Notes on Proof by Induction

Zack Jorquera

Comments and Corrections Welcome!

Contents

1 Introduction

This document will serve as a reference for how to use proof by induction and what we expect in your own proofs on the homework. It is not required that you use the formatting seen in this document. However, if you are still familiarizing yourself with the proof by induction technique, then it is highly recommended that you set up your proofs as we do in this document to help structure your proofs. The .tex file used for this document can be found on canvas/piazza.

2 Proof by Induction

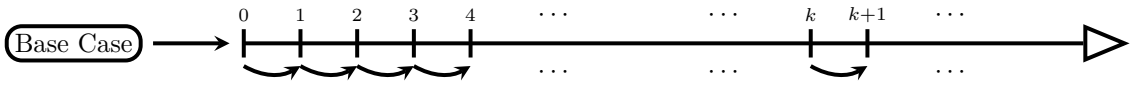
It is often necessary to prove that a statement holds for any non-negative integer. For example, we may want to prove that for any non-negative integer, $n \in \mathbb{Z}_{\geq 0}$, the following holds (note, we are using $\mathbb{Z}_{\geq 0} = \{0, 1, 2, \dots\}$)

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

This sort of statement can be very challenging to prove for every $n \in \mathbb{Z}_{\geq 0}$ due to the infinite nature of the integers. We could try to prove this by considering any arbitrary $n \in \mathbb{Z}_{\geq 0}$ and proving that it is true directly. While definitely possible in this case, this type of proof ignores the statement's structure. Namely, we have that $\sum_{i=0}^{n+1} i = (n+1) + \sum_{i=1}^{n} i$. Using this structure, we can build a proof using smaller instances of the problem. In particular, it is easy to prove that the above claim is true for small n, say $n = 0, 1, 2, 3$. These will serves as a starting point for a chain of implications that will, in turn, prove the statement for the whole domain.

2.1 Weak Induction

Intuitively, we can think of this as a sequence of dominos [Ham19]. We have to knock over the first domino (proving the initial/base cases), and then if the dominos are close enough together, each domino will knock over the next in the sequence creating a chain of implications. Pictorially, this chain of implication looks like the following.



More concretely, an inductive proof has three components (adapted from [Lev23]): The base cases, the inductive hypothesis, and the inductive step.

1. **Bases Cases** - We first verify that the statement holds for the minimal case(s) (the ones that start the chain of implications).
2. **Inductive Hypothesis** - We want to show that *if* some earlier case(s) satisfy the statement, *then* so do the subsequent cases. The inductive hypothesis is the *if* part of this if-then statement. We do this by assuming that the statement holds for some or all earlier cases.
3. **Inductive Step** - We use the inductive hypothesis to prove that the subsequent cases also hold. This is the *then* part of the if-then statement.

As an example, we can then use this proof technique to prove the following proposition.

Proposition 2.1. *For all $n \in \mathbb{Z}_{\geq 0}$, we have that:*

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

Proof. We prove this by induction over $n \in \mathbb{Z}_{\geq 0}$.

Base Case: We verify that the proposition holds for $n = 0$. We have that $\sum_{i=0}^{0} i = 0$ which is equal to $\frac{0\cdot(0+1)}{2} = 0$. And thus, the proposition holds for $n = 0$.

Inductive Hypothesis: Fix some $k \geq 0$ and suppose that the proposition holds for the $n = k$ case, i.e.,

$$\sum_{i=0}^{k} i = \frac{k(k+1)}{2}$$

Inductive Step: Consider the sum of integers from 0 to $k+1$, we want to show the following.

$$\sum_{i=0}^{k+1} i = \frac{(k+1)(k+2)}{2}$$

We do this in the following way using the inductive hypothesis.

$$\begin{aligned}\sum_{i=0}^{k+1} i &= (k+1) + \sum_{i=0}^{k} i \\ &= (k+1) + \frac{k(k+1)}{2} \quad \text{(by inductive hypothesis)} \\ &= \frac{2(k+1) + k(k+1)}{2} \\ &= \frac{k^2 + 3k + 2}{2} \\ &= \frac{(k+1)(k+2)}{2}\end{aligned}$$

And thus, by induction, the proposition holds for all $n \in \mathbb{Z}_{\geq 0}$. ■

Remark 2.2. There are many ways to write an inductive hypothesis. For example, we could have said something like "For some arbitrary $k \geq 0$, assume that the proposition holds." Or we could continue using n instead of introducing a new variable k. However, be careful. A common mistake is to assume the conclusion. That is, we assume that the entire proposition is true. As an incorrect example, I could have said, "Suppose that for all $k \geq 0$, the proposition is true." This, however, is incorrect because this is exactly what we are trying to prove (replace k with n). This may seem pedantic, but it is important to have the correct wording as otherwise, it can lead to proving incorrect statements. Instead, we want to give a proof for the inductive step for an arbitrary k by assuming the previous case(s). This distinction is subtle, but important. Using the domino analogy, we only want to suppose that the previous domino(s) fell and not that all the dominos fell.

The above proof by induction is an example of *weak induction*, the most basic form of induction. In short, weak induction is when we only have a single base case, and the inductive hypothesis only assumes the statement is true for some fixed k.

Now that we have looked at an example, we can formalize this model of induction a bit more. We state the Law (or Axiom) of Weak Induction, which is why induction works.

Definition 2.3 (Law of Weak Induction)**.** Let $P(n)$ be a statement regarding a non-negative integer, $n \in \mathbb{Z}_{\geq 0}$. If

1. $P(0)$ is true and
2. $\forall k \geq 0 : P(k) \to P(k+1)$ is true

then $\forall n \geq 0 : P(n)$ is true.

This law is not proven and is instead typically given as an axiom. It is for this reason we give it as a definition. Additionally, this is also the reason why you should always say something along the lines of, "And thus, by induction, the proposition holds for all $n \in \mathbb{Z}_{\geq 0}$" at the end of your inductive proof.

Remark 2.4. It is often the case that some statements are not true for small values of $n < c$ bellow some fixed constant $c \in \mathbb{Z}_{\geq 0}$ but are true for all $n \geq c$. We can still use the Law of Weak Induction in these cases. If we seek to prove that $\forall n \geq c : P(n)$, then we could instead consider the statement $Q(n) : P(n + c)$. Then proving that $\forall n \geq 0 : Q(n)$ is equivalent to $\forall n \geq c : P(n)$.

Finally, we dissect the law of weak induction to understand how our proof of Proposition 2.1 fits into this framework. We first prove the base case, this is the $P(0)$ part. Then we fixed a k and showed that

$P(k) \to P(k+1)$. This is both the inductive hypothesis (assuming $P(k)$ for some fixed k) and the inductive step (proving the implication itself). Note that we never wrote the $\forall k \geq 0$ part in the proof. This is because it is implicit. That is, in the inductive hypothesis and inductive step, when we fix an arbitrary k, our proof applies equally as well for any $k \geq 0$. All in all, this satisfies the second condition, $\forall k \geq 0 : P(k) \to P(k+1)$.

To emphasise this point more, let's say we are trying to prove the following proposition. We can then use use the law of induction in a proof in the following way.

Proposition (Template Proposition)**.** $\forall n \geq 0 : P(n)$ *is true.*

Template proof. We prove that $\forall n \geq 0 : P(n)$ is true by induction over $n \geq 0$.

Base Case: Verify $P(0)$ is true.

Inductive Hypothesis: Fix a $k \geq 0$ and suppose that $P(k)$ is true.

Inductive Step: Show that $P(k+1)$ is true assuming the inductive hypothesis (i.e., that $P(k) \to P(k+1)$).

Then, by the law of (weak) induction, we have proven that $\forall n \geq 0 : P(n)$. ■

2.2 Strong Induction

So far, we have looked at weak induction. Another form of induction that is widely used is *strong induction*. In short, strong induction allows us to make a stronger assumption for the inductive hypothesis. That is, we can assume the statement holds for all cases $m \leq k$, for some fixed k, and not just for k alone. Before we go further into this technique, we state the Law of Strong Induction.

Definition 2.5 (Law of Strong Induction)**.** Let $P(n)$ be a statement regarding a non-negative integer, $n \in \mathbb{Z}_{\geq 0}$. If

1. $P(0)$ is true and
2. $\forall k \geq 0 : (\forall j \leq k : P(k)) \to P(k+1)$ is true

then $\forall n \geq 0 : P(n)$ is true.

As we did for weak induction, we can look at a template proof using strong induction.

Proposition (Template Proposition)**.** $\forall n \geq 0 : P(n)$ *is true.*

Template proof. We prove that $\forall n \geq 0 : P(n)$ is true by induction over $n \geq 0$.

Base Case: Verify $P(0)$ is true.

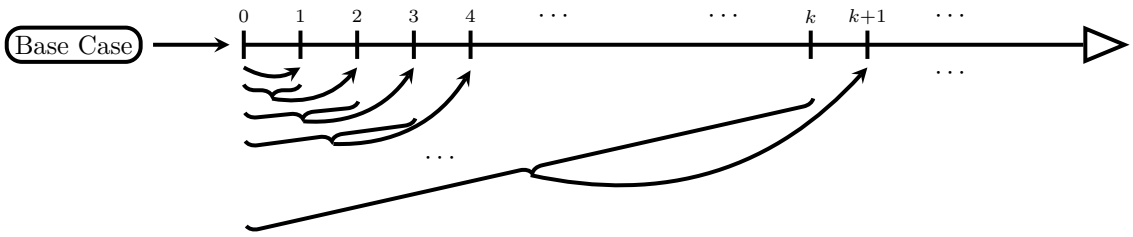
Inductive Hypothesis: Fix a $k \geq 0$ and suppose that $\forall j \leq k : P(k)$ is true.

Inductive Step: Show $P(k+1)$ is true assuming the inductive hypothesis (i.e., $(\forall j \leq k : P(k)) \to P(k+1)$).

Then, by the law of (strong) induction, we have proven that $\forall n \geq 0 : P(n)$. ■

Notice the only difference between strong and weak induction is the inductive hypothesis. Pictorially, this makes the chain of implications look like the following.

Remark 2.6. Somewhat surprisingly, strong induction and weak induction are equally as powerful. That is, any proof using strong induction can be converted into a proof using weak induction and vice versa. This is evident by constructing the statement $Q(n) : \forall 0 \leq k \leq n : P(k)$, then using weak induction to prove $\forall n \geq 0 : Q(n)$ is equivalent to using strong induction to prove $\forall n \geq 0 : P(n)$. However, in practice, it may be easier to use one over the other. We highlight this in the next example.



As a side note, in section Section 3, we will see what are called loop invariant proofs, which use weak induction under the hood. That is, they assume only that the loop invariant is true at the start of the loop and then prove that it is true at the end of the loop (or at the start of the next iteration, depending on how you phrase it). In comparison, many proofs for recursive algorithms use strong induction. That is, they assume that the algorithm works for subproblems of smaller sizes.

We can then give some examples of using strong induction.

Proposition 2.7. *Let F_n for $n \geq 0$ denote the nth Fibonacci number, which is defined as $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$. And let L_n for $n \geq 0$ denote the nth Lucas number, which is defined as $L_0 = 2, L_1 = 1, L_n = L_{n-1} + L_{n-2}$ for $n \geq 2$. We have that for all $n \geq 1$,*

$$L_n = F_{n-1} + F_{n+1}$$

For reference, we write the first few numbers for both sequences:

n	0	1	2	3	4	5	6	7
F_n	0	1	1	2	3	5	8	13
L_n	2	1	3	4	7	11	18	29

Proof. We prove this by strong induction over $n \geq 1$.

Base Case: We verify that the proposition holds for $n = 1$ and $n = 2$. First, we can verify that $L_1 = 1$ and $F_0 + F_2 = 0 + 1 = 1$. Next, we can verify that $L_2 = 3$ and $F_1 + F_3 = 1 + 2 = 3$. And thus, the proposition holds for $n = 1$ and $n = 2$.

Inductive Hypothesis: Fix some $k \geq 2$ and suppose that the proposition holds for the k and $k - 1$ cases:

$$L_k = F_{k-1} + F_{k+1}$$
$$L_{k-1} = F_{k-2} + F_k$$

Inductive Step: Consider the $(k + 1)$th Lucas number, which we want to show is $L_{k+1} = F_k + F_{k+2}$. We do this in the following way.

$$\begin{aligned} L_{k+1} &= L_k + L_{k-1} && \text{(by definition of the Lucas numbers)} \\ &= F_{k-1} + F_{k+1} + F_{k-2} + F_k && \text{(by inductive hypothesis)} \\ &= F_k + F_{k+2} && \text{(by definition of the Fibonacci numbers)} \end{aligned}$$

And thus, by induction, the proposition holds for all $n \geq 1$. ■

Remark 2.8. In this proof, we used two base cases and supposed the proposition held for both $n = k$ and $n = k-1$ for some fixed $k \geq 2$, but why is this needed? You will notice that in the inductive step, we applied the inductive hypothesis to both L_k and L_{k-1}. If we only supposed that the proposition held for $n = k$, then we would be left with $L_{k+1} = F_{k-1} + F_{k+1} + L_{k-1}$, which wouldn't help us prove the proposition for $n = k + 1$. Next, for the base cases, if we were only to have one base case, for $n = 1$, then the inductive hypothesis would have to consider some fixed $k \geq 1$. For the smallest value of $k = 1$, this would have us suppose that $L_0 = F_{-1} + F_1$, which doesn't make sense as negative Fibonacci numbers haven't been defined in this proposition. We note that this still fits into the framework given by the law of strong induction as $P(1)$ implies that $P(0) \to P(1)$.

2.3 A Longer Example

Next, we will look at the tournament ranking problem [1]. So far, all the proofs in this document have only concerned numerical equalities. However, this is a computer science class, and so we want to prove things that are more relevant to algorithms. The tournament ranking is a nice introduction to this.

Proposition 2.9 (The Tournament Ranking Problem)**.** *Consider a tennis tournament of $n \geq 1$ players, denoted by $\{P_1, P_2, \ldots, P_n\}$. For each pair of distinct players, say (P_i, P_j) (for $i \neq j$), they play a match where one player wins (there are no ties). We say $P_i \prec P_j$ if player P_j beats P_i in their match. Then for any outcome of the $\binom{n}{2}$ matches, there is always an ordering of the players:*

$$P_{i_1} \prec P_{i_2} \prec \cdots \prec P_{i_n}$$

Here, $(i_1, i_2, \ldots, i_n)$ denotes a permutation of $(1, 2, \ldots, n)$. Note, we do not *require that an ordering be transitive, that is, for an ordering $P_1 \prec P_2 \prec P_3$ does, this does* not *imply $P_1 \prec P_3$, rather only that $P_1 \prec P_2$ and $P_2 \prec P_3$.*

Before we prove this, it is useful to give the idea of the proof. That is, for our tournament of n players, which, for notation we will call PLAYERS $= \{P_1, P_2, \ldots, P_n\}$, we fix player P_1 and consider all the players that lost to player P_1 and all the players that beat P_1. That is, we consider all the players that can go where the ??s are:

$$?? \prec P_1 \prec ??$$

Furthermore, the number of players that can go where the ??s are is strictly less than n as they can't include P_1. This defines a recursive structure of the problem. Namely, if we can find an ordering of these subsets of players and put them where the question marks are, we would get an ordering for all n players.

To make the proof easier, we will consider the base case to be when there are no players in the tournament. Even though we only need to prove this proposition for $n \geq 1$, we will find it to be much easier to prove it for $n \geq 0$. This may seem weird at first, but everything works out as far as the induction step is concerned.

Proof. We prove this by strong induction over the number of players in the tournament, $n \geq 0$.

Base Case: We verify that there exists an ordering for no player, $\emptyset$. The ordering is nothing.

Inductive Hypothesis: Fix some $k \geq 0$ and suppose that for any tournament of $0 \leq m \leq k$ players there exists an ordering of the players.

Inductive Step: Consider a tournament of $k+1$ players and let PLAYERS $= \{P_1, P_2, \ldots, P_k, P_{k+1}\}$ be the set of the players. We then fix P_1 and consider the set of players that lost to P_1, which we will call L, and the set of players that beat P_1, which we will call W. These sets can be defined by

$$L = \{P \in \text{PLAYERS} \mid P \prec P_1\}$$

$$W = \{P \in \text{PLAYERS} \mid P_1 \prec P\}$$

Because $P_1 \notin L$ and $P_1 \notin W$ we have that $s = |L| \leq k$ and $t = |W| \leq k$. Furthermore, because no player can both lose and win to P_1 and every player had a match with P_1, they are disjoint, and their union is all players minus P_1. We can then use the induction hypothesis to get an ordering of the players in L and W (that is, we consider "sub-tournaments" of only the matches between the players in L and W). Let the orderings for L and W, guaranteed by the inductive hypothesis, be given by the following, respectively:

$$P_{\ell_1} \prec P_{\ell_2} \prec \cdots \prec P_{\ell_s}$$

$$P_{w_1} \prec P_{w_2} \prec \cdots \prec P_{w_s}$$

[1] Quick integrity check. This problem is often given as homework; if you use this solution at any step while doing your homework for a future or current class, you should always cite your sources.

Finally, we can construct the final ordering as follows. Note, if either L or W are the empty set, then we ignore their orderings and let P_1 be at one or both of the ends of the ordering.

$$P_{\ell_1} \prec \cdots \prec P_{\ell_s} \prec P_1 \prec P_{w_1} \prec \cdots \prec P_{w_s}$$

All that is left to check is that $P_{\ell_s} \prec P_1$ and $P_1 \prec P_{w_1}$. These follow from the fact that $P_{\ell_s} \in L$ and thus lost to P_1 and that $P_{w_1} \in W$ and thus beat P_1.

Thus, by induction, there is always an ordering for any tournament of $n \geq 1$ players. ■

Remark 2.10. In many ways, the inductive step can be seen as proving the correctness of a recursive algorithm for finding an ordering. This algorithm is a divide-and-conquer style strategy that can be roughly written in the following way: We first divide the problem into the losing and winning sets, then we find an ordering for them recursively, and finally, we combine them together to get an ordering for the whole set. Then, proving the proposition is equivalent to proving the correctness of this algorithm (which is written in pseudo-code below). Note we store this ordering in a list data structure that supports list concatenation with the + operator.

Algorithm 2.11 (Tournament Ranking Algorithm)**.**

```
1: procedure TournamentRanking(PLAYERS)
2:     if PLAYERS = ∅ then return [ ]
3:     else
4:         fix P_1 ∈ PLAYERS
5:         L ← {P ∈ PLAYERS | P ≺ P_1}
6:         W ← {P ∈ PLAYERS | P_1 ≺ P}
7:         L_ord ← TournamentRanking(L)
8:         W_ord ← TournamentRanking(W)
9:         return L_ord + [P_1] + W_ord
```

2.4 Problems/Exercises

This is a list of some of my go to problems. I have the solutions for some of them.

Problem 2.1. Prove the following for all $n \in \mathbb{Z}_{>0}$:

$$\sum_{i=1}^{n} i(i+1) = \frac{n(n+1)(n+2)}{3}$$

Problem 2.2. Prove the following: Let $n \in \mathbb{Z}_{>0}$, then for two sequences of positive numbers, $(a_i)_{i\in[n]}$ and $(b_i)_{i\in[n]}$ we have that

$$\min_{i\in[n]}\left(\frac{a_i}{b_i}\right) \leq \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} b_i} \leq \max_{i\in[n]}\left(\frac{a_i}{b_i}\right)$$

[Hint: First prove the $n = 2$ case, then break up your inductive case into two cases that allow you to use the result of the $n = 2$ case directly. Note, your base case should still be for $n = 1$.]

Problem 2.3. Prove the following: Let F_n for $n \geq 0$ denote the nth Fibonacci number. Fix an $n \in \mathbb{Z}_{\geq 0}$, then we have the following

$$\sum_{i=0}^{n} F_n = F_{n+2} - 1$$

Problem 2.4. Prove that all connected graphs on n vertices have at least $n - 1$ edges.

Problem 2.5. Prove that any planar graph can be colored in 6 colors. You may use, without proof, the fact that every planar graph always has at least one vertex with degree no more than 5 (this is evident by bounding the average degree using Euler's formula, $|V| - |E| + |F| = 2$, and the fact that $2|E| \geq 3|F|$, which in turn follows from the hand shake lemma).

Problem 2.6 (I'll take n please)**.** You're favorite dumpling shop only sells dumplings in amounts of 3,5, or 8. However, you are very hungry and would like an arbitrary number of dumplings. You recall that your friend once said they used a trick to order exactly 9 or 10 dumplings by ordering three orders of three or two orders of five. Prove that for any value of $n \geq 8$ there is a way to order exactly that many dumplings.

3 Loop Invariant Proofs

One common variant of induction seen in computer science is a loop invariant proof. In short, these are used when an algorithm has a loop, and you want to prove something about the algorithm. For the proof, you do (weak) induction over the loop to prove that some statement that depends on the loop number is true for each loop and after the loop has terminated. This statement is referred to as a loop invariant.

In addition to a loop invariant, we will also have a statement that determines whether we are still in the loop or not. We will call this the loop condition. If our algorithm is a loop of the form **while** $C(i)$ **do**, then $C(i)$ is the loop condition. Here, i is the loop number.

One common confusion with loop invariants is whether the loop invariant should be true at the beginning of the loop or at the end. It turns out that either type of loop invariant works, and, in fact, they are equivalent up to a '+1' in the loop number. However, there are subtle differences between how the proof is set up using two models. In this section, we discuss both. We suggest that you pick one model and stick with it in your own proofs. It is rarely the case that one is easier than the other, as was true for weak and strong induction. To paint a picture of what a loop invariant is, consider the following bare-bones example.

Example 3.1. We break down an iterative algorithm into parts that will help us understand the structure of a loop invariant. We use "Initialization Step," "Loop Step," and "Termination Step" to denote arbitrary code that you would have to reason about in the proof.

Any loop can be converted to have the structure of the following example. We let $I_b(i)$ denote the loop invariant that is true at the beginning of the loop and $I_e(i)$ be the loop invariant that is true at the end of the loop. Additionally, i will denote the iteration number, which is explicit in the following example but may not be in general.

```
1: Initialization Step
2: assert I_e(0)
3: i ← 1
4: while C(i) do
5:     assert I_b(i)
6:     Loop Step
7:     assert I_e(i)
8:     i ← i + 1
9: assert I_b(i)
10: Termination Step
```

Note that while both variations serve that same purpose, we don't, in general, have that $I_b(i) = I_e(i)$ but instead have that $I_b(i+1) = I_e(i)$. Moreover, $I_b(0)$ need not be defined and same for $I_e(n+1)$, where n is the last time that $C(n)$ is true, i.e., $C(n+1)$ is false.

Before formally defining a loop invariant, we will want a little more structure on the statement, $C(i)$. Even though it is never used in the loop, it will be useful to have $C(0)$ defined to be true. Moreover, when the

loop terminates, i.e., $C(n)$ is false for some $n \geq 1$, we will want to enforce that all larger integers, $k \geq n$, $C(k)$ also be false. That is, we will enforce that $\forall n \geq 1 : \neg C(n) \to \forall k \geq n : \neg C(k)$. We note that neither of these has any bearing on how $C(i)$ is used in the loop itself, but nonetheless, we do this to make the mathematical definitions easier.

With the loop condition, $C(i)$, in hand, we can now formally define a loop invariant. We take special consideration as to whether it should be true at the start or the end of the loop, even though both are equivalent, up to how they relate to $C(i)$.

Definition 3.2 (Loop Invariant)**.** A *loop invariant* is a statement $I(n)$ regarding a non-negative integer $n \in \mathbb{Z}_{\geq 0}$ that is true until $C(n)$ is false. More formally, if $I(n)$ is defined to be true at the beginning of the loop, then $I(n)$ is a loop invariant if $\forall n \geq 1 : C(n-1) \to I(n)$. If, on the other hand, $I(n)$ is defined to be true at the end of the loop, then $I(n)$ is a loop invariant if $I(0) \wedge \forall n \geq 1 : C(n) \to I(n)$.

Remark 3.3. In both cases, you should think of $C(n)$ as determining if the loop will be run. In the first case (beginning of loop case), we want the loop invariant to be true during the loop and after the loop has terminated. So if n is the first value that makes $C(n)$ false, then we still want $I(n)$ to be true. In the other case (end of loop case), if $C(n)$ is false, then we wouldn't have run that loop, and thus we only ensure that $I(n-1)$ is true.

In order to prove a loop invariant, we use induction. However, this time, we have to incorporate the loop condition into the proof. For this, we consider the modified law of (weak) induction. We note that it is, again, equally as powerful as the law of (weak) induction. However, we state the following as propositions to emphasize how they follow from the Law of induction, Definition 2.3. We, of course, continue to break this down into two cases, whether the loop invariant is defined to be true at the beginning or the end of the loop.

Proposition 3.4 (Beginning of Loop Case)**.** *Let $I(n)$ and $C(n)$ be statements regarding a non-negative integer, $n \in \mathbb{Z}_{\geq 0}$, where $C(0)$ is defined to be true and $\forall n \geq 0 : \neg C(n) \to \forall k \geq n : \neg C(k)$. Then, if*

1. *$I(1)$ is true and*
2. *$\forall k \geq 1 : C(k) \wedge I(k) \to I(k+1)$ is true*

then $\forall n \geq 1 : C(n-1) \to I(n)$ is true (i.e., $I(n)$ is loop invariant, at the beginning of the loop).

Next, we give the variant of Proposition 3.4 for loop invariants that are true at the end of the loop.

Proposition 3.5 (End of Loop Case)**.** *Let $I(n)$ and $C(n)$ be statements regarding a non-negative integer, $n \in \mathbb{Z}_{\geq 0}$, where $C(0)$ is defined to be true and $\forall n \geq 0 : \neg C(n) \to \forall k \geq n : \neg C(k)$. Then, if*

1. *$I(0)$ is true and*
2. *$\forall k \geq 0 : C(k+1) \wedge I(k) \to I(k+1)$ is true*

then $\forall n \geq 0 : C(n) \to I(n)$ is true (i.e., $I(n)$ is loop invariant, at the end of the loop).

Remark 3.6. We quickly remark on the differences in the inductive steps for the two definitions. In Proposition 3.4, the inductive hypothesis is $I(k) \wedge C(k)$. You can think of this as saying that if the loop invariant is true at the beginning of loop k and the loop will be executed for iteration k (i.e., $C(k)$), then the loop invariant should be true at the start of the next iteration or after termination.

Similarly, in Proposition 3.5 the inductive hypothesis is $I(k) \wedge C(k+1)$. You can think of this as saying that if the loop invariant is true at the end of the kth loop and the loop will be executed for one more iteration (i.e., $C(k+1)$), then the loop invariant should be true at the end of the next iteration.

As with before, we give a template proof using this structure. For now, we only give it for Proposition 3.4. The other variant is used in the proof of Proposition 3.9.

Proposition (Template Proposition)**.** $\forall n \geq 1 : C(n-1) \to I(n)$ *is true (i.e.,* $I(n)$ *is a loop invariant that is true at the beginning of the loop and after termination).*

Template proof. We prove that $\forall n \geq 1 : C(n-1) \to I(n)$ is true by induction over $n \geq 1$.

Base Case: Verify $I(1)$ is true.

Inductive Hypothesis: Fix a $k \geq 1$ and suppose that $C(k) \wedge I(k)$ is true.

Inductive Step: Show that $C(k) \wedge I(k) \to I(k+1)$.

Then, by the law of (weak) induction, we have proven that $\forall n \geq 1 : C(n-1) \to I(n)$. ■

Now that we have discussed proving that a loop invariant is a loop invariant, we will want to use it to prove the correctness of an algorithm. This follows in three parts. First, you need to identify a loop invariant. Second, prove the loop invariant. Finally, use the loop invariant and, in particular, the one after the algorithm terminates, to prove the correctness of the algorithm. In order to do this last step, you will also need to prove that the algorithm terminates, i.e., that $\exists n \geq 0 : \neg C(n)$.

Remark 3.7. Often a loop invariant proof is phrased in three parts. Initialization, Maintenance, and Termination. The first two steps are equivalent to proving that a loop invariant is indeed a loop invariant. That is, they are the base case and the inductive step, respectively. Then the termination step has you prove that the algorithm terminates and then use the loop invariant to prove the correctness of the algorithm. While this structure is common, we won't use it explicitly in these notes.

To make this more clear and to finish this section, we give an example proof of correctness using loop invariants. For this, we consider the following simple algorithm.

Algorithm 3.8 (Linear Search Algorithm)**.**

```
1: procedure LinearSearch(A[1, ..., n], x)
2:     for k from 1 to n do
3:         if A[k] = x then
4:             return k
5:     return −1
```

We will prove the following.

Proposition 3.9. *The linear search algorithm, given in Algorithm 3.8, correctly returns an index* $k \in [n] = \{1, \ldots, n\}$ *such that* $A[k] = x$ *or* -1 *is no such index exists.*

Proof. We do this using the following loop invariant that is true before the loop and at the end of each loop, $I(k) : \forall i \in [k] = \{1, \ldots, k\} : A[i] \neq x$. In words, the subarray $A[1, \ldots, k]$ does not contain the element x anywhere. We also define the loop condition to be, $C(k) : k \leq n \wedge A[k] \neq x$. We then prove this is a valid loop invariant by induction over $k \geq 1$. That is, we prove that $\forall k \geq 1 : C(k) \to I(k)$.

Base Case: We first verify that $I(0) : \forall i \in [0] = \emptyset : A[i] \neq x$ is true. This is the case because the universal quantifier is vacuously true.

Inductive Hypothesis: Fix a $k \geq 0$ and suppose that $C(k+1) \wedge I(k)$ is true. That is $k+1 \leq n \wedge A[k+1] \neq x \wedge \forall i \in [k] = \{1, \ldots, k\} : A[i] \neq x$.

Inductive Step: We then show that the inductive hypothesis implies that $I(k+1) = \forall i \in [k+1] = \{1, \ldots, k+1\} : A[i] \neq x$ is true. This follows directly from the loop invariant and the loop condition.

$$\begin{aligned} k+1 \leq n \wedge A[k+1] \neq x \wedge \forall i \in [k] : A[i] \neq x &\to \forall i \in [k] : A[i] \neq x \wedge A[k+1] \neq x \\ &\to \forall i \in [k+1] : A[i] \neq x \end{aligned}$$

Then, by induction, we have proven that $\forall k \geq 1 : C(k) \to I(k)$, i.e., that $I(k)$ is a loop invariant, which is true at the end of the loop.

Finally, we prove the claim. In particular, the loop must terminate as we add one to k after each iteration, and so eventually, $k > n$ or $A[k] = x$. If $k > n$, then the loop invariant for $I(n)$ tells us that $x \notin A$, and thus the algorithm will correctly return -1. If $A[k] = x$, then the algorithm will terminate and return k, as it found such an index. ■

Remark 3.10. Interestingly, the loop invariant was primarily used to prove the correctness of the case where the algorithm returns -1. This is to say that the loop invariant had little to do with the exact statement of the proposition we were proving and instead implied the proposition with some extra work. Additionally, note that, while $A[k] \neq x$ wasn't in the loop directly, it is important for determining when the loop terminates. Therefore, we need to add it it the loop condition.

3.1 Problems/Exercises

Problem 3.1. Prove that the alternate linear search algorithm, given bellow, correctly returns an index $k \in [n] = \{1, \ldots, n\}$ such that $A[k] = x$ or -1 is no such index exists.

Algorithm (Alternate Linear Search Algorithm)**.**

```
1: procedure LinearSearch(A[1, ..., n], x)
2:     i ← −1
3:     for k from 1 to n do
4:         if A[k] = x then
5:             i ← k
6:     return i
```

Problem 3.2. Prove that the iterative version of binary search works correctly. That is, for any sorted (ascending) input array $A[1, \ldots, n]$ on n elements of $\mathbb{Z}$ and an element $x \in \mathbb{Z}$, the binary search algorithm given bellow will find an index $i \in [n] = \{1, \ldots, n\}$ such that $A[i] = x$ or if none exist then the algorithm gives -1.

Algorithm (Binary Search Algorithm)**.**

```
1: procedure BinarySearch(A[1, ..., n], x)
2:     ℓ ← 1
3:     r ← n
4:     while ℓ ≤ r do
5:         m ← ℓ + ⌊(r−ℓ)/2⌋
6:         if A[m] = x then
7:             return m
8:         else if A[m] < x then
9:             ℓ ← m + 1
10:        else
11:            r ← m − 1
12:    return −1
```

Problem 3.3. You are given a stack of origami paper, all of different sizes. You want to sort them by their size so that you have the largest paper on the bottom and the smallest on top. You can pick up only the top few papers, flip them, and then put them back on top of the pile. You have the idea to

1. Find the largest piece of paper, then flip it and the papers above it.
2. Flip the pile starting at the ith paper from the bottom; here i is the number of times you did this step.

3. Repeat steps (1) and (2) until the stack is sorted, but for (1), find the next largest paper.

Prove this algorithm terminates and that it correctly sorts the papers.

Additional Reading and Reference

For a more in-depth resource, Richard Hammacks's *Book of Proof* (Chapter 10) is great [Ham19]. Additionally, one of my past TAs, Michael Levet, has an excellent resource on mathematics in computer science [Lev23], the first chapter of which is on proof by induction.

Inspiration for earlier drafts of these notes came, in part, from these two sources.

[Ham19] Richard Hammack. *Book of Proof*. Richard Hammack, 2019. ISBN: 978-0-9894721-3-5. URL: https://www.people.vcu.edu/~rhammack/BookOfProof/. 2, 12

[Lev23] Michael Levet. *CSCI 3104 Algorithms - Lecture Notes*. Michael Levet, 2023. URL: https://michaellevet.github.io/Algorithms_Notes.pdf. 2, 12