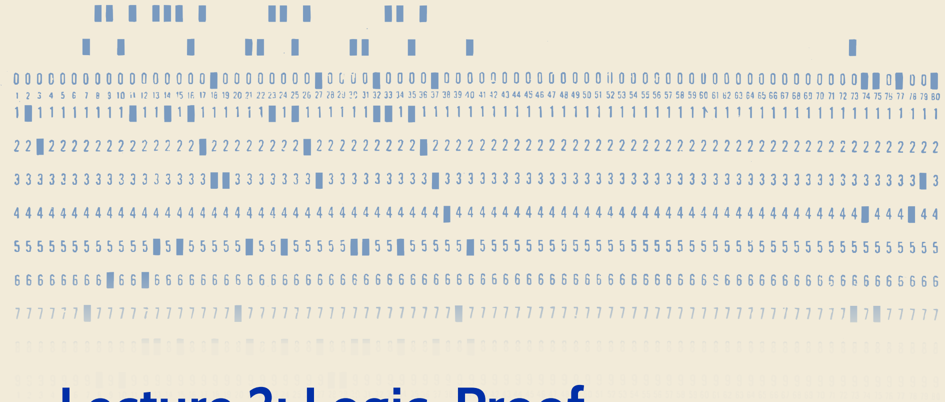
COMP526: Efficient Algorithms

- Your Instructor:
  Will Rosenbaum
  George Holt 2.16B
  `w.rosenbaum@liverpool.ac.uk`
- Module Website:
  `willrosenbaum.com/teaching/2024f-comp-526`
  - The authoritative source for module information about COMP526.
- Poll Everywhere: `pollev.com/comp526`
  - Used for in-class participation and attendance
  - Use U of L credentials to log in
- CampusWire: `https://campuswire.com/p/GBB00CD7A`
  - Invite code: **4796**
  - Used for announcements and asynchronous discussion (outside of lecture)



Lecture 2: Logic, Proof Techniques & Induction

COMP526: Efficient Algorithms

Will Rosenbaum
University of Liverpool

Updated: October 8, 2024

Announcements

1. First quiz released tomorrow, due Friday
    - Administered through Canvas
    - One question, multiple choice
    - 20 minutes
    - Covers basic (today's lecture, this week's tutorial, posted notes)
2. Programming Assignment 1 released next week
    - Due 13 November
3. Participation Confirmation: Pending

Meeting Goals

- Motivate the need for proofs in CS
- Introduce the mechanics of propositional and predicate logic
- Describe proof techniques and applications
- Introduce mathematical induction
- Analyze algorithm correctness with loop invariants

A Scenario

The Setup:

- You are contracted by a (virtual) casino to audit their code
- The casino spent millions of £££ developing an AI to play their card games
- They believe their AI is *unbeatable*
  - on average the casino will win
  - this ensures their business is profitable
- The gaming AI company even provided a *mathematical proof* that their strategies will win on average

Photo Credit: OpenAI DALL·E

A Scenario

The Setup:

- You are contracted by a (virtual) casino to audit their code
- The casino spent millions of £££ developing an AI to play their card games
- They believe their AI is *unbeatable*
  - on average the casino will win
  - this ensures their business is profitable
- The gaming AI company even provided a *mathematical proof* that their strategies will win on average

Photo Credit: OpenAI DALL·E

Unfortunately the casino found a group of users that were consistently beating the AI and winning a significant amount of money. Hence, they called in the experts: you!

Shuffling Cards

You find that the casino was using the following procedure to shuffle a (virtual) deck of cards:

procedure SHUFFLE(A, n) ▷ shuffle a deck A of n cards
  for $i = 1, \ldots, n$ **do** ▷ iterate over indices
    $j \leftarrow$ RANDOM($1, n$) ▷ pick random index
    SWAP(A, i, j) ▷ swap values at i and j
  end for
end procedure

Shuffling Cards

You find that the casino was using the following procedure to shuffle a (virtual) deck of cards:

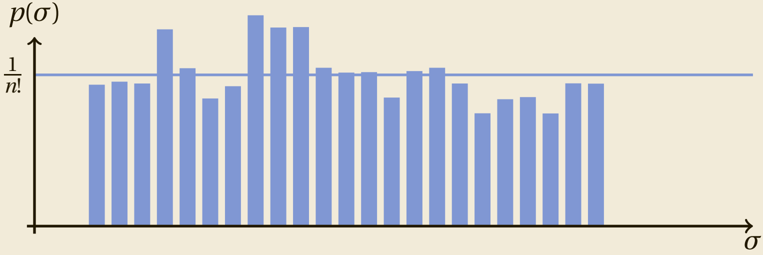
procedure SHUFFLE(A, n) $\triangleright$ shuffle a deck A of n cards
  for $i = 1, \ldots, n$ **do** $\triangleright$ iterate over indices
    $j \leftarrow$ RANDOM($1, n$) $\triangleright$ pick random index
    SWAP(A, i, j) $\triangleright$ swap values at i and j
  end for
end procedure

PollEverywhere Question

- I think SHUFFLE is fine.
- SHUFFLE is maybe reasonable?
- SHUFFLE is definitely problematic.
- I do not understand SHUFFLE.

pollev.com/comp526

Testing Shuffle



What Gives?

What is the problem here?

Two Challenges

Challenge 1

Give a *simple* argument that SHUFFLE could not possibly generate all permutations of cards with equal probability.

Challenge 2

Argue that the modified shuffle algorithm on the right does generate a uniformly random shuffling of the elements of A.

procedure FYKSHUFFLE(A, n)
    for $i = 1, \ldots, n$ **do**
        $j \leftarrow$ RANDOM$(1, i)$
        SWAP(A, i, j)
    end for
end procedure

Who is to Blame?

A Question

Having found a problem in the SHUFFLE subroutine, who is at fault? The casino? The AI consultant?

Who is to Blame?

A Question

Having found a problem in the SHUFFLE subroutine, who is at fault? The casino? The AI consultant?

The Moral

In order to make trustworthy conclusions about algorithms we must:

1. Assert our assumptions about the system
2. State our (desired) conclusions precisely
3. Argue that our conclusions follow logically from our assumptions

Goal: any system that fulfills our assumptions will also satisfy our conclusions.

Roadmap

1. Formal Reasoning through Logic (today)
    - Basic language of logic: propositions and predicates
    - Proof techniques
    - Mathematical induction
2. Our Computational Model (Thursday)
3. Algorithms (Rest of the Semester)

Propositions, Connectives, and Formulae

- A (logical) proposition is a declarative sentence that can take the value true(T) or false(F)
  - $P =$ "it is raining"
  - $Q =$ "I am wearing a jacket"
  - $R =$ "I am soaked"

Propositions, Connectives, and Formulae

- A (logical) proposition is a declarative sentence that can take the value true(T) or false(F)
  - P = "it is raining"
  - Q = "I am wearing a jacket"
  - R = "I am soaked"
- logical connectives allow us to combine propositions into more complex statements
  - $\wedge$ = "and"
  - $\vee$ = "or"
  - $\neg$ = "not"
  - $\implies$ = "implies" or "if… then"
  - $\iff$ = "if and only if"

Propositions, Connectives, and Formulae

- A (logical) proposition is a declarative sentence that can take the value true(T) or false(F)
  - P = "it is raining"
  - Q = "I am wearing a jacket"
  - R = "I am soaked"
- logical connectives allow us to combine propositions into more complex statements
  - $\wedge$ = "and"
  - $\vee$ = "or"
  - $\neg$ = "not"
  - $\implies$ = "implies" or "if…then"
  - $\iff$ = "if and only if"
- A (Boolean) formula is a statement composed of propositions and logical quantifiers:

$$\varphi = P \wedge \neg Q \implies R$$

Truth Tables

A truth table expresses the values of a formula φ for all possible input propositional values

P	Q	$P \land Q$	$P \lor Q$	$\neg P$	$P \implies Q$	$P \iff Q$
T	T	T	T	F	T	T
T	F	F	T	F	F	F
F	T	F	T	T	T	F
F	F	F	F	T	T	T

We can think of the truth table as *defining* the logical connectives.

Satisfiability

A formula is…

… satisfiable if there is an assignment of truth values to its constituent propositions such that φ evaluates to T.

Satisfiability

A formula is…

… satisfiable if there is an assignment of truth values to its constituent propositions such that φ evaluates to T.

… a contradiction if *no* assignment of truth values makes φ evaluate to T.

Satisfiability

A formula is…

… satisfiable if there is an assignment of truth values to its constituent propositions such that φ evaluates to T.

… a contradiction if *no* assignment of truth values makes φ evaluate to T.

… a tautology if *every* assignment of truth values makes φ evaluate to T.

Satisfiability

A formula is…

- … satisfiable if there is an assignment of truth values to its constituent propositions such that φ evaluates to T.
- … a contradiction if *no* assignment of truth values makes φ evaluate to T.
- … a tautology if *every* assignment of truth values makes φ evaluate to T.

PollEverywhere Question

Which of the following expressions is satisfiable, a contradiction, and a tautology?

1. $P \implies P \vee Q$
2. $(P \wedge Q) \wedge (P \implies \neg Q)$
3. $(P \wedge \neg Q) \vee (\neg P \wedge Q)$

pollev.com/comp526

Logical Equivalence

We say that logical formulae φ and ψ are logically equivalent and write $\varphi \equiv \psi$ if $\varphi \iff \psi$ is a tautology.

Logically Equivalent to Implication

The following expressions are logically equivalent

1. $P \implies Q$
2. $\neg(P \wedge \neg Q)$
3. $\neg P \vee Q$

Logical Equivalence

We say that logical formulae φ and ψ are logically equivalent and write $\varphi \equiv \psi$ if $\varphi \iff \psi$ is a tautology.

Logically Equivalent to Implication

The following expressions are logically equivalent

1. $P \implies Q$
2. $\neg(P \wedge \neg Q)$
3. $\neg P \vee Q$

More Logical Equivalence

The following expressions are also logically equivalent

1. $P \iff Q$
2. $(P \implies Q) \wedge (Q \implies P)$

Logical Equivalence

We say that logical formulae φ and ψ are logically equivalent and write $\varphi \equiv \psi$ if $\varphi \iff \psi$ is a tautology.

Logically Equivalent to Implication

The following expressions are logically equivalent

1. $P \implies Q$
2. $\neg(P \wedge \neg Q)$
3. $\neg P \vee Q$

More Logical Equivalence

The following expressions are also logically equivalent

1. $P \iff Q$
2. $(P \implies Q) \wedge (Q \implies P)$

Note. Two formulae are logically equivalent precisely when they have the same truth table.

- The two formulas agree on all inputs

Some Important Equivalences

Double Negation

$$P \equiv \neg\neg P$$

DeMorgan's Laws

$$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$$
$$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$$

Some Important Equivalences

Double Negation

$$P \equiv \neg\neg P$$

DeMorgan's Laws

$$\neg(P \land Q) \equiv \neg P \lor \neg Q$$
$$\neg(P \lor Q) \equiv \neg P \land \neg Q$$

Exercise

Write a simpler expression equivalent to $\neg(P \implies Q)$.

Predicates and Quantifiers

A logical predicate P is a function from a domain U to the values $\{T, F\}$:

- For each $x \in U$, $P(x)$ is a proposition

Examples of Predicates

1. $U = \mathbf{N} = \{0, 1, 2, \ldots\}$, $P(x) =$ "x is an even number"
2. $U =$ days of the year, $P =$ "it rained in Liverpool on the day"
3. $U =$ set of inputs for an algorithm, $P =$ algorithm outputs satisfying some property

Predicates and Quantifiers

A logical predicate P is a function from a domain U to the values $\{T, F\}$:

- For each $x \in U$, $P(x)$ is a proposition

Examples of Predicates

1. $U = \mathbf{N} = \{0, 1, 2, \ldots\}$, $P(x) =$ "x is an even number"
2. $U =$ days of the year, $P =$ "it rained in Liverpool on the day"
3. $U =$ set of inputs for an algorithm, $P =$ algorithm outputs satisfying some property

Predicates can be quantified to yield new propositions:

- universal quantifier $\forall x P(x)$: "for all x, $P(x)$"
- existential quantifier $\exists x P(x)$: "there exists x such that $P(x)$"

Negating Quantified Expressions

Quantifiers can be negated as follows:

- $\neg(\forall x \varphi(x)) \iff \exists x \neg\varphi(x)$
- $\neg(\exists x \varphi(x)) \iff \forall x \neg\varphi(x)$

Negating Quantified Expressions

Quantifiers can be negated as follows:

- $\neg(\forall x \varphi(x)) \iff \exists x \neg\varphi(x)$
- $\neg(\exists x \varphi(x)) \iff \forall x \neg\varphi(x)$

Unbounded Sets of Numbers

Suppose U is a set of numbers. Consider the formula $\varphi = \forall x \exists y [y > x]$.

- How do you interpret φ?
- What about its negation $\neg\varphi$?

Logical Equivalence and Proof Techniques

Recall: our main goal is to show that

$$\{\text{assumptions}\} \implies \{\text{conclusions}\}$$

Proof techniques are *logical strategies* for deriving logical inferences.

Logical Equivalence and Proof Techniques

Recall: our main goal is to show that

$$\{\text{assumptions}\} \implies \{\text{conclusions}\}$$

Proof techniques are *logical strategies* for deriving logical inferences.

Techniques for proving $P \implies Q$

Direct Proof assume P and derive Q

Logical Equivalence and Proof Techniques

Recall: our main goal is to show that

$$\{\text{assumptions}\} \implies \{\text{conclusions}\}$$

Proof techniques are *logical strategies* for deriving logical inferences.

Techniques for proving $P \implies Q$

Direct Proof assume P and derive Q

Proof by Contraposition $(P \implies Q) \equiv (\neg Q \implies \neg P)$

Logical Equivalence and Proof Techniques

Recall: our main goal is to show that

$$\{\text{assumptions}\} \implies \{\text{conclusions}\}$$

Proof techniques are *logical strategies* for deriving logical inferences.

Techniques for proving $P \implies Q$

Direct Proof assume P and derive Q

Proof by Contraposition $(P \implies Q) \equiv (\neg Q \implies \neg P)$

Proof by Contradiction $(P \implies Q) \equiv ((P \wedge \neg Q) \implies \text{false})$

Logical Equivalence and Proof Techniques

Recall: our main goal is to show that

$$\{\text{assumptions}\} \implies \{\text{conclusions}\}$$

Proof techniques are *logical strategies* for deriving logical inferences.

Techniques for proving $P \implies Q$

Direct Proof assume P and derive Q

Proof by Contraposition $(P \implies Q) \equiv (\neg Q \implies \neg P)$

Proof by Contradiction $(P \implies Q) \equiv ((P \wedge \neg Q) \implies \text{false})$

Proof by Exhaustion $(P \implies Q) \equiv (P \wedge A \implies Q) \wedge (P \wedge \neg A \implies Q)$ (A is any predicate)

Logical Equivalence and Proof Techniques

Recall: our main goal is to show that

$$\{\text{assumptions}\} \implies \{\text{conclusions}\}$$

Proof techniques are *logical strategies* for deriving logical inferences.

Techniques for proving $P \implies Q$

Direct Proof assume P and derive Q

Proof by Contraposition $(P \implies Q) \equiv (\neg Q \implies \neg P)$

Proof by Contradiction $(P \implies Q) \equiv ((P \wedge \neg Q) \implies \text{false})$

Proof by Exhaustion $(P \implies Q) \equiv (P \wedge A \implies Q) \wedge (P \wedge \neg A \implies Q)$ (A is any predicate)

Exercise

Show that all of the above are logical equivalences.

Example: Direct Proof

Proposition

Suppose n is a natural number. If n^2 is divisible by 4, then n is divisible by 2.

Direct proof.

- Suppose n^2 is divisible by 4: $n^2 = 4N$ for some natural number N.

Example: Direct Proof

Proposition

Suppose n is a natural number. If n^2 is divisible by 4, then n is divisible by 2.

Direct proof.

- Suppose n^2 is divisible by 4: $n^2 = 4N$ for some natural number N.
- Since n^2 is divisible by 4, it is also divisible by 2. In particular $n^2 = 2N'$ with $N' = 2N$.
- Since 2 is a prime number and $N = n \cdot n$ is divisible by n is divisible by 2.
    - Fact about prime numbers: if a prime number p divides a product $a \cdot b$, then p divides a or p divides b.
- Since n is divisible by 2, n is an even number.

□

Example: Proof by Contraposition

Proposition

Suppose n is a natural number. If n^2 is divisible by 4, then n is divisible by 2.

Proof by Contraposition.

- Suppose n is not even, i.e., n is odd.

Example: Proof by Contraposition

Proposition

Suppose n is a natural number. If n^2 is divisible by 4, then n is divisible by 2.

Proof by Contraposition.

- Suppose n is not even, i.e., n is odd.
- Write $n = 2k + 1$ for some k.
- Then $n^2 = (2k+1)^2 = 4k^2 + 4k + 1$.
- Therefore, n^2 is not divisible by 4.

$\square$

Example: Proof by Contradiction

Proposition

Suppose n is a natural number. If n^2 is divisible by 4, then n is divisible by 2.

Proof by Contradiction.

- Suppose the statement is false–i.e., that n^2 is divisible by 4 and n is not even.

Example: Proof by Contradiction

Proposition

Suppose n is a natural number. If n^2 is divisible by 4, then n is divisible by 2.

Proof by Contradiction.

- Suppose the statement is false–i.e., that n^2 is divisible by 4 and n is not even.
- Since n is not even, we can write $n = 2k+1$.
- Therefore, $n^2 = (2k+1)^2 = 4k^2+4k+1$.
- However, $4k^2+4k+1 = n^2$ is not divisible by 4, which contradicts the hypothesis that n^2 was divisible by 4.

$\square$

Example: Proof by Exhaustion

Proposition

Suppose n is a natural number. If n^2 is divisible by 4, then n is divisible by 2.

Proof by Exhaustion.

Use the case C = "n is even."

Example: Proof by Exhaustion

Proposition

Suppose n is a natural number. If n^2 is divisible by 4, then n is divisible by 2.

Proof by Exhaustion.

Use the case $C =$ "n is even."

Case 1 Suppose n is even, i.e., $n = 2k$.

- Then $n^2 = (2k)^2 = 4k^2$.
- Therefore n^2 is divisible by 4
- Since n^2 is divisible by 4 and n is even, the conclusion holds.

Example: Proof by Exhaustion

Proposition

Suppose n is a natural number. If n^2 is divisible by 4, then n is divisible by 2.

Proof by Exhaustion.

Use the case $C =$ "n is even."

Case 1 Suppose n is even, i.e., $n = 2k$.

- Then $n^2 = (2k)^2 = 4k^2$.
- Therefore n^2 is divisible by 4
- Since n^2 is divisible by 4 and n is even, the conclusion holds.

Case 2 Suppose n is not even, i.e., $n = 2k + 1$.

- Then $n^2 = (2k+1)^2 = 4k^2 + 4k + 1$.
- Therefore n^2 is not divisible by 4.
- Since n^2 is not divisible by 4, the conclusion holds.

$\square$

Evaluating the Proofs

PollEverywhere Question

Which proof seemed simplest/most natural to you?

- Direct Proof
- Proof by Contraposition
- Proof by Contradiction
- Proof by Exhaustion

pollev.com/comp526

Proving the Infinite

So Far

- *Generic* techniques/strategies for proofs
- Not specific to any particular application domain

Proofs for Algorithms

- Correctness: "For every input x, the output of an algorithm A on input x satisfies {some specification}."
- Running time: "For every input x, A performs at most {some number} operations on input x"

Proving the Infinite

So Far

- *Generic* techniques/strategies for proofs
- Not specific to any particular application domain

Proofs for Algorithms

- Correctness: "For every input x, the output of an algorithm A on input x satisfies {some specification}."
- Running time: "For every input x, A performs at most {some number} operations on input x"

Two Features

1. We must reason about infinite sets of events (i.e., all possible inputs).
2. We must infer globally correct behavior by analyzing individual local steps of an algorithm.

Mathematical Induction

The Principle of Mathematical Induction

Let P be a predicate over the natural numbers $\mathbf{N} = \{0, 1, 2, \ldots\}$. Suppose P satisfies

- Base case: $P(0)$ is true.
- Inductive step: For every $i \in \mathbf{N}$, $P(i) \implies P(i+1)$.

Then for every $n \in \mathbf{N}$, $P(n)$ is true. In strictly symbolic notation:

$$(P(0)) \wedge (\forall i [P(i) \implies P(i+1)]) \implies \forall n P(n).$$

Mathematical Induction

The Principle of Mathematical Induction

Let P be a predicate over the natural numbers $\mathbf{N} = \{0, 1, 2, \ldots\}$. Suppose P satisfies

- Base case: $P(0)$ is true.
- Inductive step: For every $i \in \mathbf{N}$, $P(i) \implies P(i+1)$.

Then for every $n \in \mathbf{N}$, $P(n)$ is true. In strictly symbolic notation:

$$(P(0)) \wedge (\forall i [P(i) \implies P(i+1)]) \implies \forall n P(n).$$

Moral Justification:

Loop Invariants

Loop Invariants

Given an algorithm A containing a loop, a loop invariant is a predicate P on the iterations of the loop such that for each iteration i, $P(i)$ is satisfied at the end of the i-th iteration of the loop.

Loop Invariants

Loop Invariants

Given an algorithm A containing a loop, a loop invariant is a predicate P on the iterations of the loop such that for each iteration i, $P(i)$ is satisfied at the end of the i-th iteration of the loop.

An Uninteresting Example

Consider the following procedure

```
procedure COUNT(n)                ▷ count to n
    t ← 0
    for i = 1,...,n do            ▷ iterate over indices
        t ← t + 1
    end for
    return t
end procedure
```

Loop Invariants

Loop Invariants

Given an algorithm A containing a loop, a loop invariant is a predicate P on the iterations of the loop such that for each iteration i, $P(i)$ is satisfied at the end of the i-th iteration of the loop.

An Uninteresting Example

Consider the following procedure

```
procedure COUNT(n)                ▷ count to n
    t ← 0
    for i = 1,...,n do            ▷ iterate over indices
        t ← t + 1
    end for
    return t
end procedure
```

Loop Invariant:
After iteration i, t stores the value i.

Loop Invariants

Loop Invariants

Given an algorithm A containing a loop, a loop invariant is a predicate P on the iterations of the loop such that for each iteration i, $P(i)$ is satisfied at the end of the i-th iteration of the loop.

An Uninteresting Example

Consider the following procedure

1: **procedure** COUNT(n) ▷ count to n
2: $t \leftarrow 0$
3: **for** $i = 1, \ldots, n$ **do** ▷ iterate over indices
4: $t \leftarrow t + 1$
5: **end for**
6: **return** t
7: **end procedure**

Loop Invariant: After iteration i, t stores the value i.

Proof.

Induct on t. Base case: t initialized to 0. Inductive step: clear. □

A More Interesting Example

Consider the following subroutine:

procedure MinIndex((a, i, k)) ▷ Find the index of the minimum value stored in array a between indices i and k.

  $m \leftarrow i$

  for $j = i, i+1, \ldots, k$ **do**

    if $a[j] > a[m]$ **then**

      $m \leftarrow j$

    end if

  end for

  return m

end procedure

PollEverywhere Question

What loop invariant does the loop in MinIndex satisfy that will help us analyze its behavior?

pollev.com/comp526

A More Interesting Example

```
procedure MinIndex((a, i, k))    ▷ Find the index of the minimum value stored in array a between indices i and k.
    m ← i
    for j = i, i + 1, ..., k do
        if a[j] < a[m] then
            m ← j
        end if
    end for
    return m
end procedure
```

A More Interesting Example

procedure MININDEX((a, i, k)) ▷ Find the index of the minimum value stored in array a between indices i and k.
    $m \leftarrow i$
    for $j = i, i+1, \ldots, k$ **do**
        if $a[j] < a[m]$ **then**
            $m \leftarrow j$
        end if
    end for
    return m
end procedure

Loop Invariant

After iteration j, m stores the index of the minimum value of a between indices i and j.

A More Interesting Example

procedure $\textsc{MinIndex}((a, i, k))$ ▷ Find the index of the minimum value stored in array a between indices i and k.

  $m \leftarrow i$

  for $j = i, i+1, \ldots, k$ **do**

    if $a[j] < a[m]$ **then**

      $m \leftarrow j$

    end if

  end for

  return m

end procedure

Loop Invariant

After iteration j, m stores the index of the minimum value of a between indices i and j.

Proof.

Induct on j

- Base case: $j = i$.
- Inductive step: $j \implies j+1$

$\square$

Further Application

Consider the following algorithm that uses MinIndex as a subroutine:

procedure SelectionSort(a, n) $\triangleright$ Sort the array a of size n
    for $i = 1, 2, \ldots, n$ **do**
        $j \leftarrow$ MinIndex(a, i, n)
        Swap(a, i, j)
    end for
end procedure

Exercise (Tutorials)

Show that SelectionSort correctly sorts any array a of length n. Specifically:

- Find a suitable loop invariant satisfied by SelectionSort
- Prove your loop invariant holds (by induction)
- Argue that your loop invariant implies the final array is sorted

Induction and Recursion

Induction is essential in reasoning about *recursively defined* methods.

A Recursive Method

```
procedure MYSTERY(n)
    if n = 1 then
        return 1
    end if
    return 2n − 1 + MYSTERY(n − 1)
end procedure
```

PollEverywhereQuestion

What is the output of MYSTERY(5)?

pollev.com/comp526

Analysis of a Mystery

```
procedure Mystery(n)
    if n = 1 then
        return 1
    end if
    return 2n − 1 + Mystery(n − 1)
end procedure
```

Analysis of a Mystery

```
procedure MYSTERY(n)
    if n = 1 then
        return 1
    end if
    return 2n − 1 + MYSTERY(n − 1)
end procedure
```

Claim

For all n, MYSTERY(n) returns the value n^2.

Analysis of a Mystery

procedure $\textsc{Mystery}(n)$
  if $n = 1$ **then**
    return 1
  end if
  return $2n - 1 + \textsc{Mystery}(n-1)$
end procedure

Claim

For all n, $\textsc{Mystery}(n)$ returns the value n^2.

Proof.

Induction on n. Base Case: $n = 1$.
Inductive step: Suppose $\textsc{Mystery}(n) = n^2$. Then

$$\begin{aligned}\textsc{Mystery}(n+1) &= 2n + 1 \\ &\quad + \textsc{Mystery}(n) \\ &= 2n + 1 + n^2 \\ &= (n+1)^2.\end{aligned}$$

$\square$

Next Time

- Machines and Models
  - What can computers do?
  - And how efficiently?
- Asymptotic Notation

Scratch Notes