Algorithm Analysis

Comparing Algorithms

Comparing Algorithms

How can we quantify an algorithm?

Comparing Algorithms

How can we quantify an algorithm?

- Memory Usage
- Runtime

Comparing Algorithms

How can we quantify an algorithm?

- Memory Usage
- Runtime

Determining exact runtime/memory usage might be difficult

Comparing Algorithms

How can we quantify an algorithm?

- Memory Usage
- Runtime

Determining exact runtime/memory usage might be difficult

Compare the runtimes to each other in terms of the input size

Comparing Algorithms

How can we quantify an algorithm?

- Memory Usage
- Runtime

Determining exact runtime/memory usage might be difficult

Compare the runtimes to each other in terms of the input size

Main Question:

> How does resource consumption **scale** with input?

Comparing Algorithms (cont.)

Computer Scientists have created a runtime **ordering**

Comparing Algorithms (cont.)

Computer Scientists have created a runtime **ordering**

The main metric you will see is Big-Oh

Comparing Algorithms (cont.)

Computer Scientists have created a runtime **ordering**

The main metric you will see is Big-Oh

- Examples include O(N), O(1), O(log(N))

Comparing Algorithms (cont.)

Computer Scientists have created a runtime **ordering**

The main metric you will see is Big-Oh

Examples include O(N), O(1), O(log(N))

Big-Oh is an upper bound. It's used to make that statement that a function will grow no worse than some other function.

Comparing Algorithms (cont.)

Computer Scientists have created a runtime **ordering**

The main metric you will see is Big-Oh

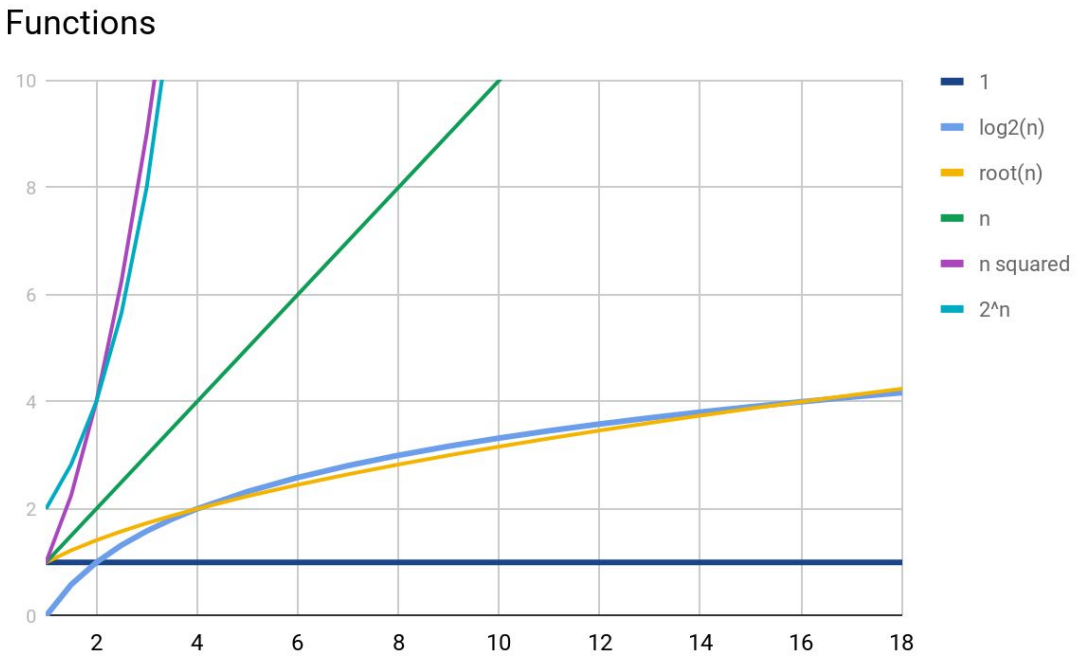
Examples include O(N), O(1), O(log(N))

Big-Oh is an upper bound. It's used to make that statement that a function will grow no worse than some other function.

Formally we would say that

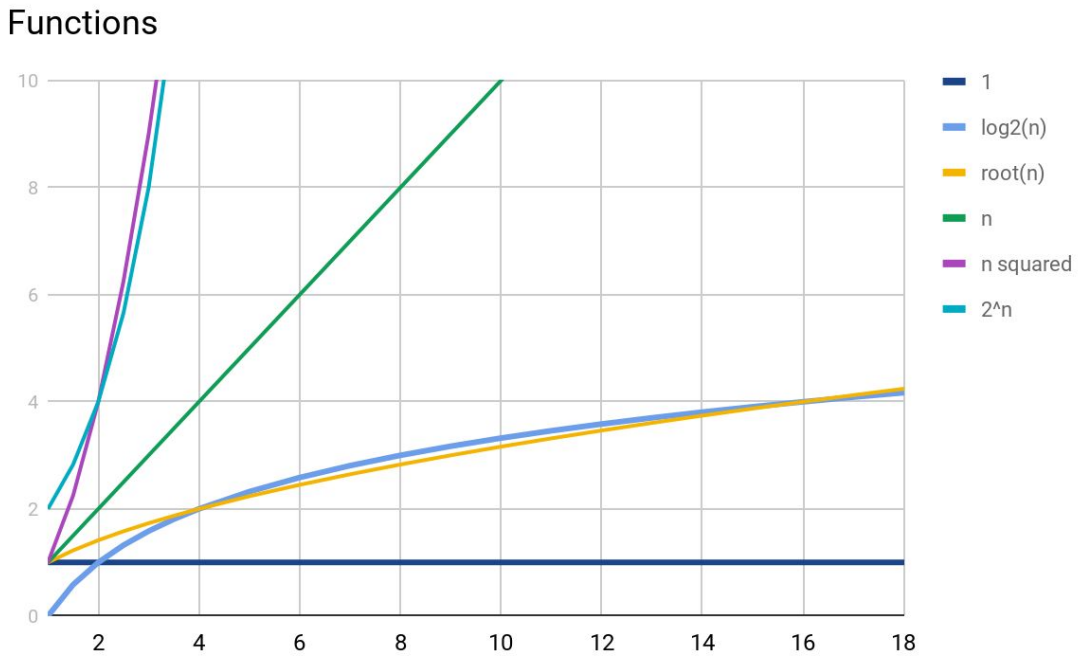
$$f(x) \in \mathrm{O}(g(x)) \equiv \exists\ n_0, c\ \forall\ n > n_0\ (f(n) < cg(n))$$

The Big O Picture



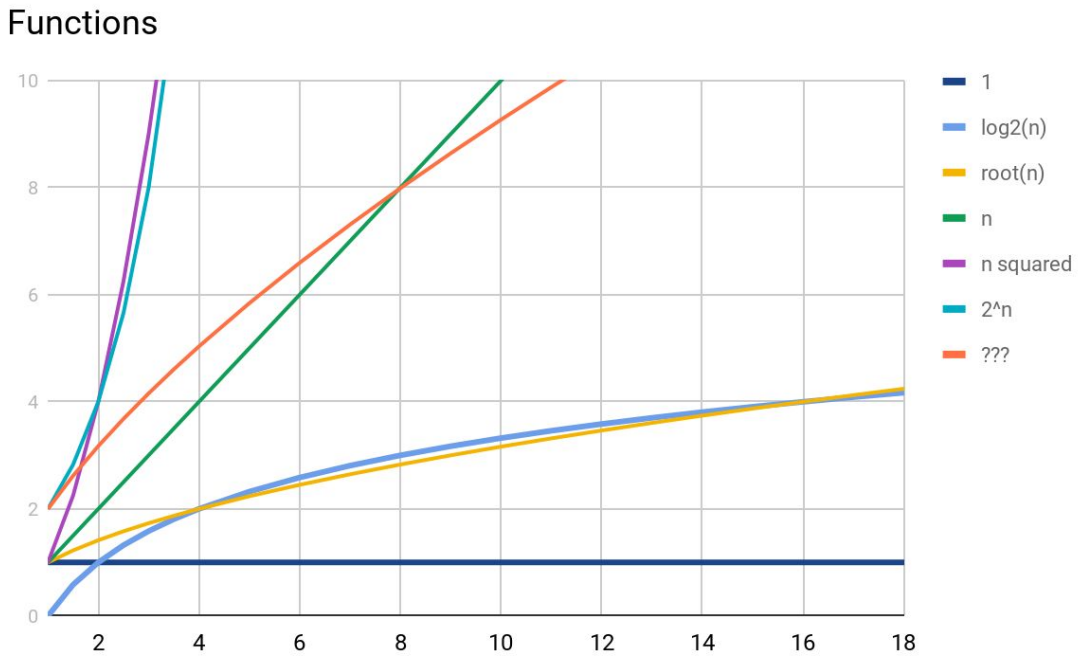
The Big O Picture

Which function is above us for large values of n?



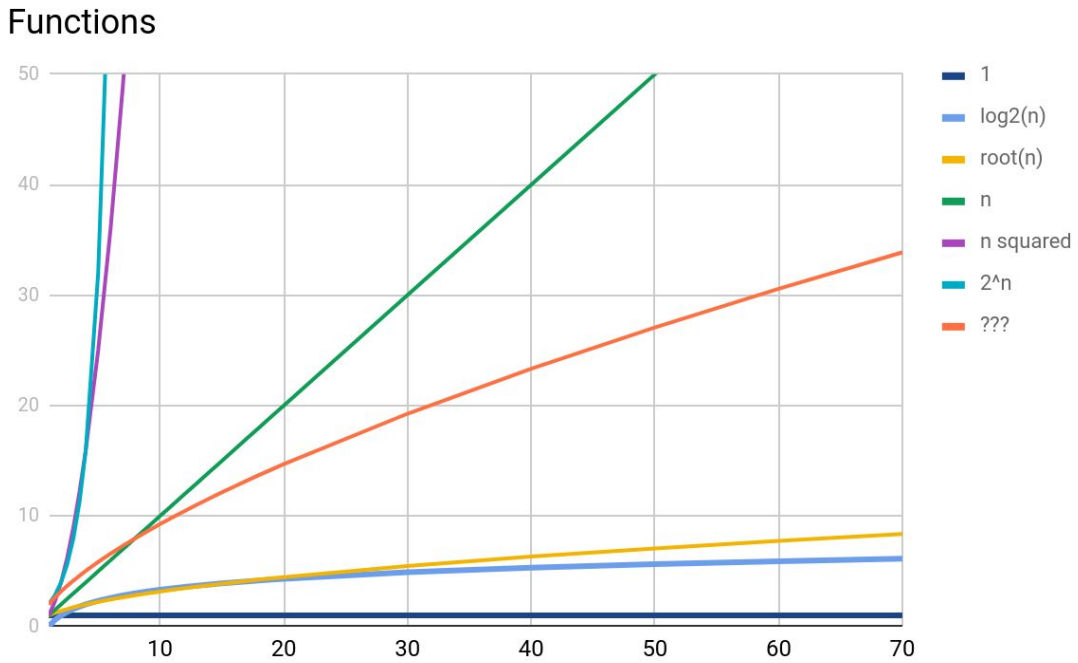
The Big O Picture

Which function is above us for large values of n?



The Big O Picture

Which function is above us for large values of n?



Determine Big-Oh Practically

Determine Big-Oh Practically

Assumption 1: Computers ideally work at a near constant frequency (operations per second)

Determine Big-Oh Practically

Assumption 1: Computers ideally work at a near constant frequency (operations per second)

Assumption 2: Simple statements (e.g. x = y, i++, s.charAt(index), 7 < 3) should take a small constant number of operations. For ease assume 1.

Determine Big-Oh Practically

Assumption 1: Computers ideally work at a near constant frequency (operations per second)

Assumption 2: Simple statements (e.g. x = y, i++, s.charAt(index), 7 < 3) should take a small constant number of operations. For ease assume 1.

Convert a program into a number of operations for some input

Determine Big-Oh Practically

Assumption 1: Computers ideally work at a near constant frequency (operations per second)

Assumption 2: Simple statements (e.g. x = y, i++, s.charAt(index), 7 < 3) should take a small constant number of operations. For ease assume 1.

Convert a program into a number of operations for some input

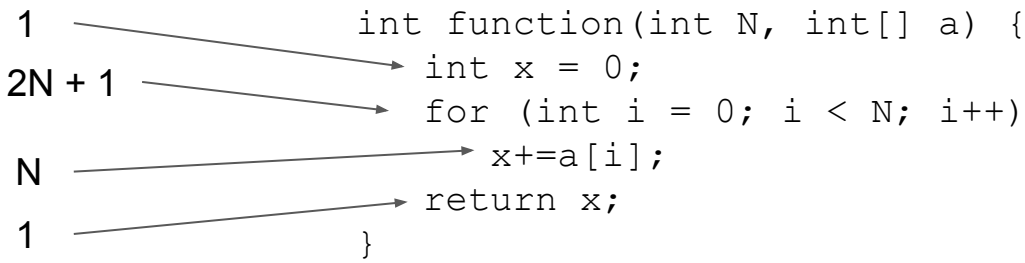
```
int function(int N, int[] a) {
   int x = 0;
   for (int i = 0; i < N; i++)
      x+=a[i];
   return x;
}
```

Determine Big-Oh Practically

Assumption 1: Computers ideally work at a near constant frequency (operations per second)

Assumption 2: Simple statements (e.g. x = y, i++, s.charAt(index), 7 < 3) should take a small constant number of operations. For ease assume 1.

Convert a program into a number of operations for some input

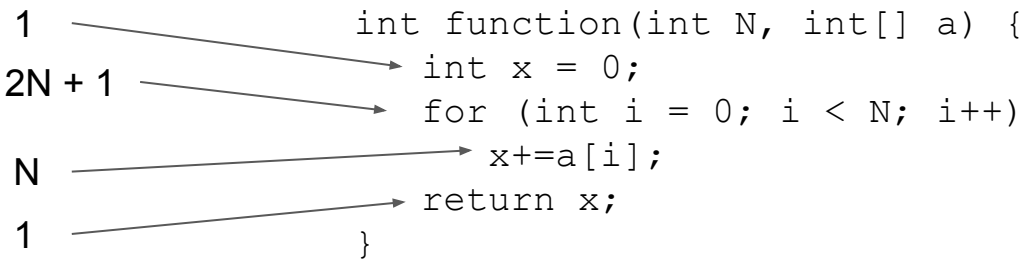


Determine Big-Oh Practically

Assumption 1: Computers ideally work at a near constant frequency (operations per second)

Assumption 2: Simple statements (e.g. x = y, i++, s.charAt(index), 7 < 3) should take a small constant number of operations. For ease assume 1.

Convert a program into a number of operations for some input



3N+3 total

Determine Big-Oh Practically

Assumption 1: Computers ideally work at a near constant frequency (operations per second)

Assumption 2: Simple statements (e.g. x = y, i++, s.charAt(index), 7 < 3) should take a small constant number of operations. For ease assume 1.

Convert a program into a number of operations for some input.

The runtime should be some constant times this value ideally.

3N+3 total

Determine Big-Oh Practically

Assumption 1: Computers ideally work at a near constant frequency (operations per second)

Assumption 2: Simple statements (e.g. x = y, i++, s.charAt(index), 7 < 3) should take a small constant number of operations. For ease assume 1.

Convert a program into a number of operations for some input.

The runtime should be some constant times this value ideally.

But what matters is the largest growing term.

3N+3 total

Determine Big-Oh Practically

Assumption 1: Computers ideally work at a near constant frequency (operations per second)

Assumption 2: Simple statements (e.g. x = y, i++, s.charAt(index), 7 < 3) should take a small constant number of operations. For ease assume 1.

Convert a program into a number of operations for some input.

The runtime should be some constant times this value ideally.

But what matters is the largest growing term.

3**N**+3 total

Determine Big-Oh Practically (cont.)

We can remove all other terms and constant factors.

3**N**+3 total

Determine Big-Oh Practically (cont.)

We can remove all other terms and constant factors.

THAT is what the Big-Oh can be represented as.

3**N**+3 total $\in$ O(N)

Comparing Algorithms Example

Suppose a program takes N^2 solutions when N is less than 10, but Nlog(N) when N is greater than 10. What would you say the Big-Oh runtime is in terms of N?