{
   mid=( low + high)/2;
   if(A[mid]==k)
            {
        printf("Found Location:=%d\n", mid);
        exit(0);
            }
   else if(A[mid]>k)
            {
         high=mid-1;
            }
   else
         low=mid+1;
}
if (low>high)
{
printf("Not Found");
}
}
```

Output on monitor:

```
Enter number of value n:=10
Enter elements of array A[0]:=10
Enter elements of array A[1]:=20
Enter elements of array A[2]:=30
Enter elements of array A[3]:=40
Enter elements of array A[4]:=50
Enter elements of array A[5]:=60
Enter elements of array A[6]:=70
Enter elements of array A[7]:=80
Enter elements of array A[8]:=90
Enter elements of array A[9]:=100
Enter the value of Key:=70
Found Location:=6
```

Mechanism for Unsuccessful Case

In the unsuccessful case, the searching data value (key item) is not found within the sorted array.

Ascending Order

A	10	20	30	40	50	60	70	80	90	100
	0	1	2	3	4	5	6	7	8	9

Let number of elements n=10, key =35, First element's location low=0, Last element's location high=9.

low	high	mid= ⌊(low + high)/2⌋	A[mid]==key	A[mid]>key	A[mid]< key
0	9	4	50==35; F	50>35;T	×
0	mid-1=3	1	20==35; F	20>35;F	20<25;T
mid+1=2	3	2	30==35; F	30>35;F	30<35;T
mid+1=3	3	3	40==35; F	40>35;T	×
3	mid-1=2	since, low>high=3>2, hence not 'Found'		×	×

Write a program in C for iterative binary search algorithm (in case of ascending order)

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
        int i,k,n,A[10],low,high,mid;
        printf("Enter number of value n:=");
        scanf("%d",&n);
        for (i=0;i<n;i++)
        {
                printf("Enter elements of array A[%d]:=",i);
                scanf("%d",&A[i]);
        }
printf("Enter the value of Key:=");
scanf("%d",&k);
low =1;
high =n;
while (low<=high)
{
   mid=( low + high)/2;
   if(A[mid]==k)
        {
      printf("Found Location:=%d\n", mid);
      exit(0);
        }
   else if(A[mid]>k)
        {
       high=mid-1;
        }
   else
       low=mid+1;
}
if (low>high)
{
printf("Not Found\n");
}
}
```

Output on monitor:

```
Enter number of value n:=10
Enter elements of array A[0]:=10
Enter elements of array A[1]:=20
Enter elements of array A[2]:=30
Enter elements of array A[3]:=40
Enter elements of array A[4]:=50
Enter elements of array A[5]:=60
Enter elements of array A[6]:=70
Enter elements of array A[7]:=80
Enter elements of array A[8]:=90
Enter elements of array A[9]:=100
Enter the value of Key:=35
Not Found
```

Recursion

Recursion means "defining a problem in terms of itself". This can be a very powerful tool in writing algorithms. Recursion comes directly from Mathematics, where there are many examples of expressions written in terms of themselves. For example, the Fibonacci sequence is defined as: F(i) = F(i-1) + F(i-2).

Recursion is the process of defining a problem (or the solution to a problem) in terms of (a simpler version of) itself.

For example, we can define the operation "find your way home" as:

1. If you are at home, stop moving.
2. Take one step toward home.
3. "Find your way home".

Here the solution to finding your way home is two steps (three steps). First, we don't go home if we are already home. Secondly, we do a very simple action that makes our situation simpler to solve. Finally, we redo the entire algorithm. The following examples are explained to make clear about recursive algorithm.

- → Factorial Number
- → Fibonacci Number

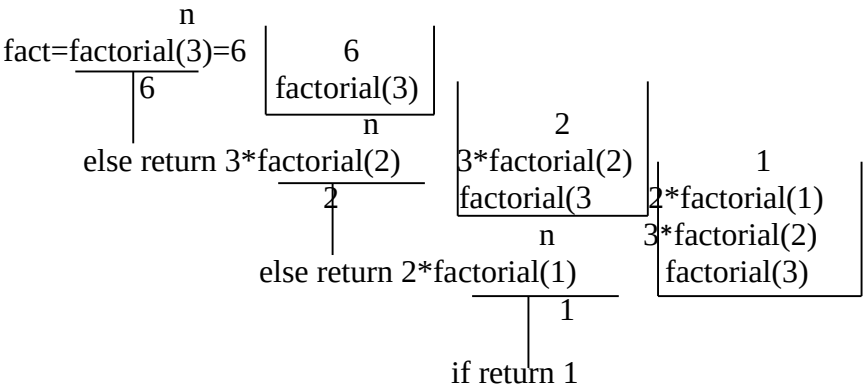
Recursive algorithm to find a value of a factorial Number.

```
void main()
{
        int n,fact;
          printf("Enter the number:=");
          scanf("%d",&n);
          printf("\nThe consecutive factorial values:=");
          fact=factorial(n);
          printf("\n\nThe factorial value:=");
          printf("%ld\n",fact);
}
long factorial(int n)
{
```

```
        if(n==1)
        {
        printf("%d\t",n);
                return 1;
        }
        else
        {
          n=n*factorial(n-1);
          printf("%d\t",n);
          return n;
        }
}
```

Recursive tree to find a value of a factorial number.

Let, n=3



Write a C program to find the factorial value of a number using recursive algorithm.

```
#include<stdio.h>
long factorial(int n);
void main()
{
        int n,fact;
        printf("Enter the number:=");
        scanf("%d",&n);
        printf("\nThe consecutive factorial values:=");
        fact=factorial(n);
        printf("\n\nThe factorial value:=");
        printf("%ld\n",fact);
}
long factorial(int n)
{
        if(n==1)
        {
        printf("%d\t",n);
                return 1;
        }
        else
        {
          n=n*factorial(n-1);
```

```
            printf("%d\t",n);
            return n;
          }
}
```

Output on monitor:

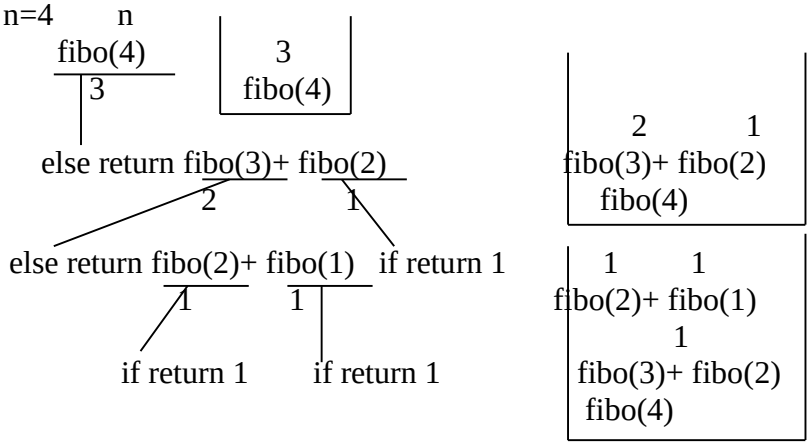
```
Enter the number:=5
The consecutive factorial values:=1    2     6     24    120

The factorial value:=120
```

Recursive algorithm for fibonacci number.

```
main()
{
int n,c,fnum;
printf("Enter the value of n:=");
scanf("%d",&n);
fnum=fibo(n);
printf("\nThe Fibonacci Number:=%d\n",fnum);
printf("The  Fibonacci Series:=");
for ( c = 1 ; c <= n ; c++ )
  {
    printf("%d ", fibo(c));
  }
}
long fibo(int n)
{
if(n==1||n==2)
   return 1;
else
  return (fibo(n-1)+fibo(n-2));
}
```

Recursive tree for fibonacci number

Let, n=4



Write a C program to find the Fibonacci series of a number using recursive algorithm.

```
#include<stdio.h>
long fibo(int n);
void main()
{
int n,c,fnum;
fflush(stdout);
printf("Enter the value of n:=");
fflush(stdin);
scanf("%d",&n);
fnum=fibo(n);
printf("\nThe Fibonacci Number:=%d\n",fnum);
printf("The  Fibonacci Series:=");
for ( c = 1 ; c <= n ; c++ )
  {
    printf("%d ", fibo(c));
  }
printf("\n");
}
long fibo(int n)
{
if(n==1||n==2)
return 1;
else
return (fibo(n-1)+fibo(n-2));

}
```

Output On Monitor:

```
Enter the value of n:=4
The Fibonacci Number:=3
The  Fibonacci Series:=1 1 2 3
```