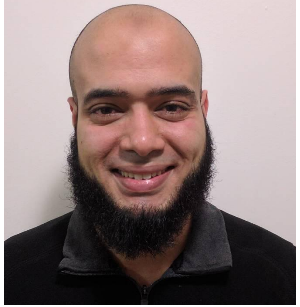
Data Structures

Asymptotic Complexity 3

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Big O notation: Some math

- Assume your code takes: 9N+17 steps ⇒ O(N) Order
- There is some **constant C**, where for any input size N: 9N+17 < CN
- Actually O(n) means there is some constant multiplied in this n
  - For example, let C = 30
  - Then 9N + 17 < 30N for **ANY** N
- What does this imply?
- Big O is an **Upper limit** to the number of steps regardless of these constants and factors in 9N+17
  - So 30N is **always bigger** than 9N + 17. So It is O(n)

Big O notation: an upper bound

- Assume we have a function F(N).
- Its total number of steps T(N) = N + 2N + 5N^2
  - Clearly T(N) is O(N^2), but what is proper C?
  - Let's try C = 6. This means T(N) < 6N^2 for any N ?

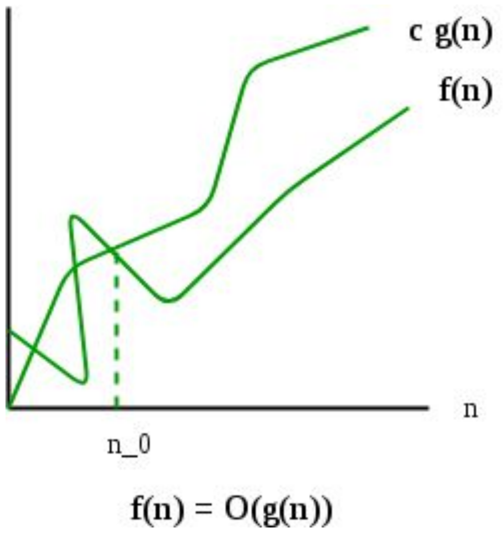
	T(N) = N + 2N + 5N^2	6N^2
N = 1	1 + 2 x 1 + 5 x 1 x 1 = 8	6 x 1 x 1 = 6 ⇒ 8 < 6? NO
N = 2	2 + 2 x 2 + 5 x 2 x 2 = 26	6 * 2 * 2 = 24 ⇒ 26 < 24? NO
N = 3	3 + 2 x 3 + 5 x 3 x 3 = 54	6 x 3 x 3 = 54 ⇒ 54 < 54? No
N = 4	4 + 2 x 4 + 5 x 4 x 4 = 92	6 x 4 x 4 = 96 ⇒ 92 < 96? YES
N = 5	5 + 2 x 5 + 5 x 5 x 5 = 140	6 x 5 x 5 = 150 ⇒ YES

Big O notation: an upper bound

- In the previous table, N = 1, 2, 3, our C was not good
- But, starting from 4, T(N) is always less than 6N^2
- Let's state O() more **formally**
  - T(N) is O(G(N)) IFF we could find:
    - n0 < N
    - Constant C, such that T(N) < C * G(N) for any N > n0
  - In our case:
    - T(N) = N + 2N + 5N^2 ⇒ Total number of steps
    - G(N) = N^2 ⇒ Our guessed order O(N^2)
    - **n0 = 4** ⇒ The starting point
    - **C = 6** ⇒ The constant

Big O notation: an upper bound

- As you see, starting from some point n0
- Our order function g(n) is always higher than f(n) with a specific C
  - So it is an **upper** function
- Note if some C is working well, any higher also
  - So, if C=6 represents an upper function...
  - Then this is true for C=7,8,9 and higher too!
- Note if g(n) is upper bound, then higher ones also
  - E.g. if g(n) is O(n^2), then O(n^3) and O(n^4) are upper bound too
  - We always use the most tightly bound one (O(n^2) here)



Img src

Enough math

- In practice
  - We don’t compute or care a lot about this X
  - Just follow the tips from the last lectures to compute the order like a pro!
- C idea is helpful in building understanding that order is an **upper function**
- If you did not understand the previous slides well = totally ok
  - Skip and repeat at the end of the course

Same order

- Consider the 2 functions f1 and f2
- Both of them are O(n^3)
- This means they **grow cubic** in time, which is too much!
- But in practice, do they take the same amount of time?

```python
def f1(n=1000):     # O(n^3
    cnt = 0
    for i in range(n):
        for j in range(n):
            for k in range(n):
                cnt += 1


def f2(n=1000):     # O(n^3)
    cnt = 0
    for i in range(n):
        for j in range(i, n, 1):
            for k in range(j, n, 1):
                cnt += 1
```

Same order

- In terms of operations:
  - For n = 1000
  - F1 = 1000,000,000
  - F2 = 167,167000
- The key point:
  - We can have 2 codes of the same order, e.g. O(n^3)
  - However, one of them is still faster
  - E.g. C1 = 7 vs C2 = 2
  - Smaller constant ⇒ faster
- Tip: build code with small C :)

```python
def f1(n=1000):      # O(n^3
    cnt = 0
    for i in range(n):
        for j in range(n):
            for k in range(n):
                cnt += 1
    # cnt = 1000,000,000


def f2(n=1000):      # O(n^3
    cnt = 0
    for i in range(n):
        for j in range(i, n, 1):
            for k in range(j, n, 1):
                cnt += 1

    return cnt
    # cnt 167,167000
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."