~~// creates a variable x and sets it to 3~~

var ~~x~~ = 3

count

NO

// The purpose of count is ...

// and the reason 3 is initial

Big-O analysis/computational complexity

- quantify speed/memory/etc analytically

```
val x = 3
var y = x + 5
if (x < y) {
    y = y + 2
}
```

Count assignments: 3

additions: 2

Comparisons: 1

Counting everything is typically very complicated, and counter-productive.

Instead, we typically count something <u>dominant</u>

- something that scales with the total amount of work

```
fun findit(list: List<Int>, value: Int): Int {
    for (i in list.indices) {
        if (list[i] == value) {
            return i;
        }
    }
    return -1;
}
```

How much work?

Pick one thing to count that scales w/ size of data

-count key comparisons

↖ "search key"

Best case: finds it as first item

\# of Comparisons = $\underline{1}$

Worst case: looks at all items

\# of comparisons = $\underline{n}$ ← # of items in list

What about average case?
In general, hard to figure out
it depends on likelihood of
individual cases to happen

```
fun checkForDups(list: List<Int>): Boolean {
    for (i in list.indices) {
        for (j in list.indices) {
            if (list[i] == list[j] && i != j) {
                return true;
            }
        }
    }
    return false;
}

println(checkForDups(listOf(1,2,3)))
```

n times

n times every time the outer loop runs

Count Key comparisons:

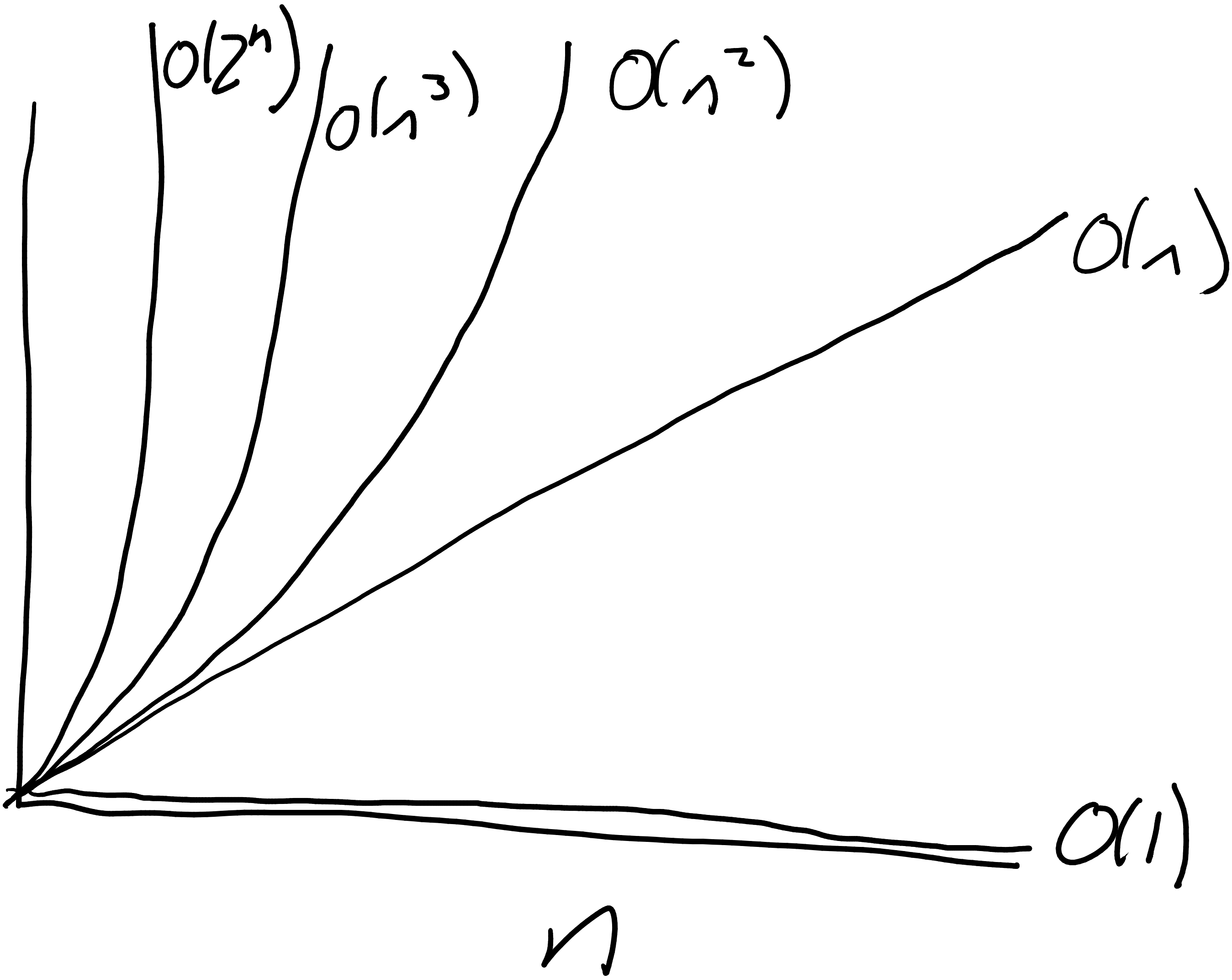
$= n * n = n^2$

Two algorithms:

- first was $O(n)$ work
- second was $O(n^2)$ work

$O(\cdot)$ = "order of"

~~A~~ A $O(n^2)$ alg is worse than $O(n)$ alg



$O(1)$ constant

$O(n)$ linear

$O(n^2)$ quadratic 1, 4, 9, 16, 25, 36

$O(n^3)$ cubic

$O(2^n)$ exponential 1, 2, 4, 8, 16, 32

Big O lets you get rid of irrelevant smaller pieces

$n^2 + 2$ is $O(n^2)$

small/insignification

$3n$ is $O(n)$

$n^2 + 2n + 1$ is $O(n^2)$

5 is $O(1)$

Binary search - cut list in half as you look for something

are you 9? X
are < 9?

~~1, 2, 4,~~ 5, 7, 9, 11

is 9 in it?

Binary search cuts in half the # of items at every step. So the amount of work is approx the number of times I can cut the list in half until I have one item.

$n^2 + 2n$
$\in O(n^2 + n)$?

8 items
↳ 4
↳ 2
↳ 1

3 steps

$2^3 = 8$

16 items
↳ 8
↳ 4
↳ 2
↳ 1

4 steps

$2^4 = 16$

n items
↳ $n/2$
↳ $n/4$
⋮
↳ 1

how many steps?

$\log_2 n$

how many times can you divide by 2, until you get to 1?

Binary search: $O(\log n)$

x

$O(1)$

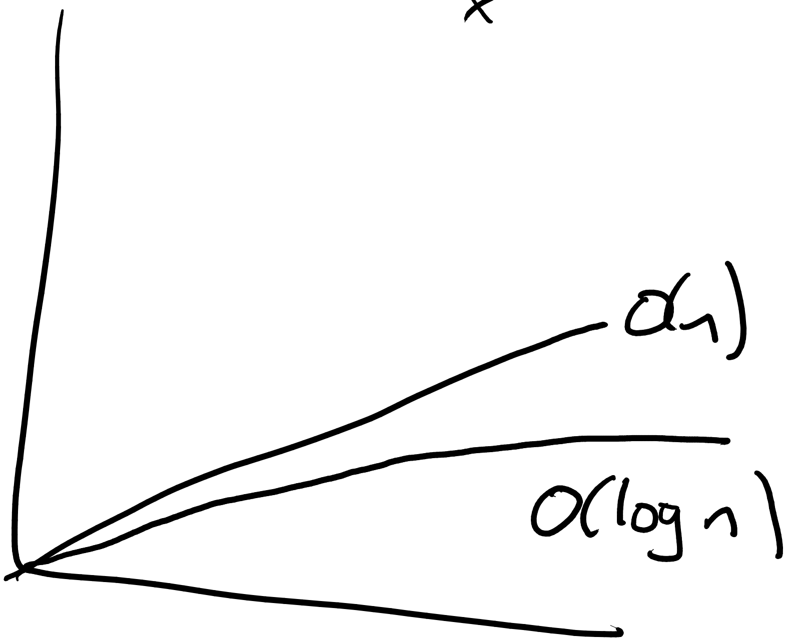
$O(n)$

$O(\log n)$

$O(n \log n)$

$O(n^2)$

$\vdots$



Mathematical defn of O

What does it mean to say that

$$5n^2 + \frac{n}{3} \text{ is } O(n^2) \text{ ?}$$

Try to come up with our own definition

Maybe it's really simple:

maybe it means

$$5n^2 + \frac{n}{3} \le n^2 \quad ?$$

(not true)

A closer definition (not quite right)

There is a constant C so that

$$5n^2 + \frac{n}{3} \le Cn^2$$

(not quite right but close)

We only care about happens as n gets big.

we don't care if this expression is false for small n.

True defn:

$$\underbrace{5n^2+\frac{n}{3}}_{f(n)} \text{ is } O(\underbrace{n^2}_{g(n)})$$

means there is some constant C so that

$$5n^2+\frac{n}{3} \le Cn^2$$

for all n bigger than some threshold N.

Defn:

$f(n)$ is $O(g(n))$

means there is some constant C so that

$$f(n) \leq C g(n)$$

for all $n \geq N$

fixed

Let's show that

$$5n^2 + \frac{n}{3} \text{ is } O(n^2)$$

Can we find a constant C so that

$$\frac{5n^2 + \frac{n}{3}}{n^2} \le \frac{Cn^2}{n^2}$$

Solve for C

$$5 + \frac{n}{3n^2} \le C$$

$$5 + \frac{1}{3n} \le C$$

Cant find a C so that

$$C \geq 5 + \frac{1}{3n}$$ (for all n big enough)

Can I find a C

so that

$$C \geq 5 + \frac{1}{3n}$$

for n big enough

For all $n \geq 1$

$$\frac{1}{3n} \text{ is always } < 1$$

So $C \geq 5 + 1 = 6$

Pick $C = 6$ and then

$C \geq 5 + \frac{1}{3n}$ for

all $n \geq 1$, and so

$5n^2 + \frac{n}{3}$ is $O(n^2)$

Can we show that

$n^2 + 3$ is not $O(1)$?

Can we find a C so that

$$n^2 + 3 \le C \cdot 1$$

for n big enough?

Solve for C

$$C \ge n^2 + 3$$

Can I find a C that is always bigger than n^2+3, no matter how big n is?

No.