Induction and Recursion Loop Invariants and Program Correctness

Rachel Fishman

Koffman 7

Epp 6.1-6.2, 7.1-7.2, 5.2-5.3, 5.5

7) Recursion

- recursive algorithm: the original problem is split into one or more simpler versions of itself.
- General Recursive Algorithm:

```
1. if the problem can be solved directly for the current value of n
2.          Solve it
3. else    recursively apply the algorithm to one or more problems
   involving smaller values of n
4.          combine the solutions to the smaller problems to get the
   solution to the original
```

Steps to Design a Recursive Algorithm

- Characteristics of Recursive Solution:
  - There must be at least one case (base care), for a small value of n, that can be solved directly
  - A problem of a given size (say, n) can be split into one or more clammer versions of the same problem (the recursive case)
- Therefore:
  - Recognize the base case and provide a solution
  - Devise a strategy to split the problem into smaller versions of itself. Each recursive case must make progress toward the base case.
  - Combine the solutions to the smaller problems in such a way that each larger problem is solved correctly.

Proving Recursive Function is Correct

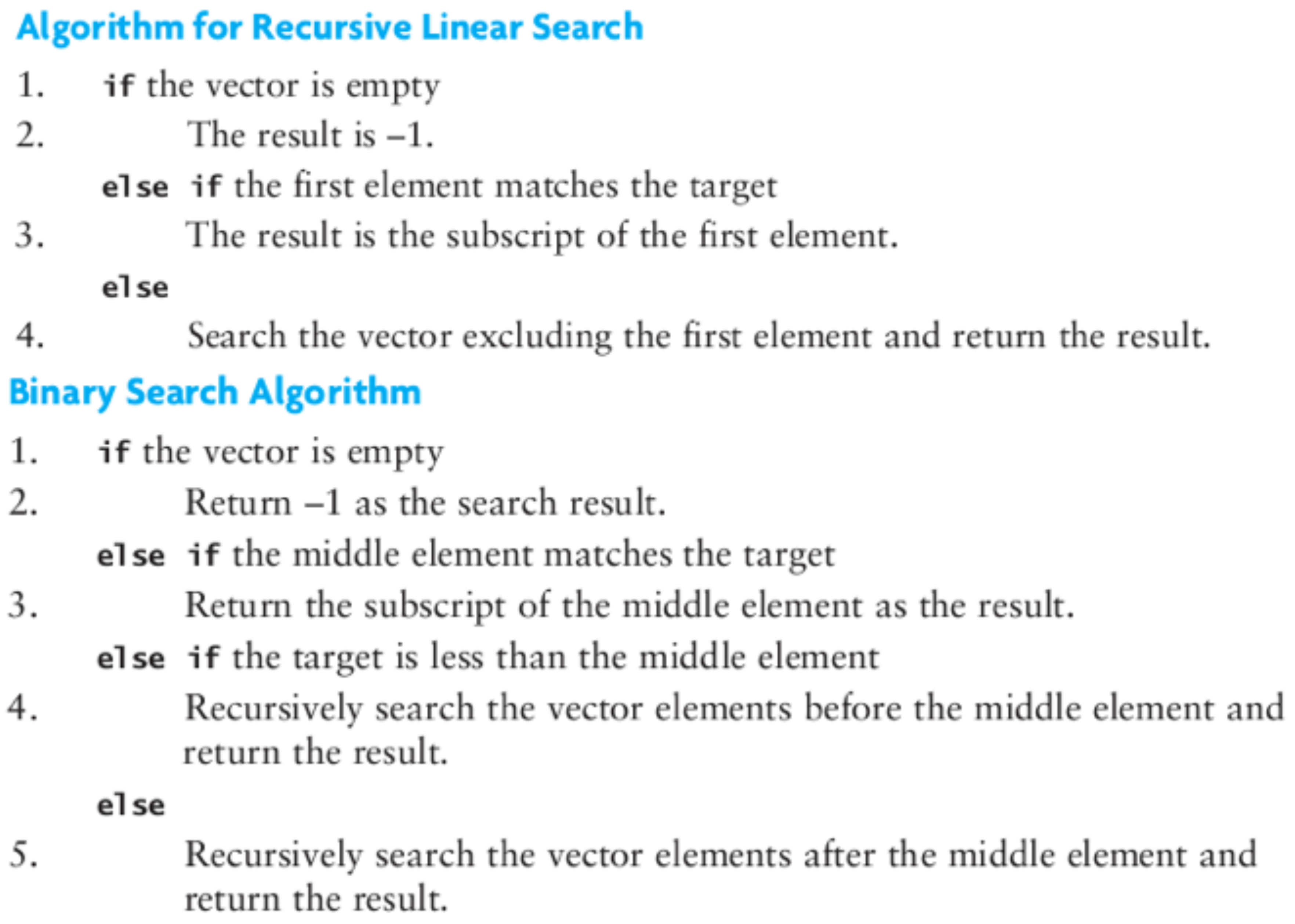
- Must verify by proof by induction
  - Prove that the theorem is true for the base case of (usually) n=0 or n=1.
  - Show that if the theorem is assumed true for n, then it must be true for n+1

Recursion v. Iteration

- Iteration
  - a loop repetition condition in the loop header determines whether we repeat the loop body or exit the loop. Repeat the the loop body while the repetition condition is true.
- Recursion
  - Test for base case. stop the recursion when the base case is reached and we execute the function body again when the condition is false.
  - easier to read and write
- Tail Recursion: single recursive call in the last line
- Efficiency: both are O(n) because the number of loop repetitions or recursive calls increase linearly with n.
  - iterative probably faster because the overhead for loop repetition
  - do recursive over iterative because easy to read and debug easier than such a small time difference
- Storage
  - recursive requires more memory than iterative because the need to save local variables and parameters on a stack

Recursive Search

- Vector search: simplest way to search a vector; O(n) linear
- Binary search: only on sorted arrays. works like linear search where the stopping cases are when the vector is empty, when the vector element being examined matches the target; O(log n) because eliminate half the elements with each call



Algorithm for Recursive Linear Search

```
1.  if the vector is empty
2.      The result is –1.
    else if the first element matches the target
3.      The result is the subscript of the first element.
    else
4.      Search the vector excluding the first element and return the result.
```

Binary Search Algorithm

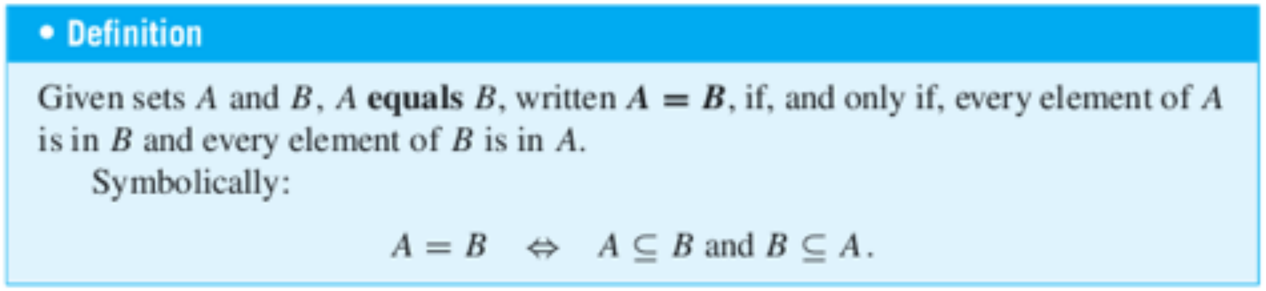
```
1.  if the vector is empty
2.      Return –1 as the search result.
    else if the middle element matches the target
3.      Return the subscript of the middle element as the result.
    else if the target is less than the middle element
4.      Recursively search the vector elements before the middle element and
        return the result.
    else
5.      Recursively search the vector elements after the middle element and
        return the result.
```

Review

- To prove that a recursive algorithm is correct
    - verify that the base case is recognized and solved correctly
    - verify that each recursive case makes progress toward the base case
    - verify that if all smaller problems are solved correctly, then the original problem must also be solved correctly
- operating systems use activation frames, stored on a stack, to keep track of argument values and return points during recursive function calls. Activation frames can be used to trace the execution of a sequence of recursive function calls.
- mathematical sequences and formulas that are defined recursively can be implemented naturally as recursive functions
- backtracking is a technique that enables you to write programs that can be used to explore different alternative paths in a search for a solution

6.1) Set Theory: Definitions and Element Method of Proof

- A = {x∈ S | P(x)}, the set of all x in S such that P of x.

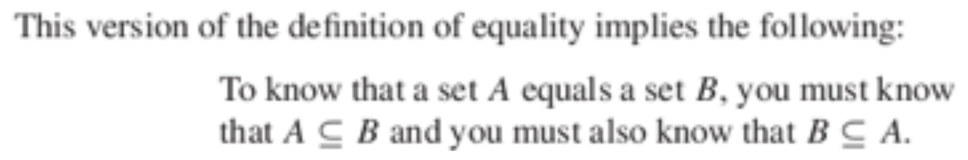


• Definition

Given sets A and B, A **equals** B, written $\boldsymbol{A = B}$, if, and only if, every element of A is in B and every element of B is in A.

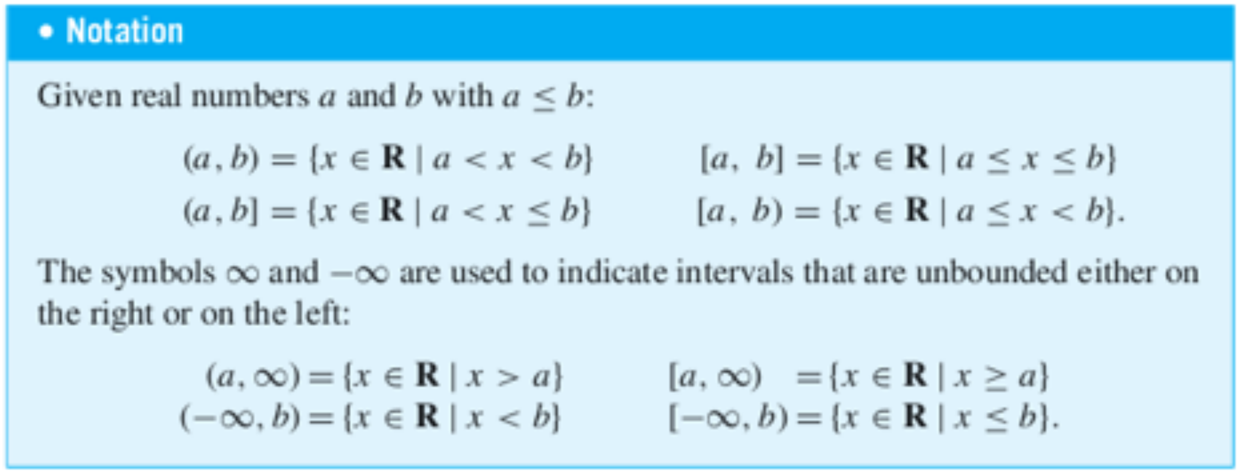
Symbolically:

$$A = B \quad \Leftrightarrow \quad A \subseteq B \text{ and } B \subseteq A.$$



This version of the definition of equality implies the following:

> To know that a set A equals a set B, you must know that $A \subseteq B$ and you must also know that $B \subseteq A$.



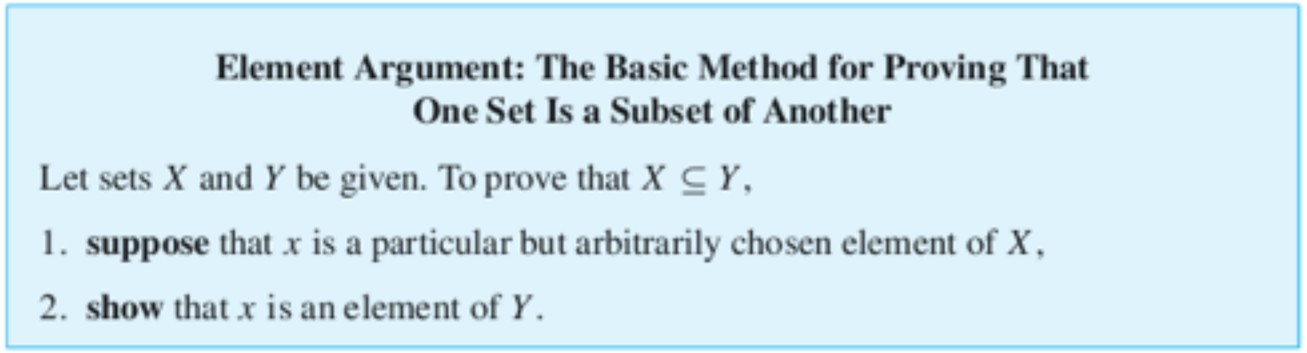
• Notation

Given real numbers a and b with $a \le b$:

$$(a, b) = \{x \in \mathbf{R} \mid a < x < b\} \qquad [a,\ b] = \{x \in \mathbf{R} \mid a \le x \le b\}$$
$$(a, b] = \{x \in \mathbf{R} \mid a < x \le b\} \qquad [a,\ b) = \{x \in \mathbf{R} \mid a \le x < b\}.$$

The symbols ∞ and $-\infty$ are used to indicate intervals that are unbounded either on the right or on the left:

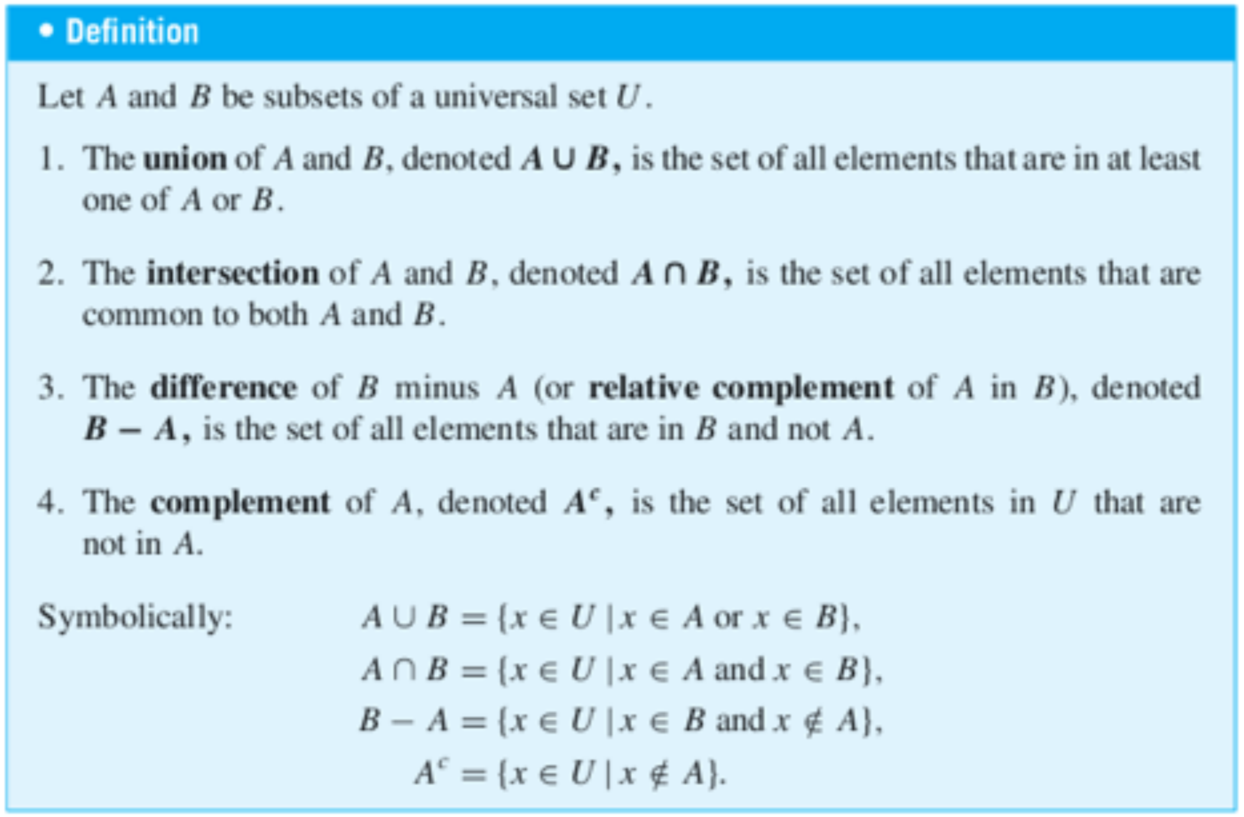
$$(a, \infty) = \{x \in \mathbf{R} \mid x > a\} \qquad [a, \infty) \ = \{x \in \mathbf{R} \mid x \ge a\}$$
$$(-\infty, b) = \{x \in \mathbf{R} \mid x < b\} \qquad [-\infty, b) = \{x \in \mathbf{R} \mid x \le b\}.$$



Element Argument: The Basic Method for Proving That One Set Is a Subset of Another

Let sets X and Y be given. To prove that $X \subseteq Y$,

1. **suppose** that x is a particular but arbitrarily chosen element of X,
2. **show** that x is an element of Y.



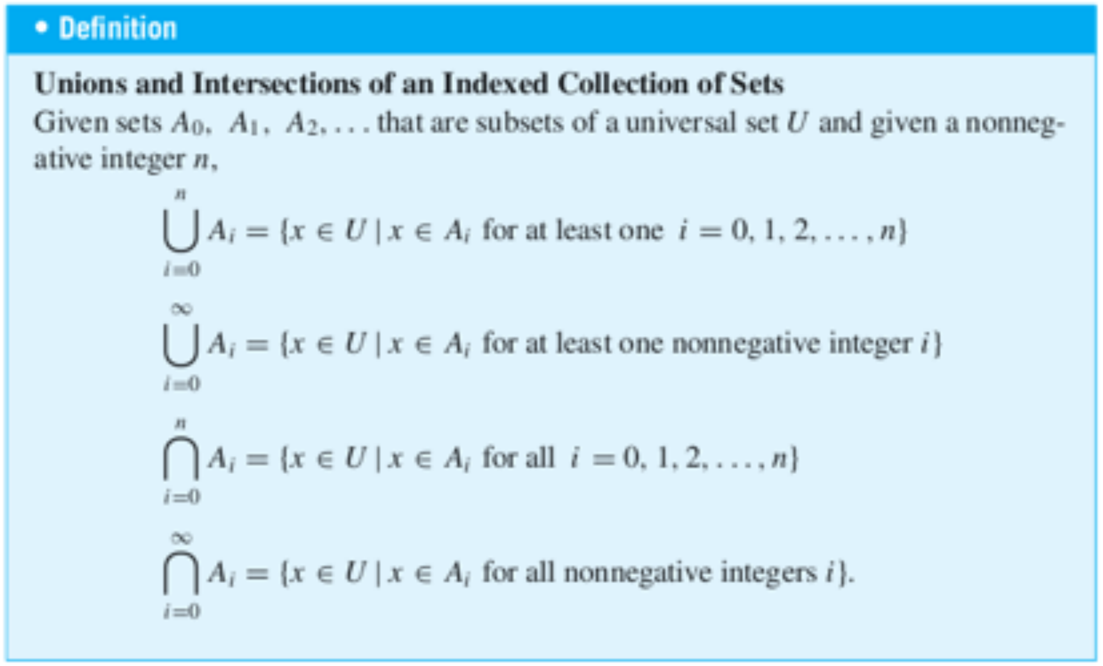
• Definition

Let A and B be subsets of a universal set U.

1. The **union** of A and B, denoted $\boldsymbol{A \cup B}$, is the set of all elements that are in at least one of A or B.
2. The **intersection** of A and B, denoted $\boldsymbol{A \cap B}$, is the set of all elements that are common to both A and B.
3. The **difference** of B minus A (or **relative complement** of A in B), denoted $\boldsymbol{B - A}$, is the set of all elements that are in B and not A.
4. The **complement** of A, denoted $\boldsymbol{A^c}$, is the set of all elements in U that are not in A.

Symbolically:

$$A \cup B = \{x \in U \mid x \in A \text{ or } x \in B\},$$
$$A \cap B = \{x \in U \mid x \in A \text{ and } x \in B\},$$
$$B - A = \{x \in U \mid x \in B \text{ and } x \notin A\},$$
$$A^c = \{x \in U \mid x \notin A\}.$$



• Definition

Unions and Intersections of an Indexed Collection of Sets

Given sets $A_0, A_1, A_2, \ldots$ that are subsets of a universal set U and given a nonnegative integer n,

$$\bigcup_{i=0}^{n} A_i = \{x \in U \mid x \in A_i \text{ for at least one } i = 0, 1, 2, \ldots, n\}$$

$$\bigcup_{i=0}^{\infty} A_i = \{x \in U \mid x \in A_i \text{ for at least one nonnegative integer } i\}$$

$$\bigcap_{i=0}^{n} A_i = \{x \in U \mid x \in A_i \text{ for all } i = 0, 1, 2, \ldots, n\}$$

$$\bigcap_{i=0}^{\infty} A_i = \{x \in U \mid x \in A_i \text{ for all nonnegative integers } i\}.$$

6.2) Properties of Sets

- one set of x is a subset of another set Y and so to prove them you suppose that c is any particular but arbitrarily chosen element of X and you show that x is an element of Y.

Theorem 6.2.1 Some Subset Relations

1. *Inclusion of Intersection:* For all sets A and B,

$$\text{(a) } A \cap B \subseteq A \quad \text{and} \quad \text{(b) } A \cap B \subseteq B.$$

2. *Inclusion in Union:* For all sets A and B,

$$\text{(a) } A \subseteq A \cup B \quad \text{and} \quad \text{(b) } B \subseteq A \cup B.$$

3. *Transitive Property of Subsets:* For all sets A, B, and C,

$$\text{if } A \subseteq B \text{ and } B \subseteq C, \text{ then } A \subseteq C.$$

Theorem 6.2.2 Set Identities

Let all sets referred to below be subsets of a universal set U.

1. *Commutative Laws:* For all sets A and B,

$$\text{(a) } A \cup B = B \cup A \quad \text{and} \quad \text{(b) } A \cap B = B \cap A.$$

2. *Associative Laws:* For all sets A, B, and C,

$$\text{(a) } (A \cup B) \cup C = A \cup (B \cup C) \quad \text{and}$$
$$\text{(b) } (A \cap B) \cap C = A \cap (B \cap C).$$

3. *Distributive Laws:* For all sets, A, B, and C,

$$\text{(a) } A \cup (B \cap C) = (A \cup B) \cap (A \cup C) \quad \text{and}$$
$$\text{(b) } A \cap (B \cup C) = (A \cap B) \cup (A \cap C).$$

4. *Identity Laws:* For all sets A,

$$\text{(a) } A \cup \emptyset = A \quad \text{and} \quad \text{(b) } A \cap U = A.$$

5. *Complement Laws:*

$$\text{(a) } A \cup A^c = U \quad \text{and} \quad \text{(b) } A \cap A^c = \emptyset.$$

6. *Double Complement Law:* For all sets A,

$$(A^c)^c = A.$$

7. *Idempotent Laws:* For all sets A,

$$\text{(a) } A \cup A = A \quad \text{and} \quad \text{(b) } A \cap A = A.$$

8. *Universal Bound Laws:* For all sets A,

$$\text{(a) } A \cup U = U \quad \text{and} \quad \text{(b) } A \cap \emptyset = \emptyset.$$

9. *De Morgan's Laws:* For all sets A and B,

$$\text{(a) } (A \cup B)^c = A^c \cap B^c \quad \text{and} \quad \text{(b) } (A \cap B)^c = A^c \cup B^c.$$

10. *Absorption Laws:* For all sets A and B,

$$\text{(a) } A \cup (A \cap B) = A \quad \text{and} \quad \text{(b) } A \cap (A \cup B) = A.$$

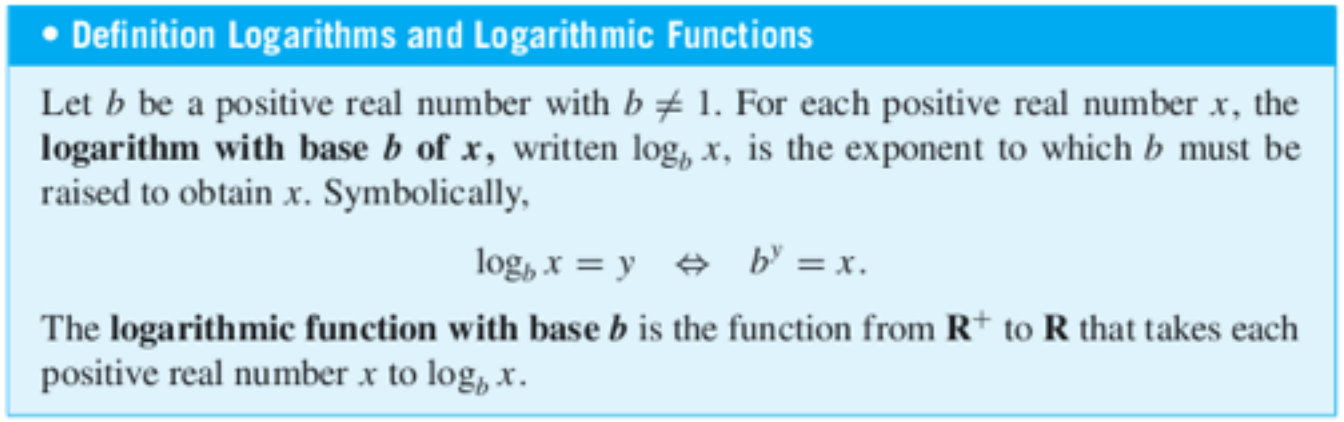
11. *Complements of U and $\emptyset$:*

$$\text{(a) } U^c = \emptyset \quad \text{and} \quad \text{(b) } \emptyset^c = U.$$

12. *Set Difference Law:* For all sets A and B,

$$A - B = A \cap B^c.$$

7.1) Functions Defines of General Sets



• Definition Logarithms and Logarithmic Functions

Let b be a positive real number with $b \neq 1$. For each positive real number x, the **logarithm with base *b* of *x*,** written $\log_b x$, is the exponent to which b must be raised to obtain x. Symbolically,

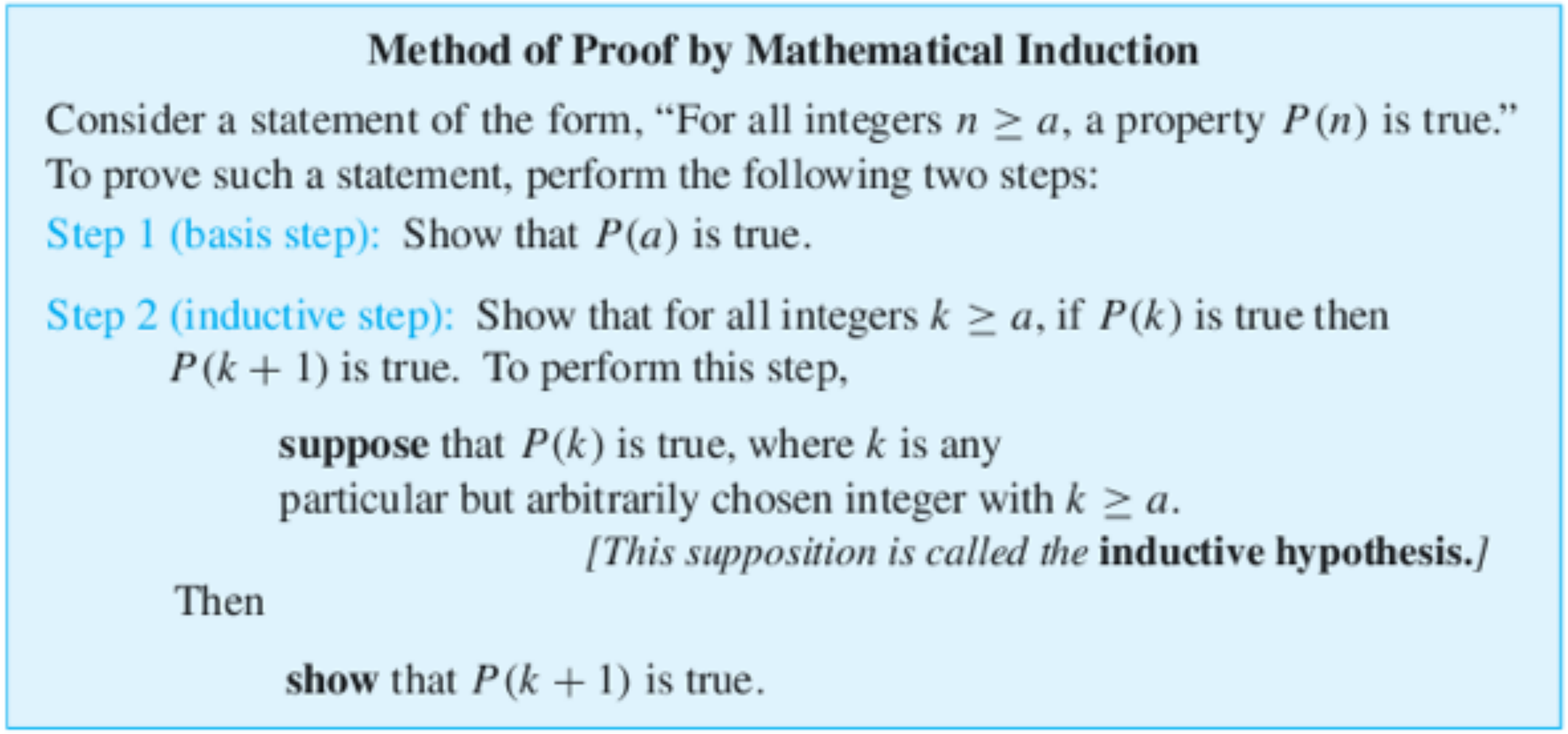
$$\log_b x = y \quad \Leftrightarrow \quad b^y = x.$$

The **logarithmic function with base *b*** is the function from $\mathbf{R}^+$ to $\mathbf{R}$ that takes each positive real number x to $\log_b x$.

- um, this was boring…

5.2/3) Mathematical Induction I/II

- used to check conjectures about the outcomes of processes that occur repeatedly and according to definite patterns
- Principle of Mathematical Induction:
  - Let P(n) be a property that is defined for integers n, and let a be a fixed integer. Suppose that following statements are true
  - 1) P(a) is true
  - 2) For all integers k≥a, if P(k) is true then P(k+1) is true.
  - Then the statement for all integers n≥a, P(n) is true



Method of Proof by Mathematical Induction

Consider a statement of the form, "For all integers $n \geq a$, a property $P(n)$ is true." To prove such a statement, perform the following two steps:

Step 1 (basis step): Show that $P(a)$ is true.

Step 2 (inductive step): Show that for all integers $k \geq a$, if $P(k)$ is true then $P(k+1)$ is true. To perform this step,

suppose that $P(k)$ is true, where k is any particular but arbitrarily chosen integer with $k \geq a$.

[This supposition is called the ***inductive hypothesis.****]*

Then

show that $P(k+1)$ is true.

For all integers $n \geq 1$,

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

Show that P(1) is true:

To establish $P(1)$, we must show that

$$1 = \frac{1(1+1)}{2} \qquad \leftarrow P(1)$$

But the left-hand side of this equation is 1 and the right-hand side is

$$\frac{1(1+1)}{2} = \frac{2}{2} = 1$$

also. Hence $P(1)$ is true.

Show that for all integers $k \geq 1$, if P(k) is true then P(k + 1) is also true:
[Suppose that P(k) is true for a particular but arbitrarily chosen integer $k \geq 1$. That is:] Suppose that k is any integer with $k \geq 1$ such that

$$1 + 2 + 3 + \cdots + k = \frac{k(k+1)}{2} \qquad \leftarrow P(k)$$

inductive hypothesis

[We must show that $P(k+1)$ is true. That is:] We must show that

$$1 + 2 + 3 + \cdots + (k+1) = \frac{(k+1)[(k+1)+1]}{2},$$

or, equivalently, that

$$1 + 2 + 3 + \cdots + (k+1) = \frac{(k+1)(k+2)}{2}. \qquad \leftarrow P(k+1)$$

[We will show that the left-hand side and the right-hand side of $P(k+1)$ are equal to the same quantity and thus are equal to each other.]

The left-hand side of $P(k+1)$ is

$$\begin{aligned}
&1 + 2 + 3 + \cdots + (k+1) \\
&= 1 + 2 + 3 + \cdots + k + (k+1) && \text{by making the next-to-last term explicit} \\
&= \frac{k(k+1)}{2} + (k+1) && \text{by substitution from the inductive hypothesis} \\
&= \frac{k(k+1)}{2} + \frac{2(k+1)}{2} \\
&= \frac{k^2+k}{2} + \frac{2k+2}{2} \\
&= \frac{k^2+3k+1}{2} && \text{by algebra.}
\end{aligned}$$

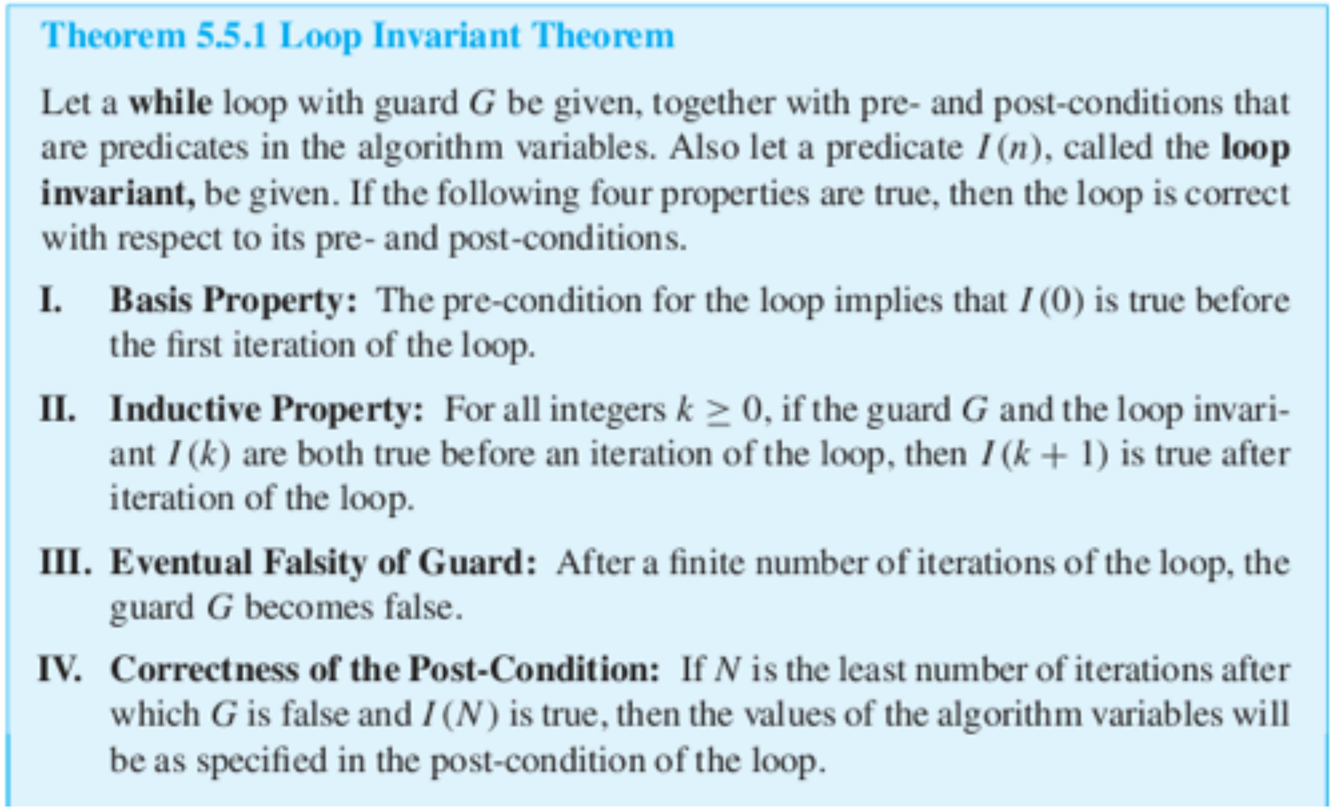
And the right-hand side of $P(k+1)$ is

$$\frac{(k+1)(k+2)}{2} = \frac{k^2+3k+1}{2}.$$

Thus the two sides of $P(k+1)$ are equal to the same quantity and so they are equal to each other. Therefore the equation $P(k+1)$ is true *[as was to be shown]*.
[Since we have proved both the basis step and the inductive step, we conclude that the theorem is true.]

Correctness of Algorithms

- pre-condition: initial state; post-condition: final state
- Loop Invariant: a predicate with domain a set of integers, which satisfies the condition: For each iteration of the loop, if the predicate is true before the iteration, then it is true after the iteration. Furthermore, if the predicate satisfies the following two additional conditions, the loop will be correct with respect to its pre- and post-conditions:
    - It is true before the first iteration of the loop
    - If the loop terminates after a finite number of iterations, the truth of the loop invariant ensures the truth of the post-condition of the loop.



Theorem 5.5.1 Loop Invariant Theorem

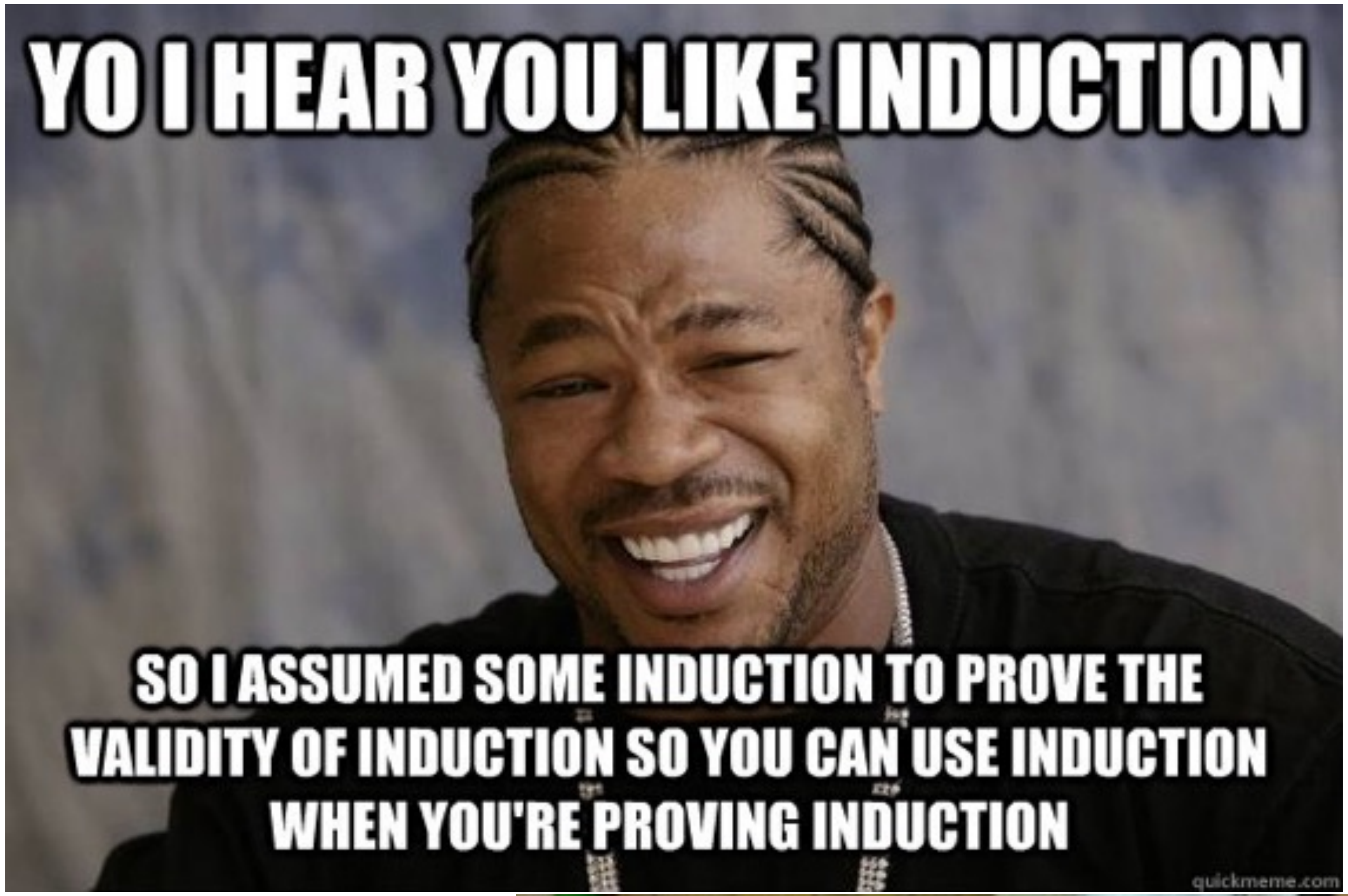
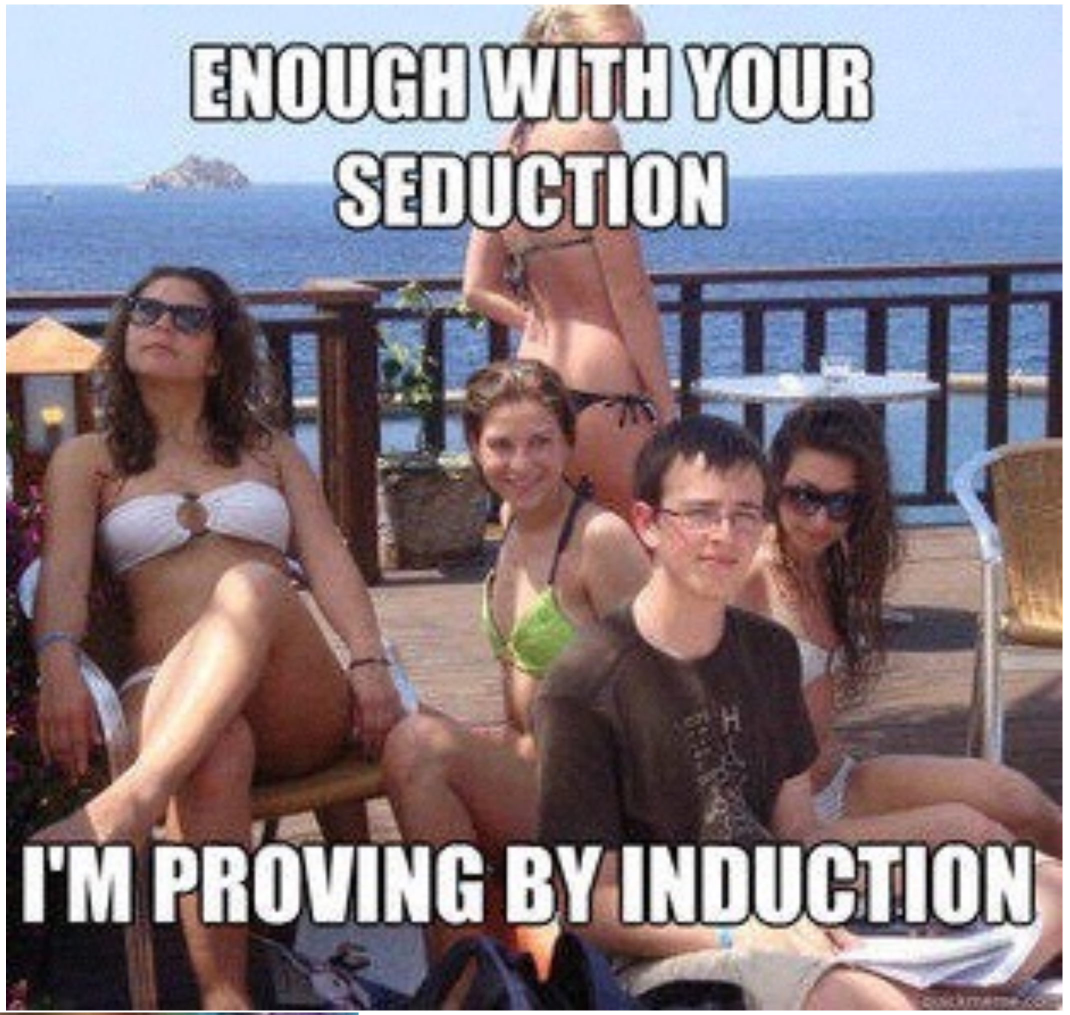
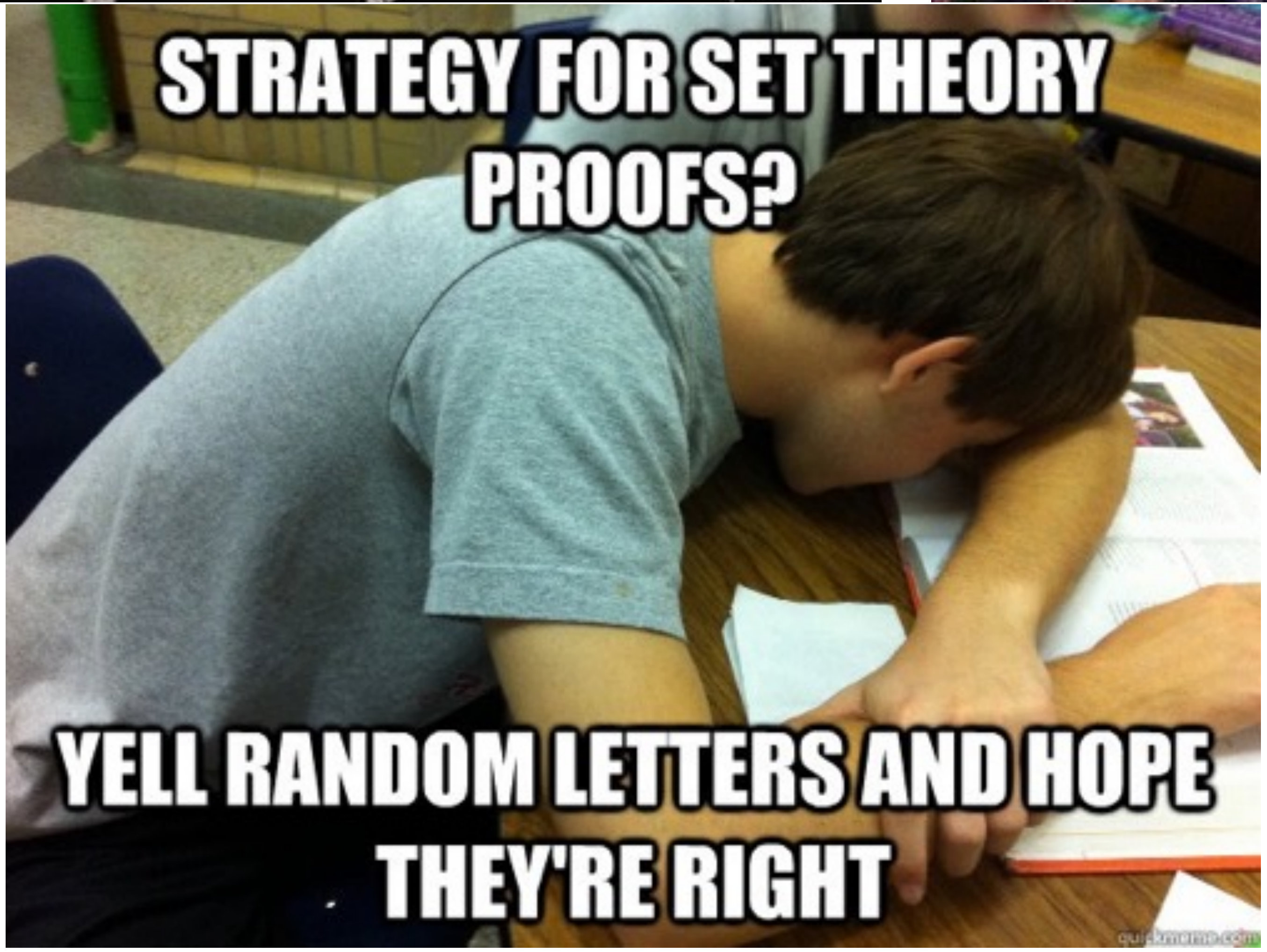
Let a **while** loop with guard G be given, together with pre- and post-conditions that are predicates in the algorithm variables. Also let a predicate $I(n)$, called the **loop invariant,** be given. If the following four properties are true, then the loop is correct with respect to its pre- and post-conditions.

I. Basis Property: The pre-condition for the loop implies that $I(0)$ is true before the first iteration of the loop.

II. Inductive Property: For all integers $k \geq 0$, if the guard G and the loop invariant $I(k)$ are both true before an iteration of the loop, then $I(k+1)$ is true after iteration of the loop.

III. Eventual Falsity of Guard: After a finite number of iterations of the loop, the guard G becomes false.

IV. Correctness of the Post-Condition: If N is the least number of iterations after which G is false and $I(N)$ is true, then the values of the algorithm variables will be as specified in the post-condition of the loop.


YO I HEAR YOU LIKE INDUCTION
SO I ASSUMED SOME INDUCTION TO PROVE THE VALIDITY OF INDUCTION SO YOU CAN USE INDUCTION WHEN YOU'RE PROVING INDUCTION
quickmeme.com

ENOUGH WITH YOUR SEDUCTION
I'M PROVING BY INDUCTION
quickmeme.com

STRATEGY FOR SET THEORY PROOFS?
YELL RANDOM LETTERS AND HOPE THEY'RE RIGHT
quickmeme.com