LESSON 2:

# Big-O Analysis

# Time Complexity

Big O notation

$O(\ )$ ← parent function

$N$ = number of inputs

$T(N)$ →

$T$ = time it takes for algorithm to run

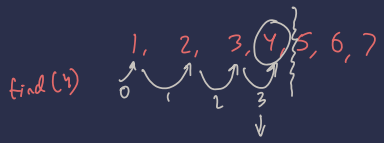
$$T(N) = O(f(N))$$

parent function

$N \quad N^2 \quad 2^N \quad \log(N) \quad \ldots$

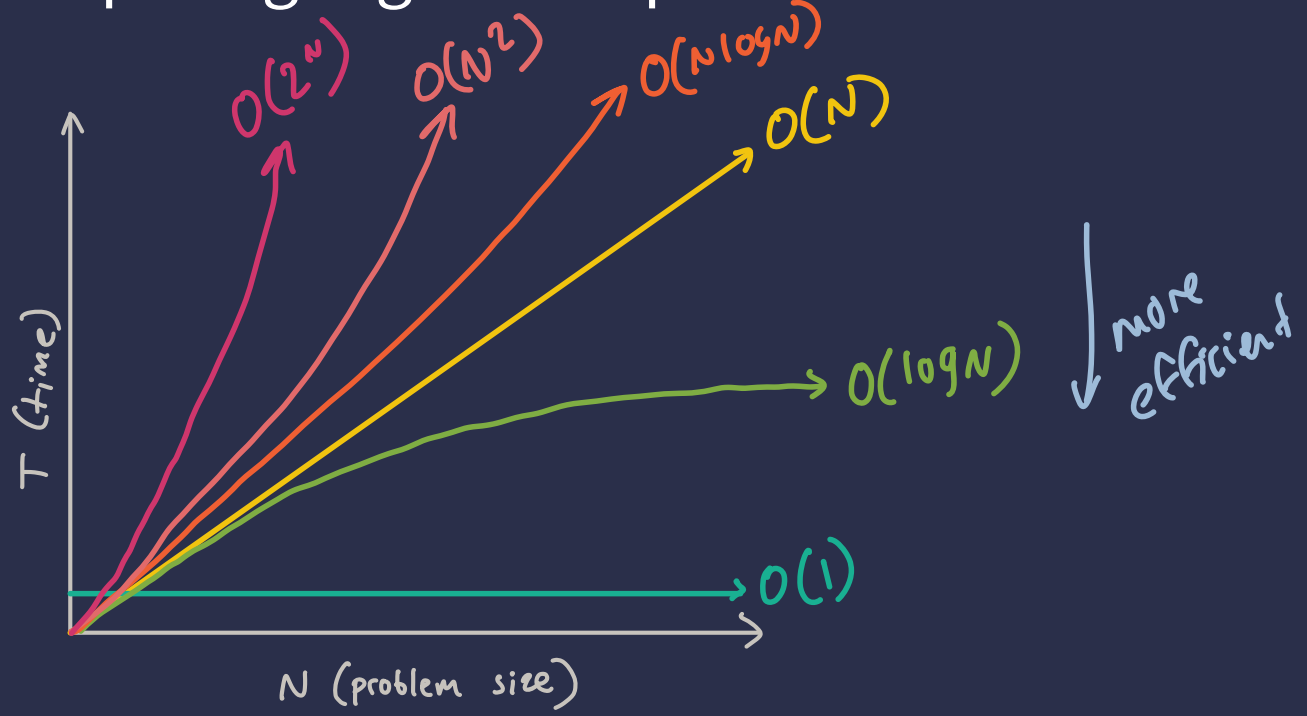
$$T(N) \leq cf(N)$$

upper bound



less efficient

# Comparing Big-O Complexities

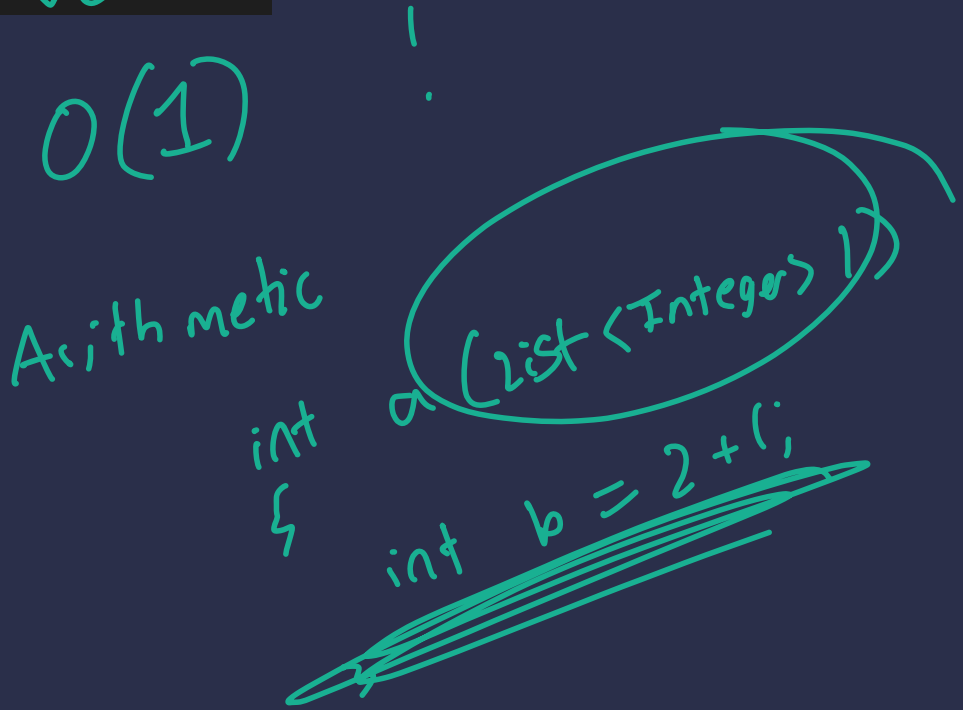


# Analyzing Algorithms

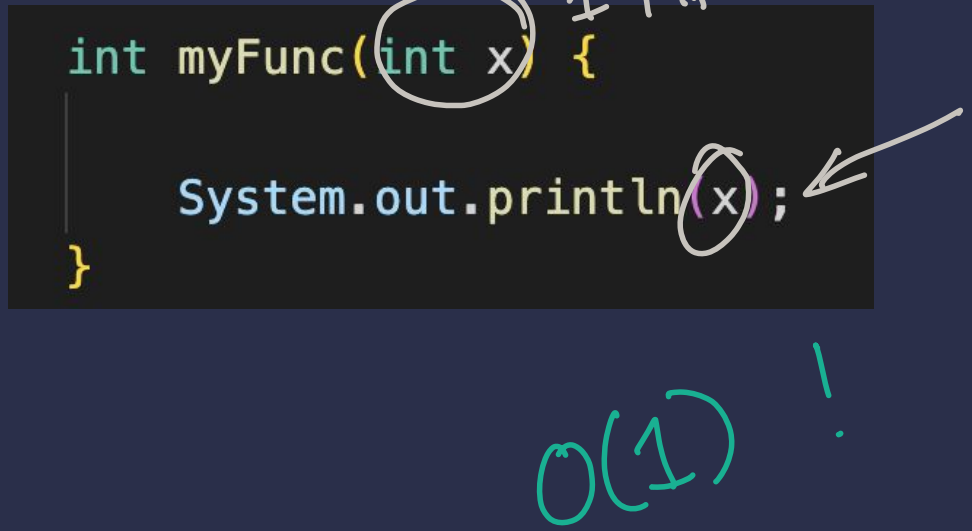
```
int a = 2 + 1;
```

O(1)

Arithmetic



# Analyzing Algorithms



# Analyzing Algorithms

```
if( x < 4) {
    doSomething(); O(log N)
}
else {
    doSomethingElse(); O(N)
}
```

max(O(1), O(log N), O(N)) = O(N)

① Condition
② if - block
③ else block

max(O(condition), O(if - block), O(else - block))

# Analyzing Algorithms

$N * O(N) = O(N^2)$

$0, 1, 2, 3, \ldots, N-1 = N$ times

```
for(int i = 0; i < N; i++) {
    doSomething();  O(1)  O(N)
}
```

$O(1)$ $O(1)$ $O(1)$ ... $N$ times

$O(\text{for loop}) = N * O(1) = O(N*1) = O(N)$

# of iterations * body

# Analyzing Algorithms

$0 \longrightarrow N-1$ $N$ times

```
for(int i = 0; i < N; i++) {
    for(int j = N; j > 0; j--) {

        doSomething();
    }
}
```

$O(\log N)$

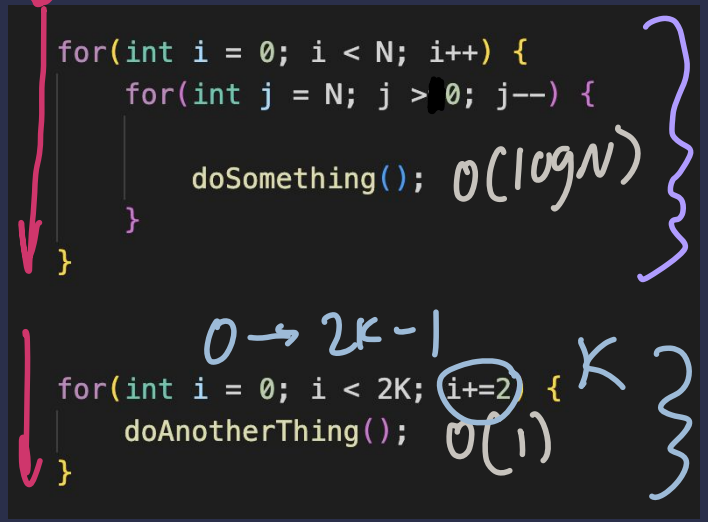
$N \rightarrow 1$ $N$ times

$$N * (N * (O(\log N)))$$

$$= N(O(N \log N)) = O(N^2 \log N)$$

# Analyzing Algorithms

Consecutive +



```
for(int i = 0; i < N; i++) {
    for(int j = N; j > 0; j--) {

        doSomething();
    }
}

for(int i = 0; i < 2K; i+=2) {
    doAnotherThing();
}
```



$O(N^2 \log N) + O(K) = O(N^2 \log N + K)$

$O(N^2 \log N) + O(N)$

$O(N^2 \log N + N)$

$O(N^2 \log N)$

If 2nd loop uses N

# In Review

**Single Statement** O(1) ← * arithmetic, comparison

* running method → O(?) depending on the method

**Block of Statements** O(1st) + O(2nd) + ...

**if-Statements** max(O(condition), O(if), O(else if), O(else))

**Loops** N (# of iters) * O(body)

**Nested Loops** N (# of iters outer loop) * O(inner loop)

*Recursion? : )*