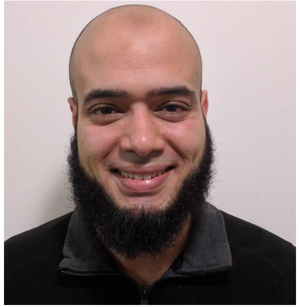
Data Structures

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 skilled programmers find the order

VERY fast?

Tips

- This code involves ~8 steps
- Ignore all constants.
- They don’t affect us
- All these are a FIXED number
- So code is just O(1)

```
void ConstantOrder1() {
    // O(1)
    int start = 6;
    int end = 100;

    int mid = (end - start) / 2;

    if (mid % 2 == 0)
        --mid;
}
```

Tips

- Too many steps?
- Yah BUT FIXED
    - Useless with very large N
- Ignore them
- Tip: Ignore anything that doesn't involve our factor N

```
void ConstantOrder2() {
    int start = 7;
    int end = 0;

    for (int i = 0; i < 1000; ++i)
        end += end * 2 + start;
}
```

Tips

- Search for loops that is based on n
- A single loop is O(n)
- Nested loop is O(n^2)
- Triple nest loop is O(n^3)
- And so on
- On right: single loop
  - Inside it fixed operations = IGNORE

```
void linear1(int n) {   // O(n)
    int sum = 0;
    for (int i = 0; i < n; i++) {
        // All below are O(1)
        int x = 2 + 3 * 4;
        sum += i;
        sum += 2;
        sum += x;
    }
}
```

Tips

- 2 parallel loops. Each is single loop
- Each depends on n
  - One 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)

```
void linear2(int n) { // O(n)
    for (int i = 0; i < 10 * n; i++)
        ConstantOrder1();

    for (int i = 0; i < 5 * n; i++)
        ConstantOrder1();

}
```

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)

```
void quadratic1(int n) { // O(n^2)
    int cnt = 0;
    for (int i = 0; i < 5 * n; ++i) {
        for (int j = 0; j < 3 * n; ++j) {
            cnt++;
            ConstantOrder1();
        }
    }
}
```

Tips

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

```
void quadratic2(int n) { // O(n^2)
    int cnt = 0;
    for (int i = 0; i < 5 * n; ++i) {
        for (int j = 0; j < 3 * n; ++j) {
            cnt++;
            ConstantOrder1();
        }
    }
    for (int i = 0; i < 10 * n; i++)
        ConstantOrder1();
}
```

Tips

- Again 2 parallel things
  - 3 nested loop
  - 1 loop
- But in 3 nested loop
  - One loop is just fixed operation
  - Again ignore constant operations
  - This 3rd loop is useless
- Total: 15000 n^2 + 10n ⇒ n^2

```
void quadratic3(int n) { // O(n^2)
    int cnt = 0;
    for (int i = 0; i < 5 * n; ++i) {
        for (int j = 0; j < 3 * n; ++j) {
            for (int k = 0; k < 1000; ++k) {
                cnt++;
                ConstantOrder1();
            }
        }
    }
    for (int i = 0; i < 10 * n; i++)
        ConstantOrder1();
}
```

Tips

- 2 parallel blocks
  - Singel loop
  - Single loop
- Ok then O(n)? No, there is a trick
- The 2nd loop is not linear in n
  - It moves 3 n^2 steps
- The order of the second loop is O(n^2)
- Tip: observe if the loop based on fixed, n, n^2 or what
  - Its value decides its order!

```
void quadratic4(int n) { // O(n^2)
    for (int i = 0; i < 10 * n; i++)
        ConstantOrder1();

    for (int i = 0; i < 3 * n * n; i++)
        ConstantOrder1();
}
```

Tips

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

```
void cubic1(int n = 1000) { // O(n^3)
    int cnt = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            for (int k = 0; k < n; ++k) {
                cnt++;
            }
        }
    }
}
```

Tips

- 2 parallel loops
  - 3 nested loops ⇒ n^3
  - 2 nested loops ⇒ n^2
    - Don’t be cheated by 1000 constant
    - IGNORE constants
- n^3 + n^2 ⇒ O(n^3)
  - Always focus on the biggest

```
void cubic2(int n) { // O(n^3)
    int cnt = 0;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < n; ++k)
                cnt++;

    for (int i = 0; i < 1000*n; ++i)
        for (int j = 0; j < 1000*n; ++j)
            cnt++;
}
```

Tips

- Why not O(n^3)?
  - First loop is n^2, then n, then n^3
  - Total O(n^6)
- Again double check if loop is based on n or what?

```
void f(int n) { // O(n^6)
    int cnt = 0;
    for (int i = 0; i < n * n; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < n * n * n; ++k)
                cnt++;

}
```

Tips

- We know f1 is o(n^3)
- Now f2 has a single loop: o(n)
  - But its body is NOT constant!
  - Its body is call that is O(n^3)
- Overall O(n^4)
- Tip
  - Imagine we copy-pasted f2 in f1
  - Now u see clearly n^4 total steps
- Tip
  - Double check if the body is FIXED or variable based on N

```
void f1(int n) { // O(n^3)
    int cnt = 0;
    for (int i = 0; i < n * n; ++i)
        for (int j = 0; j < n; ++j)
            cnt++;
}

void f2(int n) { // O(n^4)
    for (int i = 0; i < n; ++i)
        f1(i);  // n^3
}
```

Tips

- Sometimes our function depends on several variables
- Here we have total: 6nm
- Drop constants ⇒ O(nm)

```
void f3(int n, int m) {     // O(nm)
    int cnt = 0;
    for (int i = 0; i < 2 * n; ++i)
        for (int j = 0; j < 3 * m; ++j)
            cnt++;
}
```

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)

```
void f4(int n, int m) {     // O(nm + n^3)
    int cnt = 0;
    for (int i = 0; i < 2 * n; ++i)
        for (int j = 0; j < 3 * m; ++j)
            cnt++;

    for (int i = 0; i < n * n * n; ++i)
        cnt++;
}
```

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 (later)
  - 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
- But be careful from tricky codes, so don’t be so systematic
  - E.g. 3 nested while loops might actually just doing 10n steps NOT n^3
- Whenever you write a code from now one, always compute its order
  - This is how your skill will grow up
  - As it is your code, you know really what is happening

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."