RECURSION

What is base condition in recursion?

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

Example:

```
 int fact(int n)
{
   if (n < = 1) // base case
      return 1;
   else
      return n*fact(n-1);
}
```

In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.

Why Stack Overflow error occurs in recursion?

If the base case is not reached or not defined, then the stack overflow problem may arise. Let us take an example to understand this.

```
int fact(int n)
{
   // wrong base case (it may cause
   // stack overflow).
   if (n == 100)
      return 1;

   else
      return n*fact(n-1);
}
```

If fact(10) is called, it will call fact(9), fact(8), fact(7) and so on but the number will never reach 100. So, the base case is not reached. If the memory is exhausted by these functions on the stack, it will cause a stack overflow error.

What is the difference between direct and indirect recursion?

A function fun is called direct recursive if it calls the same function fun. A function fun is called indirect recursive if it calls another function say fun_new and fun_new calls fun directly or indirectly.

```
// An example of direct recursion
void directRecFun()
{
    // Some code....

    directRecFun();

    // Some code...
}

// An example of indirect recursion
void indirectRecFun1()
{
    // Some code...

    indirectRecFun2();

    // Some code...
}
void indirectRecFun2()
{
```

```
    // Some code...

    indirectRecFun1();

    // Some code...
}
```

What are the disadvantages of recursive programming over iterative programming?

Note that both recursive and iterative programs have the same problem-solving powers, i.e., every recursive program can be written iteratively and vice versa is also true. The recursive program has greater space requirements than iterative program as all functions will remain in the stack until the base case is reached. It also has greater time requirements because of function calls and returns overhead.

What are the advantages of recursive programming over iterative programming?

Recursion provides a clean and simple way to write code. Some problems are inherently recursive like tree traversals, Tower of Hanoi, etc. For such problems, it is preferred to write recursive code. We can write such codes also iteratively with the help of a stack data structure. For example refer Inorder Tree Traversal without Recursion, Iterative Tower of Hanoi.

Tail Recursion: If a recursive function calling itself and that recursive call is the last statement in the function then it's known as Tail Recursion. After that call the recursive function performs nothing. The function has to process or perform any operation at the time of calling and it does nothing at returning time.

```
// Java code Showing Tail Recursion
class Main {

// Recursion function
static void fun(int n)
{
    if (n > 0)
    {
    System.out.print(n + " ");

    // Last statement in the function
    fun(n - 1);
    }
}

// Driver Code
public static void main(String[] args)
{
    int x = 3;
    fun(x);
}
```

```
}
```

Time Complexity For Tail Recursion : O(n)

Space Complexity For Tail Recursion : O(n)

Note: Time & Space Complexity is given for this specific example. It may vary for another example.

// Converting Tail Recursion into Loop

```
import java.io.*;
class Main
{
static void fun(int y)
{
   while (y > 0) {
      System.out.print(" "+ y);
      y--;
   }
}

// Driver code
public static void main(String[] args)
{
   int x = 3;
   fun(x);
```

```
}
}
```

Time Complexity: O(n)

Space Complexity: O(1)

Note: Time & Space Complexity is given for this specific example. It may vary for another example.

So it was seen that in case of loop the Space Complexity is O(1) so it was better to write code in loop instead of tail recursion in terms of Space Complexity which is more efficient than tail recursion.

Why space complexity is less in case of loop?

Before explaining this I am assuming that you are familiar with the knowledge that's how the data stored in main memory during execution of a program. In brief, when the program executes, the main memory divided into three parts. One part for code section, the second one is heap memory and another one is stack memory. Remember that the program can directly access only the stack memory, it can't directly access the heap memory so we need the help of pointer to access the heap memory.

Let's now understand why space complexity is less in case of loop?

In case of loop when function "(void fun(int y))" executes there only one activation record created in stack memory(activation record created for only 'y' variable) so it takes only 'one' unit of memory inside stack so it's space complexity is O(1) but in case of recursive function every time it calls itself for each call a separate activation record created in stack. So if there's 'n' no of call then it takes 'n' unit of memory inside stack so it's space complexity is O(n).

Head Recursion: If a recursive function calling itself and that recursive call is the first statement in the function then it's known as Head Recursion. There's no statement, no operation before the call. The function doesn't have to process or perform any operation at the time of calling and all operations are done at returning time.

Example:

```
import java.io.*;


class Main{


// Recursive function
static void fun(int n)
{
   if (n > 0) {


      // First statement in the function
      fun(n - 1);


      System.out.print(" "+ n);
   }
}


// Driver code
public static void main(String[] args)
{
   int x = 3;
   fun(x);
```

```
}
}
```

Time Complexity For Head Recursion: O(n)

Space Complexity For Head Recursion: O(n)

Note: Time & Space Complexity is given for this specific example. It may vary for another example.

Note: Head recursion can't easily convert into loop as Tail Recursion but it can. Let's convert the above code into the loop.

```
import java.util.*;
class Main
{
// Recursive function
static void fun(int n)
{
   int i = 1;
   while (i <= n) {
      System.out.print(" "+ i);
      i++;
   }
}
```

```
// Driver code
public static void main(String[] args)
{
    int x = 3;
    fun(x);
}
}
```