Algorithms & Data Structures I Week 11 Lecture Note

Notebook:	Algorithms & Data Structures I		
Created:	2020-10-21 4:14 PM	**Updated:**	2021-01-16 3:22 PM
Author:	SUKHJIT MANN		

Cornell Notes	**Topic:** What makes a good algorithm? Part 1	Course: BSc Computer Science Class: CM1035 Algorithms & Data Structures I [Lecture] Date: January 15, 2021

Essential Question:

What makes a good algorithm?

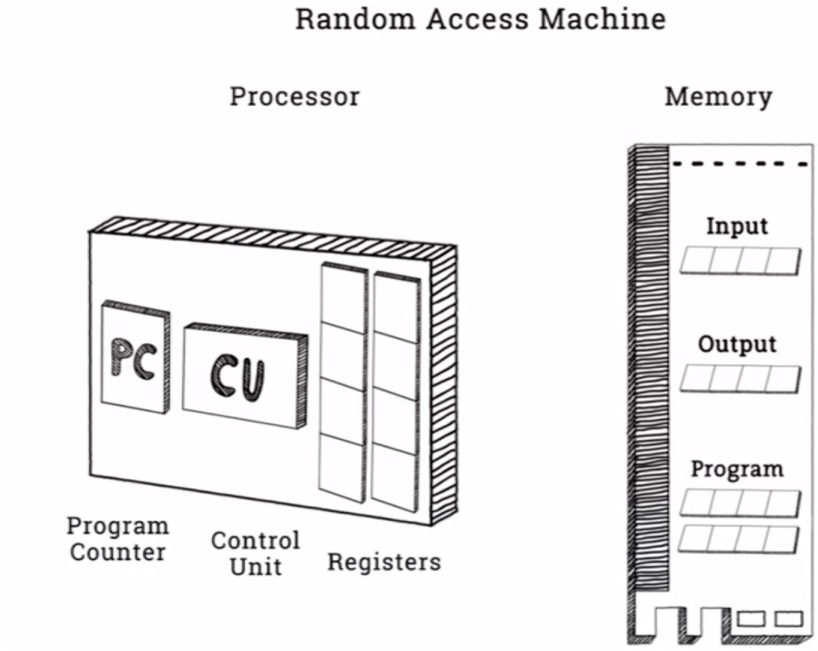
Questions/Cues:

- What basic architecture do all modern computers follow?
- What is RAM and external memory?
- What data does the CPU store?
- What is Random Access Machine model or RAM model?
- What is the control unit capable of?
- What is a register?
- How do we compare algorithms using the RAM model?
- What kinds of mathematical functions do we encounter in counting the number of operations for an algorithm?
- What is Big O Notation?
- What is the mathematical definition of Big O?
- What is the Big O hierarchy?

Notes

- Von Neumann architecture = the basic architecture followed by all modern day computers. The structure is as follows:
    1. CPU (Central Processing Unit)
    2. RAM (Random Access Memory)
    3. External forms of memory (Hard Drives, SSD, flash drives, SD cards and so on)
    - RAM is the memory that's accessible to a CPU and is quite mutable(changing) as a result. If many computations are being done, then the values stored in RAM can change frequently. The external memory is good for storing larger files in a stable manner
    - When we want to perform computations on the data in external memory we just load it into RAM at which point the CPU can access it
    - At the CPU/RAM level of the architecture, data is represented as binary digits or bits (The numbers 0 and 1)
        - Most computers deal with not individual bits but a byte or a collection of 8 bits
    - So at its most basic level, a computer stores a byte in its memory with each byte having its own address

        - This basically how RAM works, its made up of bytes with each byte stored in a register, each with its own address. A computer can any of these registers if given the relevant address
        - In these registers, RAM can stored the computer program to be implemented as well the data upon which a program will be implemented. Furthermore, it can store the output of the program
- CPU (data stored) = CPU has multiple data registers just like RAM, but the number of registers is limited. Some of the registers store data upon which computations will be done and other registers keep track of where the CPU is in the implementation of a program as well as the description of that particular action in the program
    - On these registers, the CPU can perform computational tasks, tasks such as arithmetic operations like addition and subtraction as well as logical operations like AND, OR, along with if-then statements permitting conditional operations
- RAM model = where random access refers to how the machine can access the data in any register in one single operation, as long as the machine is given the particular index of that register. So if we pick a register randomly, we can access it directly. The RAM is as follows:



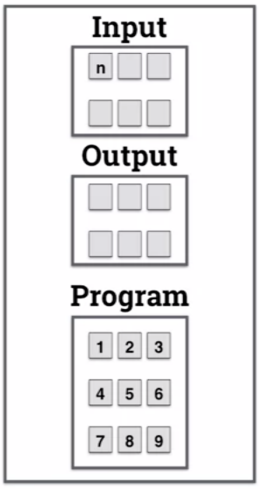
    - The processor (CPU) its going to do computational tasks
        - Program counter which is a register that is going to store a number
        - Control unit which is going to be doing all of the computation by handling the data and producing other data
        - Registers which is going to be where we temporarily store data for a control unit to handle
    - Memory(RAM)
        - The input is going to be the data which is going to be the input to a problem
        - The output is going to the output from solving the problem
        - The program is going to be a store of all of the instructions that are going to be implemented by the control unit, just like a computer program when coding
- Control Unit = The control unit can read data from the input, it can write data to the output, and it can also read data which are instructions in the program. Then it can edit the program counter, helping it keep track of where it is in the program. The control unit can also handle the data inside the registers
    - Control unit can read, write and copy values of memory units. Also it can do simple arithmetic such as addition, subtraction, multiplication and division. It can also perform conditional operations (if...then...)
    - Each individual operation is done in time-step, for example one tick in a time clock means one operation performed
- Registers = Each memory unit can store an arbitrary integer, this integer must be non-negative for the Program counter because the PC is keeping of it is in the program, so it needs to track of where it is in

the set of instructions. Depending on the value stored in all of these memory units, the control unit will do a particular operation, it can also write new values to these memory units.

****Note** the "size" of the program that's stored in the memory is not necessarily the number of time-steps in any implementation of that program

- Comparing Algorithms = we can account for how many operations are carried out by the RAM machine for a particular algorithm and then compare it with the number of operations carried out by another algorithm. We assume that each operation is carried out in sequence, one after the other, and every instance there is an operation, this happens in unit of time or one time-step. If two operations are carried out in the RAM model then two time-steps have elapsed

```
function Factorial(n)
    a ← 1
    for 1 ≤ i ≤ n do
        a ← a × i
    end for
    return a
end function
```




```
function Factorial(n)
    a ← 1
    for 1 ≤ i ≤ n do
        a ← a × i
    end for
    return a
end function
```

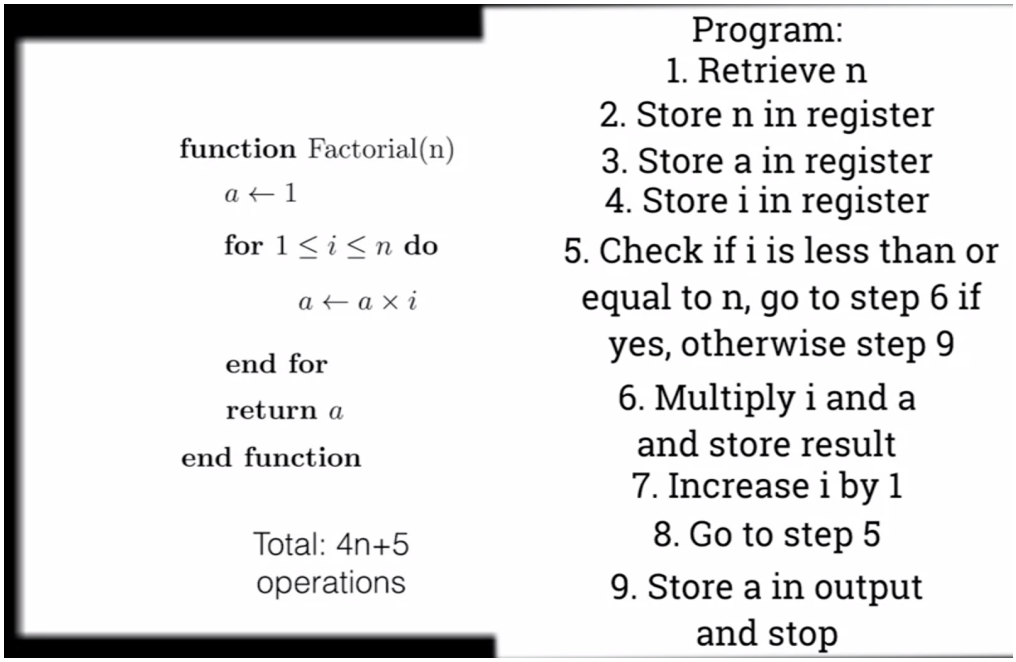
Steps 5-8 repeat n times

Program:

1. Retrieve *n*
2. Store *n* in register
3. Store *a* in register
4. Store *i* in register
5. Check if *i* is less than or equal to *n*, go to step 6 if yes, otherwise step 9
6. Multiply *i* and *a* and store result
7. Increase *i* by 1
8. Go to step 5
9. Store *a* in output and stop

- We see in this factorial algorithm that as n increases more time-steps are required to implement the algorithm, and so we need more time to complete the computation
- In this way, time can be seen as a resource that gets used up in any implementation. We are interested in how much of this resource we need to devote to a particular task
- Another resource worth being concerned about is the amount of memory or space need to complete a task. We can look at the number of registers required to complete a task within the RAM model and call this the space
- An algorithm gets expensive depending on how much space it uses as the input gets larger

- In the study of algorithms, time and space are the main resources that we wish to quantify in implementations of algorithms. Sometimes algorithms may require many time-steps but very little space. The space requirements in an algorithm always give a lower bound on the number of time-steps
    - In simple terms, storing data in a register can be done in a single time-step. So storing n numbers in n registers will require at least n time-steps. This way, the number of time-steps will always be larger than the space requirements



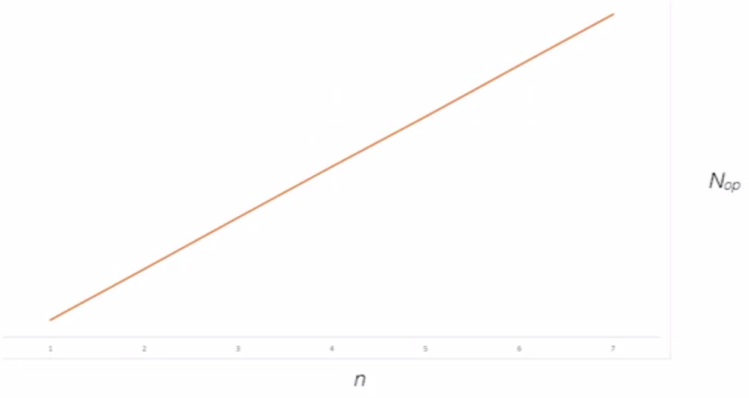
- The number of operations required to implement the algorithm depends on n or is a function of n. In other words, for every input n to our algorithm, the number of operations required to complete the algorithm is a function of n

Number of operations depends on n

$$N_{op} = f(n)$$

For factorial:

$$N_{op} = an + b$$



Quadratic: $f(n) = an^2 + bn + c$

Polynomial: $f(n) = an^k + bn^{k-1} + \ldots + cn + d$

Exponential: $f(n) = 2^n$ $f(n) = 3^n$

Logarithmic: $f(n) = log_2 n$ $f(n) = log_3 n$

    - the number of operations is a function of the input in general, depending on the function used, it will tell us the resources required to complete the algorithm as the input gets larger
- Big O Notation = ignores constants in favour of dependencies on the variables, such as n and focusing on the fastest-growing part of a function
    - If we have a constant function with no dependencies on n, we write O(1)
    - Big O represents a class or set of functions
    - $O(\log_2 n) = O(\log_3 n)$

Bases

$O(log_3 n)$

Rewrite into base 2 using change of base formula

$log_2 n = log_3 n / log_3 2$

$log_3 n$ is just $log_2 n$ multiplied by a constant

We don't need to worry about this constant

$O(log_3 n) = O(log_2 n)$

    - For all log bases, we can have just one Big O class O(log n)
    - 4^n and 2^n are two different Big O classes

$$f(n) \in O(g(n))$$

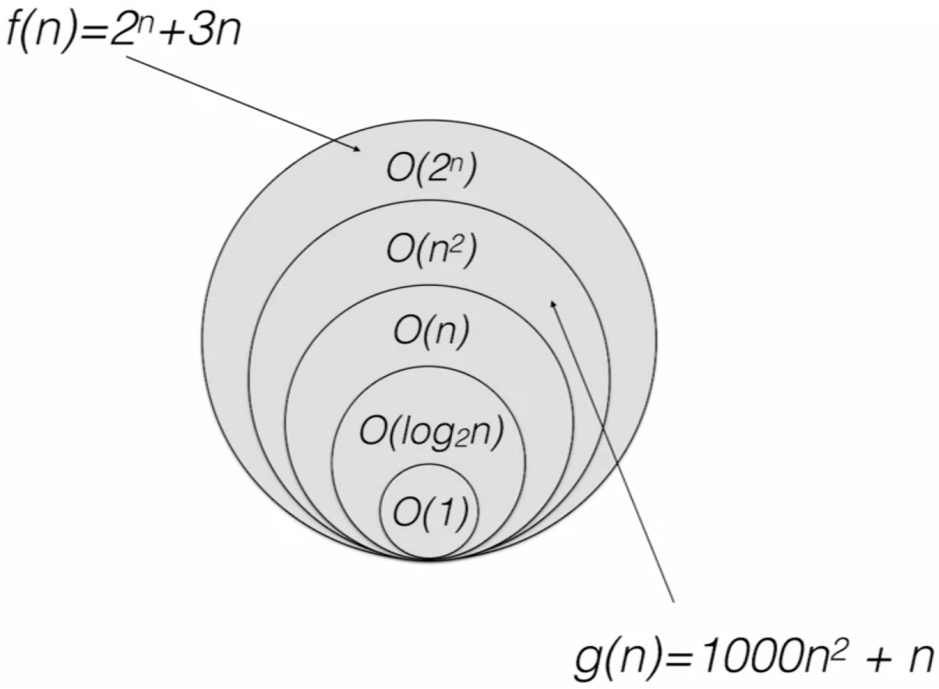
$$\exists\ k > 0$$

$$\exists\ n_0$$

Such that

$$\forall\ n > n_0$$

$$|f(n)| \leq k \cdot g(n)$$



- If one algorithm with input n takes n^2 time-steps and another takes 2^n time-steps, both functions in n live inside O(2^n) but clearly n^2 takes far fewer time-steps. So in algorithmic analysis, we always to make sure of the smallest Big O class in which a function lives

Summary

In this week, we learned about the RAM model, the different kinds of functions used in comparing algorithms, Big O Notation and so on.