MAKE SCHOOL

# Algorithm Analysis

CS 1.3 - Core Data Structures

## What is algorithm analysis?

In Computer Science, **algorithm analysis** is the study of the performance of algorithms. It answers the question on how many resources an algorithm consumes.

MAKE SCHOOL

Your program requires **resources** that come in the form of:

1. **Time**
2. **Space**

MAKE SCHOOL

So in other words, we need to question:

1. How much **time** does it take for an algorithm to run?

2. How much **space** does it take for an algorithm to run?

## Time Complexity

## Space Complexity

# Importance of Algorithm Analysis

MAKE SCHOOL

- Estimate how much time required to run an algorithm
- Estimate how much space required to run an algorithm
- Compare the performance of different algorithms without being reliant on specific configurations and hardware.
- Analyze how the time changes when the input size grows
- Helps choose the most efficient algorithm  to solve a particular problem

**Example**: An algorithm that takes too long to run could render results that are outdated or useless to a program that require time-sensitive information.

MAKE SCHOOL

# Time Complexity

MAKE SCHOOL

## What is time complexity?

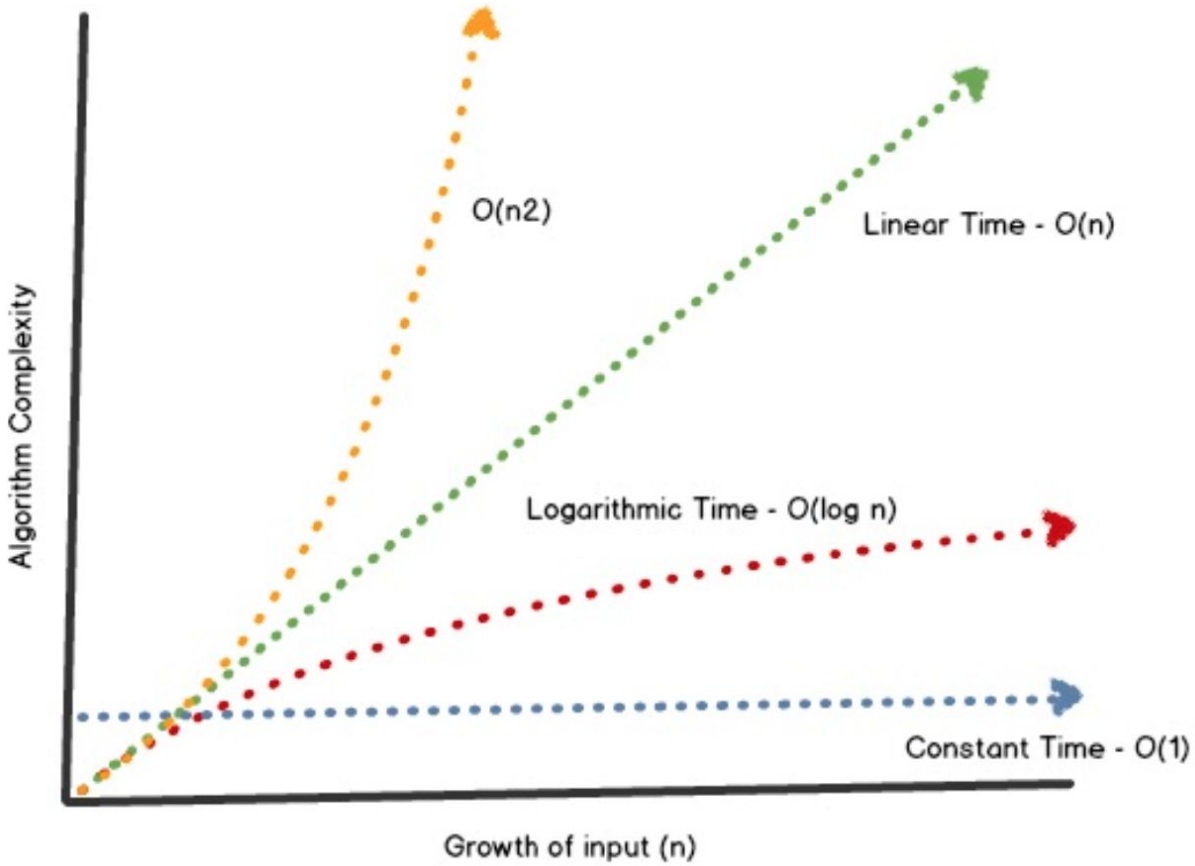
The time complexity of an algorithm is the amount of time taken by the algorithm to complete its process as a function of its input length, *n*.

MAKE SCHOOL

## What is time complexity?

The time complexity of an algorithm is the amount of time taken by the algorithm to complete its process as a function of its input length, *n*.

Refers to functions in mathematics: $f(x)$


O(n2)
Linear Time - O(n)
Logarithmic Time - O(log n)
Constant Time - O(1)
Algorithm Complexity
Growth of input (n)

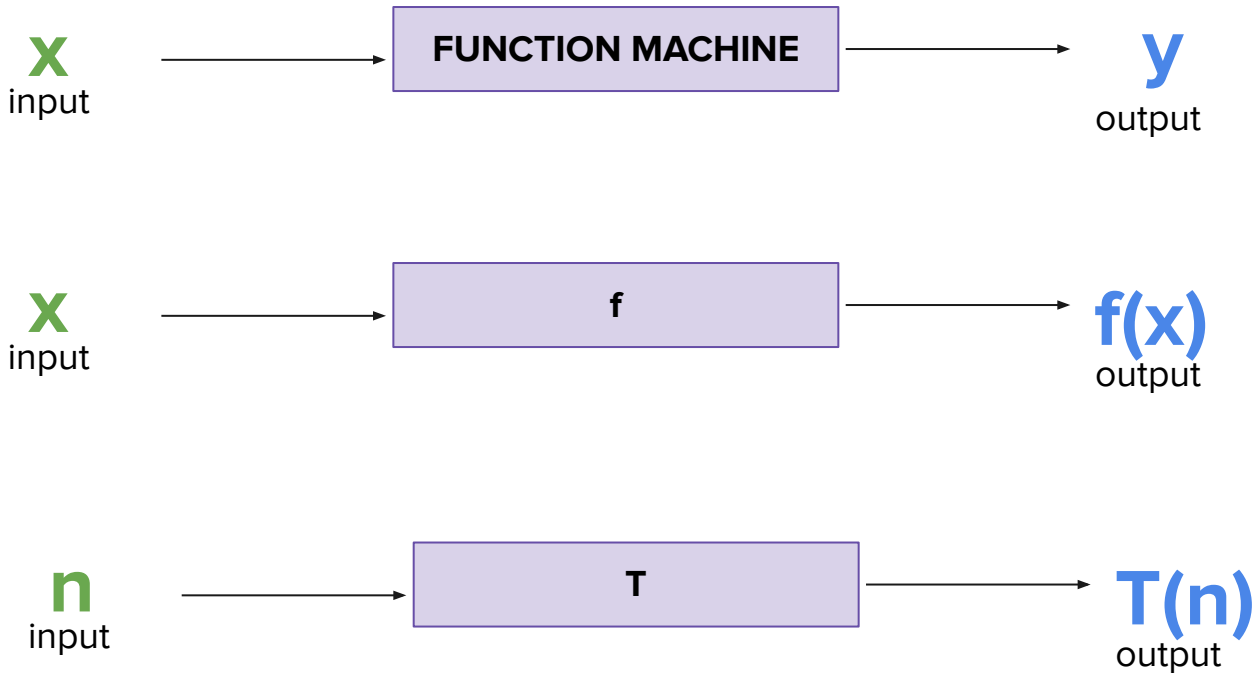
MAKE SCHOOL

$f(x)$

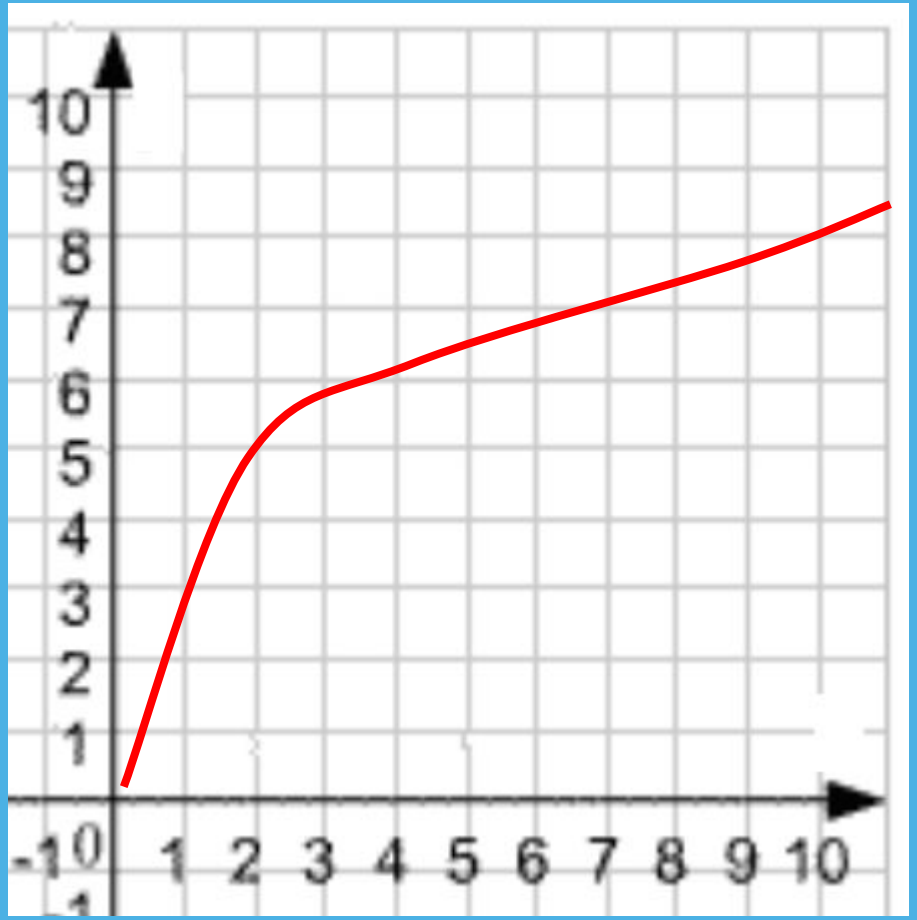
$T(n)$

**Math**

**Time Complexity**




**For time complexity, input is labeled as “n”, the function machine is labeled as “T” to stand for time, and the output therefore is T(n).**




# T(n)

## Time Complexity

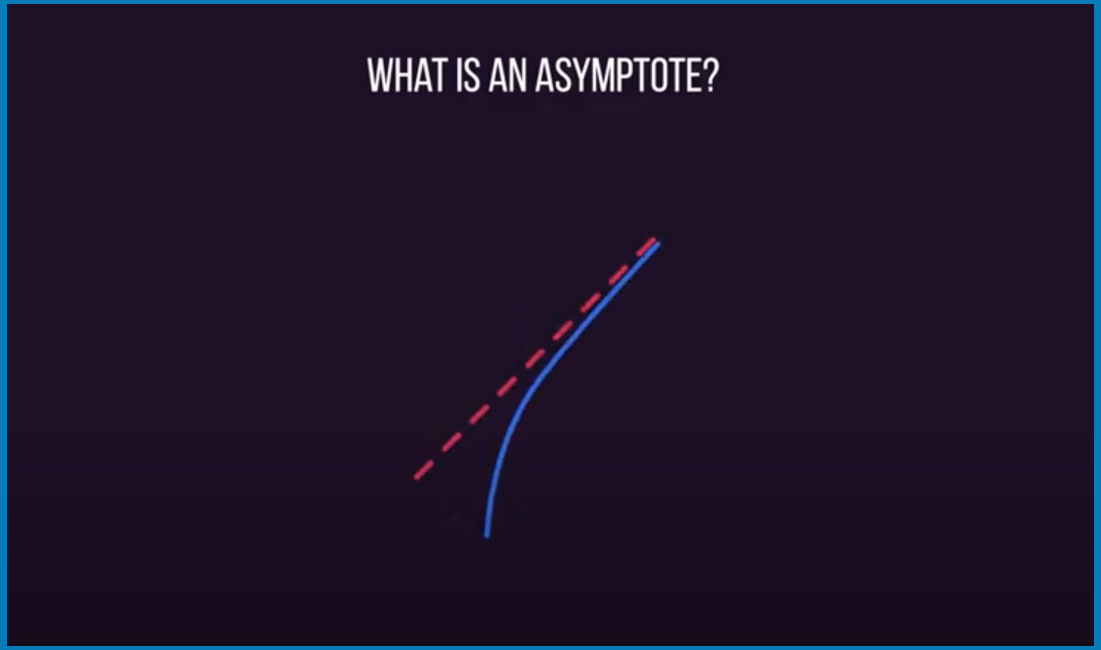
MAKE SCHOOL

The time complexity of an algorithm is commonly expressed using **asymptotic notations**.

The time complexity of an algorithm is commonly expressed using **asymptotic notations**.

MAKE SCHOOL

# Math Review: Asymptotes

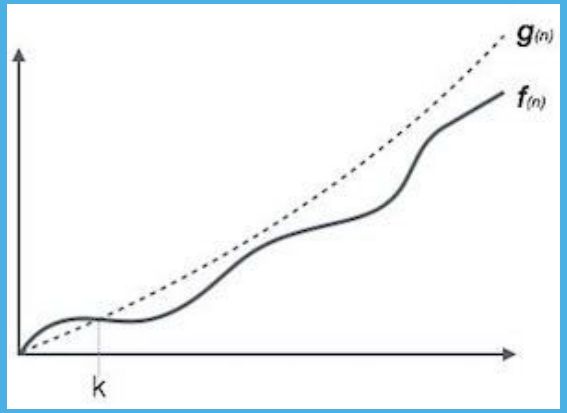



🔗 **What is an Asymptote?**

The time complexity of an algorithm is commonly expressed using **asymptotic notations**.

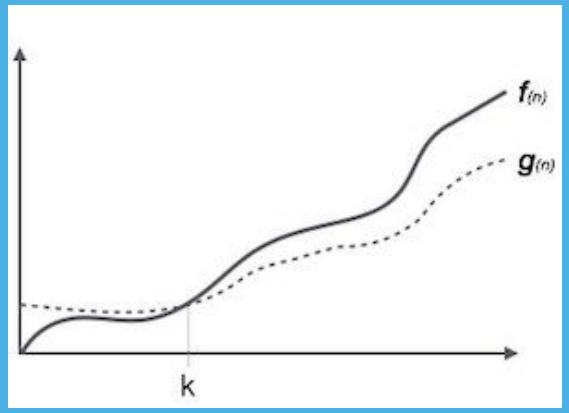
There are three common asymptotic notations:

- Big O
- Big Theta
- Big Omega



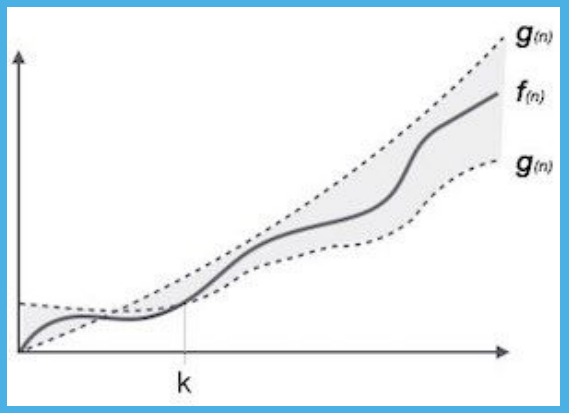

# Big O

Upper Bound Asymptote
“Worst Case”



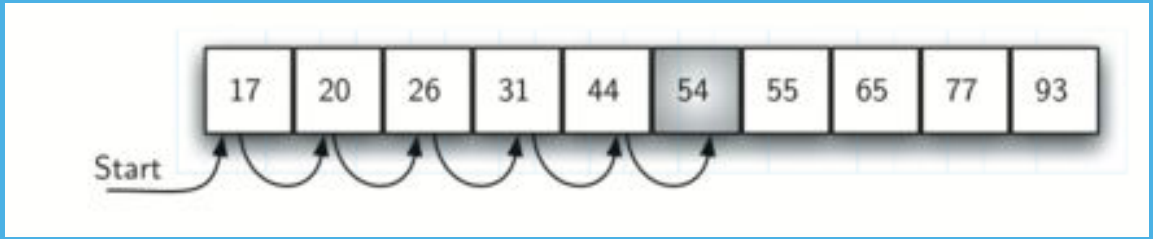

# Big Ω

Lower Bound Asymptote
“Best Case”



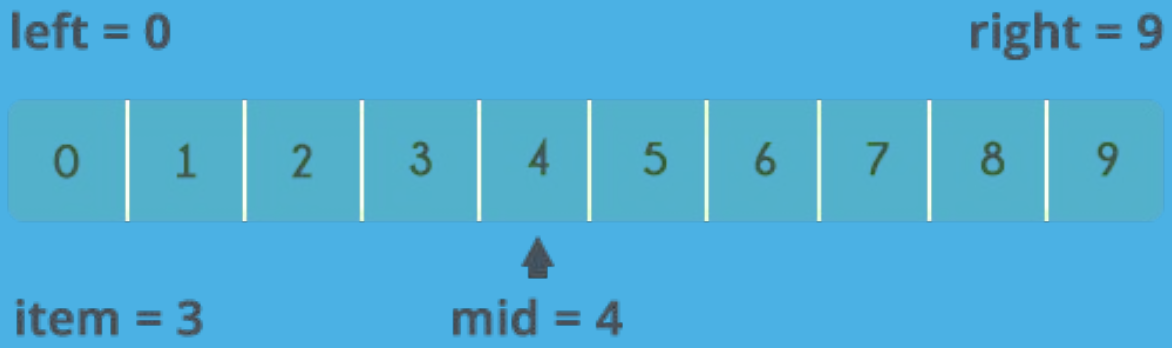

# Big θ

Average Bound Asymptote
“Average Case”



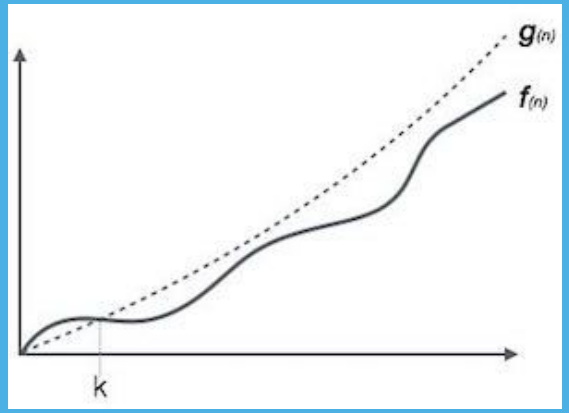

# Sequential Search / Linear Search

- Best Case (Big Ω): Target value is at index  0.
- Worst Case (Big O): Target value is not in list.
- Average Case (Big θ): Target value in middle of list




# Binary Search

- Best Case (Big Ω): Target value is in middle.
- Worst Case (Big O): Target value is not in list.
- Average Case (Big θ): Target value in middle of list




# Big O

Upper Bound Asymptote

“Worst Case”

# Big O Notation

*Using not-boring math to measure code's efficiency*

*20 minute pair breakout*

MAKE SCHOOL

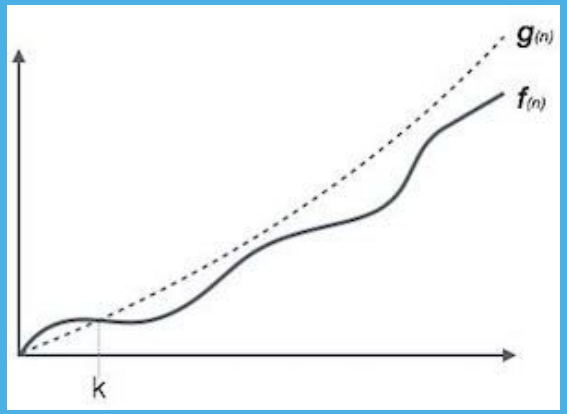
1. **Read the linked article**

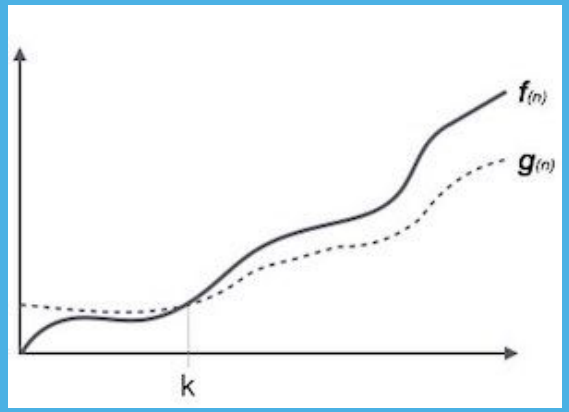
2. **Answer the following questions in your group**:

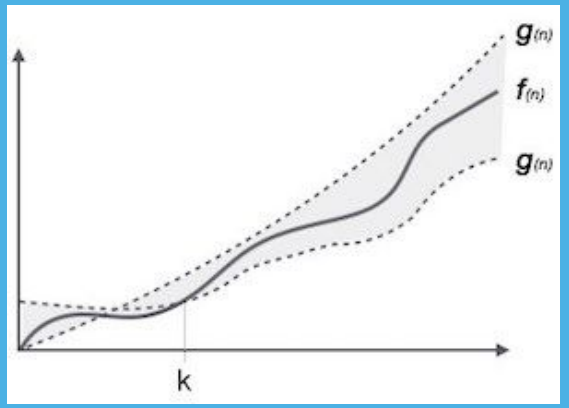
    a. **When should you optimize** an algorithm for an **early stage product**?

    b. **When should you optimize** an algorithm for a **popular product**?

    c. Should **all** algorithms be optimized? **Why** or **why not**? **Defend** your answer.




# Big O

Upper Bound Asymptote
“Worst Case”

# Big Ω

Lower Bound Asymptote
“Best Case”

# Big θ

Average Bound Asymptote
“Average Case”

MAKE SCHOOL

Time complexity estimates the time to run an algo rithm. It's calculated by counting elementary opera tions.

# Big O Counting Rules

MAKE SCHOOL

- **O(1)** - set of non-recursive and non-loop statements.
- **O(n)** - loops that are incremented/decremented by a constant rate
- **O($n^c$)** - number of times the innermost statement is executed. (2 for-loops would be O($n^2$)
- **O(log n)** - loop variables are divided by a constant amount.
- **O(log log n)** - loop variable is reduced/increased exponentially.

# Let's calculate the time complexity of this algorithm

MAKE SCHOOL

```
def sum_list(num_lst):
    total = 0
    for num in num_list:
        total += num
    return total
```

**T(n) =**

# Let's calculate the time complexity of this algorithm

```
def sum_list(num_lst):
    total = 0
    for num in num_list:
        total += num
    return total
```

$$T(n) = O(1) +$$

# Let's calculate the time complexity of this algorithm

```
def sum_list(num_lst):
    total = 0
    for num in num_list:
        total += num
    return total
```

$$T(n) = O(1) + O(n) +$$

# Let's calculate the time complexity of this algorithm

```
def sum_list(num_lst):
    total = 0
    for num in num_list:
        total += num
    return total
```

$$T(n) = O(1) + O(n) + O(1)$$

# Let's calculate the time complexity of this algorithm

MAKE SCHOOL

$$T(n) = O(1) + O(n) + O(1)$$

**Simplifies to....**

$$T(n) = O(n) + O(2)$$

**But Big O simplifies this even more...**

$$T(n) = O(n)$$

# General Counting Rules for Big O 

MAKE SCHOOL

- Ignore constants, for example:
  - O(2)  is reduced to O(1)
  - O(2n) is reduced to O(n)
  - O(2 log n) is reduced to O(log n)
- We only care about the most dominant term in the equation.

  Here is the order of dominance:

  **O(1) < O(logn) < O(n) < O(nlogn) < O($n^2$) < O(2^n) < O(n!)**

# General Counting Rules for Big O

MAKE SCHOOL

- Ignore constants, for example:
  - O(2) is O(1)
  - O(2n) is O(n)
  - O(n/2) is O(n)
- We only care about the most dominant term in the equation.

  Here is the order of dominance:

  **O(1) < O(logn) < O(n) < O(nlogn) < O($n^2$) < O(2^n) < O(n!)**

MAKE SCHOOL

# Big Theta and Omega

# Asymptotic Notation

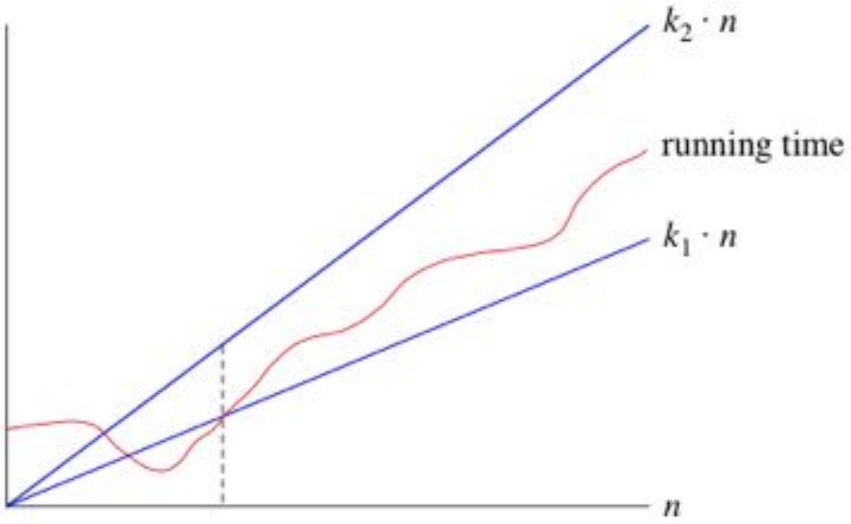
MAKE SCHOOL

So far we have only learned about big O but there's more to the story...

- Big-Θ notation: tight bound
- Big-Ω notation: lower bound
- Big O notation: upper bound

The term asymptotic means approaching a value or curve arbitrarily closely (i.e., as some sort of limit is taken).

# Big-Θ

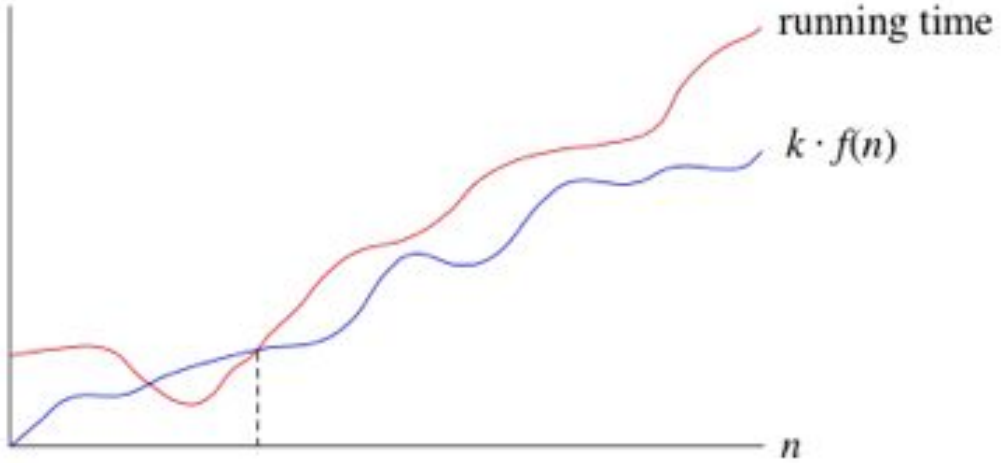



From Khan Academy

MAKE SCHOOL

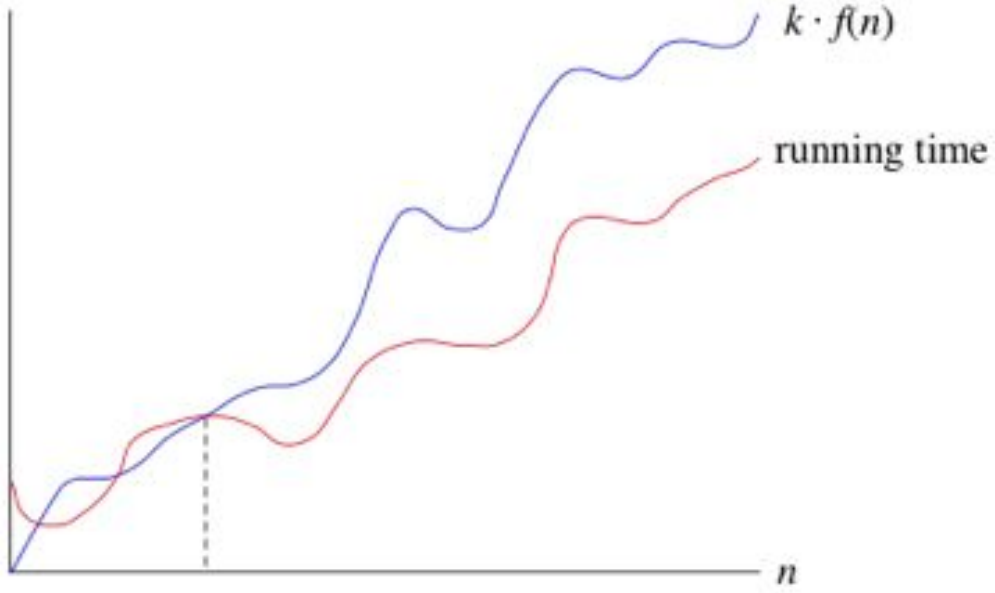
$\Theta(g(n)) = \{f(n)$: there exist positive constants c1, c2 and n0 such that $0 <= c1*g(n) <= f(n) <= c2*g(n)$ for all $n >= n0\}$

# Big-Ω




From Khan Academy

# Big O




From Khan Academy

MAKE SCHOOL

# A comparison

MAKE SCHOOL

# Space Complexity

MAKE SCHOOL

## What is space complexity?

The space complexity is the amount of space (i.e. memory) taken by the algorithm to run as a function of its input length, *n*.

MAKE SCHOOL

## What is space complexity?

The space complexity is the amount of space (i.e. memory) taken by the algorithm to run as a function of its input length, *n*.

Refers to functions in mathematics: $f(x)$

MAKE SCHOOL

## Space includes:

- Space used by the input, *n*.
- Auxiliary space.


Space Complexity
0:01 / 5:13

# Space Complexity

MAKE SCHOOL

# Resources & Credits

# Resources & Credits

- Big O Notation: Interview Cake