

CS 112

Introduction to Data Structures

WEEK 08

Algorithm Analysis and Big-Oh

Eric Araújo

Calvin University · Fall 2026

This Week

- 1 Counting operations
- 2 From counts to classes
- 3 Big-Oh notation
- 4 Why constants drop out
- 5 The curves that matter **KEY**
- 6 Reading complexity off code
- 7 Best, worst, and amortized

How Long Does This Take?

```
v[i]                // 3 operations, always  
v.push_back(it)     // 5 operations if there's room  
                   // ~3n + 5 if it must grow  
list.push_front(it) // 4 operations, always
```

Counting exact operations is honest but useless for comparing algorithms, the number depends on the compiler, the machine, the mood of the cache. What we actually care about is **how the cost changes as n grows**.

Two Things We Don't Care About

Constant factors

$$3n + 5$$
$$100n + 2000$$

Both grow *proportionally to n*. Double n, double the work.

Lower-order terms

$$n^2 + 50n + 700$$

At $n = 1000$, the n^2 term is 20× everything else combined.

So we throw both away and keep only the term that dominates. What's left is the **order of growth**.

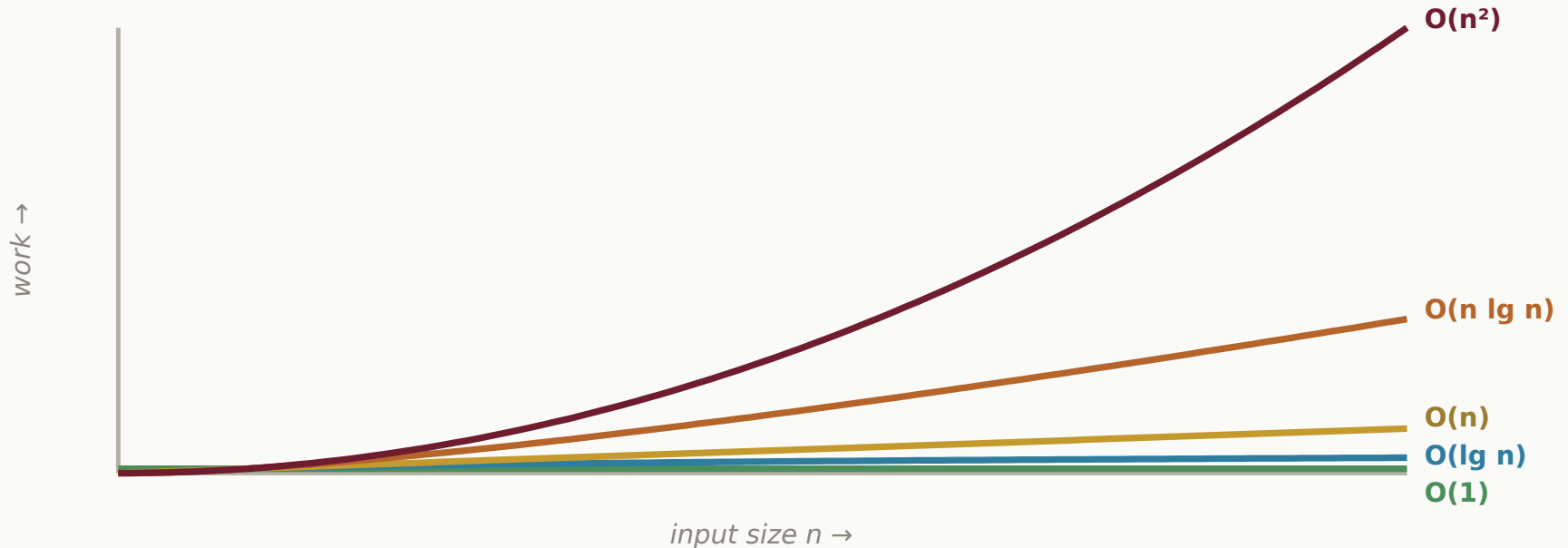
Big-Oh Notation

We write **$O(f(n))$** : "on the order of f of n ".

Exact count	Big-Oh	Name
3	$O(1)$	constant
$\log_2 n + 4$	$O(\lg n)$	logarithmic
$3n + 5$	$O(n)$	linear
$2n \lg n + n$	$O(n \lg n)$	linearithmic
$n^2 + 50n + 700$	$O(n^2)$	quadratic
2^n	$O(2^n)$	exponential

Big-Oh is an *upper bound on the shape of the growth*, not a promise about seconds.

The Curves That Matter



Near the origin the differences look academic. That is exactly why they surprise people in production, the gap only opens up once the data gets big.

TALK TO YOUR NEIGHBOR · TTYN

An algorithm takes $4n + 200$ steps. Its complexity is...

A. $O(4n)$

B. $O(n)$

C. $O(n + 200)$

D. $O(200)$

Reading It Off the Code

```
for (int i = 0; i < n; ++i) {  
    total += a[i];  
}
```

One loop over $n \rightarrow O(n)$

```
for (int i = 0; i < n; ++i) {  
    for (int j = 0; j < n; ++j) {  
        if (a[i] == a[j]) ...  
    }  
}
```

Loop inside a loop $\rightarrow O(n^2)$

Rule of thumb: loops *in sequence* add (keep the biggest), loops *nested* multiply. Halving the problem each pass gives you a $\lg n$.

TALK TO YOUR NEIGHBOR · TTYN

What is the complexity of this loop?

```
for (int i = 1; i < n; i = i * 2) {  
    cout << i << endl;  
}
```

- A. $O(n)$
- B. $O(n^2)$
- C. $O(\lg n)$
- D. $O(n \lg n)$

What We've Built So Far

Operation	Dynamic array	Linked list
index <code>v[i]</code>	$O(1)$	$O(n)$
append	$O(1)$ amortized	$O(1)$
prepend	$O(n)$	$O(1)$
insert / remove at position	$O(n)$	$O(n)$
search (unsorted)	$O(n)$	$O(n)$
traverse everything	$O(n)$	$O(n)$

The same table you filled in by counting, now in the language everyone uses.

Why This Matters: Searching

One million items, and you need to find one.

Approach	Complexity	Comparisons
Linear search	$O(n)$	up to 1,000,000
Binary search (sorted)	$O(\lg n)$	about 20

Fifty thousand times less work, from the same data, just organised differently. That is the entire argument for this course.

Demo



Timing $O(n)$ against $O(n^2)$ on real data.

(same laptop, same data, wildly different patience)

TALK TO YOUR NEIGHBOR · TTYN

Algorithm A is $O(n^2)$, algorithm B is $O(n \lg n)$. Which should you use?

- A.** Always B; it has the better complexity
- B.** Always A, simpler code runs faster
- C.** B for large n ; for small n , A may well be faster
- D.** Whichever is easier to write

Overloading << for Your Container

```
// in List.h
ostream& operator<<(ostream &out, const List &list);

// in List.cpp
ostream& operator<<(ostream &out, const List &list) {
    list.writeTo(out);
    return out;
}
```

Week 3's rule again: a free function, taking the stream on the left, returning the stream so it chains. Traversing the whole list to print it is $O(n)$, as you'd expect.

Week 08 Recap

- Big-Oh describes **how work grows with n** , not seconds on a clock
- Drop constant factors and lower-order terms: keep the dominant one
- Nested loops multiply; halving the problem gives **$\lg n$**
- $O(1) < O(\lg n) < O(n) < O(n \lg n) < O(n^2) < O(2^n)$
- At a million items the difference is a second versus an afternoon

Next: a09 →

This deck stands on earlier CS112 materials by
Joel Adams and **Victor Norman** · adapted and extended by **Eric Araújo**

