Algorithms & Data Structures I Week 15 Lecture Note

Notebook:	Algorithms & Data Structures I
Created:	2020-10-21 4:14 PM
Updated:	2021-01-19 5:56 PM
Author:	SUKHJIT MANN

Cornell Notes	**Topic:** Recursion, Part 1	Course: BSc Computer Science Class: CM1035 Algorithms & Data Structures I [Lecture] Date: January 19, 2021

Essential Question:

What is recursion?

Questions/Cues:

- What is decrease and conquer?
- What is the recursive pseudocode for classic recursive problem of Fibonacci numbers?
- How can the Euclidean or GCD algorithm be written recursively?
- How can we implement the Linear Search Algorithm recursively?
- How do we implement Bubble Sort recursively?
- How do we implement Insertion Sort recursively?
- How do store the permutations of ABC recursively?

Notes

- Decrease and conquer = for a general problem, we can start with an algorithm to solve a particular simpler instance of this problem for a particular input. Then for arbitrary inputs to the problem, we reduce our problem to the already solved simpler instance; recursion is what we use to do the reduction. We applying the algorithm within itself until we get down to the simpler instance or base case.

$$n! = n \times (n-1) \times (n-2) \times ... \times 2 \times 1$$

function Factorial(n)
    if $n = 0$ **then**
        return 1
    end if
    return $n \times$ Factorial(n-1)
end function

$$F_n = F_{n-1} + F_{n-2}$$

function Fibonacci(n)
    if $n \leq 2$ **then**
        return 1
    end if
    return Fibonacci(n-1) + Fibonacci(n-2)
end function

- A recursive function calls itself inside its body

function GreatestCommonDivisor(a, b)

 if $a = b$ **then**

  return a

 else if $a > b$ **then**

  return GreatestCommonDivisor($a - b, b$)

 else

  return GreatestCommonDivisor($a, b - a$)

 end if

end function

$$a = g \times x$$

$$b = g \times y$$

$$a - b = g \times (x - y)$$

```
function Search(v, l, item)
    n ← LENGTH[v]
    if l > n then
        return FALSE
    else if v[l] = item then
        return TRUE
    end if
    return Search(v, l + 1, item)
end function

function LinearSearch(v, item)
    return Search(v, 1, item)
end function
```

function Swap($vector, i, j$)

$x \leftarrow vector[j]$

$vector[j] \leftarrow vector[i]$

$vector[i] \leftarrow x$

return $vector$

end function

function Sort($vector, r$)
    if $r \leq 1$ **then**
        return $vector$
    end if
    for $1 \leq j \leq r - 1$ **do**
        if $vector[j+1] < vector[j]$ **then**
            Swap($vector, j, j+1$)
        end if
    end for
    Sort($vector, r - 1$)
    return $vector$
end function
function BubbleSort($vector$)
    $n \leftarrow$ LENGTH[$vector$]
    return Sort($vector, n$)
end function

function Shift($vector, i, j$)

  if $i \leq j$ **then**

    return $vector$

  end if

  $store \leftarrow vector[i]$

  for $0 \leq k \leq (i - j - 1)$ **do**

    $vector[i - k] \leftarrow vector[i - k - 1]$

  end for

  $vector[j] \leftarrow store$

  return $vector$

end function

function Sort($vector, r$)

  if $r \leq 1$ **then**

    return $vector$

  end if

  Sort($vector, r - 1$)

  $j \leftarrow r \quad i \leftarrow r$

  while $(vector[i] < vector[j - 1]) \wedge (j > 1)$ **do**

    $j \leftarrow j - 1$

  end while

  Shift($vector, i, j$)

  return $vector$

end function

function InsertionSort($vector$)

  $n \leftarrow \mathrm{LENGTH}[vector]$

  return Sort($vector, n$)

end function

```
function Permutations(vector)
    if LENGTH[vector] ≤ 1 then
        return vector as a dynamic array
    end if
    new DynamicArray s
    for 1 ≤ i ≤ LENGTH[vector] do
        new Vector v(LENGTH[vector] − 1)
        v[1 : i − 1] ← vector[1 : i − 1]
        v[i : LENGTH[vector] − 1] ← vector[i + 1 : LENGTH[vector]]
        new DynamicArray w ← Permutations(v)
        new Vector p(LENGTH[vector])
        p[1] ← vector[i]
        for 1 ≤ j ≤ LENGTH[w] do
            p[2 : LENGTH[vector]] ← w[j]
            s[LENGTH[s] + 1] ← p
        end for
    end for
    return s
end function
```

Summary

In this week, we learned about decrease and conquer recursion.