Algorithm Analysis (CCDSALG)

Pau Rivera

Write a script that will get the factorial of any given whole number.

```
value = int(input("Please input an integer: "))
factorial = 1
factor = value

while factor > 1:
    factorial = factorial * factor
    factor = factor - 1

print("The factorial of", value, "is", factorial)
```

Write a script that will get the factorial of any given whole number.

← Problem

```
value = int(input("Please input an integer: "))
factorial = 1
factor = value

while factor > 1:
    factorial = factorial * factor
    factor = factor - 1

print("The factorial of", value, "is", factorial)
```

← ?

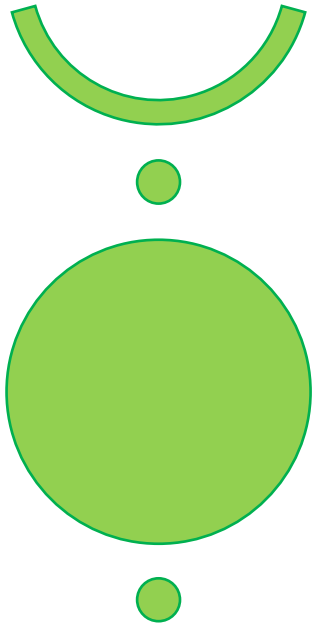
Write a script that will get the factorial of any given whole number.

← Problem

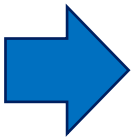
```
value = int(input("Please input an integer: "))
factorial = 1
factor = value

while factor > 1:
    factorial = factorial * factor
    factor = factor - 1

print("The factorial of", value, "is", factorial)
```

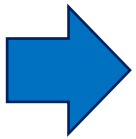
← Solution! ALGORITHM

This is the focus!



Input





Output

ALGORITHMS

Example

Input: a sequence of n numbers $\{a_1, a_2, \dots, a_n\}$

Output: a permutation (reordering) $\{a'_1, a'_2, \dots, a'_n\}$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Solution: ???

ALGORITHMS

Example

Input: a sequence of n numbers $\{a_1, a_2, \dots, a_n\}$

Output: a permutation (reordering) $\{a'_1, a'_2, \dots, a'_n\}$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Solution: Sorting algorithms

ALGORITHMS

Image credits

ALGORITHMS

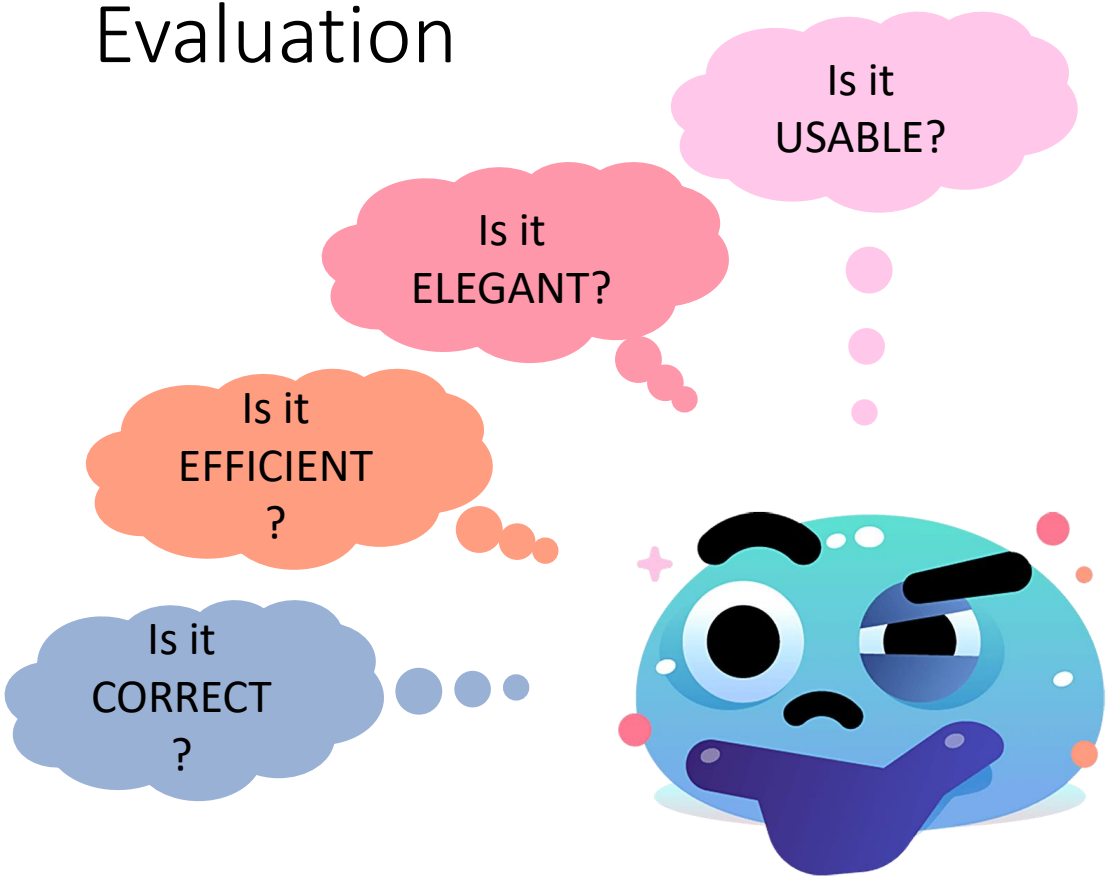
- There can be **more than one algorithm** to solve a given problem.
- An algorithm can be **implemented using different programming languages** on different platforms

Evaluation

Understanding how solutions that solve the same problem can differ

Evaluation

4 questions to ask when evaluating a solution



Evaluation

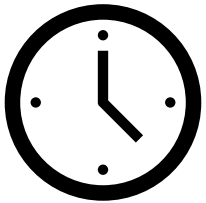
- **Correctness**
  - Does your solution actually solve the problem you set out to solve?
- **Efficiency**
  - Does it use resources reasonably?
- **Elegance**
  - Is it simple yet effective?
- **Usability**
  - Does it provide a satisfactory way for the target audience to use it?

Is it **EFFICIENT**?

- use an **acceptable amount of resources** in solving the problem

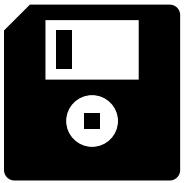
Efficiency

- Computers utilize **resources** (memory, computation)



Time

Duration of an algorithm's run time from start to end

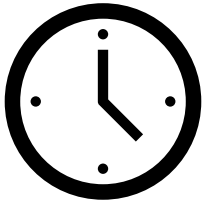


Space

Amount of memory needed to process

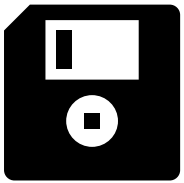
Efficiency

- Computers utilize **resources** (memory, computation)



Time

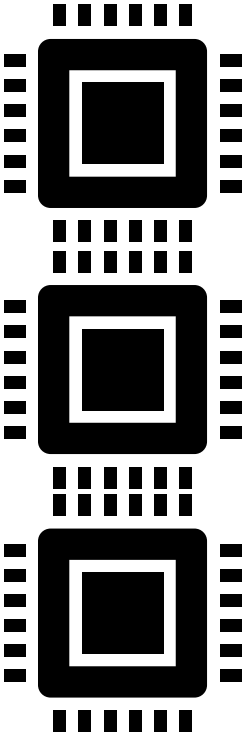
What parts of the algorithm affects the runtime?



Space

How does the choice of data structure affect the runtime?

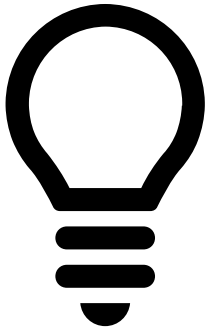
ANALYZING ALGORITHMS



Assumptions

- Instructions are executed **ONE AT A TIME**, no concurrent operations.
- Instructions are implemented following instructions commonly found in real computers (load, store, copy, add, subtract, multiply, divide, remainder, floor, ceiling, return, etc.).

ANALYSIS METHODS



A priori Analysis

Obtains a function bounding the time complexity through from mathematical facts.

A posteriori Analysis

Study the exact time and space required for execution using actual experiments

Conditionals

```
if (a > 0 && a + b < 30)
      c = a * a * a;
else if (a > 0)
      c = a * a;
else if (a == 0)
      c = a * a * a / 5 - 20;
else
      c = a;
```

Say you have an array with data. You want to traverse it using a loop, how would you determine the number of iterations?

Iterations: Nested Loops

```
x = 0;
m = A.length;

for (i=0; i < m; i++)
     x += A[i];
```

Say you have an array with data. You want to traverse it using a loop, how would you determine the number of iterations?

A posteriori Analysis

	Input Size	
	$n = 10$	$n = 100$
Algorithm 1	1 sec	10 secs
Algorithm 2	3 secs	15 secs

Which of the two algorithms is more efficient?

A posteriori Analysis

	Input Size		
	$n = 10$	$n = 100$	$n = 1000$
Algorithm 1	1 sec	10 secs	100 secs
Algorithm 2	3 secs	15 secs	30 secs

Which of the two algorithms is more efficient?

A posteriori Analysis

It is impossible to know the **exact amount of time** to execute any command unless the following are known:

- Machine for execution
- Machine instruction set
- Time required by each machine instruction
- Translation of the compiler from source to machine language

Algorithm Analysis

We want each comparison to be hardware-independent.

We can't use runtimes because they aren't easily replicable over different systems.

But we can approximate running time (**T(n)**) by counting the number of instructions(?) in an algorithm.

A priori Analysis

Let A[i] be the ith number on the list $(a_1, a_2, \ldots, a_n)$

```
[1] max, min = A[1]
[2] for i = 2 to n
[3]      if A[i] > max then
[4]                max = A[i]
[5]      if A[i] < min then
[6]                min = A[i]
[7] return max + min
```

```
[1] 2
[2] n
[3] n - 1
[4] n - 1
[5] n - 1
[6] n - 1
[7] 1

Total: 5n-1 = O(n)
```

Assumptions:

- Instructions are executed sequentially
- Each instruction takes 1-time unit

Algorithm Analysis

- Time taken by an algorithm grows with the input size
  - **Input size** – Number of items in the input
  - **Running time** – Number of steps executed

Amount of data

So far in school…

In reality...

Algorithm Analysis

- Time taken by an algorithm grows with the input size
  - **Input size** – Number of items in the input
  - **Running time** – Number of steps executed

- Different scenarios can also affect the Running time
  - Best case scenario
  - Average case scenario
  - Worst case scenario

Let's apply linear search algorithm to find #1 from a list of numbers

Average case

Best case

Worst case

Disclaimer! This depends on the algorithm used ☺

Algorithm Analysis

- Time taken by an algorithm grows with the input size
  - **Input size** – Number of items in the input
  - **Running time** – Number of steps executed

- Our concern: **Rate of Growth** or **Order of Growth**

Which one is better?

Look at these two frequency count polynomials:

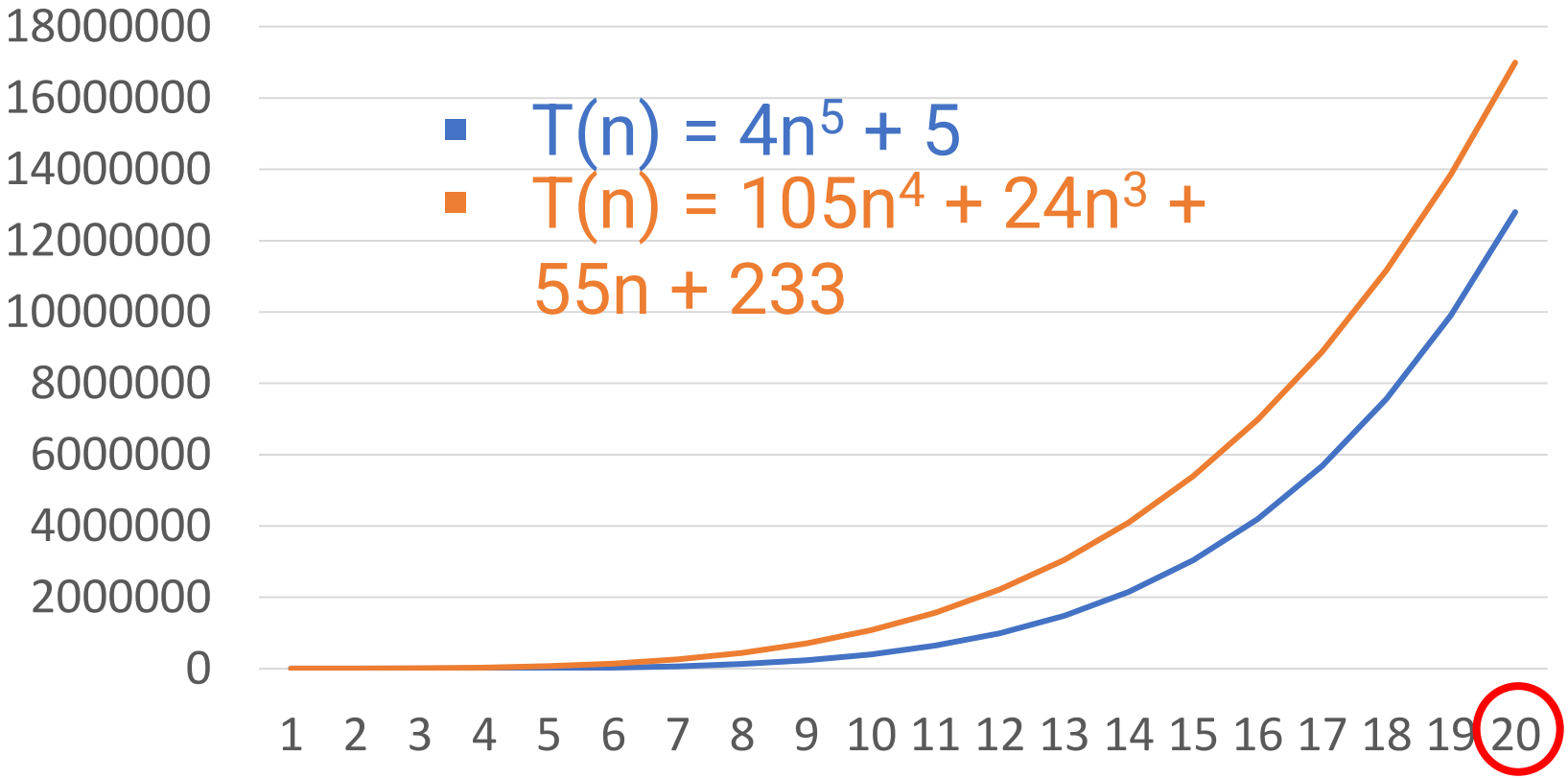
Algo 1: $T(n) = 4n^5 + 5$

Algo 2: $T(n) = 105n^4 + 24n^3 + 55n + 233$

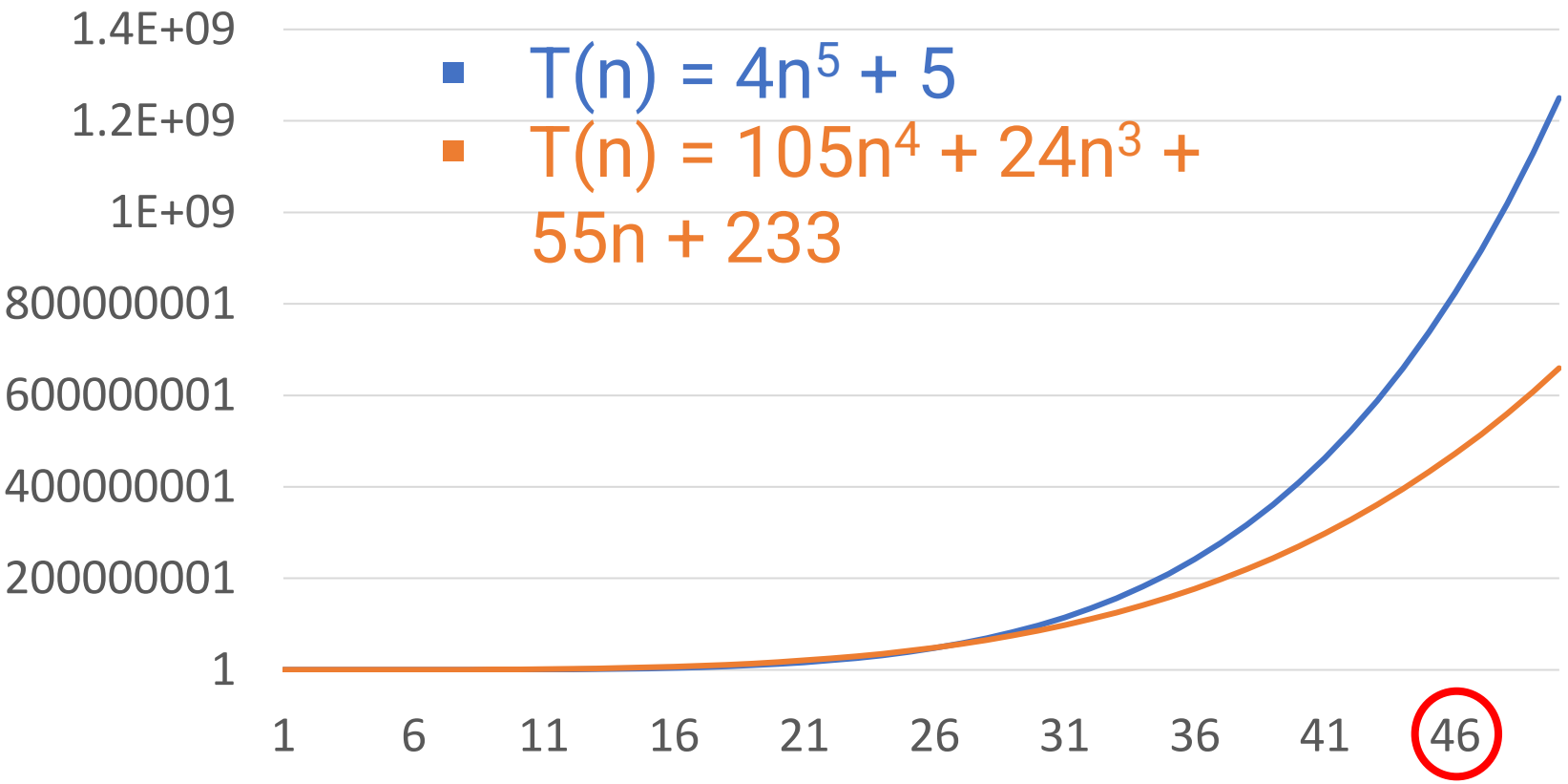
where n = number of iterations

Which one would you rather use?

Growth Rate


18000000
16000000
14000000
12000000
10000000
8000000
6000000
4000000
2000000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
T(n) = 4n^5 + 5
T(n) = 105n^4 + 24n^3 + 55n + 233

Growth Rate


1.4E+09
1.2E+09
1E+09
800000001
600000001
400000001
200000001
1
1
6
11
16
21
26
31
36
41
46
T(n) = 4n^5 + 5
T(n) = 105n^4 + 24n^3 + 55n + 233

Order of Growth

Just look at the biggest term or the term with the highest order:

$$T(n) = 105n^4 + 24n^3 + 55n + 233$$

If you work with bigger values, all other smaller terms affect the final result less.

$$T(n) = 4n^5 + 5$$

Big O Notation

The Big O notation is also known as *asymptotic* notation, and is taken as the input to a function that increases without bound. The output is a function of O, where the input to O is a descriptor to the function's rate of growth at **worst case**.

Big O Notation

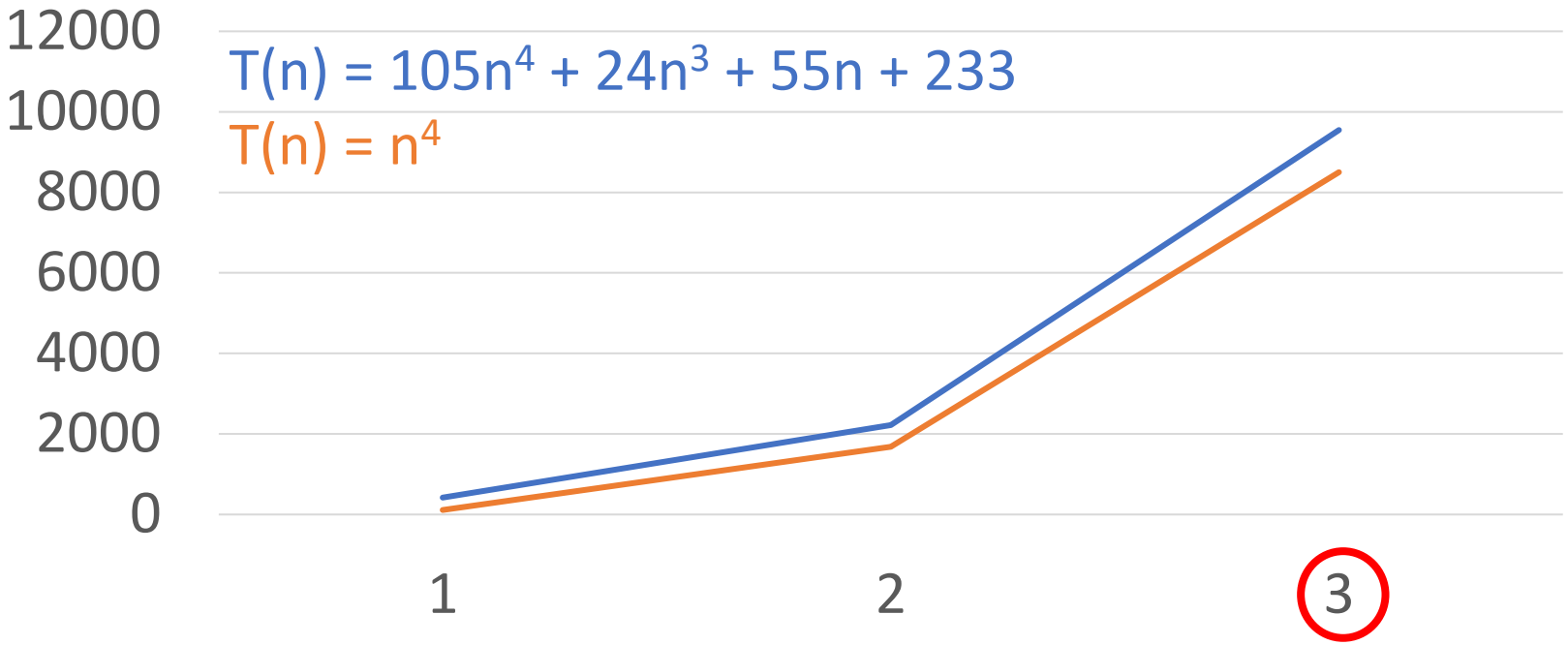
The different possible rates of growth:

- O(1), constant growth
- O(log n), logarithmic growth
- O(n log n), linear logarithmic growth
- O(n^c), polynomial growth
  - O(n), linear growth
  - O(n^2), quadratic growth
- O(2^n), exponential growth

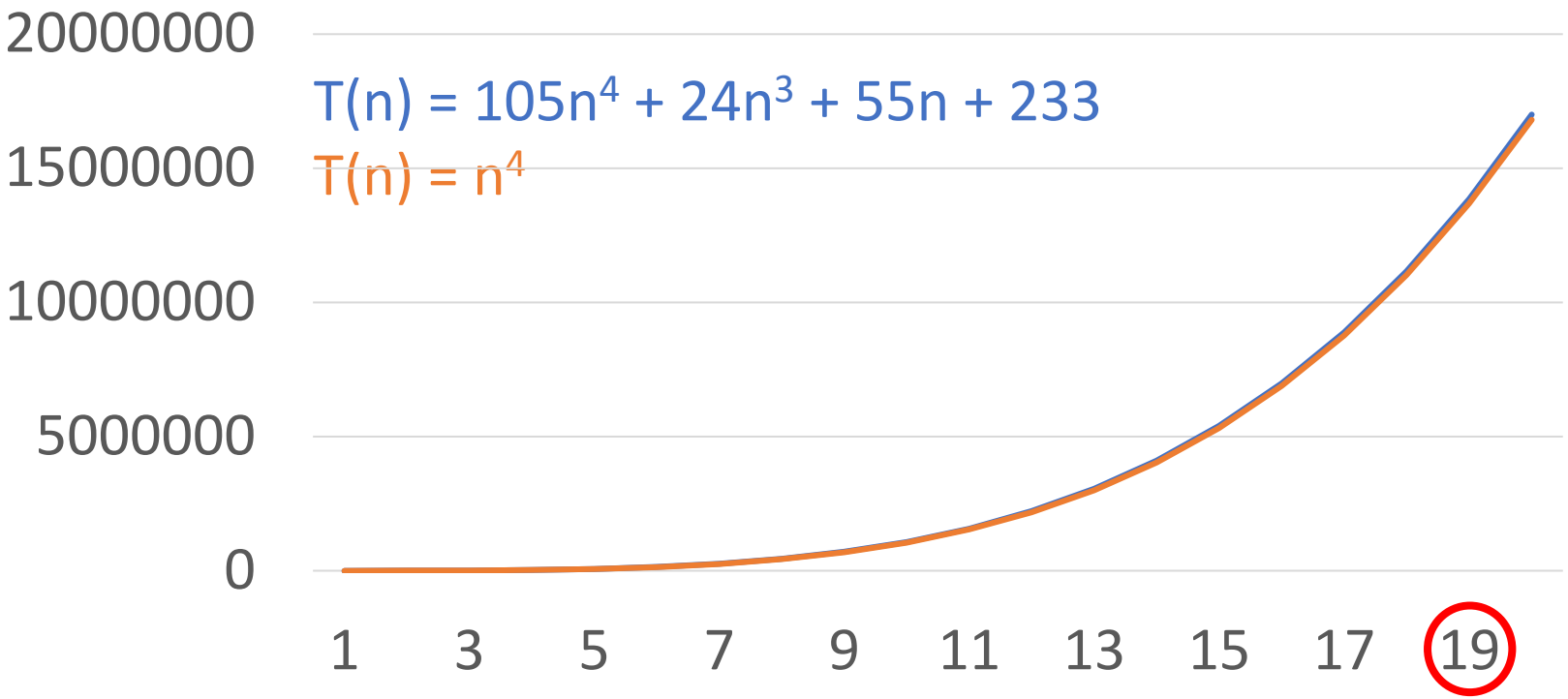
Big O Notation

- The order of growth is a function of the dominant term
- The **dominant term** (term with the fastest growth rate) in the function determines the behaviour of the algorithm
- The dominant term is the term that contributes the most significant increase in f(n) as n increases
- The **coefficient** of the dominant term is **ignored**

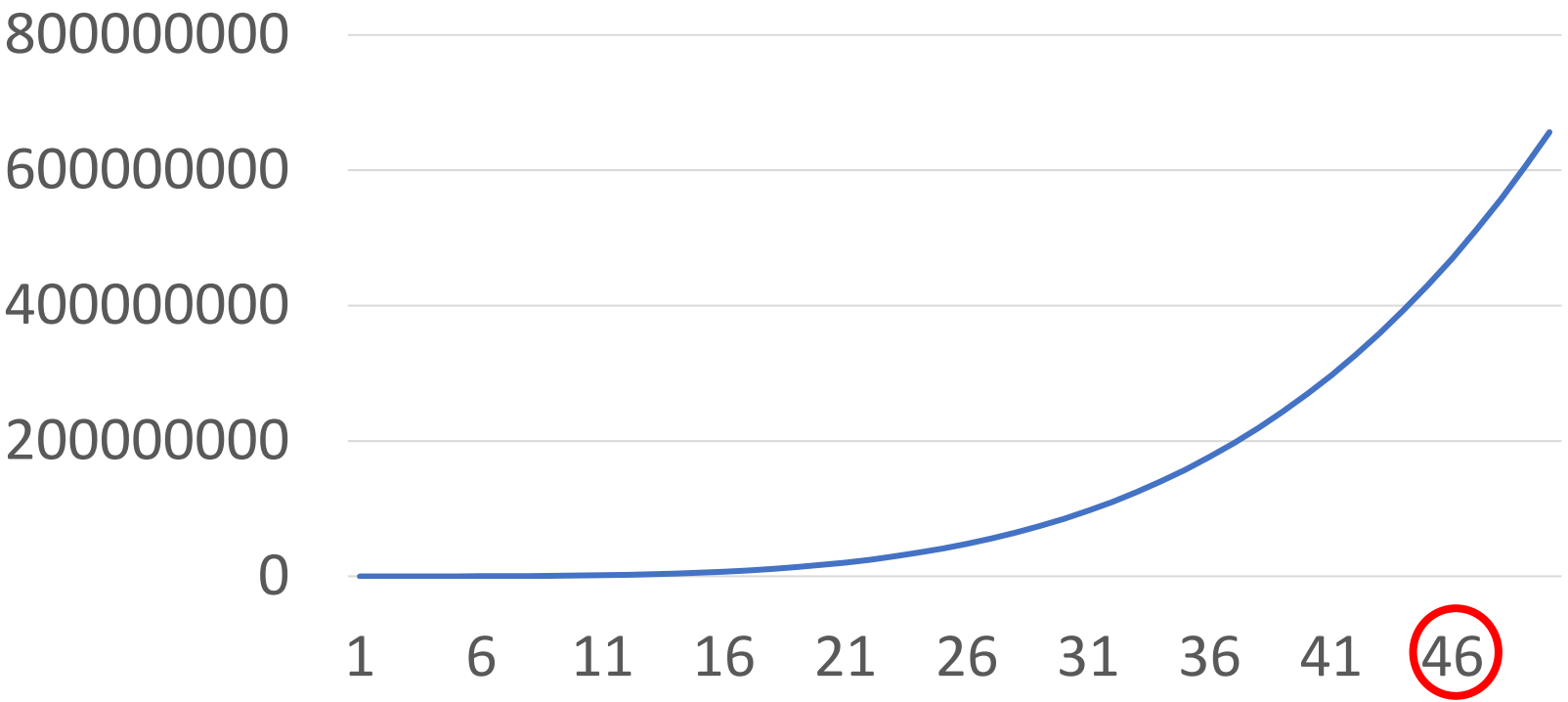
Why dominant term only?



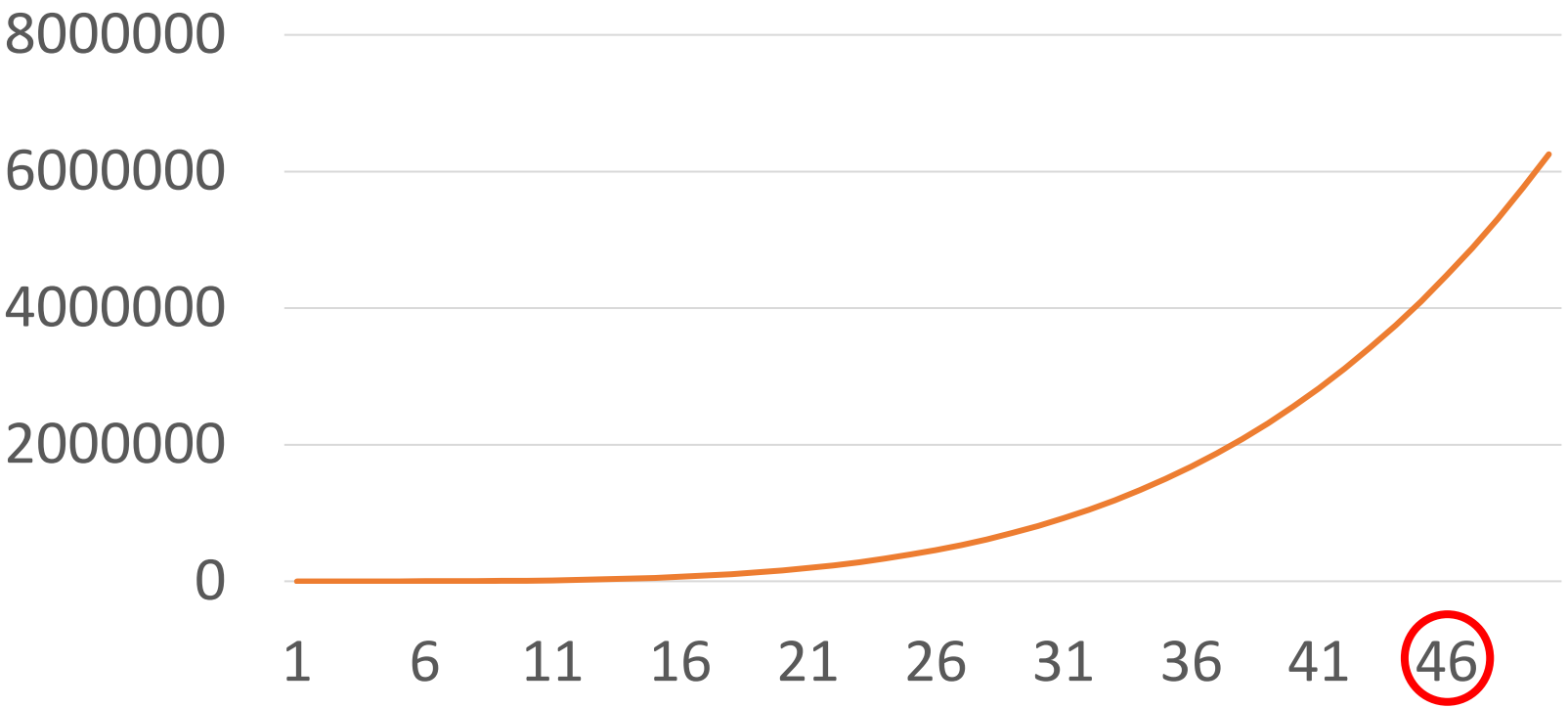
Why dominant term only?



Why ignore the coefficient?



Why ignore the coefficient?

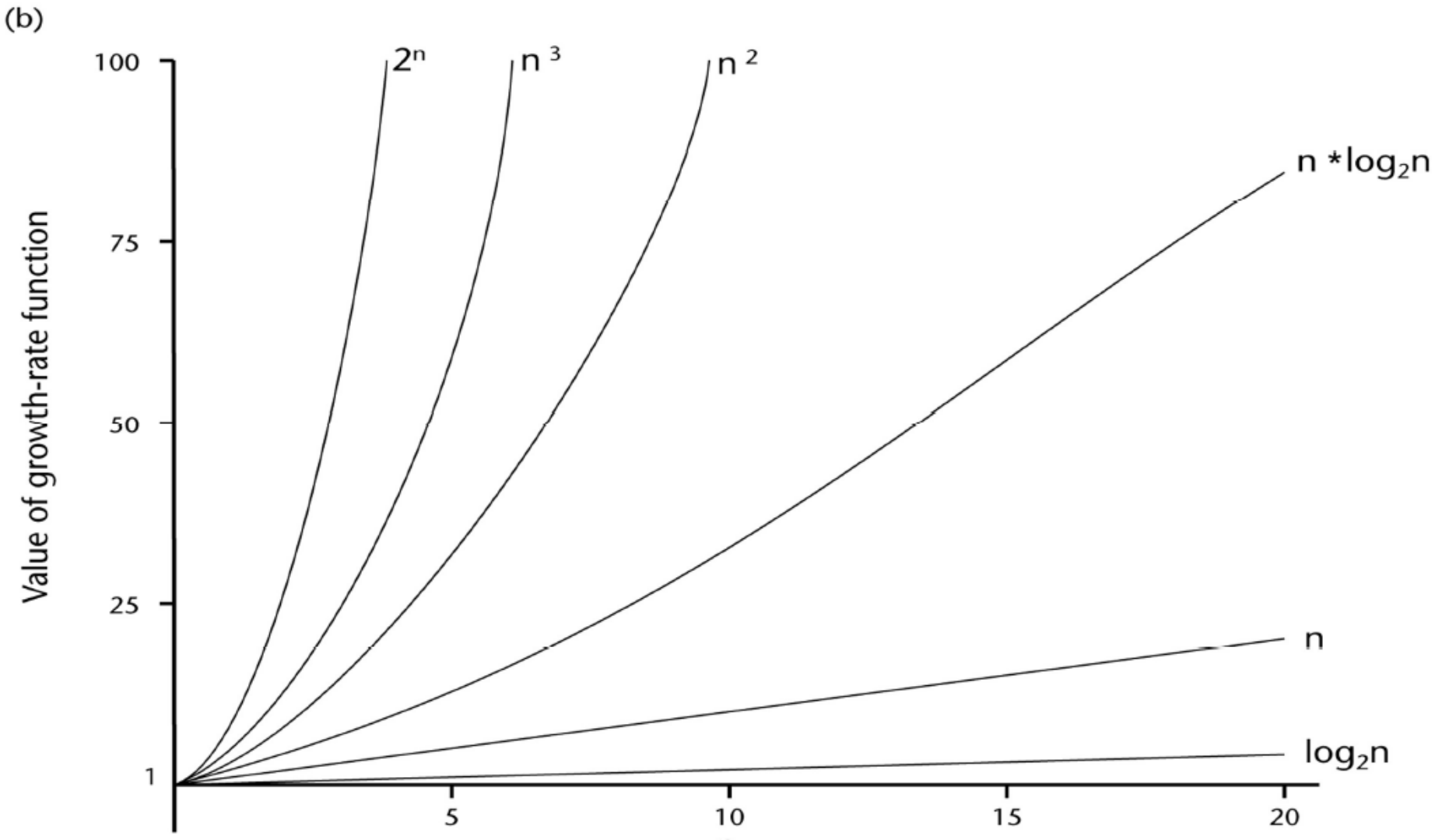


Big O Notation

What if our frequency count is like this:

$$5 + 100n^2 \log n + 40n^5 + 2^n$$

Which is the dominant term?


(b)
Value of growth-rate function
100
75
50
25
1
5
10
15
20
2^n
n^3
n^2
$n * \log_2 n$
n
$\log_2 n$

Big O Notation

- Any **exponential** function of n dominates any **polynomial** function of n
- A **polynomial degree k** dominates a **polynomial of degree m** iff **k > m**
- Any **polynomial** function of n dominates any **logarithmic** function of n
- Any **logarithmic** function of n dominates a **constant** term

Questions? ☺