CS 170 Tutorial #1

Invariants and Proofs of Correctness
Why do we use induction?

Property $P(n)$: sum of first $n$ natural numbers is $1/2*n*(n+1)$
We want to prove $P(n)$ for all natural numbers $n$.

Strategy:
Prove $P(0)$, $P(1)$, $P(2)$, …

Better Strategy:
Use induction!
Choose Induction Hypothesis to be $P$
1. Base case: Prove $P(0)$
2. Induction case: $P(k) \Rightarrow P(k+1)$
Reasoning about algorithms with loops

```
x = c; y = 0;
while (x > 0) {
    x--;  
    y++;  
}
```

**Property:** y equals c after the loop terminates

**Strategy:**
Compute state after iteration #1, iteration #2, ...
Prove that state after last iteration has y = c

**Better Strategy:**
Use induction (over number of iterations)
- **Base case:** Prove induction hypothesis holds on loop entry
- **Induction case:** Assuming induction hypothesis holds after k iterations, prove it holds after k+1 iterations
Step 1: Construct an Inductive Hypothesis

We can generalize from examples...

- On loop entry: $x = c$, $y = 0$
- After iteration 1: $x = c - 1$, $y = 1$
- After iteration 2: $x = c - 2$, $y = 2$
- ...
Step 2: Prove that Loop Invariant is Inductive

1. Base case: loop invariant \( x + y = c \) holds on loop entry
   \[ \text{True} \]

2. Inductive case:
   Assume loop invariant holds after \( k \) iterations:
   \[ y = k, \quad x = c - y = c - k \]
   After the \((k+1)\)st iteration, \( y = k + 1, \quad x = c - k - 1 \)
   Therefore, \( x + y = k + 1 + c - k - 1 = c \)
   \[ \text{True} \]
Step 3: Proving correctness property using loop invariant

```c
x = c; y = 0;
while (x > 0) {
    x--;  
    y++; 
}
```

- Use loop invariant to prove correctness property that $y = c$ after loop terminates

After final iteration: $x = 0$

We also know our loop invariant holds: $x + y = c$

Therefore, $y = c$. 
Practice Problems

• Divide into groups of 2-3
Problem 1

Consider the following piece of code:

```java
y = 0;
for (i = 0; i <= n; i++) {
    y += 2^i;
}
return y;
```

What is the value of `y` after the loop termination?

(Hint: Try to find a loop invariant that holds at the start of each loop iteration)
Aside: For loops

```plaintext
for (i = 0; i <= n; i++) {
    // invariant: I(i) is true
    ... loop body ...
}

is equivalent to:

i := 0
loopstart:
    // invariant: I(i) is true
    if i > n: goto end
    ... loop body ...
    i := i+1
goto loopstart
end
```
Step 1 : Run a few iterations

```c
y = 0;
for (i = 0; i <= n; i++) {
    y += 2^i;
}
```

At the start of each iteration:
- $i = 0 : y_0 = 0$
- $i = 1 : y_1 = 1$
- $i = 2 : y_2 = 1 + 2 = 3$
- $i = 3 : y_3 = 1 + 2 + 4 = 7$
- $i = 4 : y_4 = 1 + 2 + 4 + 8 = 15$
- ...

Any pattern?
Step 1 : Run a few iterations

```java
y = 0;
for (i = 0; i <= n; i++) {
    y += 2^i;
}
```

At the start of each iteration:

- $i = 0 : y_0 = 0 = 2^0 - 1$
- $i = 1 : y_1 = 1 = 2^1 - 1$
- $i = 2 : y_2 = 1 + 2 = 3 = 2^2 - 1$
- $i = 3 : y_3 = 1 + 2 + 4 = 7 = 2^3 - 1$
- $i = 4 : y_4 = 1 + 2 + 4 + 8 = 15 = 2^4 - 1$
Step 1: Run a few iterations

\[
y = 0;
\text{for} \ (i = 0; i <= n; i++) \{
    y += 2^i;
\}
\]

At the start of each iteration:

- \( i = 0 : y_0 = 0 = 2^0 - 1 \)
- \( i = 1 : y_1 = 1 = 2^1 - 1 \)
- \( i = 2 : y_2 = 1 + 2 = 3 = 2^2 - 1 \)
- \( i = 3 : y_3 = 1 + 2 + 4 = 7 = 2^3 - 1 \)
- \( i = 4 : y_4 = 1 + 2 + 4 + 8 = 15 = 2^4 - 1 \)

It looks like \( y_i = 2^i - 1 \) is a good candidate for loop invariant
Step 2: Prove that loop invariant is inductive

- **Base case**
  \[ i = 0 : y_0 = 2^0 - 1 = 0 \quad \checkmark \]

- **Inductive step**
  Assume that at the start of the \( i \)-th iteration \( y_i = 2^i - 1 \)
  Then, at the start of the \((i+1)\)-th iteration we will have:
  \[ y_{i+1} = y_i + 2^i = 2^i - 1 + 2^i = 2 \times 2^i - 1 = 2^{i+1} - 1 \quad \text{Q.E.D.} \]
Step 3: Loop invariant at the last iteration

- When the loop terminates $i = n + 1$. Thus after the loop execution we have:
  
  $$y = 2^{n+1} - 1$$
Problem 2: Binary Search
You’ve all seen this a billion times.

But how do we prove that it’s correct?

Given that A is sorted and A contains target, prove that `binary_search(A, target)` always returns target’s index within A

Use Loop Invariants!!
Step 1: Hypothesize a Loop Invariant

```python
def binary_search(A, target):
    lo = 0
    hi = len(A) - 1
    while lo <= hi:
        mid = (lo + hi) / 2
        if A[mid] == target:
            return mid
        elif A[mid] < target:
            lo = mid + 1
        else:
            hi = mid - 1
```

Say we’re searching for 14 in the following array A

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-5</td>
<td>10</td>
<td>14</td>
<td>33</td>
<td>42</td>
<td>42</td>
</tr>
</tbody>
</table>

1\textsuperscript{st} step: lo = 0, hi = 6, mid = 3

2\textsuperscript{nd} step: lo = 0, hi = 2, mid = 1

3\textsuperscript{rd} step: lo = 2, hi = 2, mid = 2

At each step of the while loop, lo and hi \textit{surrounded} the actual location of where 14 is! This was always true!

\textbf{THIS IS OUR LOOP INVARIANT.}
Step 1: Construct Loop Invariant

At each iteration of the while loop, \( \text{lo} \) and \( \text{hi} \) are such that:

\[
A[\text{lo}] \leq \text{target} \leq A[\text{hi}]
\]
Step 2: Prove that loop invariant is inductive

• Base Case: when the algorithm begins, $lo = 0$ and $hi = \text{len}(A) - 1$. $lo$ and $hi$ enclose ALL values, so \text{target} must be between $lo$ and $hi$.

• Inductive Hypothesis: suppose at any iteration of the loop, $lo$ and $hi$ still enclose the \text{target} value.

• Inductive Step:
  – Case 1: If $A[mid] > \text{target}$, then the target must be between $lo$ and $mid$
    • We update $hi = mid - 1$
  – Case 2: If $A[mid] < \text{target}$, then the target must be between $mid$ and $hi$
    • we update $lo = mid + 1$
  – In either cases, we preserve the inductive hypothesis for the next loop
Step 3: Prove correctness property using loop invariant

• Notice for each iteration, \( \text{lo} \) always increases and \( \text{hi} \) always decreases. These value will converge at a single location where \( \text{lo} = \text{hi} \).

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
-5 & 10 & 14 & 33 & 42 & 42 & 42 \\
\end{array}
\]

• By the induction hypothesis, \( A[\text{lo}] \leq \text{target} \leq A[\text{hi}] \).

Food for thought: How will the proof change if \( \text{target} \) isn’t in the array?
Problem 3: array reversal
In-place Array Reversal

//inputs: array A of size n
void reverse_array(int *A, int n):
    int i = (n - 1) / 2;
    int j = n / 2;
    int tmp;
    while (i >= 0 && j <= (n - 1))
        tmp = A[i];
        A[i] = A[j];
        A[j] = tmp;
        i--;
        j++;

Prove that array A of size n is reversed as a result of invoking reverse_array(A, n)
Step 1: Hypothesize a Loop Invariant

Before iteration of the while loop, i and j are such that:

**A[i+1 : j-1] is reversed**

Or more formally,

new_A[i+1 : j-1] = reverse(old_A[i+1 : j-1])

where,

reverse([]) = []
reverse([a0]) = [a0]
reverse([a0, a1, ...]) = [reverse([a1,...]), a0]
Step 2: Prove that loop invariant is inductive

• Loop invariant: $A[i+1 : j-1]$ is reversed

• Base Case: Upon loop entry, $j - 1 < i + 1$. Invariant holds trivially.

• Inductive case:
  At the start of k-th iteration, assume that $A[i+1 : j-1]$ is reversed.
  Therefore, at the start of (k+1)-th iteration, we can prove that $A[i+1 : j-1]$ is reversed.
Step 3: Prove correctness property using loop invariant

- After the loop terminates, i = -1 and j = n.
- Loop invariant tells us that A[i+1 : j-1] is reversed.
- Therefore, A[0:n-1] is reversed.

QED