Notes on Chapter 9 - A Simplistic Introduction to Algorithmic Complexity

Swarup Tripathy *

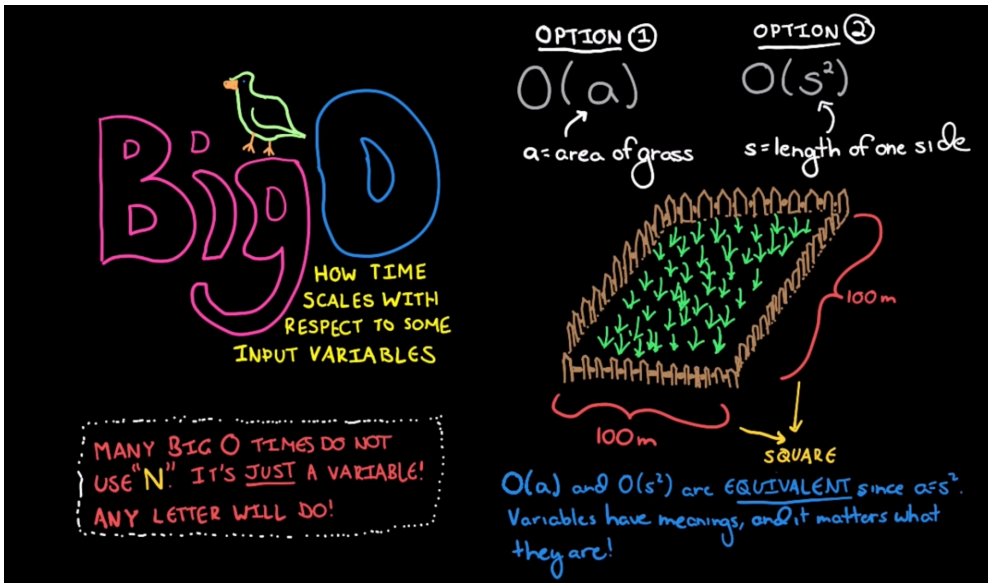
May 2022

A curated list of important points for my reference.

1. Exhaustive enumeration was so slow as to be impractical for many combinations of x and epsilon. For example, evaluating *squarerootExhaustive(100,0.0001)* requires roughly one billion iterations of the while loop. In contrast, evaluating *squarerootBi(100,0.001)* takes roughly twenty iterations of a slightly more complex while loop.
2. Asymptotic Notation describes the Complexity of an algorithm as the size of its inputs approaches Infinity.
3. Following rules of thumb in describing the asymptotic Complexity of an algorithm
   - If the running time is the sum of multiple terms, keep the one with the larges growth rate, and drop the others
   - If the remaining term is a product, drop any constants.
4. **Big O Notation**
   - So, the story begins in 2009 in a region of south africa when there was a slow internet issue in the office.
   - They wanted to test how much time will it take for the internet to send a chunk of data to another office located 50 miles away as compared to a bird carrying that chunk of data.
   - The Bird won, and it went all over the press that the bird was faster than the internet but this was something of a ridiculous experiment.
   - Everything depends upon the chunk of data as in for the bird this would remain the same while for the internet this would be something of an additional load. It's not going to take longer for the bird since the usb stick carries more data.

*John V Guttag

- So of course for a sufficiently large file the pigeon's going to win.
- We define the pigeon transfer speed as *O(1)* → it's constant time wrt the size of the input. It doesn't take any longer with more input.
- Whereas the Internet Transfer time is *O(n)* → it scales linearly with respect to the amount of input. Twice amount of data will take roughly twice the amount of time.
- BIG O → How time scales with respect to some input variables.



How much time does it take to mow the grass off the field.

- Important rules of Big O
  - Different Steps get added

```
function something(){
    doStep1(); // O(a)
    doStep2(); // O(b)
}
// after adding their complexity it becomes O(a+b)
```

  - Drop Constants

```
function minMax1(array){
    min,max = NULL
    for each e in array
        min = MIN(e,min)
    for each e in array
        max = MAX(e,max)
}
```

```
// Above: individual complexity O(n) and O(n) and
// as a whole remains O(n) and not O(2n) <= drop the constant 2.

        function minMax1(array){
            min,max = NULL
            for each e in array
                min = MIN(e,min)
                max = MAX(e,max)
        }
// Above: complexity altogether is O(n)
```

– Different inputs → Different Variables

```
        int intersectionSetSize(arrayA, arrayB){
            int count=0;
            for a in arrayA{
                for b in arrayB{
                    if a==b{
                        count = count + 1
                    }
                }
            }
            return count
        }
// Above: complexity is not O(n^2) rather it will be O(a.b)
```

– Drop non-dominate terms

```
        function example(arrray){
            max=NULL
            for each a in array{
                max = MAX(a,max)
            }
            print max
            for each a in array{
                for each b in array{
                    print a,b
                }
            }
        }
// Above: here complexity as discusses before could be O(n+n^2) but
// here n^2 dominates so we neglect n and
// final Big O notation will be O(n^2)
```

5. Generating Power Set
    - empty powerSet list
    - i ranges from 0 to 2**len(given list)
    - generating binary representation of i
    - empty list subset
    - j ranges from 0 to len(given list)
    - whenever j encounters a '1' in the binary representation then it'll append that digit from the given list to subset
    - further the subsets are then appended to the powerset list
6. Logarithmic algorithm are almost as good as constant time ones.
7. Most of a time a linear algorithm is acceptable efficient.