Recall our recursive multiply algorithm:

```
PRECONDITION:
  x and y are both binary bit arrays of length n,
n a power of 2.
POSTCONDITION:
  Returns a binary bit array equal to
  the product of x and y.

 REC_MULTIPLY_2( x, y):
  if (len(x) == 1):
     return (x[0]*y[0])

  xl = x[n/2:n]
  xh = x[0:n/2]
  yl = x[n/2:n]
  yh = x[0:n/2]

   p1 = REC_MULTIPLY_2(xh, yh)
   p2 = REC_MULTIPLY_2(xh+xl, yh+yl)
   p3 = REC_MULTIPLY_2(xl, yl)

  p2 = BINARY_ADD(p2,-p1, -p3)
  p2 = SHIFT(p2, n/2)
  p1 = SHIFT(p1,n)
  return BINARY_ADD(p1,p2,p3)
```

Let's prove that `REC_MULTIPLY_2(x,y)` does indeed return the *product* of x and y.

Proof by complete induction.

1. **Define** $P(n)$:

   RTP: $P(n)$ is true for all even $n \in \mathbb{N}$ and $n = 1$.

2. **Base Case:**

3. **Inductive Hypothesis:**

4. **Inductive Step:**

Another Recursive Algorithm–Quicksort

PRECONDITION:
A is an array/list of integers.
POSTCONDITION:
Returns the integers of A sorted in increasing order.

```
def QUICKSORT(A):
  if (len(A)==1 or len(A)==0):
    return (A)
  else:
    # make A[0] the pivot
    L, M, U = [], [], []
    for value in A:
      if (A[0] < value):
        L.append(value)
      elif (A[0] == value):
        M.append(value)
      else:
        U.append(value)
    final = QUICKSORT(L)
    final.append(M)
    final.append(QUICKSORT(U))
    return(final)
end Quicksort
```

Q: Which type of *induction* should we use to prove that `Quicksort` is correct?

Correctness of Quicksort

Proof. By *complete induction* on $n =$ `len(A)`.

1. **Define $P(n)$:**

2. **IH**: .

3. **Case 1.** $n = 0$ or $n = 1$

4. **Case 2.** $n \geq 2$

$\square$

Correctness of Iterative Algorithms

Q: What is an *iterative algorithm*?

Typical Iterative Algorithm Structure

```
Precondition:  Requirements about the input.
Postcondition:  Requirements about the output.

variable declarations
various statements
begin loop
   more statements
   ⋮
   exit statement
   more statements
end loop
more statements
```

Q: Which part of the algorithm is the hardest to prove *correct*?

To prove an algorithm is correct, we need to

- prove that the *looping* portion has a *well defined behavior*
- and that this *behavior* ensures that the *postcondition* is *met*.

The *well defined behavior* is called a *loop invariant*.

Q: How do we express a *loop invariant*?

Q: How does the *postcondition* relate to $P(n)$?

Q: Does this remind you of something else we have seen?

.

Keys to Proving an Iterative Algorithm Correct

We will use a *3-step* process.

1. **Partial Correctness**

2. **Proof of Termination**

3. **Total Correctness**

A Toy Example

PRECONDITION: Input is a natural number **x**.
POSTCONDITION: Output is 2^x.

```
def POWER( x ):
  current = 1
  count = 0
  while ( count < x ):
    current = current*2
    count = count+1
  return(current)
```

Q: What is the *exit condition*?

Q: What is the *invariant*? ie., what is true *each iteration* of the *loop*?

Let's look at a few iterations of the loop to find a pattern:

n	P(n)	
0	current =	count =
1	current =	count =
2	current =	count =
3	current =	count =
4	current =	count =
⋮		
i	current =	count =

Notation:

We will represent the *value* of a variable x during the k^{th} *iteration* of the loop by x_k.

What is the *loop invariant* $P(k)$?

$P(k)$:

Q: How is this related to the *postcondition*?

Partial Correctness: *Prove* $P(k)$ *true*.

Proof.

Base Case:

Inductive Hypothesis

Inductive Step

If there is not an $(i+1)^{st}$ iteration then $P(i+1)$ is trivially true. why??.

Otherwise, there exists an $(i+1)^{st}$ iteration:

$$
\begin{aligned}
current_{i+1} &= \\
&= \\
&=
\end{aligned}
\qquad\qquad
\begin{aligned}
count_{i+1} &= \\
&=
\end{aligned}
$$

Therefore $P(i)$ holds for all $i \in \mathbb{N}$.

Showing Termination

Theorem 2.5 (in the notes) *Every decreasing sequence of natural numbers is finite.*

Q: How does Theorem 2.5 follow from the *Well Ordering Principle*?

- Consider defining $d_i = x$ - count$_i$.
- What do we know about d_i versus d_{i+1}:
- How does the *exit condition* relate to d_i?
- How do we kow that the *loop* must *terminate*?

Q: Given that the *loop invariant* holds and that the *loop terminates*, is the *post condition* met when the *exit condition* is *true*?

□

To show *termination* define a *decreasing sequence* of *natural* numbers and use the *W.O.P.*

Multiplication – Take 2

PRECONDITION: $m \in \mathbb{N}, n \in \mathbb{Z}$.
POSTCONDITION: Returns the value $m \cdot n$.

```
MULTIPLY( m, n )
    int x = m;
    int y = n;
    int z = 0;
    while (x ≠ 0)
      if (x mod 2 = 1)
        z = z+y;
      x = x div 2;
      y = y · 2;
    return z;
```

Q: Why does **MULTIPLY**(n,m) work?

Consider:

- $x \cdot y = y + y + y + \cdots + y$ (x times)
- If x is even then $x \cdot y = 2(y + y + y + \cdots + y)$ ($x/2$ times)
- If x is odd then $x \cdot y = \ldots$
- Once $x = 0$, $xy =$

Q: Why might we want to use such an algorithm to multiply?

For *correctness*, we need to prove *three* things:

1.

2.

3. .

Partial Correctness

Q: Which variables would we *expect* to partake in the *loop invariant*?

Loop Invariant: $P(i)$: "*If the i^{th} iteration exists then*

$$mn = z_i + x_i y_i$$

Q: Why do we *believe* this loop invariant?

Claim: $P(i)$ is true for all $i \in \mathbb{N}$.

Proof.

1. **Base Case:**

2. **Induction Hypothesis:** Assume that $P(i)$ is true for *arbitrary* $i \in \mathbb{N}$.

3 **Induction Step:** **RTP:** $P(i+1)$ is true.

Assume that the $(i+1)^{st}$ iteration exists:

- Since $P(i)$ is true:

- If $x_i \mod 2 \neq 1$ then:

  $z_{i+1} =$

  $x_{i+1} =$ $y_{i+1} =$

  Therefore, $z_{i+1} =$

- If $x_i \mod 2 = 1$ then:

  $z_{i+1} =$

  $x_{i+1} =$ $y_{i+1} =$

  Therefore,

$$z_{i+1} = $$

$$= $$

$$= $$

Therefore, $P(i)$ holds for all $i \in \mathbb{N}$. $\square$

Termination

Q: How can we show that the loop *terminates*?

Q: What is such a *sequence*?

Claim: The loop will terminate.

Proof.