# Analysis of Algorithms

## Sorting in linear time

Arturo Calderón (the TA guy)

September 5, 2020

# Outline

# Outline

# Algorithm efficiency

**When measuring an algorithm efficiency we must consider:**

- Speed.
- Memory usage.
- Scalability.

# Algorithm efficiency

**When measuring an algorithm efficiency we must consider:**

- Speed.
- Memory usage.
- Scalability.

**Measuring speed!**

- Speed is measured in terms of the number of operations relative to the size of the input.

# Algorithm efficiency

**When measuring an algorithm efficiency we must consider:**

- Speed.
- Memory usage.
- Scalability.

**Measuring speed!**

- Speed is measured in terms of the number of operations relative to the size of the input.

# Algorithm efficiency

**When measuring an algorithm efficiency we must consider:**

- Speed.
- Memory usage.
- Scalability.

**Measuring speed!**

- Speed is measured in terms of the number of operations relative to the size of the input.

# Asymptotic bounds

**Intuition**

An asymptotic bound is a curve that represents the limit of a function.

# Asymptotic bounds

**Intuition**

An asymptotic bound is a curve that represents the limit of a function.

**For the purpose of analyzing the speed of an algorithm, tree typical asymptotic bounds are used.**

1. Big $O$ (Upper bound)
2. Big $\Omega$(Lower bound)
3. Big $\Theta$(Expected bound)

# Asymptotic bounds

**Intuition**

An asymptotic bound is a curve that represents the limit of a function.

**For the purpose of analyzing the speed of an algorithm, tree typical asymptotic bounds are used.**

1. Big $O$ (Upper bound)
2. Big $\Omega$(Lower bound)
3. Big $\Theta$(Expected bound)

# Asymptotic bounds

### Intuition

An asymptotic bound is a curve that represents the limit of a function.

### For the purpose of analyzing the speed of an algorithm, tree typical asymptotic bounds are used.

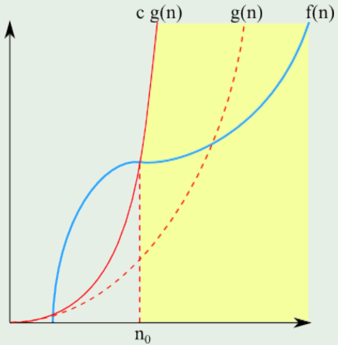
1. Big $O$ (Upper bound)
2. Big $\Omega$(Lower bound)
3. Big $\Theta$(Expected bound)

# Outline

# Big O (Upper bound)

## Intuition

- Let $f(n)$ and $g(n)$ be two real valued functions, lets build intuition on the meaning of $f(n) \in O(g(n))$.



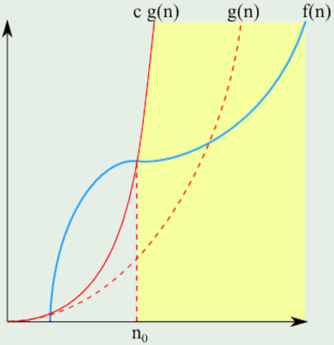

## Definition

- $f(n) \in O(g(n))$ if there exists $c, n_0 > 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

# Big O (Upper bound)

## Intuition

- Let $f(n)$ and $g(n)$ be two real valued functions, lets build intuition on the meaning of $f(n) \in O(g(n))$.



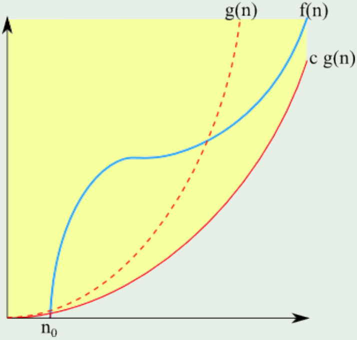

## Definition

- $f(n) \in O(g(n))$ if there exists $c, n_0 > 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

# Big Ω (Lower bound)

## Intuition

- Let $f(n)$ and $g(n)$ be two real valued functions, lets build intuition on the meaning of $f(n) \in \Omega(g(n))$.



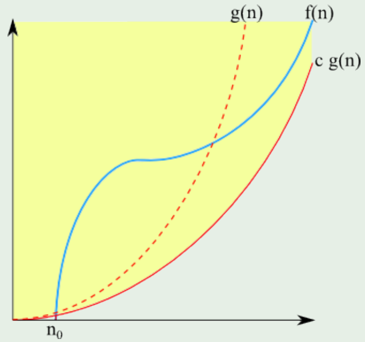

## Definition

- $f(n) \in \Omega(g(n))$ if there exists $c, n_0 > 0$ such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$.

# Big $\Omega$ (Lower bound)

## Intuition

- Let $f(n)$ and $g(n)$ be two real valued functions, lets build intuition on the meaning of $f(n) \in \Omega(g(n))$.



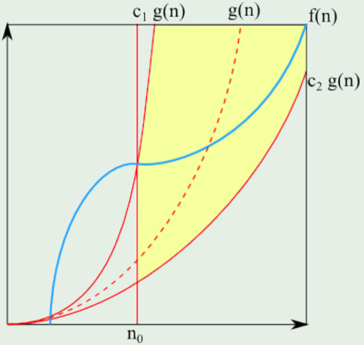

## Definition

- $f(n) \in \Omega(g(n))$ if there exists $c, n_0 > 0$ such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$.

# Big Θ (Expected bound)

## Intuition

- Let $f(n)$ and $g(n)$ be two real valued functions, lets build intuition on the meaning of $f(n) \in \Theta(g(n))$.



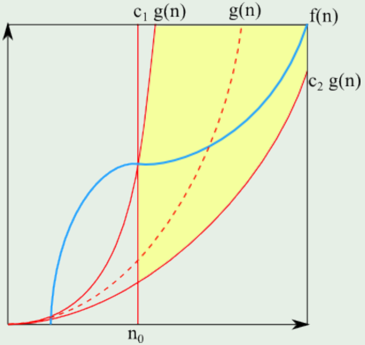

## Definition

- $f(n) \in O(g(n))$ if there exists $c, n_0 > 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

# Big Θ (Expected bound)

## Intuition

- Let $f(n)$ and $g(n)$ be two real valued functions, lets build intuition on the meaning of $f(n) \in \Theta(g(n))$.




## Definition

- $f(n) \in O(g(n))$ if there exists $c, n_0 > 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

# Outline

# Correctness of an algorithm

## Loop invariant and loop conditional

- A loop invariant is a condition that is necessarily true immediately before and immediately after each iteration of a loop.
- A loop conditional is a statements that controls the termination of the loop.
- Both loop invariant and loop conditional must be different conditions.

# Correctness of an algorithm

## Loop invariant and loop conditional

- A loop invariant is a condition that is necessarily true immediately before and immediately after each iteration of a loop.
- A loop conditional is a statements that controls the termination of the loop.
- Both loop invariant and loop conditional must be different conditions.

## Facts!

- To prove an algorithm is correct we must find a loop conditional that ensures the algorithm terminates.
- The loop invariant mus be true:
  - Before the loop starts
  - Before each iteration of the loop
  - After the loop terminates

# Correctness of an algorithm

## Loop invariant and loop conditional

- A loop invariant is a condition that is necessarily true immediately before and immediately after each iteration of a loop.
- A loop conditional is a statements that controls the termination of the loop.
- Both loop invariant and loop conditional must be different conditions.

## Facts!

- To prove an algorithm is correct we must find a loop conditional that ensures the algorithm terminates.
- The loop invariant mus be true:
  - Before the loop starts
  - Before each iteration of the loop
  - After the loop terminates

# Correctness of an algorithm

## Loop invariant and loop conditional

- A loop invariant is a condition that is necessarily true immediately before and immediately after each iteration of a loop.
- A loop conditional is a statements that controls the termination of the loop.
- Both loop invariant and loop conditional must be different conditions.

## Facts!

- To prove an algorithm is correct we must find a loop conditional that ensures the algorithm terminates.
- The loop invariant mus be true:
  - Before the loop starts
  - Before each iteration of the loop
  - After the loop terminates

# Correctness of an algorithm

## Loop invariant and loop conditional

- A loop invariant is a condition that is necessarily true immediately before and immediately after each iteration of a loop.
- A loop conditional is a statements that controls the termination of the loop.
- Both loop invariant and loop conditional must be different conditions.

## Facts!

- To prove an algorithm is correct we must find a loop conditional that ensures the algorithm terminates.
- The loop invariant mus be true:
  - Before the loop starts
  - Before each iteration of the loop
  - After the loop terminates

# Correctness of an algorithm

## Loop invariant and loop conditional

- A loop invariant is a condition that is necessarily true immediately before and immediately after each iteration of a loop.
- A loop conditional is a statements that controls the termination of the loop.
- Both loop invariant and loop conditional must be different conditions.

## Facts!

- To prove an algorithm is correct we must find a loop conditional that ensures the algorithm terminates.
- The loop invariant mus be true:
  - Before the loop starts
  - Before each iteration of the loop
  - After the loop terminates

# Correctness of an algorithm

## Loop invariant and loop conditional

- A loop invariant is a condition that is necessarily true immediately before and immediately after each iteration of a loop.
- A loop conditional is a statements that controls the termination of the loop.
- Both loop invariant and loop conditional must be different conditions.

## Facts!

- To prove an algorithm is correct we must find a loop conditional that ensures the algorithm terminates.
- The loop invariant mus be true:
  - Before the loop starts
  - Before each iteration of the loop
  - After the loop terminates

# Correctness of an algorithm

## Loop invariant and loop conditional

- A loop invariant is a condition that is necessarily true immediately before and immediately after each iteration of a loop.
- A loop conditional is a statements that controls the termination of the loop.
- Both loop invariant and loop conditional must be different conditions.

## Facts!

- To prove an algorithm is correct we must find a loop conditional that ensures the algorithm terminates.
- The loop invariant mus be true:
  - Before the loop starts
  - Before each iteration of the loop
  - After the loop terminates

# Outline

# Well, now you know the basics. Time to work!

**From Dasgupta's Algorithms book, exercise 0.1**

Using the definition in each of the following situations indicate wether $f = O(g)$, or $f = \Omega(g)$, or both (in which case $f = \Theta(g)$).

1. $f(n) = n - 100$, $g(n) = n - 200$
2. $f(n) = n2^n$, $g(n) = 3^n$

# Well, now you know the basics. Time to work!

## From Dasgupta's Algorithms book, exercise 0.1

Using the definition in each of the following situations indicate wether $f = O(g)$, or $f = \Omega(g)$, or both (in which case $f = \Theta(g)$).

1. $f(n) = n - 100$, $g(n) = n - 200$
2. $f(n) = n2^n$, $g(n) = 3^n$

# Well, now you know the basics. Time to work!

**From Dasgupta's Algorithms book, exercise 0.1**

Using the definition in each of the following situations indicate wether $f = O(g)$, or $f = \Omega(g)$, or both (in which case $f = \Theta(g)$).

1. $f(n) = n - 100$, $g(n) = n - 200$
2. $f(n) = n2^n$, $g(n) = 3^n$

# Exercise

**Let's try this one!**

Show that $\sum_{k=1}^{n} \frac{1}{k^2}$ is bounded by a constant. (help me here!).

# Exercise

## From Cormen's book exercise 2.1-3

Consider the searching problem:

*Input:* A sequence of $n$ numbers $A = \langle a_1, a_2, ..., a_n \rangle$ and a value $v$.

*Output:* An index $i$ such that $v = A[i]$ or the special value $NIL$ if $v$ does not appear in $A$.

Write pseudocode for linear search, which scans through the sequence, looking for $v$. Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.

# Exercise

**From Cormen's book exercise 2.3-4**

We can express insertion sort as a recursive procedure as follows. In order to sort $A[1...n]$, we recursively sort $A[1...n-1]$ and then insert $A[n]$ into the sorted array. Write a recurrence for the running time of this recursive version of insertion sort.

# Exercise

**From Cormen's book exercise 3.1-7**

Prove that $o\left(g\left(n\right)\right) \cap \omega\left(g\left(n\right)\right)$ is the empty set.

# Exercise

**From Cormen's book exercise 3.2-8**

Show that $k \ln k = \Theta(n)$ implies that $k = \Theta\left(\frac{n}{\ln n}\right)$.

# Exercise

## From Cormen's book exercise 4.3-1, 4.3-6

1. Show that the solution of $T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1$ is $O(\lg n)$.
2. Show that the solution of $T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1$ is $\Omega(n \lg n)$.

# Exercise

**From Cormen's book exercise 4.3-1, 4.3-6**

1. Show that the solution of $T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1$ is $O(\lg n)$.
2. Show that the solution of $T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1$ is $\Omega(n \lg n)$.