Basic Terminology: Elementary Data Organization:-

Data:-Data are simply value or set of values.

Example:-Employee Name=Ram Krishna Param.

Example:-Complete information of a Employee

Employee Name= Ram Krishna Param.

Employee Id=11111.

Employee Address=Mathura.

Data Item:-Single unit of a value is called a Data item.

Employee Name= Ram Krishna Param.

Group Item:-Data Item are divided into subgroup is called Group Item.

Example:-

Employee First Name:- Ram.

Employee Second Name:-Krishna.

Employee Last Name:-Param.

Entity:- An entity is a thing that has some properties which can take values.

Example:-

Name= Ram Krishna Param.

Gender=Male.

Where name and gender are Entity.

Field:-A field is a part of the record and contains a single piece of data for the subject of the record.

Record:-A record is a collection of field.

File:-File is collection of record.

Information:-Meaningful or processed data is called information.

Data Structure:-

Arranging the data in the proper structure or manner is called a Data Structure.

OR

Organizing data is called a Data Structure.

The following are the various operations we can perform on Data Structure.

1. Insert new data.
2. Delete existing data.
3. Update or replacement of data.
4. Searching the data(Linear, binary).
5. Sorting technique etc.

Example:-

Student Data-----> read operation-----> ArrayList/Array.

Student Data-----> insert operation-----> LinkedList.

Photo in Mobile Gallery:-Store in Stack data structure form.

Sending emails to multiple person:-Use Queue data structure form.

File Explorer:-Use Tree data structure form.

GPS Navigation System:- Use Graph data structure form.

Trending video:-Use Priority queue.

Data Structure are divided into two parts:-

Primitive Data Structure.

Non-Primitive Data Structure.

Primitive Data Structure:-The data structure which can be directly manipulated by Machine-Level instruction. Primitive Data Structure like int, char, boolean etc.

Example:-

```
int  x=10;

int  y=20;

int c=x+y;
```

Non-Primitive Data Structure:- The data structure which derived from one or more primitive data structure.

The main objective of creating of Non-Primitive Data Structure is to make a set of homogeneous or heterogeneous data elements.

Non-Primitive Data Structure divided into two parts:-

1) Linear Data structure.
2) Non-Linear Data structure.

Linear Data structure:- In this data structure the data elements are arranged sequentially or linear where each and every element is attached to its previous or next adjacent is called Linear data structure.

Example:- Array, LinkedList, Stack, Queue.

Non-Linear Data structure:- In this data structure where the data elements are not arranged sequentially or linearly are called Non-Linear Data structure.

Example:- Tree, Graph.

Abstract Data Type(ADT):- Abstract Data Type(ADT) is a data type, where data behaviour, characteristics and properties are defined but how to implementation that data, it is not defined. Every programming language provide implementation in own way.

Examples:-
Array, List, Map, Queue, Set, Stack, Table, Tree, and Vector are Abstract Data Type(ADT).

Operation on Data Structure:- There are several common operation performed on data structure like:-

1. **Traversing:-** It is the process of accessing each record of a data structure exactly once.

2. **Searching:-**In this process we find a particular element present in data structure.

3. **Inserting:-**In this process we add new element in data structure.

4. **Deleting:-**In this process we removing an existing record from a data structure.

5. **Sotring:-**It is the process of arranging the records of data structure in a specific order, such as alphabetic, ascending or descending.

6. **Merging:-**It is the process of combining the record of two different sorted data structure to produce a single sorted data set.

Algorithm:-

An algorithm is a set of finite rules or instructions, we follow that rule or instruction to perform various operations like insert, delete, update, search, sort.. etc. in a very effective manner.

OR

Step by step process to solve a problem is called an algorithm.

Example:-

Algorithm to perform the addition of two numbers.

Step:-1 Read two numbers from the user.

Step 2 Use arithmetic operations to calculate the result.

Step 3 Display the Result.

Implements the above algorithm.

```
import java. util.*;

class Addition

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

//Step:-1

System.out.println("Enter First Number");

int num1=sc.nextInt();

System.out.println("Enter Second Number");
```

```
int num2=sc.nextInt();
//Step:-2
int sum=num1+num2;
//Step:-3
System.out.println("SUM="+sum);
}
}
```

Properties or Characterstic of an Algorithm:-

1) Finite number of steps to perform any operation.
2) It should have zero or more valid and clearly defined input values.
3) It should be able to generate at least a single valid output based on valid input.
4) Each instruction in the algorithm should be defined clearly.
5) There should be no ambiguity regarding the order of execution of algorithm steps.
6) It should be able to terminate on its own.

Swapping two number:-

Approach:-1 Swapping two number by using third variable.

```
public class SwapDemo
{
public static void main(String[] args)
{
int x=10;
int y=20;
System.out.println("Before swapping"+" x = "+x+" y = "+y);
int temp=y;
y=x;
x=temp;
```

```
System.out.println("After swapping"+" x = "+x+" y = "+y);
}
}
```

Approach:-2 Swapping two number by without using third variable.

```
public class SwapDemo
{
public static void main(String[] args)
{
int x=10;
int y=20;
System.out.println("Before swapping"+" x = "+x+" y = "+y);
x=x+y;
y=x-y;
x=x-y;
System.out.println("After swapping"+" x = "+x+" y = "+y);
}
}
```

Approach:-3 Swapping two number by without using third variable and arithmetic operation.

```
package com.swap;
public class SwapDemo
{
public static void main(String[] args)
{
int x=10;
int y=20;
System.out.println("Before swapping"+" x = "+x+" y = "+y);
x=x*y;
y=x/y;
```

```
x=x/y;
System.out.println("After swapping"+" x = "+x+" y = "+y);
}
}
```

Approach:-4 Swapping two number by using bitwise XOR (^) .

```
public class SwapDemo
{
public static void main(String[] args)
{
int x=10;
int y=20;
System.out.println("Before swapping"+" x = "+x+" y = "+y);
x=x^y;
y=x^y;
x=x^y;
System.out.println("After swapping"+" x = "+x+" y = "+y);
}
}
```

Approach:-5 Swapping two number in single line .

```
public class SwapDemo
{
public static void main(String[] args)
{
int x=10;
int y=20;
System.out.println("Before swapping"+" x = "+x+" y = "+y);
x=(x+y)-(y=x);
System.out.println("After swapping"+" x = "+x+" y = "+y);
```

```
}
}
```

Find the maximum number between two number.

Algorithm:-

Step:-1 Read two number.

Step:-2 Apply Logic.

Version:-1 use condition operator.

(condition)?value1:value2

Version:-2

Math.max(a,b)

Step:-3 print the result.

Implements Algorithm.

```
import java.util.Scanner;
public class MaxNumber
{
public int max_version1(int x, int y)
{
return (x>y)?x:y;
}
public int max_version2(int x, int y)
{
return Math.max(x, y);
}
public static void main(String[] args)
{
Scanner scanner=new Scanner(System.in);
System.out.println("Enter value of X");
int x=scanner.nextInt();
```

```
System.out.println("Enter value of y");
int y=scanner.nextInt();
MaxNumber mn=new MaxNumber();
System.out.println("Mximum value from version1 "+mn.max_version1(x, y));
System.out.println("Mximum value from version2 "+mn.max_version2(x, y));
}
}
```

Write a program to find given number is Even or Odd.

Algorithm.

Step:-1 Read number.

Step:-2 Apply logic if number is divided by 2 means number is 'even' else number is odd.

```
if(number%2==0)
print even
else
print add.
```

Step:-3 display result.

```
import java.util.Scanner;
public class OddEven
{
public String checkOddEven(int number)
{
return (number%2==0)?"Even":"Odd";
}
public int max_version2(int x, int y)
{
return Math.max(x, y);
```

```
}
public static void main(String[] args)
{
Scanner scanner=new Scanner(System.in);
System.out.println("Enter Number");
int  number=scanner.nextInt();
OddEven oe=new OddEven();
System.out.println("Number  is  "+oe.checkOddEven(number));
}
}
```

Write a program to find sum of 'N' natural number.

Algorithm:-

Step:-1 Read 'n' value from the user.

Step:-2 Apply the logic to find sum of n natural number.

Logic:-1 By using loop.

```
int sum=0;
for(int  i=1;i<=n;i++)
{
sum=sum+i;
}
```

Logic:-2 By using recursion.

```
int sum(int n)
{
if(n==0)
{
return 1
}
else
```

```
{
return n*sum(n-1)
}
}
```

Logic:-3 By apply math formula n*(n+1)/2

Implementation:-

```
import java.util.Scanner;
public class SumOfN
{
public int sumLogic1(int n)
{
int sum=0;
for(int i=1;i<=n;i++)
{
sum=sum+i;
}
return sum;
}
public int sumLogic2(int n)
{
if(n==0)
{
return 1;
}
else
{
return n+sumLogic1(n-1);
}
}
```

```
public int sumLogic3(int n)
{
return n*(n+1)/2;
}
public static void main(String[] args)
{
Scanner scanner=new Scanner(System.in);
System.out.println("Enter  Number");
int  number=scanner.nextInt();
SumOfN sumOfN=new SumOfN();
System.out.println("SUM1   "+sumOfN.sumLogic1(number));
System.out.println("SUM2   "+sumOfN.sumLogic2(number));
System.out.println("SUM3   "+sumOfN.sumLogic3(number));
}
}
```

Find Max number among the number:-

Algorithm:-

Step:-1 Read a,b and c values from user.

Step2:- Apply Logic.

Logic:-1

```
(a>b && a<c)?a:((b>c)?b:c)
```

Logic:-2

```
Math.max(Math.max(number1,number2),number3))
```

Step:-3 Display the result.

```
import  java.util.Scanner;
public class MaxNumber
{
int max1(int a, int b, int c)
```

```
{
return (a>b && a>c)?a:(b>c?b:c);
}
int max2(int a, int b, int c)
{
return Math.max(Math.max(a, b),c);
}
public static void main(String[] args)
{
Scanner scanner=new Scanner(System.in);
System.out.println("First Number");
int a=scanner.nextInt();
System.out.println("Second  Number");
int b=scanner.nextInt();
System.out.println("Third Number");
int c=scanner.nextInt();
System.out.println(new MaxNumber().max1(a, b, c));
System.out.println(new MaxNumber().max2(a, b, c));
}
}
```

Find the factorial the of Nth number.

Algorithm:-

Step:-1 Read Nth value.

Step:-2 Apply logic based on your requirement.

Logic:-1

By using looping.

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

```
{
f=f*i;
}
display f
```

Logic:-2 By using recursion.

```
int fact(int n)
{
if(n==1)
{
retrun 1;
}
else
{
return n*fact(n-1);
}
}
```

```
package com.ds;
import java.util.Scanner;
public class FindFactorial
{
int fact=1;
//looping approach
public int fact1(int n)
{
for(int i=1;i<=n;i++)
{
fact=fact*i;
```

```
}
return fact;
}
//Recursion approach
public int fact2(int n)
{
if(n==1 || n==0)
{
return 1;
}
else
{
return n*fact2(n-1);
}
}
public static void main(String[] args)
{
Scanner scanner=new Scanner(System.in);
System.out.println("Enter Number");
int  n=scanner.nextInt();
if(n>0)
{
System.out.println(new FindFactorial().fact1(n));
System.out.println(new FindFactorial().fact2(n));
}
else
{
System.out.println("You can't find factorial of zero and negative
value");
```

```
}
}
}
```

To Check given number is prime or not.

Algorithm:-

Step:-1

Read number.

Step:-2 Apply Logic.

Logic:-1 By using loop.

Logic:-2 By using recursion.

Step:-3

Print the status.

Example:-

```
package com.ds;
public class CheckPrimeNumber
{
//By using Loop
public boolean isPrime1(int n)
{
int flag=0;
if(n==1 || n==0)
{
return false;
}
else
{
for(int i=2;i<=n/2;i++)
{
```

```
if(n%i==0)
{
flag++;
}
}
if(flag>0)
{
return false;
}
else
{
return true;
}
}
}

//By using recursion
public boolean isPrime2(int n,int i)
{
 if(n==1 || n==0)
{
return false;
}
else
{
if(i==1)
{
return true;
}
```

```
else if (n%i==0)
{
return false;
}
else
{
return isPrime2(n,--i);
}
}

}
public static void main(String[] args)
{
for(int i=50;i<=100;i++)
{
System.out.println("Logic:-1 "+(((new
CheckPrimeNumber().isPrime1(i))?"yes":"no"))+"\t"+"Logic:-2 "+((new
CheckPrimeNumber().isPrime2(i,i/2))?"yes":"no"));
}
}
}
```

Print Fibonacci Number:-

Algorithm:-

Step:1 Read Range of series.

Step:-2 Apply Logic and print number.

```
for(int i=0;i<=range;i++)
{
int c=a+b;
```

```
System.out.print(" "+c);
a=b;
b=c;
}

package com.ds;
import java.util.Scanner;
public class FibonacciNumber
{
public static void main(String[] args)
{
Scanner sc=new Scanner(System.in);
System.out.println("Enter Range of Fabinocci Series");
int range=sc.nextInt();
int a=0;
int b=1;
System.out.print(a+" "+b);
for(int i=0;i<=range;i++)
{
int c=a+b;
System.out.print(" "+c);
a=b;
b=c;
}
}
}
```

Print Tribonacci Number:-

Algorithm:-Algorithm is the same as Fibonacci number.

```
package com.ds;
import java.util.Scanner;
public class FibonacciNumber
{
public static void main(String[] args)
{
Scanner sc=new Scanner(System.in);
System.out.println("Enter Range of Fabinocci Series");
int range=sc.nextInt();
int a=0;
int b=1;
int c=2;
System.out.print(a+" "+b+" "+c);
for(int i=0;i<=range;i++)
{
int d=a+b+c;
System.out.print(" "+d);
a=b;
b=c;
c=d;
}
}
}
```

Performance of algorithms or Effectiveness of the algorithm we can measure by using two following two components.

1. Space Complexity.
2. Time Complexity.

Space Complexity:- The space complexity of any algorithm is the amount of memory, it needs to run to complete a task.

The space complexity of any algorithm is calculated as:-

Space complexity=fixed part + variable part.

Fixed part:- Independent of instance characteristics

Like:- Space for variables, space for constants.

Variable Part:- Dependent on the instance characteristics.

Like:- Arrays etc.

Example:-

```
algo addition( a, b, c)
{
return a+b+c;
}
```

Space complexity:-

a---------->1 unit.

b---------->1 unit.

c---------->1 unit.

Total: 3 units.

Space complexity:- 3 units.

Example:-

```
algo areaOfCircle(radius)
{
return 3.147*radius*radius;
}
```

Space complexity:-

radius--------->1 unit.

3.14 --------->1 unit.

Total: 2 units.

Space complexity:- 2 units.

Example:-

```
float area;
float areaOfCircle(float radius)
{
area=3.147*radius*radius;
return area;
}
```

area ---------- >1

radius--------->1 unit.

3.14 --------->1 unit.

Total: 3 units.

Space complexity:- 3 units.

Example:-

```
String s=”abc”;
```

s--------->1 unit

Example:-

Sum of ‘n’ natural numbers.

```
algorithm sum_of_n(n)
{
s=0;
for(i=1;i<=n;i++)
{
s=s+i;
}
return s;
```

```
}
Space complexity:-
n---------->1 unit
s---------->1 unit
i---------->1 unit
------------------
Total----->3 units
------------------
Example:-Sum of elements in an array ‘a’.
algorithm sum_array(a,n)
{
s=0;
for(i=0;i<n;i++)
{
s=s+a[i];
}
return s;
}
Space complexity ---------- >
a----------> n unit.
n----------> 1 unit.
s----------> 1 unit.
i----------> 1 unit.
--------------------
Total----->n+3 units
--------------------
```

Time Complexity:- The time complexity is the amount of computer time is need to complete the task.

Time Complexity = Execution Time.

To count time complexity by using following steps:-

1) for algorithm heading ----------->0.

2) for brace ----------->0.

3) for expression ----------->1.

4) for if condition ----------->1.

5) for loop---------->Based on the number of iterations.

Example:-Algorithm addition of three number.

```
1.algorithm addition(a,b,c)
2.{
3.return a+b+c;
4.}
```

1--------->0

2--------->0

3--------->1 unit

4--------->0

Total time complexity=1 unit

Case:-1

```
algorithm addition(a,b,c)
{
int d=a+b+c;
return d;
}
```

Case:-2

```
algorithm addition(a,b,c)
{
return a+b+c;
}
```

Both cases time complexity is the same because the calculation operation is performed only once.

But space complexity is different because in case1 uses more number of variables.

Example:-Find the max of two number.

```
1. algorithm max(a,b)
2. {
3. return (a>b)?a:b;
4. }
```

1--------->0

2--------->0

3--------->1 unit

4--------->0

Total time complexity=1 unit

Example:-Find max number between two numbers.

```
1. algorithm max(a,b)
2. {
3. if(a>b)
4. return a;
5. else
6. return b;
7. }
```

1--------->0

2--------->0

3---------->1 unit

4---------->0

5---------->0

6---------->0

7---------->0

Total time complexity=1 unit

Example:-Find max number among three numbers.

```
1. algorithm max(a,b,c)
2. {
3. if(a>b && a>c)
4. return a;
5. else if(b> && b>c)
6. return a;
7. else
8. return c;
9. }
```

1---------->0

2---------->0

3---------->1 unit

4---------->0

5---------->1

6---------->0

7---------->0

8---------->0

9---------->0

Total time complexity=2 unit

Example:-Sum of n natural number.

```
1. algorithm sum(n)
2. {
3. sum=0;
4. for(i=1;i<=n;i++)
5. {
6. sum=sum+i;
7. }
8. return sum;
9. }
```

1.-------------->0

2.-------------->0

3.-------------->0

4---------->n+1

5.-------------->0

6---------->n

7.-------------->0

8.-------------->0

9.-------------->0

Total time complexity=2n+1 unit

Example:-Sum of n even number.

```
1. algorithm sum(n)
2. {
3. sum=0;
4. for(i=1;i<=n;i++)
5. {
6.  if(n%2==0)
7.  sum=sum+i;
8. }
9. return sum;
```

```
10. }
1.-------------->0
2.-------------->0
3.-------------->0
4---------------------->n+1
5.-------------->0
6---------------------->n
7---------------------->n/2
8.-------------->0
9.-------------->0
----------------------------
Total time complexity=2n+1+n/2 unit
---------------------------
```

Example:- Find max element present in an array

```
---------------------------------------
1.algorithm maxElement(a,n)
2.{
3.max = a[0];
4.for(i=1;i<n;i++)
5.{
6.if(max<a[i])
7.{
8.max = a[i];
9.}
10.}
11.return max;
12.}
```

1-----> 0

2 ----> 0

3 ----> 0

4 ----> n

5 ----> 0

6 ----> n-1

7 ----> 0

8 ----> n-1

9 ----> 0

10 ---> 0

11 ---> 0

12 ---> 0

Time Complexity= n+n-1+n-1=3n-2

Ex1: sorting of an array

```
algorithm sort(a,n)
{
for(i=0;i<n;i++) ---------> n+1
{
for(j=i+1;j<n;j++) ---> n*(n) ---> n^2
{
if(a[i]>a[j]) ----> n^2 - 1
{
t=a[i];
a[i]=a[j];
a[j]=t;
}
```

```
}
}
}
```

Time Complexity-----> n+1+n2+n2-1 ----> 2n^2+n

Algorithm Cases:- There are three cases Best case, Average case and Worst case, to measure efficiency or performance of algorithm.

Best Case:- If we are looking for a solution, which is available at the very first location, then such type of case is called as Best Case.

Example:-

11, 12, 13, 14, 15, 16

key = 11

comparison ---> 1st comparison ---> best case

Worst Case:-

If we are looking for data, which is available at the last position or may not be available, then such type of cases are called as the Worst Case.

Example:-

11, 12, 13, 14, 15, 16

key = 16

comparison----> 6^{th} comparison, Success--->Worst Case.

key = 17

comparison ---> 6^{th} comparison, Failure--->Worst Case.

Average Case:-

If we are looking for multiple data, the time/space taken for that algorithm is calculated based on sum of the possible case.

Ex:

11, 12, 13, 14, 15, 16

key=11 ----> 1

key=12 ----> 2

key=13 ----> 3

key=14 ----> 4

key=15 ----> 5

key=16 ----> 6

key=17 ----> 6

avg case ===> total comp/num.of cases

27/7=3

Note:-We can ignore this case.

Asymptotic notation:-

it is used to measure/represent the time and space complexity of any algorithm.

1. Big-Oh ----> O
2. Omega ----> W
3. Theta ----> theta
4. Little oh ----> o
5. Little omega ----> w little omega

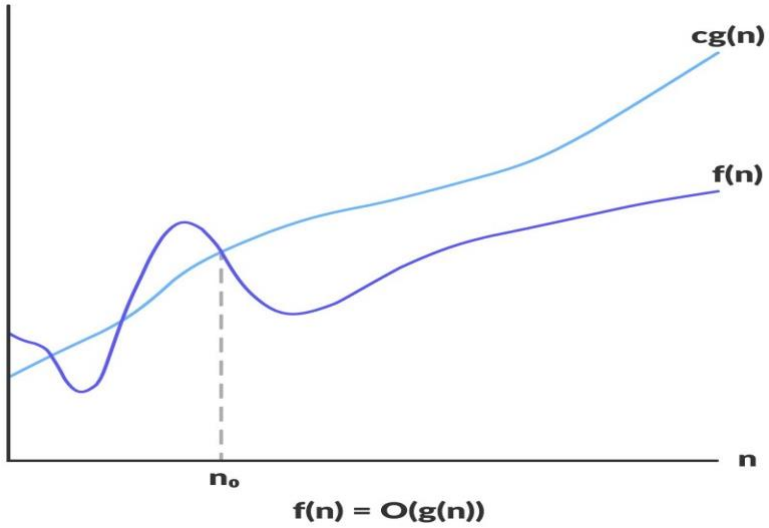
Big "Oh" (O):-

The big-oh notation is a method that is used to express the upper bound of the running time of an algorithm. It is denoted by 'O'. Using this annotation we can compute the maximum possible amount of time that an algorithm will take for it completion.

f(n) and *g(n)* are two positive function of n, where n is the size of the input data. f(n) is big-oh of g(n), if and only if there exists a positive constant C and an integer n_0, such that

f(n)<=c*g(n) for every n>=n0

Diagram:-



Example:-

f(n) = 2n+2

g(n) = n^2

	f(n)=2n+2	g(n)= n^2
n=1	4	1
n=2	6	4
n=3	8	9
n=4	10	16

From n=3 g(n) function staring increase.

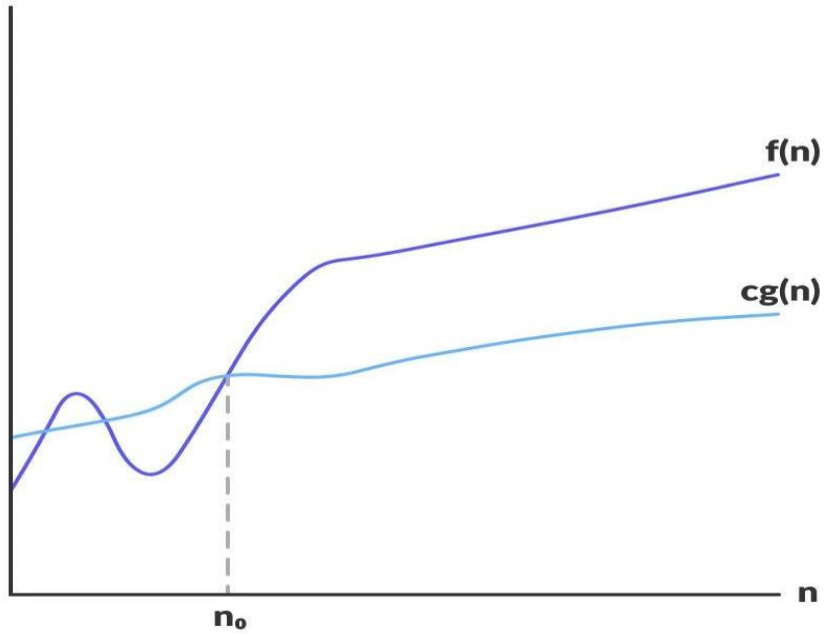
where n>=3

Omega Notation (W):-

The omega notation is a method that is used to express the lower bound of the running time of an algorithm. It is denoted by 'Ω'. Using this annotation we can compute the minimum possible amount of time that an algorithm will take for it completion.

f(n) and *g(n)* are two positive function of n, where n is the size of the input data. Then f(n) is omega of g(n), if and only if there exists a positive constant C and an integer n_0, such that

Diagram:-



f(n) = Ω(g(n))

Example:-

$f(n) = 2n^2 + 3$

g(n) = 7n

	$f(n)= 2n^2 + 3$	g(n)=7n
n=1	5	7
n=2	11	14
n=3	21	21
n=4	35	26

From n=3 g(n) function staring decrease.

where n0>=3