CSCI 2011: Induction Proofs and Recursion

Chris Kauffman

Last Updated:
Thu Jul 12 13:50:15 CDT 2018

Logistics

Reading: Rosen

- Now: 5.1 - 5.5
- Next: 6.1 - 6.5

Assignments

- A06: Post Thursday
- Due Tuesday

Quiz Thursday

- Big-O Algorithm Analysis
- Number Theory and Modulo
- Encryption
  - Caesar, Vigenere
  - Maybe some RSA
- Basic Induction Proofs

Goals

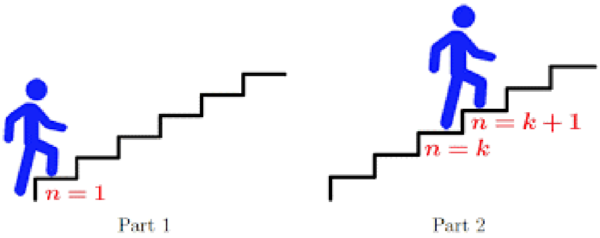
- Induction
- Recursive Structures
- Recursive Code

Principles of Mathematical Induction

- **Induction** is a proof technique based on the following principle

$$(P(1) \wedge \forall k P(k) \to P(k+1)) \to \forall n P(n)$$

- In English
  1. Show that $P(1)$ is true (**base case**)
  2. Show that if $P(k)$ is true for some value k, then $P(k+1)$ is also true (**inductive step**)
  3. Conclude that $P(n)$ is true for all positive integers n
- We will study applications induction to integers and also to **structures** such as trees which arise in CS



An Old Friend: Sum of 1 to n

Recall that we proved the following relation which has applications in algorithm analysis via a term pairing argument.

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

Will now show this via induction instead

Proof by Mathematical Induction

Base Case

- $n = 1$, have $\sum_{i=1}^{1} i = 1$ and $\frac{1(1+1)}{2} = 1$.
- So, both expressions equal 1, property holds at $n = 1$.

Induction Case

1. Assume for k that $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$
2. Show $\sum_{i=1}^{k+1} i = \frac{(k+1)((k+1)+1)}{2}$
3. Start with right side of equality and show equivalent to left

$$
\begin{aligned}
\frac{(k+1)((k+1)+1)}{2} &= \frac{(k+1)(k+2)}{2} && \text{Expand} \\
&= \frac{(k+1)\cdot k+(k+1)\cdot 2}{2} && \text{Distribute} \\
&= \frac{k(k+1)}{2} + (k+1) && \text{Divide} \\
&= \left(\sum_{i=1}^{k} i\right) + (k+1) && \textbf{Inductive Hypothesis (1)} \\
&= \sum_{i=1}^{k+1} i && \text{Def. of Summation}
\end{aligned}
$$

By Base/Inductive Cases, true for all positive integers. ■

Exercise: Inductive Proof for Sums of Odds

Notice that sums first n odd integers seem to follow a pattern

$$\begin{array}{rlrl} 1 & = 1 & 1+3+5+7 & = 16 \\ 1+3 & = 4 & 1+3+5+7+9 & = 25 \\ 1+3+5 & = 9 & 1+3+...+(2n-1) & = n^2 \end{array}$$

Use a Proof by Induction to show that

$$\sum_{i=1}^{n}(2i-1) = n^2$$

Clearly show both

- **Base Case**
  - Show property holds for $n = 1$
- **Inductive Step**
  - Assume fact $\sum_{i=1}^{k}(2i-1) = k^2$
  - Show that $\sum_{i=1}^{k+1}(2i-1) = (k+1)^2$

Answers: Inductive Proof for Sums of Odds

Base Case

- $n = 1$, have $\sum_{i=1}^{1}(2i - 1) = 1$ and $1^2 = 1$.
- So, both expressions equal 1, property holds at $n = 1$.

Induction Case

1. Assume fact $\sum_{i=1}^{k}(2i - 1) = k^2$
2. Show that $\sum_{i=1}^{k+1}(2i - 1) = (k + 1)^2$
3. Start with right side of equality and show equivalent to left

$$
\begin{array}{lll}
(k+1)^2 & = k^2 + 2k + 1 & \text{Expand} \\
 & = (\sum_{i=1}^{k}(2i-1)) + 2k + 1 & \textbf{IH (1)} \\
 & = (\sum_{i=1}^{k}(2i-1)) + 2(k+1) - 1 & \text{Rearrange} \\
 & = (\sum_{i=1}^{k+1}(2i-1)) & \text{Def. of Summation}
\end{array}
$$

By Base/Inductive Cases, true for all positive integers. ■

Exercise: Size of Power Set

- Recall the **power set** of A, $\mathcal{P}(A)$ is defined to be the set of all subsets of A
- For a finite set like $A = \{1, 2, 3\}$

$$\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

- The size of the power set seems to follow a pattern:

$$\begin{array}{llll}
|A| = 0 & |\mathcal{P}(A)| = 1 & |A| = 4 & |\mathcal{P}(A)| = 16 \\
|A| = 1 & |\mathcal{P}(A)| = 2 & |A| = 5 & |\mathcal{P}(A)| = 32 \\
|A| = 2 & |\mathcal{P}(A)| = 4 & \text{...} & \text{...} \\
|A| = 3 & |\mathcal{P}(A)| = 8 & |A| = N & |\mathcal{P}(A)| = 2^N
\end{array}$$

- Prove this relation with proof by induction.
- Note the base case is at $N = 0$ this time
- **Hint:** for A with $|A| = N$, add a new element a to get $|A \cup a| = N + 1$, consider all subsets of A with and without a

Answers: Size of Power Set (Base Case)

Show that for finite set A, IF $|A| = N$ THEN $|\mathcal{P}(A)| = 2^N$

Base Case

1. For $|A| = 0$, A must be the empty set $\emptyset$.
2. The only subset of $\emptyset$ is itself so $\mathcal{P}(\emptyset) = \{\emptyset\}$ which has 1 element.
3. So $|\mathcal{P}(A)| = 1$ and $2^0 = 1$ so the property holds at $N = 0$

Answers: Size of Power Set (Inductive Case)

Show that for finite set A, IF $|A| = N$ THEN $|\mathcal{P}(A)| = 2^N$

Inductive Case

1. Assume fact IF $|A| = k$ THEN $|\mathcal{P}(A)| = 2^k$
2. Show IF $|A| = k+1$ THEN $|\mathcal{P}(A)| = 2^{k+1}$
3. Let $A = S \cup \{a\}$ so that $|S| = k$
   - S is one element smaller than A
4. From (3), know that $\mathcal{P}(A) = \mathcal{P}(S \cup \{a\})$
5. To form $\mathcal{P}(S \cup \{a\})$, use $\mathcal{P}(S)$: If set $X \subseteq \mathcal{P}(S)$ then
   - $X \subseteq \mathcal{P}(S \cup \{a\})$
   - $X \cup \{a\} \subseteq \mathcal{P}(S \cup \{a\})$

   So, 2 subsets in $\mathcal{P}(A)$ for every 1 in $\mathcal{P}(S)$
6. By 5, $|\mathcal{P}(A)| = |\mathcal{P}(S)| \cdot 2$.
7. By **IH** (1), know that $|\mathcal{P}(S)| = 2^k$.
8. Combine (6)/(7) to get $|\mathcal{P}(A)| = 2^k \cdot 2 = 2^{k+1}$.

By Base/Inductive Cases, true for all positive integers. $\blacksquare$

Stronger Induction Assumptions

- Standard induction assumes $P(k)$ and shows $P(k+1)$ in the Inductive Step
- **Strong Induction** makes a stronger assumption
  - Assume $P(1) \wedge P(2) \wedge \cdots \wedge P(k)$
  - Show $P(k+1)$
- Comes in handy when one needs to "look back" farther

Fibonacci Growth and Strong Induction

- The **Fibonacci Numbers** are defined recursively as

$$\text{fib}(0) = 0,\ \text{fib}(1) = 1,\ \text{fib}(N) = \text{fib}(N-1) + \text{fib}(N-2)$$

- Show that $\text{fib}(N)$ is $O(2^N)$

Base Cases

1. $\text{fib}(0) = 0$ and $2^0 = 1$, dominated
2. $\text{fib}(1) = 1$ and $2^1 = 2$, dominated

Inductive Case

1. Strong Inductive Hypothesis: Assume **both** $\text{fib}(k)$ is $O(2^k)$ and $\text{fib}(k-1)$ is $O(2^{k-1})$
2. Show $\text{fib}(k+1)$ is $O(2^{k+1})$
3. By definition $\text{fib}(k+1) = \text{fib}(k) + \text{fib}(k-1)$
4. By **IH** (1) $\text{fib}(k+1)$ is then $O(2^k + 2^{k-1})$
5. Rearranging gets $O(2^k + 2 \cdot 2^{k-1} - 2^{k-1}) = O(2^{k+1} - 2^{k-1})$ which is $O(2^{k+1})$

Exercise: Warm-up

1. What is Mathematical Induction? What parts appear in a proof involving induction?
2. What is the difference between Standard Induction and **Strong** Induction?
3. What kind of object is particularly well-suited for Proofs by Induction?

Answers: Warm-up

1. What is Mathematical Induction? What parts appear in a proof involving induction?
   - Induction is a proof technique that allows a properties be proved for all objects of a certain kind
   - Has a **Base Case** where the "smallest" objects are shown to have the property
   - Has an **Induction Case** where it is assumed that a smaller object has the property and this leads to a slightly larger object having the property
2. What is the difference between Standard Induction and **Strong** Induction?
   - Standard Induction assumes only $P(k)$ and shows $P(k+1)$ holds
   - Strong Induction assumes $P(1) \wedge P(2) \wedge P(3) \wedge \cdots \wedge P(k)$ and shows $P(k+1)$ holds
   - Stronger because more is assumed but Standard/Strong are actually identical
3. What kind of object is particularly well-suited for Proofs by Induction?
   - Objects with recursive definitions often have induction proofs

Exercise: Fibonacci Lower Bound

Show that for $N \geq 3$,

$$\text{fib}(N) > \alpha^{N-2}$$

with $\alpha = \frac{1+\sqrt{5}}{2}$

- Use a proof by induction, strong hypothesis
- Multiple Base Cases to support strong induction
- Inductive Step exploits looking back by 2 fib numbers
- Use the fact that

$$\alpha^2 = \alpha + 1$$

Answer: Fibonacci Lower Bound

Show for $N \geq 3$ and $\alpha = \frac{1+\sqrt{5}}{2}$ with $\alpha^2 = \alpha + 1$, that

$$\mathrm{fib}(N) > \alpha^{N-2}$$

Base Case

1. $N = 3$, $\mathrm{fib}(3) = 2$ and $\alpha^{3-2} = \alpha^1 = 1.618\ldots$, check
2. $N = 4$, $\mathrm{fib}(4) = 3$ and $\alpha^{4-2} = \alpha^2 = 2.618\ldots$, check

Inductive Case

1. Strong IH: Assume facts

$$\begin{aligned} \mathrm{fib}(k) &> \alpha^{k-2} \\ \mathrm{fib}(k-1) &> \alpha^{k-3} \end{aligned}$$

2. Show $\mathrm{fib}(k+1) > \alpha^{k-1}$
3. By def of $\mathrm{fib}(N)$ and IH (1)

$$\begin{aligned} \mathrm{fib}(k+1) &= \mathrm{fib}(k) + \mathrm{fib}(k-1) \\ &> \alpha^{k-2} + \alpha^{k-3} \end{aligned}$$

4. Fact: $\alpha^2 = \alpha + 1$
5. RHS of (3) becomes

$$\begin{aligned} \alpha^{k-2} + \alpha^{k-3} &= (\alpha+1)\alpha^{k-3} \\ &= \alpha^2 \cdot \alpha^{k-3} \\ &= \alpha^{k-1} \end{aligned}$$

6. So $\mathrm{fib}(k+1) > \alpha^{k-1}$ ■

Exercise: Classes Scheduling

- A Classes Hall is open 09:00 (9am) to 17:00 (5pm)
- Professors have submitted classes they want to schedule
- Each submission has start/end times (s_i, e_i)
- Classroom management wants to **maximize** the number of classes offered
- Determine Max number of classessss that can be scheduled sample data
- What algorithm works for this? *Hint: Try sorting...*

Class#	Start	End
1	15	17
2	9	12
3	11	14
4	13	17
5	14	17
6	9	10
7	11	12
8	12	14
9	12	15
10	9	11
11	11	13
12	16	17
13	14	16
14	10	14

Greedy Approaches to Class Scheduling

- Sort by a start or end time
- Greedy selection: earliest non-conflicting class

Sort by Start Time

Class#	Start	End	
6	9	10	1
10	9	11	
2	9	12	
14	10	14	2
7	11	12	
11	11	13	
3	11	14	
8	12	14	
9	12	15	
4	13	17	
13	14	16	3
5	14	17	
1	15	17	
12	16	17	4

4 classes scheduled

Sort by End Time

Class#	Start	End	
6	9	10	1
10	9	11	
2	9	12	
7	11	12	2
11	11	13	
14	10	14	
3	11	14	
8	12	14	3
9	12	15	
13	14	16	4
4	13	17	
5	14	17	
1	15	17	
12	16	17	5

5 classes scheduled

Exercise: Greedy Algorithm for Class Scheduling

- Previous example suggests the following algorithm
- Analyze complexity and give worst case Big-O runtime
- Speculate: Is this algorithm correct?
  - YES: will always schedule the maximum # classes
  - NO: Some array T will result in fewer than maximum classes scheduled
- How would one prove correctness

```
select_classes(T[] : int pair array){
  # T are (start,end) time pairs
  sort(L) by end times      # ????
  S = empty list
  (prev_start, prev_end) = (-1,-1)
  append(S, L[0])           # ????
  for(i=1; i<length(L); i++){
     (cur_start,cur_end) = T[i]
     if(T[i] COMPATIBLE){   # ????
       append(S, L[i])      # ????
       (prev_start,prev_end) = T[i]
     }
  }
  return S : list of scheduled classes
}
```

Answers: Greedy Algorithm for Class Scheduling

- Fastest general purpose sorting algorithms are $O(N \log N)$
  - Quicksort, Mergesort, Heapsort, Timsort
- Appending to a list should be $O(1)$
- Compatibility check: one numerical comparison, constant time $O(1)$
- Total complexity: $O(N \log N)$
- Algorithm **is correct**, use a Proof by Induction

```
select_classes(T[] : int pair array){
  # T are (start,end) time pairs
  sort(L) by end times    # O(N log N)
  S = empty list
  (prev_start, prev_end) = (-1,-1)
  append(S, L[0])               # O(1)
  for(i=0; i<length(L); i++){
     (cur_start,cur_end) = T[i]
     if(cur_start >= prev_end){ # O(1)
       append(S, L[i])          # O(1)
       (prev_start,prev_end) = T[i]
     }  # O(1) work per iteration
  }     # O(N) work for loop
  return S : list of scheduled classes
} # N log N + N = O(N log N)
```

Correctness of `select_classes()`: **Base Cases**

Prove `select_classes()` algorithm

- Sort classes by end time
- Select earliest compatible classes

always schedules the maximum number of classes possible.
Proved by induction on the **maximum possible # classes** that can be scheduled.

Base Cases: Max Classes = 0 or 1

If there are no classes possible, the loop will not add any to the set and returns empty.
If there is only 1 talk possible, `select_classess()` picks the one which ends the earliest and adds it.

Correctness of `select_classes()`: **Induction Case**

1. Assume if k classess is the max possible, `select_classes()` schedules k classes (works correctly)
2. Show if $k+1$ classes is the max possible, `select_classes()` will schedule $k+1$ classes.
3. `select_classes()` sorts classes by end time: $e_1 \leq e_2 \leq \cdots \leq e_N$
4. Suppose classes with end time e_i is the earliest ending classes among the $k+1$ max classes possible.
5. Replace e_i with classes with end time e_1 which is what `select_classes()` picks first.
6. e_1 has an earlier or equal end time to e_i so still possible to schedule remaining k talks.
7. By IH (1), `select_classes()` works correctly when k classes is the max.

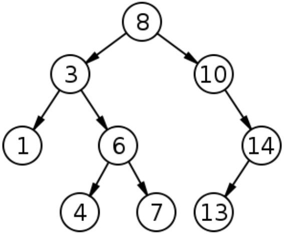
By proof of Base and Induction Cases, `select_classes()` is correct for all possible #'s of maximum classes. ■

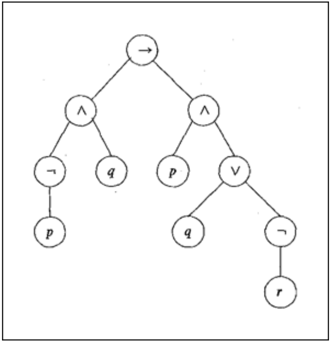
Recursively Defined Structures

- Can define a variety of objects recursively
- Some of these are numeric such as sets of integers
  - 3 is in S, if x and y are in S, $x + y$ is in S
- Others lack numerical description
  - Binary Trees
  - Logical Formulas like

$$(((\neg p) \wedge q) \rightarrow (p \wedge (q \vee (\neg r))))$$

- Note the **parse tree** for the above logic formula to the right





Structural Induction

Structural Induction is used on these objects to prove properties about them.

- Base Case deals with initial set of objects, shows property holds for all of them
- Induction Case deals with recursive definition to build up objects, shows that combining smaller objects maintains the property

Exercise: Even Parentheses

Define **well-formed logical formulas** recursively:

Base Cases

- The symbols **T**, **F** are well-formed
- Any single variables such as p or q is well-formed

Recursive Cases

If E and F are well-formed, then the following combinations are also well-formed, all of which are parenthesized

- Negation: $(\neg E)$
- And: $(E \wedge F)$
- Or: $(E \vee F)$
- Implies: $(E \rightarrow F)$

Prove that well-formed logic formulas have an **even number of parentheses**

Base Cases

1. ???
2. ???

Induction Cases

1. Assume E, F are **smaller** formulas which have k logical connectives and have an even number of parentheses
2. Show **larger** formulas with $k+1$ connective symbols created from E, F have an even number of parentheses
3. ???

Answers: Even Parentheses Base Cases

Prove that well-formed logic formulas have an **even number of parentheses**

Base Cases

1. **T** and **F** have 0 parentheses, even
2. Single variables like p have 0 parentheses, even

Answers: Even Parentheses Induction Cases

Prove that well-formed logic formulas have an **even number of parentheses**

Induction Cases

1. Assume E, F are **smaller** formulas which have k logical connectives and have an even number of parentheses
2. Show **larger** formulas with $k + 1$ connective symbols created from E, F have an even number of parentheses
3. Each way of combining symbols introduces 2 parentheses
   - 2 for Negation: $(\neg E)$
   - 2 for And: $(E \wedge F)$
   - 2 for Or: $(E \vee F)$
   - 2 for Implies: $(E \rightarrow F)$
4. By IH (1), E, F both have even # parentheses so adding them and 2 more keeps the total even.

By combination of Base and Induction Cases, all well-formed formulas have an even # of parenthesis. ■

Full Binary Trees

- Binary trees are special kinds of graphs where each vertex (node) has at most two edges (connections) to other vertices (nodes) and no cycles are formed
- **Full Binary Trees** are binary trees in which each node has a 0 or 2 children
- The set of full binary trees can be recursively defined as follows

Base Case

A single node r is a full binary tree

Recursive Case

If T_L and T_R are both full binary trees, a new full binary tree is formed by creating a root r with left child T_L and right child T_R.

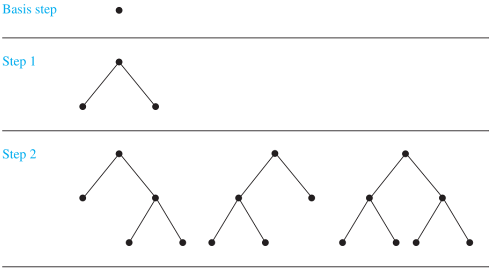


FIGURE 4 Building Up Full Binary Trees.

Exercise: Height of Binary Trees

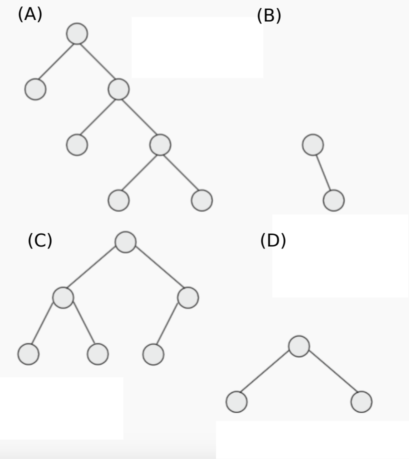
The **Height** of a binary tree is defined recursively as

- **Base Cases:** The height of the empty tree $h(\emptyset) = 0$, height of a single node/root $h(r) = 1$
- **Recursive Case:** The height of tree T with root r, left child tree T_L and right child tree T_R is

$$h(T) = 1 + \max(h(T_L), h(T_R))$$

Give a non-recursive description of the meaning of height involving the root of the tree and its leaves.

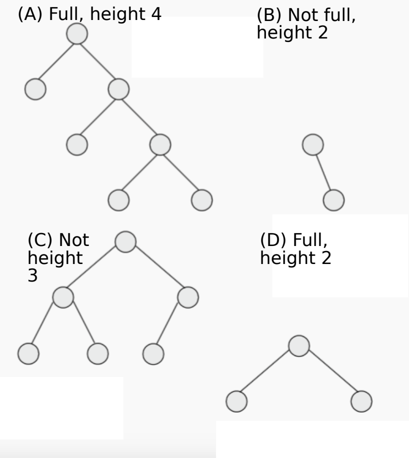
- What are the heights of the following binary trees?
- Which are full binary trees?



Answers: Height of Binary Trees

Non-recursive definition of height: Number of nodes on longest path from root to leaf.
Note: Height definitions vary

- Lecture will use height of 1 for a single node
- Textbook and others use height of 0 for a single node



Exercise: Number of Nodes in a Tree

Give a recursive definition for the number of nodes in a tree called $n(T)$

- Base Case(s)
- Recursive Case(s)

Exercise: Number of Nodes in a Tree

Give a recursive definition for the number of nodes in a tree called $n(T)$

Base Cases

The empty tree has $n(\emptyset) = 0$ nodes
(Maybe) the single node tree r has $n(r) = 1$ nodes

Recursive Case

If T_L and T_R are binary trees, then the larger tree T with root r and T_L, T_R as left/right child trees has number of nodes

$$n(T) = 1 + n(T_L) + n(T_R)$$

Exercise: Height of Full Binary Trees

Prove that the number of nodes $n(T)$ is bounded by the height $h(T)$ for **full binary trees** according to the following formula.

$$n(T) \leq 2^{h(T)} - 1$$

Use a **Proof by Induction** on the structure of Full Binary Trees.

- **Base Case:** Single node tree is full.
- **Induction Case:** Full tree formed from two other full trees.

Answers: Height of Full Binary Trees Base Case

Prove that the number of nodes $n(T)$ is bounded by the height $h(T)$ for **full binary trees** according to the following formula.

$$n(T) \leq 2^{h(T)} - 1$$

Base Case

For the single node full binary tree we have

- $n(T) = 1$ (single node)
- $h(T) = 1$ (one node on path from root to leaf)

Plugging into the formula gives

$$1 \leq 2^1 - 1$$

$$1 \leq 2 - 1$$

$$1 \leq 1$$

Answers: Height of Full Binary Trees Induction Case

Prove that the number of nodes $n(T)$ is bounded by the height $h(T)$ for **full binary trees** according to the following formula.

$$n(T) \leq 2^{h(T)} - 1$$

Induction Case

1. Assume bound holds for smaller trees T_L and T_R
   - $n(T_L) \leq 2^{h(T_L)} - 1$
   - $n(T_R) \leq 2^{h(T_R)} - 1$
2. Show bound holds for larger tree T formed by joining root r to left/right child trees T_L, T_R, that

$$n(T) \leq 2^{h(T)} - 1$$

3. Start with number of nodes and derive the following

$$\begin{aligned}
n(T) &= 1 + n(T_L) + n(T_R) && \text{Def of } n(T) \\
&\leq 1 + 2^{h(T_L)} - 1 + 2^{h(T_R)} - 1 && \text{by IH (1)} \\
&\leq 2^{h(T_L)} + 2^{h(T_R)} - 1 && \text{simplify} \\
&\leq 2 \cdot \max(2^{h(T_L)}, 2^{h(T_R)}) - 1 && \text{Def of Max} \\
&= 2 \cdot 2^{\max(h(T_L), h(T_R))} - 1 && \text{Def of Exp.} \\
&= 2 \cdot 2^{(1+\max(h(T_L), h(T_R)))-1} - 1 && \text{+1 and -1} \\
&= 2 \cdot 2^{h(T)-1} - 1 && \text{Def of } h(T) \\
&= 2^{h(T)} - 1 && \text{Simplify}
\end{aligned}$$

By combination of Base and Induction Cases, node count holds. ■

Exercise: Iterative and Recursive Fibonacci

- Recall the Fibonacci numbers
- Below are two code implementations of them
- Which is Recursive and which is Iterative?
- Which has better big-O runtime?
- Which is easier to prove **correct**?
  - How would one prove correctness...)

```
int fib(int n){
  if(n==0){
    return 0;
  }
  if(n==1){
    return 1;
  }
  else{
    int tmp1 = fib(n-1);
    int tmp2 = fib(n-2);
    return tmp1+tmp2;
  }
}
```

```
int fib(int n){
  int fi = 0;
  int f2 = 0;
  int f1 = 1;
  for(int i=0; i < n; i++){
    f2 = f1;
    f1 = fi;
    fi = f1 + f2;
  }
  return fi;
}
```

Answers: Iterative and Recursive Fibonacci

- Which is Recursive and which is Iterative?
  - Left Recursive, Right Iterative
- Which has better big-O runtime?
  - Iterative **much** more efficient as it avoids redundant computations
- Which is easier to prove **correct** - via Induction
  - Recursive: easy follows the definition of Fibonacci very closely
  - Iterative: harder, perhaps proof by induction on iteration count

```
int fib_recursive(int n){
  if(n==0){
    return 0;
  }
  if(n==1){
    return 1;
  }
  else{
    int tmp1 = fib(n-1);
    int tmp2 = fib(n-2);
    return tmp1+tmp2;
  }
}
```

```
int fib_iterative(int n){
  int fi = 0;
  int f2 = 0;
  int f1 = 1;
  for(int i=0; i < n; i++){
    f2 = f1;
    f1 = fi;
    fi = f1 + f2;
  }
  return fi;
}
```

Proof of Correctness for Recursive Fibonacci

Base Case

- For fib(0) returns 0 - check
- For fib(1) returns 1 - check

Induction Case

1. Assume `fib(n-1)` and `fib(n-2)` are correct
2. Show fib(n) is correct
3. Lines 9 and 10 return correct answers
4. Combined correctly according to definition of `fib(n)` in line 11.

```
1  int fib_recursive(int n){
2    if(n==0){
3      return 0;
4    }
5    if(n==1){
6      return 1;
7    }
8    else{
9      int tmp1 = fib(n-1);
10     int tmp2 = fib(n-2);
11     return tmp1+tmp2;
12   }
13 }
```

By combination of Base and Induction cases, implementation works correctly. ■

Proof of Correctness of Iterative Fibonacci

Much trickier as only partly follows definition of sequence. **General strategy** is along the following lines

- Show correct for cases of `n=0, n=1`
- For `n >= 2`, prove at the end of loop iteration `i` at line 9, variables hold specific values
  - `f2` is fib($i-2$)
  - `f1` is fib($i-1$)
  - `fi` is fib(i)
- Induction on `i` with base case `i=2` and induction shows updates to vars `f2,f1,fi`

```
 1  int fib_iterative(int n){
 2    int fi = 0;
 3    int f2 = 0;
 4    int f1 = 1;
 5    for(int i=0; i < n; i++){
 6      f2 = f1;
 7      f1 = fi;
 8      fi = f1 + f2;
 9    }
10    return fi;
11  }
```

Formal Methods is the study of automatically proving correctness of code. UMN has very strong researchers in this area.