Lecture 3: Intro to proofs for algorithms

Jessica Sorrell

September 2, 2025
601.433/633 Introduction to Algorithms

Today

Discuss common proof techniques for algorithms.

- Inductive arguments (weak, strong)
- Proof by contradiction
- Direct proof
- Loop invariant
- Proof by contrapositive

We'll demonstrate proof techniques by proving the correctness and running time of algorithms you've seen before.

Quicksort review

Algorithm Quicksort
Input: array A of length n

```
if n ≤ 1 then
    return A
end if
Pick some element p ∈ A as the pivot
Let L be the elements less than or equal to p, let G be the elements larger than p
L′ ← Quicksort(L)
G′ ← Quicksort(G)
return L′‖p‖G′
```

Strong Induction vs Weak Induction

Strong induction:

- Prove property holds for a base case
- e.g. Quicksort always returns a sorted array for input arrays of size $n \leq 1$

Strong Induction vs Weak Induction

Strong induction:

- Prove property holds for a base case
- e.g. Quicksort always returns a sorted array for input arrays of size $n \leq 1$
- Assume inductive hypothesis, that property holds *for all* $n \leq k$. Then show that property holds for $n = k + 1$.
- e.g. Assume Quicksort always returns a sorted array for input arrays of size $\leq k$. Show it returns a sorted array for input arrays of size $k + 1$.

Strong Induction vs Weak Induction

Strong induction:

- Prove property holds for a base case
- e.g. Quicksort always returns a sorted array for input arrays of size $n \leq 1$
- Assume inductive hypothesis, that property holds *for all* $n \leq k$. Then show that property holds for $n = k + 1$.
- e.g. Assume Quicksort always returns a sorted array for input arrays of size $\leq k$. Show it returns a sorted array for input arrays of size $k + 1$.

Weak induction:

- Prove property holds for a base case

Strong Induction vs Weak Induction

Strong induction:

- Prove property holds for a base case
- e.g. Quicksort always returns a sorted array for input arrays of size $n \leq 1$
- Assume inductive hypothesis, that property holds *for all* $n \leq k$. Then show that property holds for $n = k + 1$.
- e.g. Assume Quicksort always returns a sorted array for input arrays of size $\leq k$. Show it returns a sorted array for input arrays of size $k + 1$.

Weak induction:

- Prove property holds for a base case
- Assume inductive hypothesis, that property holds for $n = k$. Then show that property holds for $n = k + 1$.
- e.g. Assume Quicksort always returns a sorted array for input arrays of size *exactly* k. Show it returns a sorted array for input arrays of size $k + 1$.

Correctness of Quicksort - (Strong) Inductive Proof

Claim: Given an array $\boldsymbol{A}$ of length $\boldsymbol{n}$, Quicksort($\boldsymbol{A}$) returns an array with all elements of $\boldsymbol{A}$ sorted from least to greatest.

Correctness of Quicksort - (Strong) Inductive Proof

Claim: Given an array A of length n, Quicksort(A) returns an array with all elements of A sorted from least to greatest.

Proof:

- Base case: $n \leq 1$. Quicksort(A) returns A.

Correctness of Quicksort - (Strong) Inductive Proof

Claim: Given an array A of length n, Quicksort(A) returns an array with all elements of A sorted from least to greatest.

Proof:

- Base case: $n \leq 1$. Quicksort(A) returns A.
- Inductive step: Assume Quicksort(A) returns a sorted array for all A of length $\leq n$. Show it returns a sorted array for all A of length $n + 1$.

Correctness of Quicksort - (Strong) Inductive Proof

Claim: Given an array $\boldsymbol{A}$ of length $\boldsymbol{n}$, Quicksort($\boldsymbol{A}$) returns an array with all elements of $\boldsymbol{A}$ sorted from least to greatest.
Proof:

- Base case: $\boldsymbol{n} \leq \mathbf{1}$. Quicksort($\boldsymbol{A}$) returns $\boldsymbol{A}$.
- Inductive step: Assume Quicksort($\boldsymbol{A}$) returns a sorted array for all $\boldsymbol{A}$ of length $\leq \boldsymbol{n}$. Show it returns a sorted array for all $\boldsymbol{A}$ of length $\boldsymbol{n} + \mathbf{1}$.
  - Pick pivot $\boldsymbol{p} \in \boldsymbol{A}$. Let $\boldsymbol{L}$ be the elements less than or equal to $\boldsymbol{p}$, let $\boldsymbol{G}$ be the elements larger than $\boldsymbol{p}$.

Correctness of Quicksort - (Strong) Inductive Proof

Claim: Given an array A of length n, Quicksort(A) returns an array with all elements of A sorted from least to greatest.

Proof:

- Base case: $n \leq 1$. Quicksort(A) returns A.
- Inductive step: Assume Quicksort(A) returns a sorted array for all A of length $\leq n$. Show it returns a sorted array for all A of length $n+1$.
  - Pick pivot $p \in A$. Let L be the elements less than or equal to p, let G be the elements larger than p.
  - L and G are of length $\leq n$, so by inductive hypothesis, Quicksort(L) and Quicksort(G) return sorted arrays L' and G'.

Correctness of Quicksort - (Strong) Inductive Proof

Claim: Given an array A of length n, Quicksort(A) returns an array with all elements of A sorted from least to greatest.

Proof:

- Base case: $n \leq 1$. Quicksort(A) returns A.
- Inductive step: Assume Quicksort(A) returns a sorted array for all A of length $\leq n$. Show it returns a sorted array for all A of length $n+1$.
  - Pick pivot $p \in A$. Let L be the elements less than or equal to p, let G be the elements larger than p.
  - L and G are of length $\leq n$, so by inductive hypothesis, Quicksort(L) and Quicksort(G) return sorted arrays L' and G'.
  - Therefore $L' \| p \| G'$ is sorted.

Why Strong Induction?

- A weak inductive hypothesis assumes the desired property holds for $n = k$.

Why Strong Induction?

- A weak inductive hypothesis assumes the desired property holds for $n = k$.
- A strong inductive hypothesis assumes the desired property holds for all $n \leq k$.

Why Strong Induction?

- A weak inductive hypothesis assumes the desired property holds for $n = k$.
- A strong inductive hypothesis assumes the desired property holds for all $n \leq k$.
- Quicksort recursively calls itself on L and G, which we don't know the size of a priori
- In strong induction, we assume that Quicksort is correct for all arrays of size $\leq k$, so doesn't matter what the exact size L and G are, because we know they are both $\leq k$.

Correctness of Quicksort - Proof by Contradiction

Claim: Given an array $\boldsymbol{A}$ of length $\boldsymbol{n}$, Quicksort($\boldsymbol{A}$) returns an array with all elements of $\boldsymbol{A}$ sorted from least to greatest.

Correctness of Quicksort - Proof by Contradiction

Claim: Given an array A of length n, Quicksort(A) returns an array with all elements of A sorted from least to greatest.

Proof:

- Assume there exists at least one array such that Quicksort does not return a sorted array. Let A be the smallest such array and let n be the size of A.

Correctness of Quicksort - Proof by Contradiction

Claim: Given an array A of length n, Quicksort(A) returns an array with all elements of A sorted from least to greatest.

Proof:

- Assume there exists at least one array such that Quicksort does not return a sorted array. Let A be the smallest such array and let n be the size of A.
- Note $n \geq 2$.

Correctness of Quicksort - Proof by Contradiction

Claim: Given an array $\boldsymbol{A}$ of length $\boldsymbol{n}$, Quicksort($\boldsymbol{A}$) returns an array with all elements of $\boldsymbol{A}$ sorted from least to greatest.

Proof:

- Assume there exists at least one array such that Quicksort does not return a sorted array. Let $\boldsymbol{A}$ be the smallest such array and let $\boldsymbol{n}$ be the size of $\boldsymbol{A}$.
- Note $\boldsymbol{n} \geq \mathbf{2}$.
- So Quicksort($\boldsymbol{A}$) picks a pivot element $\boldsymbol{p} \in \boldsymbol{A}$, defines $\boldsymbol{L}$ and $\boldsymbol{G}$ as the elements less than or equal to $\boldsymbol{p}$ and the elements greater than $\boldsymbol{p}$ respectively, and recursively calls Quicksort on $\boldsymbol{L}$ and $\boldsymbol{G}$.

Correctness of Quicksort - Proof by Contradiction

Claim: Given an array A of length n, Quicksort(A) returns an array with all elements of A sorted from least to greatest.

Proof:

- Assume there exists at least one array such that Quicksort does not return a sorted array. Let A be the smallest such array and let n be the size of A.
- Note $n \geq 2$.
- So Quicksort(A) picks a pivot element $p \in A$, defines L and G as the elements less than or equal to p and the elements greater than p respectively, and recursively calls Quicksort on L and G .
- By assumption that A is the smallest such array, L and G are sorted.

Correctness of Quicksort - Proof by Contradiction

Claim: Given an array A of length n, Quicksort(A) returns an array with all elements of A sorted from least to greatest.

Proof:

- Assume there exists at least one array such that Quicksort does not return a sorted array. Let A be the smallest such array and let n be the size of A.
- Note $n \geq 2$.
- So Quicksort(A) picks a pivot element $p \in A$, defines L and G as the elements less than or equal to p and the elements greater than p respectively, and recursively calls Quicksort on L and G .
- By assumption that A is the smallest such array, L and G are sorted.
- Therefore $L \| p \| G$ is sorted.
- Contradiction: A is not the smallest array such that Quicksort does not return a sorted array.

Direct Proof

A direct proof argues the conclusion of a claim *directly* from its assumptions.

Direct Proof

A direct proof argues the conclusion of a claim *directly* from its assumptions.

For a statement of the form $\boldsymbol{A} \Rightarrow \boldsymbol{B}$, a direct proof shows that $\boldsymbol{B}$ follows from the logical implications of $\boldsymbol{A}$.

Quicksort Runtime - Direct Proof

Claim: Quicksort runs in time $O(n^2)$ in the worst case.

Quicksort Runtime - Direct Proof

Claim: Quicksort runs in time $O(n^2)$ in the worst case.

- Before making its two recursive calls, Quicksort compares every element of its input array to the pivot, taking time $\Theta(n)$.

Quicksort Runtime - Direct Proof

Claim: Quicksort runs in time $O(n^2)$ in the worst case.

- Before making its two recursive calls, Quicksort compares every element of its input array to the pivot, taking time $\Theta(n)$.
- The worst case for runtime occurs when the pivot is the smallest or largest element of the array.

Quicksort Runtime - Direct Proof

Claim: Quicksort runs in time $O(n^2)$ in the worst case.

- Before making its two recursive calls, Quicksort compares every element of its input array to the pivot, taking time $\Theta(n)$.
- The worst case for runtime occurs when the pivot is the smallest or largest element of the array.
- In this case, the array is partitioned into an array of size $n - 1$ and an array of size 0.

Quicksort Runtime - Direct Proof

Claim: Quicksort runs in time $O(n^2)$ in the worst case.

- Before making its two recursive calls, Quicksort compares every element of its input array to the pivot, taking time $\Theta(n)$.
- The worst case for runtime occurs when the pivot is the smallest or largest element of the array.
- In this case, the array is partitioned into an array of size $n-1$ and an array of size 0.
- This gives a recurrence $T(n) = T(n-1) + \Theta(n)$, which has solution $T(n) = \Theta(n^2)$.

Insertion Sort Review

Algorithm Insertion Sort
Input: array A of length n

```
for i ← 2 to n do
    j ← i
    while j > 1 and A[j] < A[j − 1] do
        Swap A[j] and A[j − 1]
        j ← j − 1
    end while
end for
```

Proof by Loop Invariant

Proof by loop invariant is a proof technique that establishes some useful property that is true throughout every loop of an iterative algorithm.

Proof by Loop Invariant

Proof by loop invariant is a proof technique that establishes some useful property that is true throughout every loop of an iterative algorithm.

- Initialization: the property is true at the start of the loop.

Proof by Loop Invariant

Proof by loop invariant is a proof technique that establishes some useful property that is true throughout every loop of an iterative algorithm.

- Initialization: the property is true at the start of the loop.
- Maintenance: if the property is true at the beginning of an iteration, it is true at beginning of the next iteration.

Proof by Loop Invariant

Proof by loop invariant is a proof technique that establishes some useful property that is true throughout every loop of an iterative algorithm.

- Initialization: the property is true at the start of the loop.
- Maintenance: if the property is true at the beginning of an iteration, it is true at beginning of the next iteration.
- Termination: when the loop terminates, the invariant holds and shows that the algorithm is correct.

Correctness of Insertion Sort - Proof by Loop Invariant

Claim: Given an array $\boldsymbol{A}$ of length $\boldsymbol{n}$, InsertionSort($\boldsymbol{A}$) returns an array with all elements of $\boldsymbol{A}$ sorted from least to greatest.

Correctness of Insertion Sort - Proof by Loop Invariant

Claim: Given an array A of length n, InsertionSort(A) returns an array with all elements of A sorted from least to greatest.

Proof:

- Loop invariant: at iteration i, $A[1, i-1]$ contains all elements of the original input array $A[1, i-1]$, and is sorted.

Correctness of Insertion Sort - Proof by Loop Invariant

Claim: Given an array A of length n, InsertionSort(A) returns an array with all elements of A sorted from least to greatest.
Proof:

- Loop invariant: at iteration i, $A[1, i-1]$ contains all elements of the original input array $A[1, i-1]$, and is sorted.
- Initialization - At the beginning of the first iteration $i = 2$, $A[1]$ is sorted.
- Maintenance - In a single iteration, element $A[i]$ of the input Array is moved to the left until it is no longer smaller than the element to its left, therefore at the beginning of the next iteration, $A[1, i]$ is sorted and contains exactly the same elements as $A[1, i]$ from the original input array.

Correctness of Insertion Sort - Proof by Loop Invariant

Claim: Given an array A of length n, InsertionSort(A) returns an array with all elements of A sorted from least to greatest.
Proof:

- Loop invariant: at iteration i, $A[1, i-1]$ contains all elements of the original input array $A[1, i-1]$, and is sorted.
- Initialization - At the beginning of the first iteration $i = 2$, $A[1]$ is sorted.
- Maintenance - In a single iteration, element $A[i]$ of the input Array is moved to the left until it is no longer smaller than the element to its left, therefore at the beginning of the next iteration, $A[1, i]$ is sorted and contains exactly the same elements as $A[1, i]$ from the original input array.
- Termination - When the loop terminates, $i = n$ and therefore $A[1, n]$ is sorted and contains exactly the same elements as $A[1, n]$ from the original input array. Therefore the original input array has been sorted.

Proof by Contrapositive

Proof by contrapositive is a proof technique for conditional statements. That is, statements of the form "If $\boldsymbol{A}$, then $\boldsymbol{B}$."

Proof by Contrapositive

Proof by contrapositive is a proof technique for conditional statements. That is, statements of the form "If $\boldsymbol{A}$, then $\boldsymbol{B}$."

It relies on the fact that $\boldsymbol{A} \Rightarrow \boldsymbol{B}$ is logically equivalent to $\neg \boldsymbol{B} \Rightarrow \neg \boldsymbol{A}$.

Proof by Contrapositive

Proof by contrapositive is a proof technique for conditional statements. That is, statements of the form "If $\boldsymbol{A}$, then $\boldsymbol{B}$."

It relies on the fact that $\boldsymbol{A} \Rightarrow \boldsymbol{B}$ is logically equivalent to $\neg \boldsymbol{B} \Rightarrow \neg \boldsymbol{A}$.

To prove $\boldsymbol{A} \Rightarrow \boldsymbol{B}$ by contrapositive, we show that if the negation of the conclusion is true ($\neg \boldsymbol{B}$), then the negation of the hypothesis is true ($\neg \boldsymbol{A}$).

Correctness of (one iteration of) Insertion Sort - Proof by Contrapositive

Claim: If the ith iteration of the inner WHILE loop terminates with counter value j for $j > 1$, then element $A[i]$ of the original input array is greater than or equal to $A[j-1]$.

Correctness of (one iteration of) Insertion Sort - Proof by Contrapositive

Claim: If the ith iteration of the inner WHILE loop terminates with counter value j for $j > 1$, then element $A[i]$ of the original input array is greater than or equal to $A[j-1]$.

- A: The ith iteration of the inner WHILE loop terminates with counter value j for $j > 1$

Correctness of (one iteration of) Insertion Sort - Proof by Contrapositive

Claim: If the ith iteration of the inner WHILE loop terminates with counter value j for $j > 1$, then element $A[i]$ of the original input array is greater than or equal to $A[j-1]$.

- A: The ith iteration of the inner WHILE loop terminates with counter value j for $j > 1$
- B: Element $A[i]$ of the original input array is greater than or equal to $A[j-1]$

Correctness of (one iteration of) Insertion Sort - Proof by Contrapositive

Claim: If the ith iteration of the inner WHILE loop terminates with counter value j for $j > 1$, then element $A[i]$ of the original input array is greater than or equal to $A[j-1]$.

- A: The ith iteration of the inner WHILE loop terminates with counter value j for $j > 1$
- B: Element $A[i]$ of the original input array is greater than or equal to $A[j-1]$
- Want to prove $A \Rightarrow B$

Correctness of (one iteration of) Insertion Sort - Proof by Contrapositive

Claim: If the ith iteration of the inner WHILE loop terminates with counter value j for $j > 1$, then element $A[i]$ of the original input array is greater than or equal to $A[j-1]$.

- A: The ith iteration of the inner WHILE loop terminates with counter value j for $j > 1$
- B: Element $A[i]$ of the original input array is greater than or equal to $A[j-1]$
- Want to prove $A \Rightarrow B$
- So will argue that if element $A[i]$ of the original input array is less than $A[j-1]$, then the ith iteration of the inner WHILE loop will not terminate with counter value j for $j > 1$.

Algorithm Insertion Sort
Input: array A of length n

```
for i ← 2 to n do
    j ← i
    while j > 1 and A[j] < A[j − 1] do
        Swap A[j] and A[j − 1]
        j ← j − 1
    end while
end for
```

Correctness of (one iteration of) Insertion Sort - Proof by Contrapositive

Claim: If element $A[i]$ of the original input array is less than $A[j-1]$, then the ith iteration of the inner WHILE loop will not terminate with counter value j for $j > 1$.

Correctness of (one iteration of) Insertion Sort - Proof by Contrapositive

Claim: If element $\boldsymbol{A}[\boldsymbol{i}]$ of the original input array is less than $\boldsymbol{A}[\boldsymbol{j}-\mathbf{1}]$, then the $\boldsymbol{i}$th iteration of the inner WHILE loop will not terminate with counter value $\boldsymbol{j}$ for $\boldsymbol{j} > \mathbf{1}$.

- In order for the loop to terminate at counter value $\boldsymbol{j} > \mathbf{1}$, it must hold that $\boldsymbol{A}[\boldsymbol{j}] \geq \boldsymbol{A}[\boldsymbol{j}-\mathbf{1}]$.

Correctness of (one iteration of) Insertion Sort - Proof by Contrapositive

Claim: If element $A[i]$ of the original input array is less than $A[j-1]$, then the ith iteration of the inner WHILE loop will not terminate with counter value j for $j > 1$.

- In order for the loop to terminate at counter value $j > 1$, it must hold that $A[j] \geq A[j-1]$.
- Note that inside the WHILE loop, $A[j] = A[i]$ of the original input array. Therefore if $A[i] = A[j] < A[j-1]$, the loop will not terminate with counter value j.

Algorithm Insertion Sort
Input: array A of length n

```
for i ← 2 to n do
    j ← i
    while j > 1 and A[j] < A[j − 1] do
        Swap A[j] and A[j − 1]
        j ← j − 1
    end while
end for
```