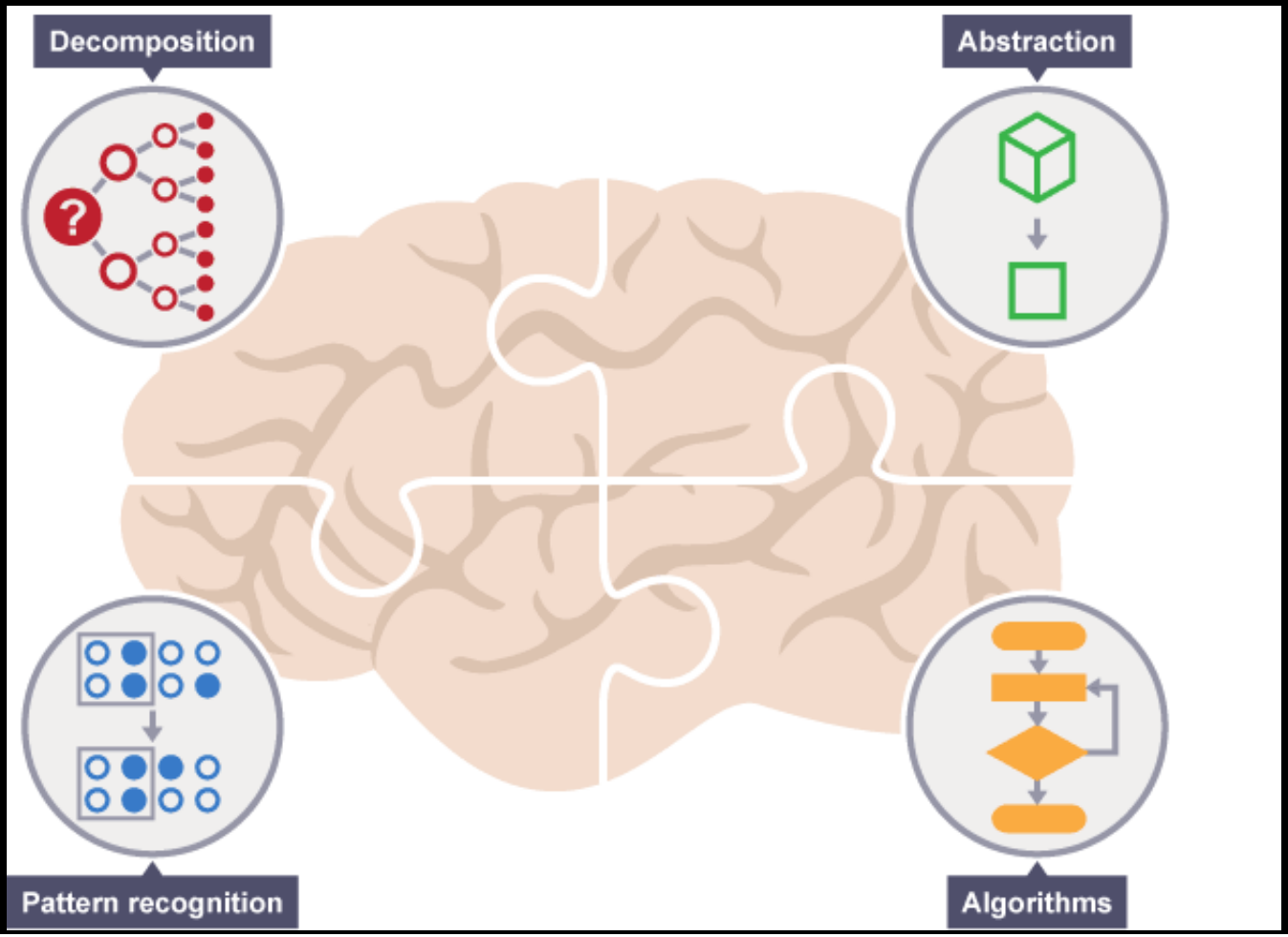
Computer Science: The Good Parts

A Practical Journey for
Early-Career Developers

Part 2

Computational Thinking



Memory

Memory

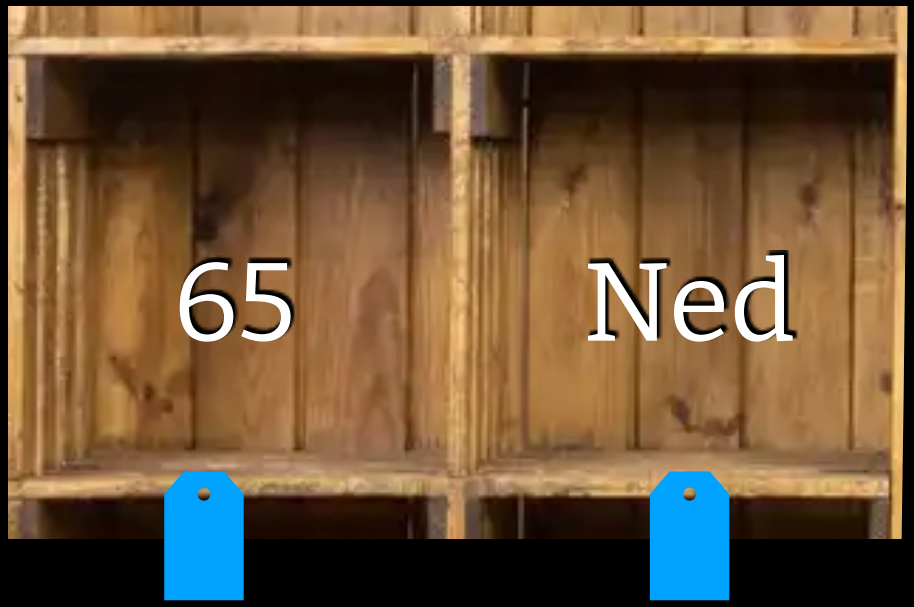
Memory

```
n = 65

name = "Ned"

print(n)

print(name)
```



Memory

```
n = 65

name = "Ned"

print(n)

print(name)
```

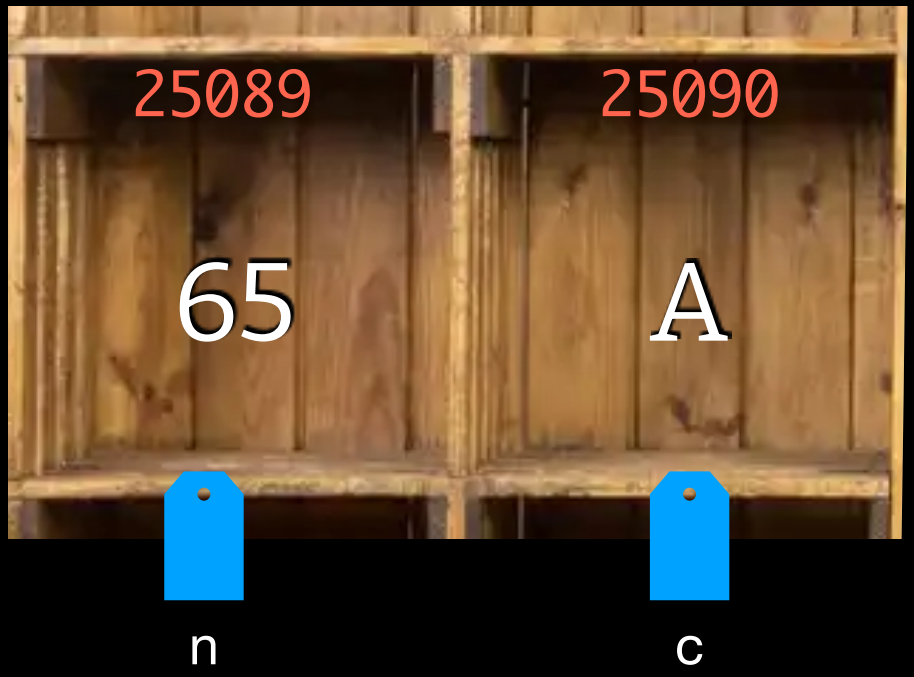
Memory

```
int n = 5;

char c = 'A';

printf(n)

printf(c)
```



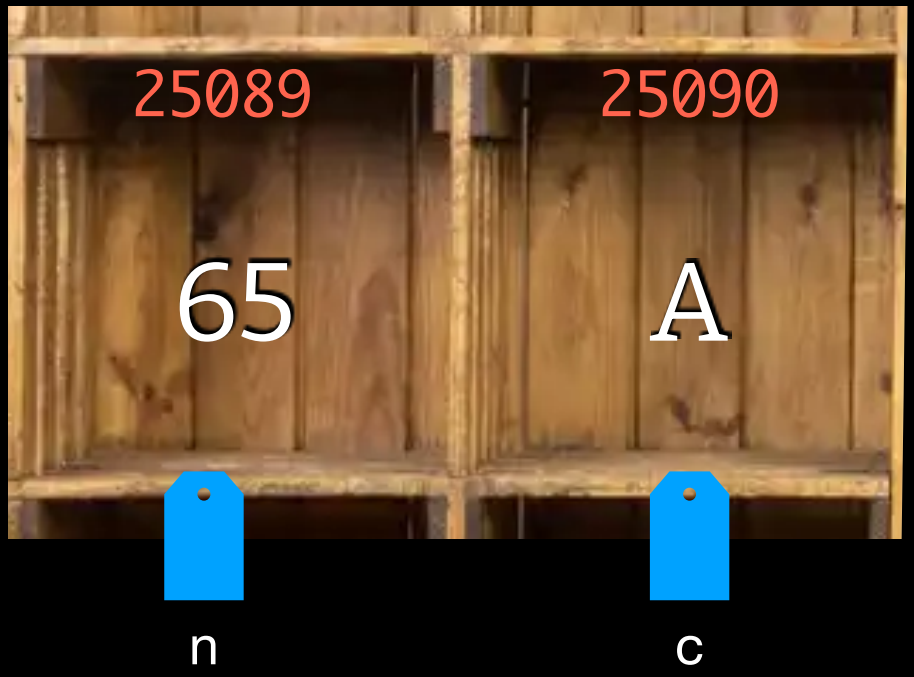
Memory

```
int n = 65;

char* c = 25090;

printf(n)

printf(c)
```



Memory

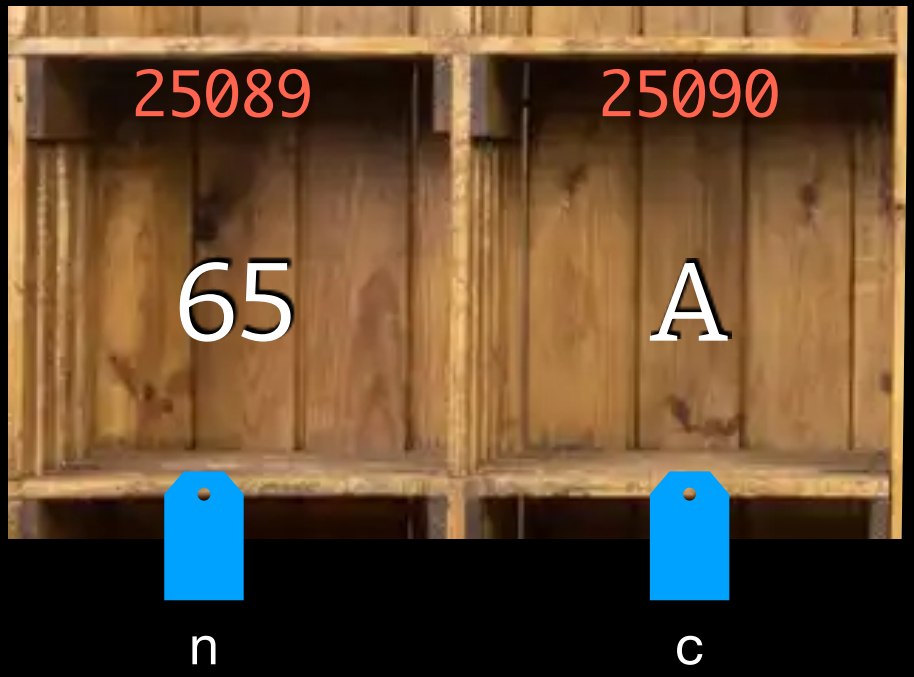
```
int n = 65;

char* c = 25090;

printf(n)

printf(c)

printf(*c)
```



"Copy on Write" (COW)

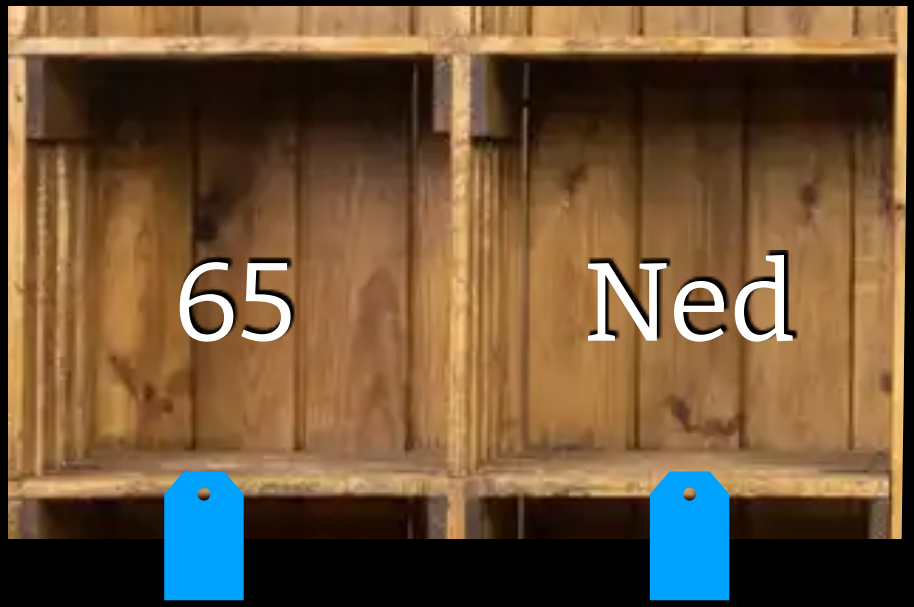
Allocation

```
n = 65

name = "Ned"

print(n)

print(name)
```



Memory

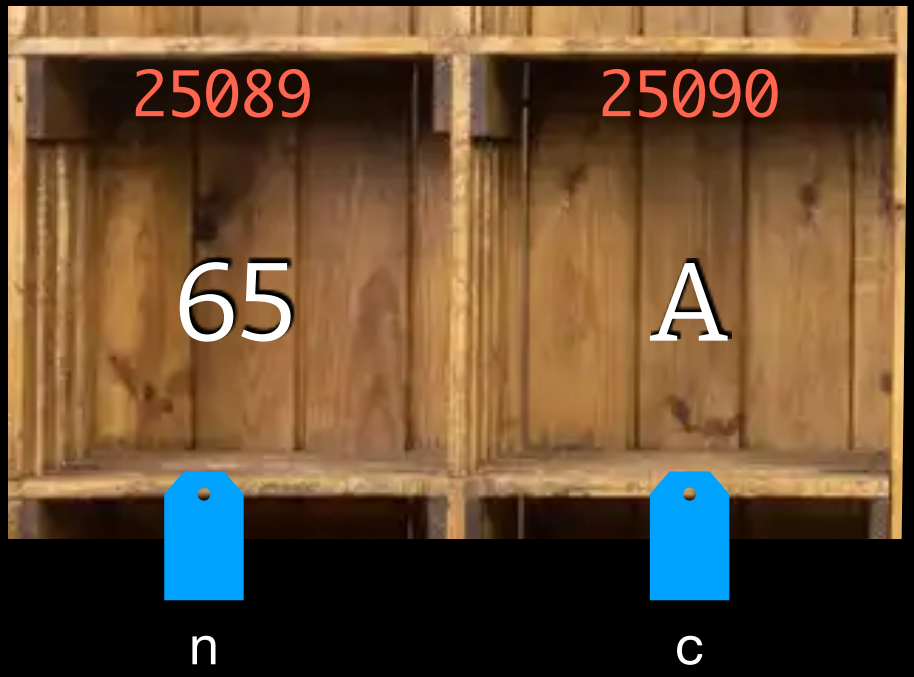
```
int n = 65;

char* c = 25090;

printf(n)

printf(c)

printf(*c)
```



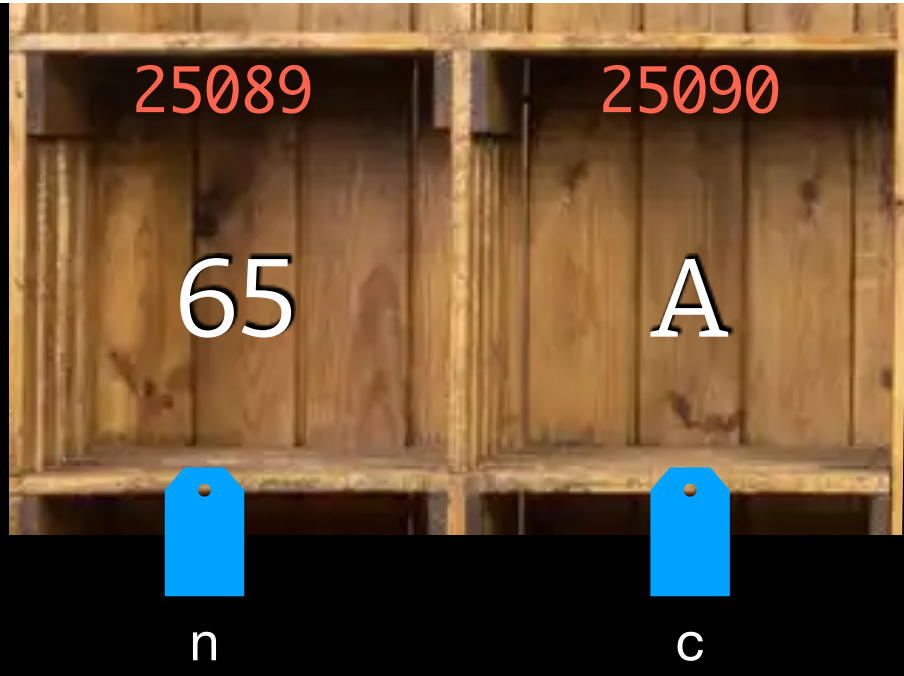
Memory

```
void do_something() {
int n = 65;

char* c = malloc(1*sizeof(char*));

*c = 'A';

printf(n) 65
printf(c)
printf(*c) A
free(c)
}
```



Release

```
def greet(name)
  print(name)
end

people = ["Kermit", "Bert", "Ernie"]

for person in people
  greet(person)
end
```

Kinds of Memory

```
local ("stack")

global ("heap")

virtual ("disk")

device
```

I lied

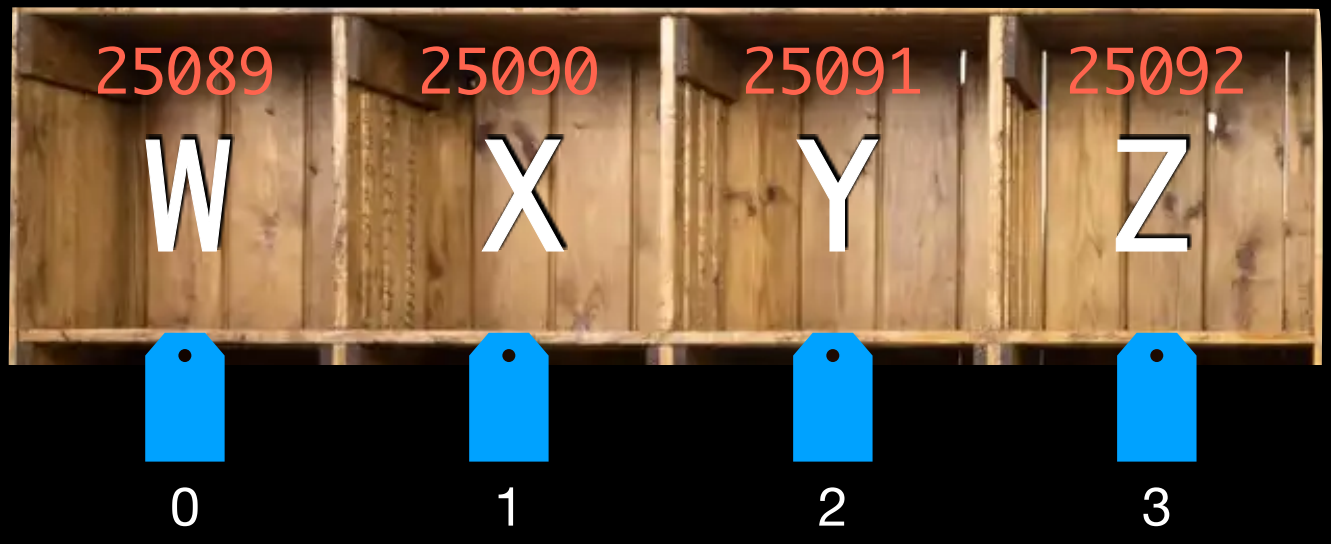
The computer doesn't really work with one memory address at a time.

When a memory address is requested, an entire *page* is retrieved (usually 2k or 4k per page).

Recursion

Data Structures

Array

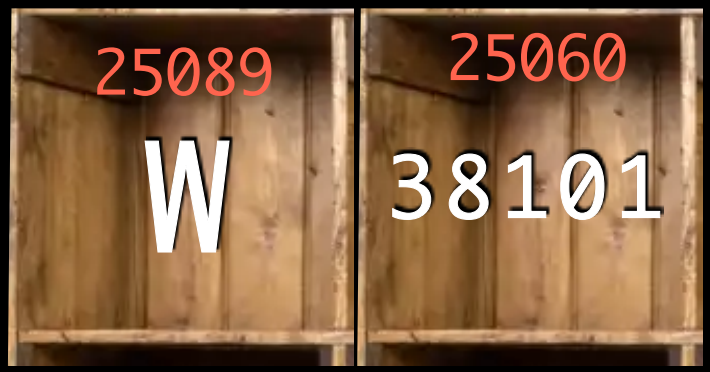


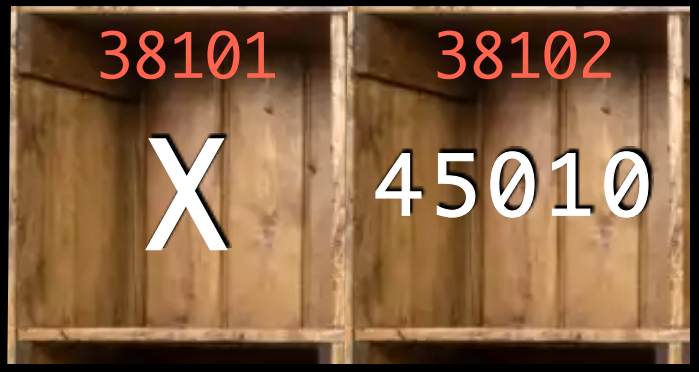
Contiguous memory slots make arrays very fast and easy to access by using offset-based memory address arithmetic.

What if... *the memory slots are not contiguous?*

What if...

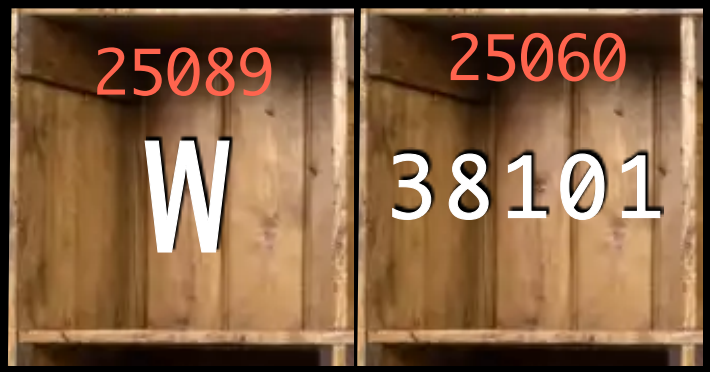
the memory slots are not contiguous?

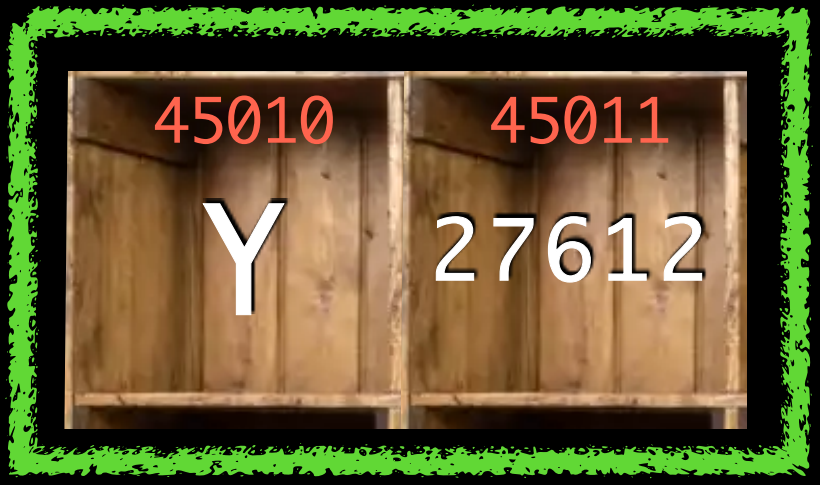


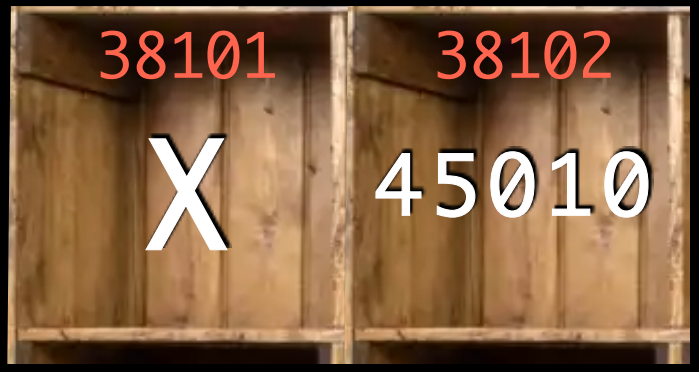


What if...

the memory slots are not contiguous?



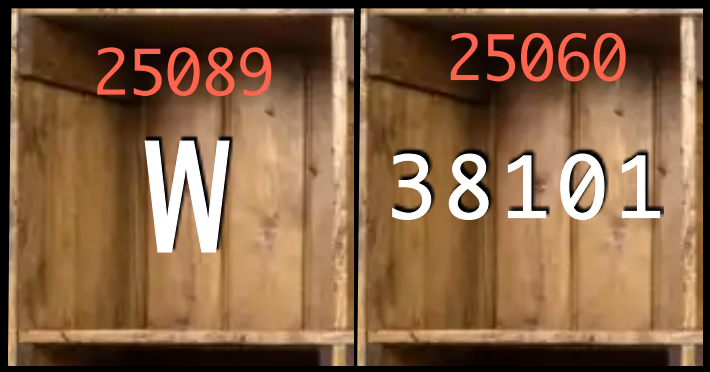


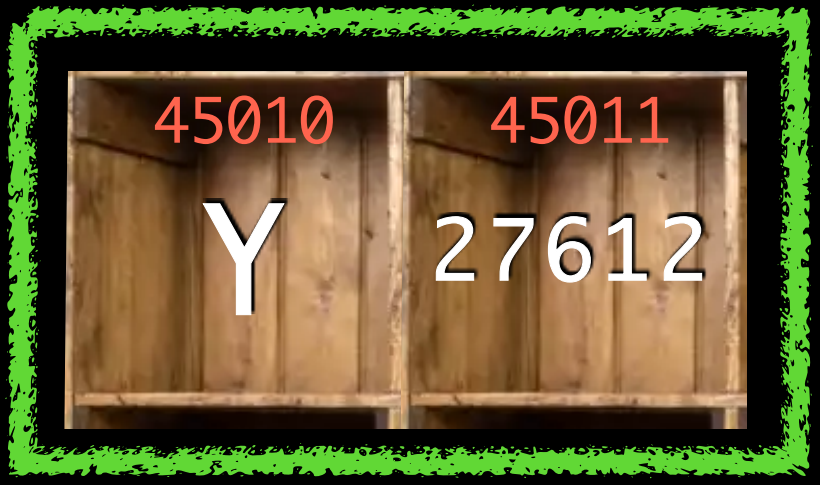


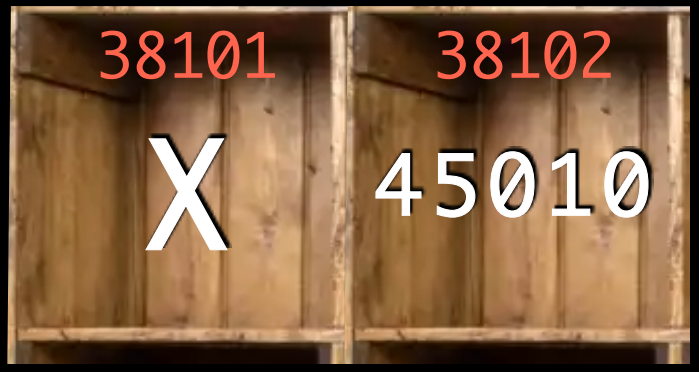
"Pass by value"

"Pass by reference"

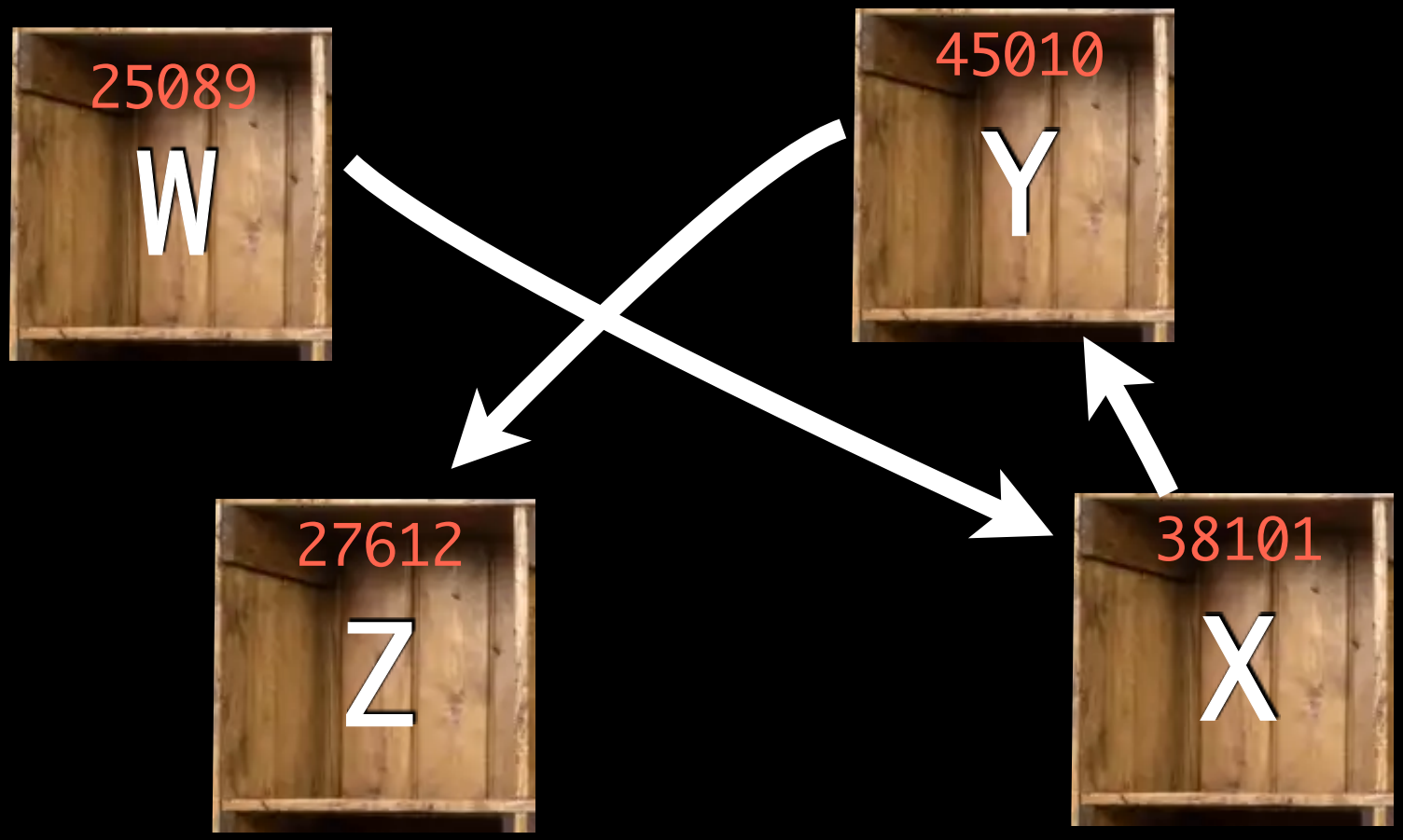
Linked Lists







Linked Lists

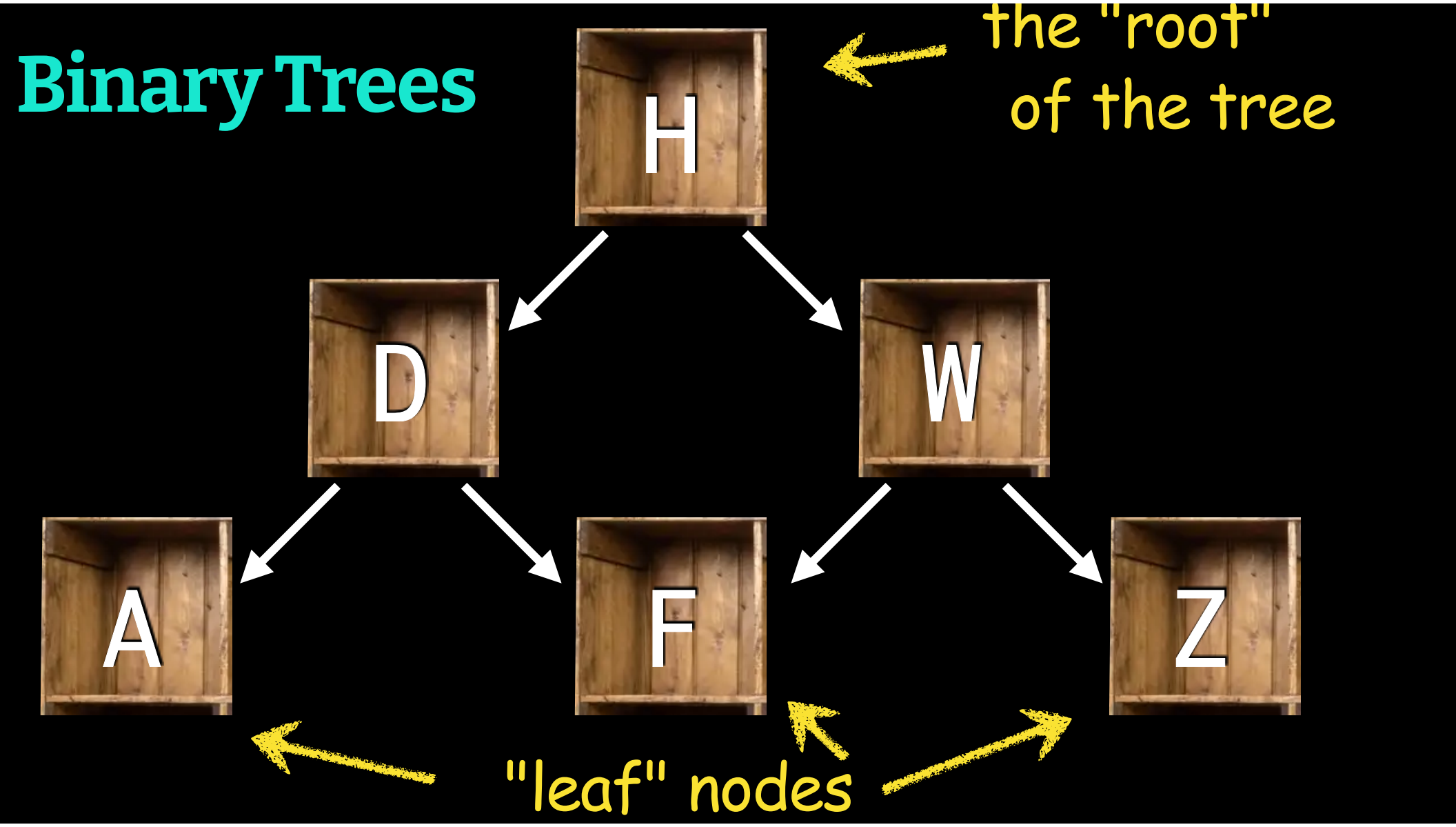


Linked Lists

Linked Lists

1_linked_list

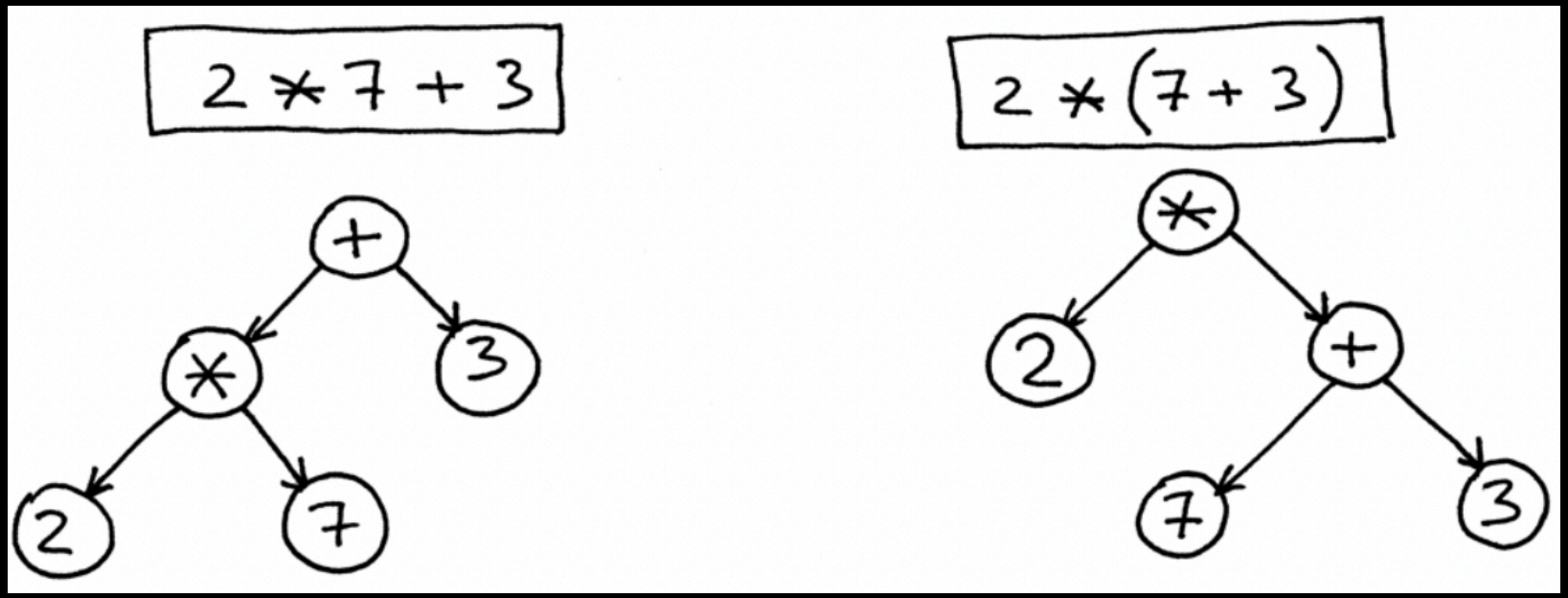
Binary Trees



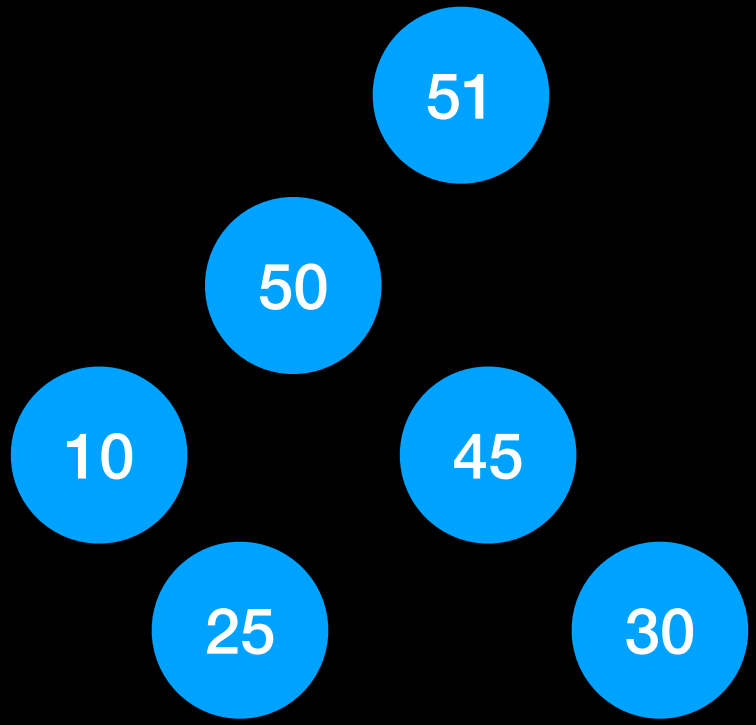
Binary Trees

2_binary_tree

Example: Abstract Syntax Tree (AST)



Example: Max Heap



Complexity Analysis

1. Big "O" Notation
2. Cyclomatic Complexity

Big "O" Notation

Many ways to implement the same algorithm. Is there a notion of "better" or "worse"?

Big "O" Notation

Let's be scientific.

How can we objectively compare one implementation to another?

Big "O" Notation

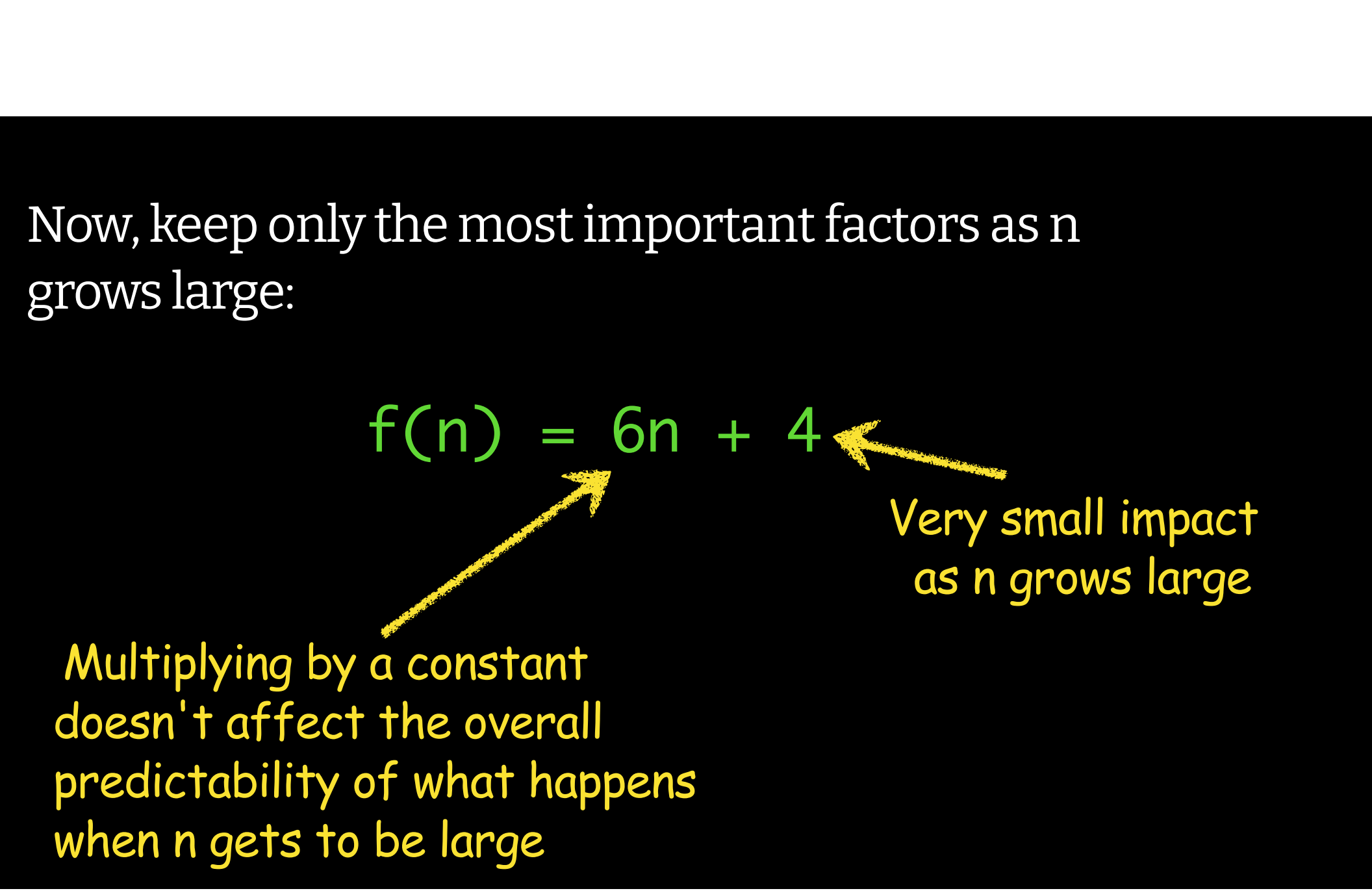
Find the maximum element in an array.

Let's count the number of "atomic" instructions in the worst-case scenario.

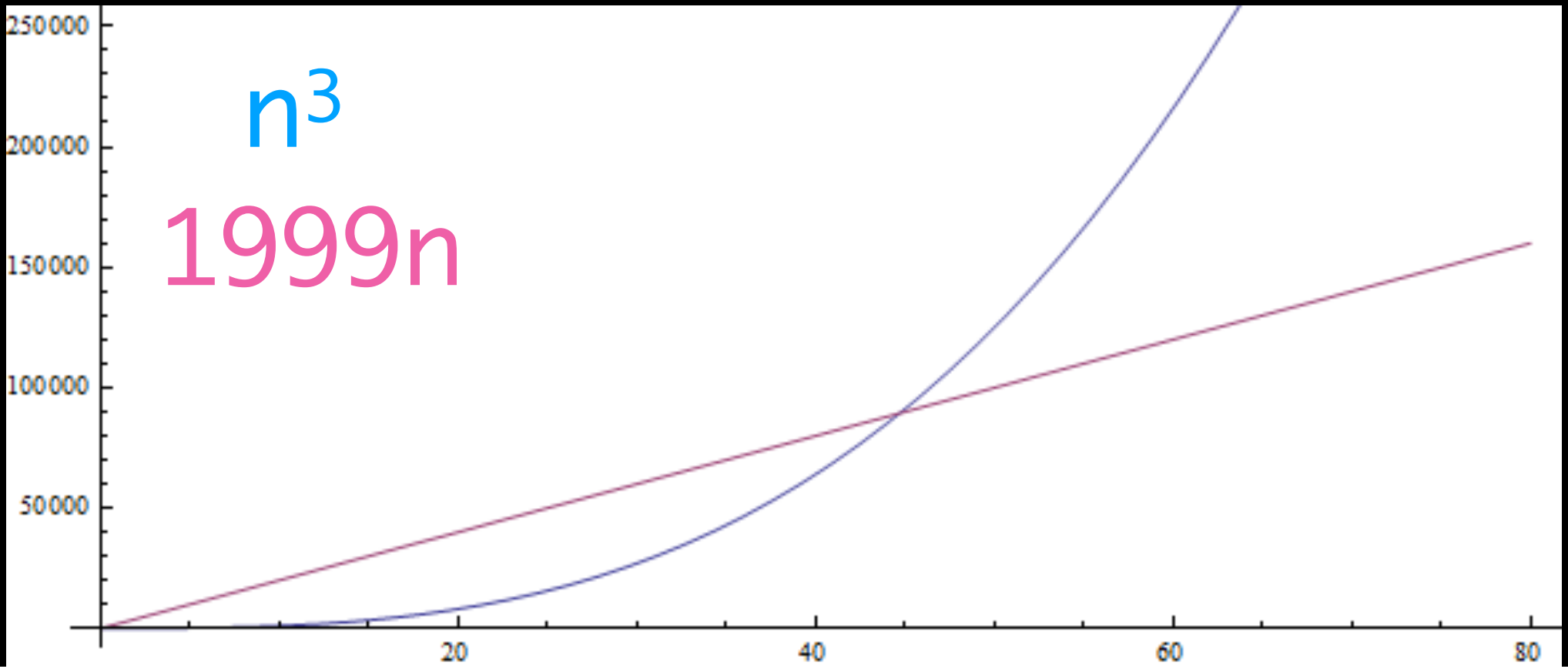
- Assigning a value to a variable
- Looking up an element in an array
- Comparing two values
- Incrementing a value
- Basic math operations

```
1  var M = A[0];
2
3  for (var i = 0; i < n; ++i) {
4    if (A[i] >= M) {
5      M = A[i];
6    }
7  }
```

Now, keep only the most important factors as n grows large:



Keeping only the largest growing term is known as finding the *asymptotic behavior*, or *O(n)*:



Examples:

f(n) = 5n + 12

f(n) = 109

f(n) = n2 + 3n + 112

f(n) = n3 + 1999n + 1337

f(n) = n +√n

Shortcuts:

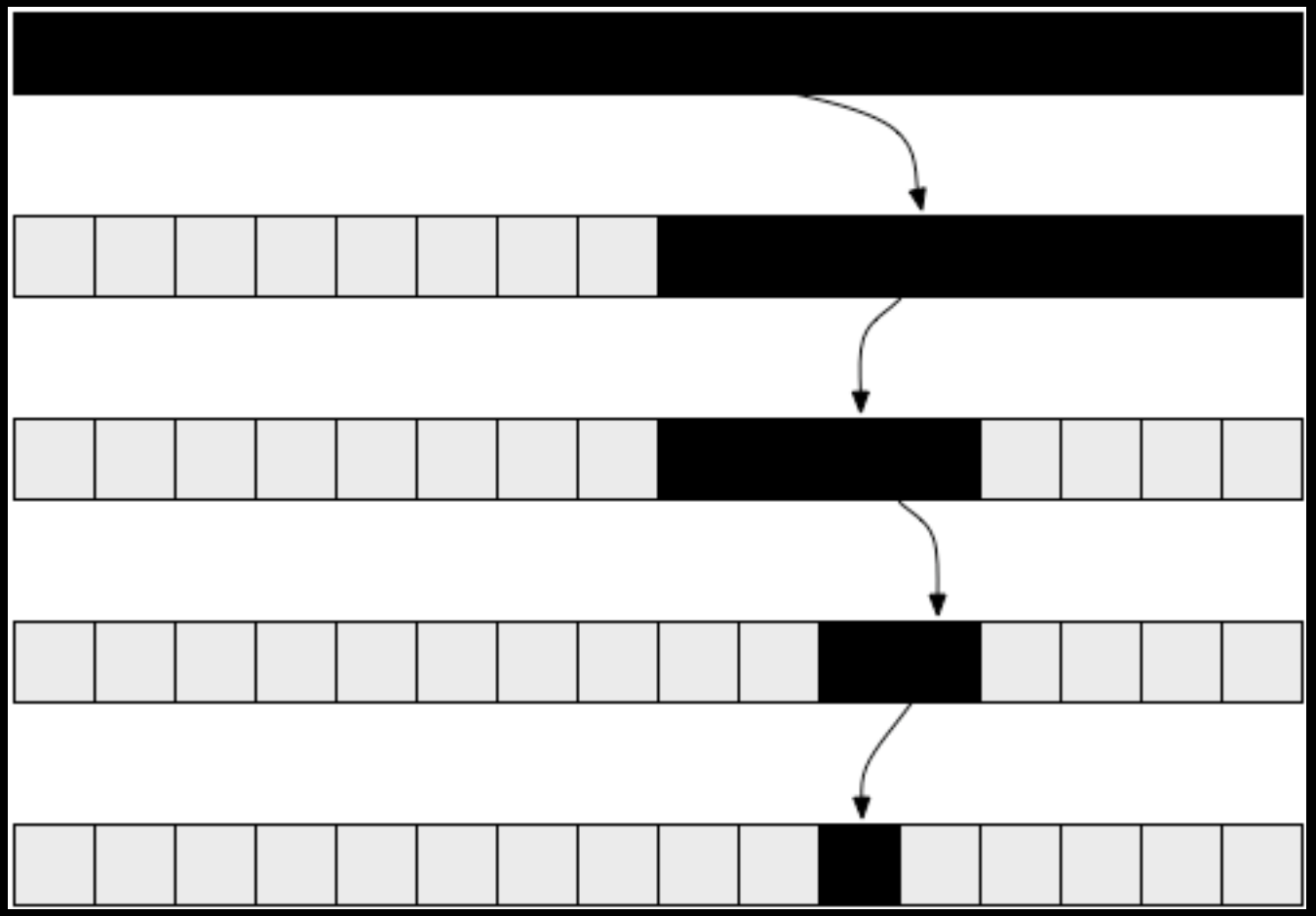
No loops: O(n) = 1

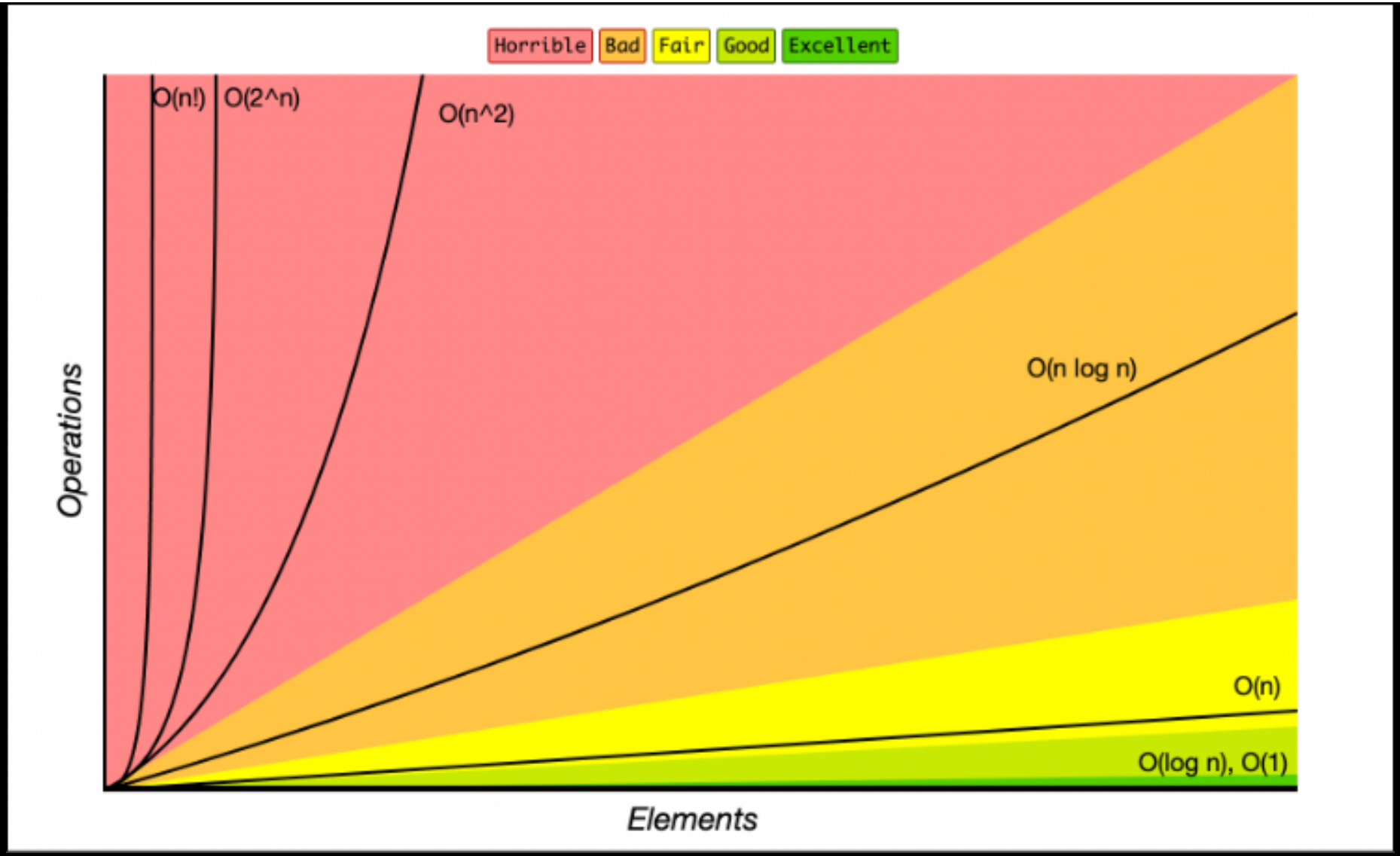
1 loop: O(n) = n

A loop within another loop: O(n) = n^2

Divide-by-half strategy: O(n) = log(n)

Binary Search: O(n) = log(n)




Horrible
Bad
Fair
Good
Excellent
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements

https://www.bigocheatsheet.com/