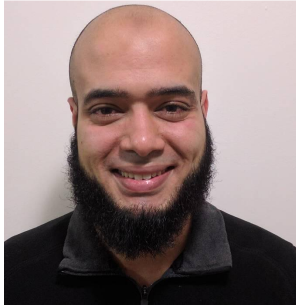
Data Structures

Asymptotic Complexity (2)

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)



How can skilled programmers find the order
VERY quickly?

Tips

- This code involves ~8 steps
- **Ignore** all constants
- Constants don't affect the overall order when n is large
- All constants are simply FIXED numbers
- So, the code is just **O(1)**

```python
def constant_order1():    # O(1)
    start = 6
    end = 100
    mid = (end - start) // 2
    if mid % 2 == 0:
        del mid
```

Tips

- Too many steps?
- Yes; but a FIXED number!
  - Useless with very large N
- Ignore them
- Tip: Ignore anything that doesn't involve our variables (e.g. n)

```
def constant_order2():
    start = 7
    end = 0
    for i in range(1000):
        end += end * 2 + start
```

Tips

- Search for loops that are based on **n**
- A single loop is O(n)
- Two nested loop is O(n^2)
- Triple nested loops is O(n^3)
- *And so on*
- This code shows a single loop
- inside it is a FIXED number of operations
  ⇒ IGNORE!

```python
def linear1(n):      # O(n)
    sum = 0
    for i in range(n):
        # All below are O(1)
        x = 2 + 3 * 4
        sum += i
        sum += 2
        sum += x
```

Tips

- 2 parallel loops.
- Each is a single loop
- Each loop depends on n
    - One is 10n and one is 5n
    - Ignore these constants
- Practically: 10n + 5n = 17n
- Ignore constants ⇒ O(n)
- Tip: what is the **deepest**?
    - A single loop ⇒ O(n)

```python
def linear2(n):        # O(n)
    for i in range(10 * n):
        constant_order1()

    for i in range(5 * n):
        constant_order1()
```

Tips

- This is 5n x 3n loop steps
  - Multiplied with some factor from all these FIXED steps
  - Overall O(n^2)
- Tip: nested loops ⇒ O(n^2)

```python
def quadratic1(n):  # O(n^2)
    cnt = 0
    for i in range(5 * n):
        for j in range(3 * n):
            cnt += 1
            constant_order1()
```

Tips

- We have 2 parallel blocks
  - Nested loops: O(n^2)
  - Linear loop: O(n)
- Tip: focus on the biggest
  - As it dominates the operations
  - n^2 + n ⇒ n^2

```python
def quadratic2(n):  # O(n^2)
    cnt = 0
    for i in range(5 * n):
        for j in range(3 * n):
            cnt += 1
            constant_order1()

    for i in range(10 * n):
        constant_order1()
```

Tips

- 2 parallel blocks
  - 3 nested loops
  - 1 loop
- But in the 3 nested loops
  - One loop is just a fixed # of operations
  - Again **ignore constant operations**
  - This 3rd loop is useless (ignore)
- Total: 15000 n^2 + 10n ⇒ n^2

```python
def quadratic3(n):  # O(n^2)
    cnt = 0
    for i in range(5 * n):
        for j in range(3 * n):
            for k in range(1000):
                cnt += 1
                constant_order1()

    for i in range(10 * n):
        constant_order1()
```

Tips

- 2 parallel blocks
  - A single loop
  - A single loop
- So O(n)? No, there is a trick
- The 2nd loop is not linear in n
  - It iterates **3 n^2** steps
- The order of the second loop is O(n^2)
- Tip: observe if the number of loop operations is fixed, n, n^2 etc...
  - This value decides the order!

```python
def quadratic4(n):  # O(n^2)
    for i in range(10 * n):
        constant_order1()

    for i in range(3 * n * n):
        constant_order1()
```

Tips

- As this code has 3 nested loops
  - Each depends on n
  - It is O(n^3)

```python
def cubic1(n):  # O(n^3)
    cnt = 0
    for i in range(n):
        for j in range(n):
            for k in range(n):
                cnt += 1
```

Tips

- 2 parallel blocks
  - 3 nested loops ⇒ n^3
  - 2 nested loops ⇒ n^2
    - Don't be distracted by the size of the constant (1000)
    - IGNORE constants
- n^3 + n^2 ⇒ O(n^3)
  - Always focus on the **biggest**

```python
def cubic2(n):  # O(n^3)
    cnt = 0
    for i in range(n):
        for j in range(n):
            for k in range(n):
                cnt += 1

    for i in range(1000 * n):
        for j in range(1000 * n):
            cnt += 1
```

Tips

- Why not O(n^3)?
    - Our loops are n^2, n, and n^3 respectively
    - Total O(n^6)
- Again, **double check** if the number of operations in the loop is based on a *fixed constant number, n, n*^2 etc...

```python
def f(n):    # O(n^6)
    cnt = 0
    for i in range(n * n):
        for j in range(n):
            for k in range(n * n * n):
                cnt += 1
```

Tips

- F1 is O(n^3)
  - n*n then n
- F2 has a single loop: O(n)
  - But its body is NOT constant!
  - The body contains a call that is O(n^3)
- Overall O(n^4)
- Tip
  - Imagine we copy-pasted f2 in f1
  - You should see clearly it is O(n^4) steps in total
- Tip
  - Double check each line of code (like a function call): is fixed or variable?

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


def f2(n):  # O(n^4)
    for i in range(n):
        f1(i)
```

Tips

- Sometimes our function depends on several variables
- Total is 6nm steps
- Drop the constant value ⇒ **O(nm)**

```python
def f3(n, m):   # O(nm)
    cnt = 0
    for i in range(2 * n):
        for j in range(3 * m):
            cnt += 1
```

Tips

- 2 parallel blocks
  - Block 1: O(nm)
  - Block 2: O(n^2)
- Which is bigger? We don't know
- Total: O(nm + n^3)

```python
def f4(n, m):   # O(nm + n^3)
    cnt = 0
    for i in range(2 * n):
        for j in range(3 * m):
            cnt += 1

    for i in range(n * n * n):
        cnt += 1
```

Polynomial Order

- Today we discussed polynomial order functions (format n^k)
  - n^0 = 1 (const), n^2, n^3 and so on
- Intuition: code is doing some hundred million steps ⇒ ~ 1 second (not really)
- From the table, the **bigger** your O() the **slower** your code
- There are other worse families
  - E.g. O(n^n) or O(n!)

	n=100	n=1000	n=1000000
O(n)	100	1000	**1000000**
O(n^2)	10000	**1000000**	Too much
O(n^3)	**1000000**	1000000000	Too much
O(n^4)	100000000	Too much	Too much

Overall

- Try to keep these tips in mind
- Don't be overly systematic. Some codes are trickier!
  - E.g. 3 nested while-loops might actually just do 10n steps. It's not necessarily O(n^3)
    - We will see several cases during my other courses
- Whenever you write a code from now on, always compute its time order
  - This is how you will develop your skill!

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”