Algorithms & Data Structures I Week 16 Lecture Note

Notebook:	Algorithms & Data Structures I		
Created:	2020-10-21 4:14 PM	**Updated:**	2021-01-19 6:41 PM
Author:	SUKHJIT MANN		

Cornell Notes	**Topic:** Recursion, Part 2	Course: BSc Computer Science Class: CM1035 Algorithms & Data Structures I [Lecture] Date: January 19, 2021

Essential Question:

What is recursion?

Questions/Cues:

- How can we implement the Binary Search Algorithm recursively?
- What is a Call Stack in programming?
- What is a stack frame?
- How can we use the concept of a call stack in developing a recursive function?
- What is stack overflow?
- How can we implement the factorial algorithm recursively using the notion of a call stack?

Notes

```
function Search(v, item, L, R)
        if L > R then
                return FALSE
        end if
        M ← ⌊(L + R)/2⌋
        if v[M] = item then
                return TRUE
        else if v[M] > item then
                R ← M − 1
        else
                L ← M + 1
        end if
        return Search(v, item, L, R)
end function

function BinarySearch(v, item)
        n ← LENGTH[v]
        R ← n
        L ← 1
        return Search(v, item, L, R)
end function
```

- Call Stack = is a stack that manages function calls. The call stack is working in the background of the program whilst running, managing what functions are called and when. The call stack also manages all the data used by a function.
    - Broadly speaking, every time a function is called, all the variables and arguments relevant to that function call are pushed into the call stack to form what is called a stack frame.
        - The order of the individual elements containing variables and arguments may vary depending on the specific implementations
    - The stack frame is a memory location in the stack and every time a function returns a value and thus terminates, this memory is cleared and the stack frame is popped from the stack
        - Every time a new function is called, a stack frame is created storing the relevant data and the value returned by function called is passed on to the next stack frame
    - In the case of recursion, even when the function calls itself, it creates a new stack frame pushing all the useful data to the stack
        - After making multiple function calls and creating new stack frames every time a function calls itself, we get to the base case. After reaching the base case, something will be returned and we pop the stack frame from the stack. The whole procedure will then be terminated when the call stack is empty and a value can then be returned

```
function recursiveSum(n) {
    if (n == 0) {
        return 0;
}
return n + recursiveSum(n-1);
}
```

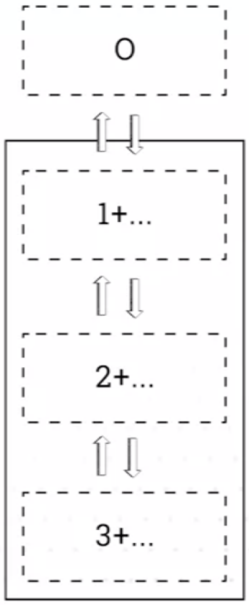
Let's call recursiveSum(3)

Return 0, pop the stack

Return 1, pop the stack

Return 3, pop the stack

Return 6, empty the stack



- The call stack needs to stored in the memory of the computer and memory is finite. So if you make too many recursive function calls (function calls within function calls), the call stack could get too big and we would have what is called stack overflow, where we run of memory with which to store the stack
    - If the program result in infinite recursion, you definitely get stack overflow, but even for programs without infinite recursion, the capacity of the stack could be exceeded

```
function CallStack(n)
    if n = 0 then
        return 1
    end if
    new Stack c
    for 1 ≤ i ≤ n do
        PUSH[i, c]
    end for
    a ← 1
    for 1 ≤ i ≤ n do
        a ← a × TOP[c]
        POP[c]
    end for
    return a
end function
```

Summary

In this week, we learned about the call stack and its use in recursion.