Middlebury

CSCI 200: Math Foundations of Computing

Spring 2026

Lecture 3W: Complexity

Goals for today:

- Characterize how functions grow as the inputs get really big.
- Use big-O notation to characterize the running time of algorithms.
- Simplify summations into closed-form expressions.
- Calculate the sum of arithmetic and geometric series.

Try to do a detailed analysis of the number of operations performed by this algorithm? (count assignment, arithmetic & comparison operators).

Assume the *worst-case*: x is not in the list a.

```
1 def find(a, x):
2   n = len(a)
3   i = 0
4   while i < n:
5     if a[i] == x:
6       return i
7     i = i + 1
8   return -1
```

	op	count
(2)	=	1
(3)	=	1
(4)	<	n+1
(5)	==	n
(7)	=	n
(7)	+	n

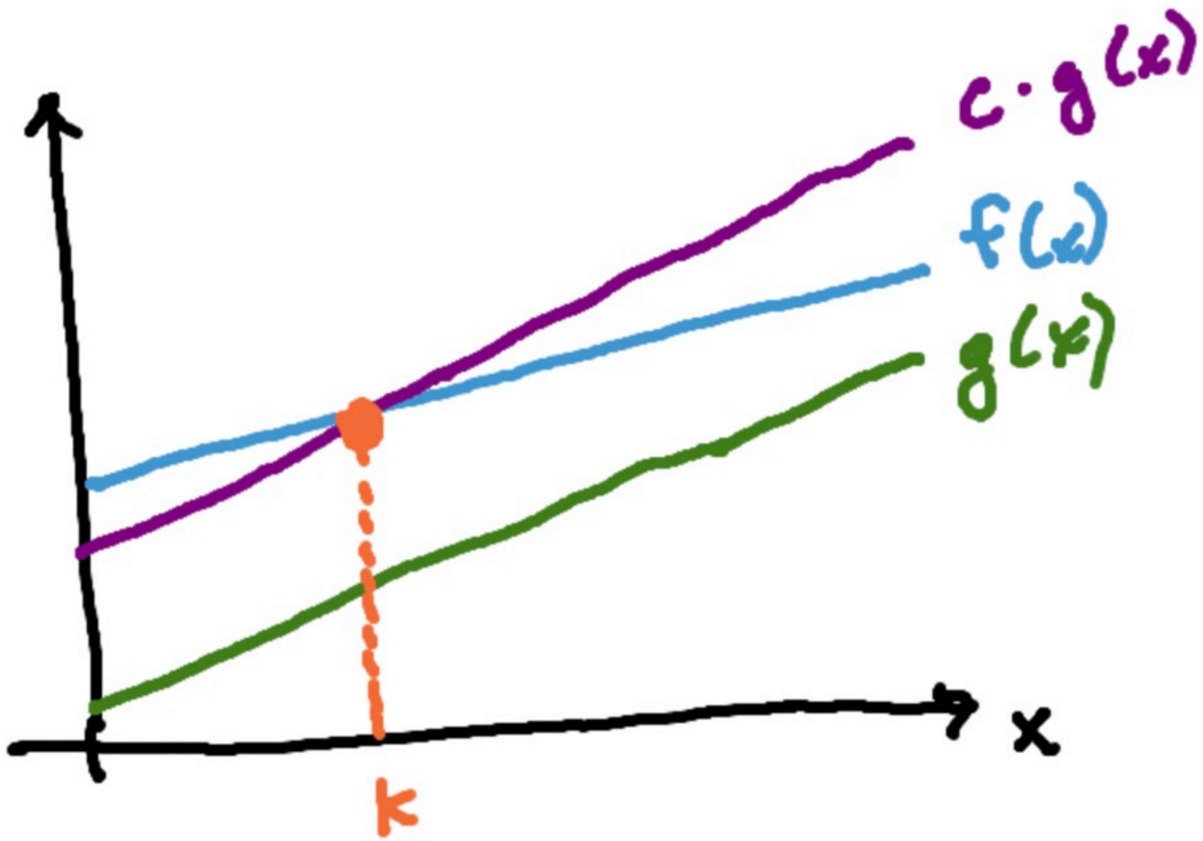
add $4n+3$ operations

We need a better way to analyze running time of algorithms.

big-O notation: Given $f,\ g\colon \mathbb{R} \to \mathbb{R}$, we say that $f(x)$ is $O(g(x))$ if-and-only-if there *exist* constants $c > 0$ and k such that

$$|f(x)| \leq c \cdot |g(x)|, \quad \forall x \geq k$$

"f(x) is eventually no larger than some constant multiple of g(x)"



① need to find c, k to satisfy this def.

② show $\lim_{x \to \infty} \frac{f(x)}{g(x)} < \infty$

Example: Show that $\overbrace{x^2 + 4x + 4}^{f}$ is $O(\overset{g}{x^2})$.

show $x^2 + 4x + 4 \le c \cdot x^2$

$0 \le (c-1)x^2 - 4x - 4$

$(c-1)x^2 - 4x - 4 \ge 0$ pick $c = 2$

$x^2 - 4x - 4 \ge 0$ for all $x \ge K$

$\boxed{\begin{array}{l} c = 2 \\ k = 5 \end{array}}$

try $k=1$ $-7 \ge 0$ X

$k=2$ $-8 \ge 0$ X

$k=3$ $-7 \ge 0$ X

$k=4$ $-4 \ge 0$ X

$k=5$ $1 \ge 0$ ✓

$\lim_{x \to \infty} \frac{f(x)}{g(x)} < \infty$ ✓

$\lim_{x \to \infty} \frac{x^2 + \cancel{4x} + \cancel{4}}{x^2} = 1 < \infty$

Which of the following statements is true? Discuss and then vote.

$a^b = e^{b\ln a}$

1. $1000000000x$ is $O(x^2)$. ✓
2. ✓ x^{10} is $O(e^x)$. $\rightarrow$ $x^{10} = e^{10\ln x} \le c \cdot e^x$ let $c=1$ $10\ln x \le x$
3. ✗ 4^x is $O(2^x)$.
4. 1000 is $O(1)$. ✓

$x = 10$: $10(2.3\ldots) \not\le 10$

$x = 20$: $10(2.99) \not\le 20$

$x = 40$: $10(3.68\ldots) \le 40$ ✓

$4^x = (2^2)^x = 2^{2x}$

$= 2^x \cancel{2^x} \le c \cdot \cancel{2^x}$

$2^x \le c$

A. 1, 2 and 3.

B. $\boxed{\text{1, 2 and 4.}}$

C. 1, 3 and 4.

D. 1, 2, 3 and 4.

How many times is append called in this algorithm to create groups, in terms of n? Start detailed and then express with big-O.

Assume we want to allow single-person groups, e.g. `("Philip", "Philip")`.

```python
# people is a list of strings, for example:
# ["Ahmed", "Alana", "Donovan", "Nico", "Tenzin", "Alex", "Jack",
#  "James", "Edie", "Aditya", "Simon", "Michael", "Hamdi", "Sarah", "Bosco", "Rick"]
def list_groups(people):
  groups = []
  n = len(people)
  for i in range(n):
    for j in range(i, n):
      groups.append((people[i], people[j]))
  return groups
```

$\sum_{i=a}^{b} f_i$ summation notation

i	# append
0	n
1	$+\ n-1$
2	$+\ n-2$
$\vdots$	$\vdots$
$n-1$	$+\ 1$

$$S = 1 + 2 + 3 + \dots + (n-1) + n = \sum_{i=1}^{n} i$$

$$S = n + (n-1) + \dots + 2 + 1$$

$$2S = n(n+1)$$

$$\boxed{S = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}}$$

arithmetic series

$O(n^2)$

How many times is **append** called in this algorithm to make a truth table, in terms of n? Start detailed and then express with big-O.

```
1  # n is the number of variables
2  def make_truth_table(n):
3    truth_table = []
4    truth_table.append([])
5    for i in range(n):
6      new_rows = []
7      for row in truth_table:
8        new_rows.append(row + [True])
9        new_rows.append(row + [False])
10     truth_table = new_rows
11   return truth_table
```

`make_truth_table(3)`

```
[True, True, True]
[True, True, False]
[True, False, True]
[True, False, False]
[False, True, True]
[False, True, False]
[False, False, True]
[False, False, False]
```

$$1 + 2 + 4 + 8 + \dots + 2^n$$

$$2^0 + 2^1 + 2^2 + 2^3 = \sum_{i=0}^{n} 2^i = S$$

mult. by (-2)

$$1 + \cancel{2} + \cancel{4} + \cancel{8} + \dots + \cancel{2^n} = S$$

$$-\cancel{2} - \cancel{4} - \cancel{8} \quad -\cancel{2^n} - 2^{n+1} = -2S$$

add $\quad (1-2)S = 1 - 2^{n+1} \quad S = \dfrac{1-2^{n+1}}{1-2}$

i	# append (lines 8,9)
0	2
1	4
2	8
⋮	⋮

general geometric series

$$\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r}$$

$O(2^n)$

Common functions used in big-O estimates.