ASSIGNMENT

EXPLAIN THE FOLLOWING:

CACHE:

This is a hardware or software component that stores data so that future requests for that data can be served faster; the data stored in a cache might be the result of an earlier computation or a copy of data stored elsewhere. A cache hit occurs when the requested data can be found in a cache, while a cache miss occurs when it cannot. Cache hits are served by reading data from the cache, which is faster than recomputing a result or reading from a slower data store; thus, the more requests that can be served from the cache, the faster the system performs.

Examples of hardware caches

CPU cache
GPU cache
Information-centric networking
The least frequent recently used (LFRU)
Disk cache
Web cache

REGISTER:

A processor register (CPU register) is one of a small set of data holding places that are part of the computer processor. A record may hold an instruction, a storage address, or any kind of data (such as a bit sequence or individual characters). Some instructions specify registers as part of the instruction. A record must be large enough to hold an instruction.

RECURSION:

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function. Using a recursive algorithm, certain problems can be solved quite easily. A recursive function solves a particular problem by calling a copy of itself and solving smaller subproblems of the original problems. Many more recursive calls can be generated as and when required. It is essential to know that we should provide a certain case to terminate this recursion process. So we can say that every time the function calls itself with a simpler version of the original problem.

Need of Recursion

Recursion is an amazing technique with the help of which we can reduce the length of our code and make it easier to read and write. It has certain advantages over the iteration technique which will be discussed later. A task that can be defined with its similar subtask, recursion is one of the best solutions for it.

Properties of Recursion:

1. Performing the same operations multiple times with different inputs.
2. In every step, we try smaller inputs to make the problem smaller.
3. A base condition is needed to stop the recursion otherwise infinite loop will occur.

How are recursive functions stored in memory?

Recursion uses more memory, because the recursive function adds to the stack with each recursive call, and keeps the values there until the call is finished. The recursive function uses the LIFO (LAST IN FIRST OUT) Structure just like the stack data structure.

In the recursive program, the solution to the base case is provided and the solution to the bigger problem is expressed in terms of more minor problems.

EG:

```
//recursion
//input: 123
//output: 3          :)
static int CountDigits(int num, int count)
      {
         if (num == 0)
         {
            return count;
         }

         return CountDigits(num / 10, ++count);
      }
static void Main(string[] args)
      {
         Console.WriteLine(CountDigits(123, 0));
       }
```

Recursive function

```
public static double Factorial(int number)
{
        if (number == 0)
        {
                return 1;
   }
   return number * Factorial(number-1);//Recursive call
}
```

FIBONNIER SERIES:

In mathematics and computing, Fibonacci coding is a universal code which encodes positive integers into binary code words. It is one example of representations of integers based on Fibonacci numbers. Each code word ends with "11" and contains no other instances of "11" before the end.
The Fibonacci Series in C# in the Fibonacci series is one of the famous sequence series. The sequence is 0, 1, 1, 2, 3, 5, 8…. The Fibonacci series starts from zero and one and the next number is the sum of two preceding numbers. It has been said that the Fibonacci Series created by Mr.Leonardo Pisano Bigollo in the 13th century. Fibonacci series is useful for some scenarios. Basically it was originally used to solve the rabbit problem i.e. The number of rabbits born from a pair. There are other problems also in which the Fibonacci sequence is useful.
Fibonacci Series Logic
As in the Fibonacci series, the number is the sum of its two preceding numbers. So if we have a Fibonacci series say 0, 1, 1, 2, 3, 5, 8, 13, 21… According to this next number would be the sum of its preceding two like 13 and 21. So the next number is 13+21=34.

Here is the logic for generating Fibonacci series

F(n)= F(n-1) +F(n-2)

Where F(n) is term number and F(n-1) +F(n-2) is a sum of preceding values.

So if we have series 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89…

According to the logic F(n)= F(n-1) +F(n-2)

F(n)= 55+89

F(n)= 144

The next term would be 144.

Various Method of creating Fibonacci Series

Fibonacci series can be generated in multiple ways.

1. Iterative Approach
This way is the easiest way to generate series.

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespaceFibonacciDemo
{
classProgram
{
staticint Fibonacci(int n)
{
intfirstnumber = 0, secondnumber = 1, result = 0;
if (n == 0) return 0; //It will return the first number of the series
if (n == 1) return 1; // it will return  the second number of the series
for (int i = 2; i<= n; i++)  // main processing starts from here
{
result = firstnumber + secondnumber;
firstnumber = secondnumber;
secondnumber = result;
}
return result;
}
staticvoid Main(string[] args)
{
Console.Write("Length of the Fibonacci Series: ");
int length = Convert.ToInt32(Console.ReadLine());
for(int i = 0; i< length; i++)
{
Console.Write("{0} ", Fibonacci(i));
}
Console.ReadKey();
}
}
}
```

2. Recursive Method
This is another method to solve this problem.

Method 1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespaceFibonacciDemo
{
classProgram
{
staticint Fibonacci(int n)
{
intfirstnumber = 0, secondnumber = 1, result = 0;
if (n == 0) return 0; //it will return the first number of the series
if (n == 1) return 1; // it will return the second number of the series
return Fibonacci(n-1) + Fibonacci(n-2);
}
staticvoid Main(string[] args)
{
Console.Write("Length of the Fibonacci Series: ");
int length = Convert.ToInt32(Console.ReadLine());
for(int i = 0; i< length; i++)
{
Console.Write("{0} ", Fibonacci(i));
}
Console.ReadKey();
}
}
}
```

Method 2

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace FibonacciSeries
{
class Program
{
public static void Fibonacci
(
int firstnumber,
int secondnumber,
int count,
int length,
)
{
if (count <= length)
{
Console.Write("{0} ", firstnumber);
Fibonacci(secondnumber, firstnumber + secondnumber, count + 1, length);
}
}
public static void Main(string[] args)
{
Console.Write("Length of the Fibonacci Series: ");
int length = Convert.ToInt32(Console.ReadLine());
Fibonacci(0, 1, 1, length);
Console.ReadKey();
}
}
}
```

Fibonacci by using Array

Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
public class Program
{
public static int[] Fibonacci(int number)
{
int[] a = new int[number];
a[0] = 0;
a[1] = 1;
for (int i = 2; i < number; i++)
{
a[i] = a[i - 2] + a[i - 1];
}
return a;
}
public static void Main(string[] args)
{
```

```
var b = Fibonacci(10);
foreach (var elements in b)
{
Console.WriteLine(elements);
}
}


}
```

How to find the Nth Term of Fibonacci Series?
Following are the methods

Method 1
Code:

```
using System;
namespace FibonacciSeries
{
class Program {
public static int NthTerm(int n)
{
if ((n == 0) || (n == 1))
{
return n;
}
else
{
return (NthTerm(n - 1) + NthTerm(n - 2));
}
}
public static void Main(string[] args)
{
Console.Write("Enter the nth term of the Fibonacci Series: ");
int number = Convert.ToInt32(Console.ReadLine());
number = number - 1;
Console.Write(NthTerm(number));
Console.ReadKey();
}
}
}
```

The above code is to find the nth term in the Fibonacci series. For example, if we want to find the 12th term in the series then the result would be 89.

Method 2
(O(Log t) Time).

There is one another recurrence formula that can be used to find t'th Fibonacci Number If t is even then = t/2:

F(t) = [2*F(k-1) + F(k)]*F(k)

If t is odd then k = (t + 1)/2

F(t) = F(k)*F(k) + F(k-1)*F(k-1)

Fibonacci matrix

After getting determinant, we will get (-1)t = Ft+1Ft-1 – Ft2

FmFt + Fm-1Ft-1 = Fm+t-1

By putting t = t+1,

FmFt+1 + Fm-1Ft = Fm+t

Putting m = t

F2t-1 = Ft2 + Ft-12

F2t = (Ft-1 + Ft+1)Ft = (2Ft-1 + Ft)Ft

To get the formula we will do the following

If t is even, put k = t/2

If t is odd, put k = (t+1)/2

So by sorting these numbers we can prevent the constantly using memory space of STACK. It gives time complexity of O(n). The recursive algorithm is less efficient.

Code:

```
int f(n) :
if( n==0 || n==1 )
return n;
else
return f(n-1) + f(n-2)
```

Now when the above algorithm run for n=4

```
fn(4)

f(3)          f(2)

f(2)   f(1)     f(1)   f(0)

f(1)  f(0)
```

So it’s a tree. For calculating f(4) we need to calculate f(3) and f(2) and so on.For a small value of 4, f(2) is calculated twice and f(1) is calculated thrice. This number of additions will grows for large numbers.

There is a conjecture that the number of additions required for calculating f (n) is f (n+1) -1.

Conclusion

Here the iteration method is always preferred because it has a faster approach to solve this kind of problem. Here we are storing the first and the second number of Fibonacci series in the previous number and previous number(these are two variables) and also we are using the current number to store the Fibonacci number.