CSC236, Summer 2004, Assignment 4, sample solutions

1. Use the first-order language of arithmetic defined in Exercise 1, Course Notes page 183, to construct the formulas below, assuming that the domain $D = \mathbb{N}$, the natural numbers. You are also free to use the additional formula $Prime(x)$, defined in Exercise 2, to express the predicate "x is prime."

   (a) Give a formula that expresses "there are infinitely many primes."

   $$\forall x \exists y (L(x,y) \wedge Prime(y))$$

   (b) Give a formula that expresses "there are infinitely many composite numbers."

   $$\forall x \exists y (L(x,y) \wedge \neg Prime(y))$$

   (c) Give a formula that expresses "$x^2 + y^2 = z^2$."

   $$\forall v \forall w \forall w ((P(x,x,v) \wedge P(y,y,w) \wedge P(z,z,u)) \rightarrow S(v,w,u))$$

   (d) Give a formula that expresses "there are k consecutive composite numbers."

   $$\exists u \exists v \exists w (L(0,u) \wedge S(u,k,v) \wedge S(v,1,w) \wedge \forall x((L(u,x) \wedge L(x,w)) \rightarrow \neg Prime(x))$$

   (e) Give a formula that expresses "Any natural-number power of 11 equals 1 mod 10."
   I define constants $\mathbf{10}^{\mathcal{N}} = 10$ and $\mathbf{11}^{\mathcal{N}} = 11$

   $$\forall x (\forall y \forall t (P(y,t,x) \rightarrow (\approx (y, \mathbf{1}) \vee \approx (y, \mathbf{11}))) \rightarrow \exists u \exists v (P(u, \mathbf{10}, v) \wedge S(v,1,x)))$$

2. State whether each formula is valid or not. Prove your claim.

   (a)
   $$\forall x \exists y\, (\forall y F(x,y) \rightarrow \exists x M(x,y)) \leftrightarrow \exists y \forall x\, (\forall y F(x,y) \rightarrow \exists x M(x,y))$$

   Claim: The formula is valid.

   Proof: I will show, using the logical equivalences from 6.6 (pages 160–163), that the formula on the left-hand side of the biconditional (LF) is logically equivalent to the one on the right (RF). This (by definition) means that $LF \leftrightarrow RF$ is valid.

[rename variables]	$\forall x \exists y\, (\forall y F(x,y) \rightarrow \exists x M(x,y))$	LEQV	$\forall x \exists y\, (\forall u F(x,u) \rightarrow \exists w M(w,y))$
	[factor quantifier]	LEQV	$\forall x\, (\forall u F(x,u) \rightarrow \exists y \exists w M(w,y))$
	[factor quantifier]	LEQV	$(\exists x \forall u F(x,u) \rightarrow \exists y \exists w M(w,y))$
	[factor quantifier]	LEQV	$\exists y\, (\exists x \forall u F(x,u) \rightarrow \exists w M(w,y))$
	[factor quantifier]	LEQV	$\exists y \forall x\, (\forall u F(x,u) \rightarrow \exists w M(w,y))$
	[rename variables]	LEQV	$\exists y \forall x\, (\forall y F(x,y) \rightarrow \exists x M(x,y))$

   Thus LF LEQV RF, which (by definition) means $LF \leftrightarrow RF$ is a tautology. QED.

(b)

$$\forall x \exists y\,(\forall y F(x,y) \rightarrow \exists y M(x,y)) \leftrightarrow \exists y \forall x\,(\forall y F(x,y) \rightarrow \exists y M(x,y))$$

CLAIM: If F is a first-order formula in which variable x does not appear, then $\forall x F$ is logically equivalent to F, and both are logically equivalent to $\exists x F$. Furthermore, both are equivalent to $\exists x F$.

PROOF: Let (S, σ) be an interpretation containing the predicate F, and let v be an arbitrary element of D (D is non-empty by definition). Then F is true in (S, σ) if and only if F is true in $(S, \sigma|^x_v)$, since mapping x to v has no effect on F. Since v is an arbitrary element of D, the truth value of F is the same as the truth value of $\forall x F$. Since the interpretation (S, σ) is arbitrary, F is logically equivalent to $\forall x F$. Furthermore, $\exists x F$ is logically equivalent to (by double negation rule) $\neg\neg\exists x F$, which is logically equivalent to (by negation of quantifier rule) $\neg\forall x \neg F$, which is logically equivalent to (by the first part of this claim) $\neg\neg F$, which is logically equivalent to (double negation) F. QED.

CLAIM: Let $LF = \forall x \exists y\,(\forall y F(x,y) \rightarrow \exists y M(x,y))$ and $RF = \exists y \forall x\,(\forall y F(x,y) \rightarrow \exists y M(x,y))$. Then LF LEQV RF, so (by definition) $LF \leftrightarrow RF$ is a valid formula.

PROOF: Apply the logical equivalence rules from page 160:

$$\begin{array}{rlll}
\text{[renaming rule]} & \forall x \exists y\,(\forall y F(x,y) \rightarrow \exists y M(x,y)) & \text{LEQV} & \forall x \exists y\,(\forall u F(x,u) \rightarrow \exists u M(x,u)) \\
& \text{[Claim proved above]} & \text{LEQV} & \forall x\,(\forall u F(x,u) \rightarrow \exists u M(x,u)) \\
& \text{[Claim proved above]} & \text{LEQV} & \exists y \forall x\,(\forall u F(x,u) \rightarrow \exists u M(x,u)) \\
& \text{[renaming rule]} & \text{LEQV} & \exists y \forall x\,(\forall y F(x,y) \rightarrow \exists y M(x,y)).
\end{array}$$

Thus LF LEQV RF, so $LF \leftrightarrow RF$ is a valid formula. QED.

(c)

$$(\forall x F(x,y) \vee \forall x M(y,x)) \leftrightarrow \forall x (F(x,y) \vee M(y,x))$$

CLAIM: The formula is not valid. Let S be the structure where $D = \mathbb{N}$, the natural numbers, $F(x,y)$ be interpreted as $x < y$, $M(y,x)$ be interpreted as $x \geq y$, and let $\sigma(y) = 5$. In interpretation (S, σ) $\forall x F(x,y)$ is false, since $F(x,y)$ is false under $\sigma|^x_7$. Also, $\forall x M(y,x)$ is false, since $M(y,x)$ is false under $\sigma|^x_3$. Thus, in interpretation (S, σ), the disjunction $(\forall x F(x,y) \vee \forall x M(y,x))$ is false. On the other hand, in interpretation (S, σ) every $x \in D$ is either less than 5 or no less than 5, so the quantified disjunction $\forall x (F(x,y) \vee M(y,x))$ is true. This exhibits an interpretation that falsifies LF but satisfies RF, so (by definition) $LF \leftrightarrow RF$ is not valid. QED.

(d)

$$(\forall x F(x,y) \wedge \forall x M(y,x)) \leftrightarrow \forall x (F(x,y) \wedge M(y,x))$$

CLAIM Let $LF = (\forall x F(x,y) \wedge \forall x M(y,x))$ and $RF = \forall x (F(x,y) \wedge M(y,x))$. Then LF LEQV RF, so $LF \leftrightarrow RF$ is a valid formula.

PROOF: Let (S, σ) be an interpretation that satisfies LF. Then (by the definition of $\wedge$) both $\forall x F(x,y)$ and $\forall x M(y,x)$ are true in (S, σ). This means that (by Definition 6.6, applied twice) for every $(v_1, v_2) \in D \times D$, $F(x,y)$ is true in $(S, \sigma|^x_{v_1})$ and $M(y,x)$ is true in $(S, \sigma|^x_{v_2})$. This implies, in the special cases where $v_1 = v_2$, that for every $v_1 \in D$, $F(x,y)$ and $M(y,x)$ are true in $(S, \sigma^x_{v_1})$, which means $\forall x (F(x,y) \wedge M(x,y))$ is satisfied. Thus LF logically implies RF.

On the other hand, let (S, σ) be an interpretation that satisfies RF. This means that $\forall x (F(x,y) \wedge M(y,x))$ is satisfied in (S, σ), so (by Definition 6.6), for every $v \in D$, $F(x,y) \wedge M(y,x)$ is satisfied in $(S, \sigma|^x_v)$. This means (by definition of $\wedge$) that for every $v \in D$ both $F(x,y)$ and $M(y,x)$ are satisfied in (S, σ^x_v). Since this is true for every $v \in D$, this means that for every $v_1 \in D$ $F(x, y)$ is satisfied in $(S, \sigma|^x_{v_1})$, and for every $v_2 \in D$ $M(y, x)$ is satisfied in $(S, \sigma|^x_{v_2})$. In other words, $(\forall x F(x, y) \wedge \forall x M(y, x))$ is satisfied, so RF logically implies LF. Since LF and RF logically imply each other, they are logically equivalent, and (by definition) $LF \leftrightarrow RF$ is a valid formula. QED.

3. Either prove the java code for `binSearch2` correct with respect to its precondition/postcondition pair, or provide a counter-example of input for which it fails.

CLAIM: $P(i)$: "If the loop has i iterations, then $-1 \leq f_i < l_i \leq A.length$, $A[0..f_i] < n$, and $A[l_i..A.length - 1] \geq n$," is true for all $i \in \mathbb{N}$.

PROOF (INDUCTION ON i): Since array A has length at least 0, after the 0th iteration (which always occurs), you have $-1 = f_i < A.length = l_i = 0$, and the empty array $A[0, f_i]$ has only elements less than n, and the empty array $A[l_i, A.length - 1]$ has only elements greater than or equal to n. Thus $P(0)$, the base case, holds.

INDUCTION STEP: Assume $P(i)$. If there is no $i + 1$th iteration, then $P(i + 1)$ holds vacuously. Otherwise the exit condition is not satisfied, so $l_i \neq f_i + 1$, which together with the IH $P(i)$ means that $l_i > f_i + 1$, or $l_i \geq f_i + 2$. The "if $(f! = l - 1)$" branch is executed, and $m_{i+1} = (f_i + l_i)/2$ is calculated, and $mid_{i+1} = (f_i + l_i)/2 \geq (f_i + f_i + 2)/2 = f_i + 1 > f_i$. On the other hand, $mid_{i+1} = (f_i + l_i)/2 \leq (l_i + l_i - 2)/2 = l_i - 1 < l_i$. Thus, $f_i < mid_{i+1} < l_i$. There are two cases to consider.

CASE 1: If $A[mid_{i+1}] \geq n$, then $l_{i+1} = mid_{i+1}$ and $f_{i+1} = f_i$, so (using $P(i)$ and $f_i < mid_{i+1} < l_i$):

$$-1 \leq f_i = f_{i+1} < m_{i+1} = l_{i+1} < l_i \leq A.length,$$

which confirms part of claim $P(i)$. Since A is sorted, the fact that $A[mid_{i+1} = l_{i+1}] \geq n$ means that $A[l_{i+1}..A.length - 1] \geq n$. By $P(i)$ we can assume that $A[0..f_i] = A[0..f_{i+1} < n$. So $P(i + 1)$ holds in this case.

CASE 2: If $A[mid_{i+1}] < n$, then $l_{i+1} = l_i$ and $f_{i+1} = mid_{i+1}$, so (using $P(i)$ and $f_i < m_{i+1} < l_i$):

$$-1 \leq f_i < f_{i+1} = mid_{i+1} < l_i \leq A.length,$$

which confirms part of claim $P(i)$. Since A is sorted, the fact that $A[mid_{i+1} = f_{i+1}] < n$ means that $A[0..f_{i+1}] < n$. By $P(i)$ we can assume that $A[l_i..A.length-1] = A[l_{i+1}..A.length-1] \geq n$. So $P(i + 1)$ holds in this case.

In both cases, $P(i) \Rightarrow P(i + 1)$, as wanted.

I conclude that $P(i)$ holds for all $i \in \mathbb{N}$.

CLAIM (PARTIAL CORRECTNESS): Suppose the precondition is satisfied and $binSearch2(A, n)$ terminates. Then, when it does, the postcondition is satisfied.

PROOF: Suppose the precondition is satisfied and $binSearch2(A, n)$ terminates after k iterations of the loop. By the exit condition we know that $l_k = f_k + 1$, so A is the disjoint union of $A[0..f_k]$ and $A[l_k..A.length - 1]$. Suppose there is some $0 \leq i \leq A.length - 1$ such that i is the smallest index with $A[i] \geq n$. By $P(k)$ $i \notin A[0..f_k]$, so it must be in $A[l_k..A.length-1]$ (forcing $A[l_k..A.length-1]$ to be non-empty, so $l_k < A.length$). By $P(k)$, $A[l_k] \geq n$, and it is the smallest index with this property, so $i = l_k$. Since the program returns $l_k = i$, it satisfies the postcondition in this case. Otherwise, if there is no such i then (by $P(k)$) the sub-array $A[0..f_k]$ is equal to A, so the sub-array $A[l_k..A.length - 1]$ is empty, so $l_k = A.length$. In this case, the program returns $l_k = A.length$, and the postcondition is satisfied. In both cases the postcondition is satisfied, as claimed. QED.

CLAIM: $P(i)$: "If the loop iterates i times, then $l_i - f_i$ is a natural number," is true for all $i \in \mathbb{N}$.

PROOF (INDUCTION ON i): Suppose $i = 0$, then $f_i = -1$ and $l_i = A.length \geq 0$, so $l_i - f_i \geq 1 > 0$, and both f_i and l_i are integers, so there difference is an integer greater than 0. Thus $P(0)$ holds.

INDUCTION STEP: Assume $P(i)$ holds. If there is no $(i+1)$th iteration of the loop, then $P(i+1)$ holds vacuously. Otherwise (proved in the process of proving the loop invariant) we have $f_i < mid_{i+1} < l_i$. If $A[mid_{i+1}] \geq n$, then the program sets $l_{i+1} = mid_{i+1}$ and $f_{i+1} = f_i$, so $1 \leq l_{i+1} - f_{i+1} = mid_{i+1} - f_i$, and $l_{i+1} - f_{i+1}$ is a natural number. If $A[mid_{i+1}] < n$, then the program sets $l_{i+1} = l_i$ and $f_{i+1} = mid_{i+1}$, so $l_{i+1} - f_{i+1} = l_i - mid_{i+1} \geq 1$, and $l_{i+1} - f_{i+1}$ is a natural number. In both cases $P(i)$ implies $P(i+1)$.

I conclude that $P(i)$ holds for all $i \in \mathbb{N}$. QED.

CLAIM (TERMINATION): If the precondition is satisfied, then $binSearch2(A, n)$ terminates.

PROOF: Suppose the precondition holds. Then (by the previous claim) $\langle l_i - f_i \rangle$ is a sequence of natural numbers. If there is an $(i+1)$th iteration of the loop, then we have two cases (using $f_i < mid_{i+1} < l_i$ from loop invariant proof): either $l_{i+1} - f_{i+1} = mid_{i+1} - f_i < l_i - f_i$, or $l_{i+1} - f_{i+1} = l_i - mid_{i+1} < l_i - f_i$. In either case, $l_{i+1} - f_{i+1} < l_i - f_i$, so the sequence $\langle l_i - f_i \rangle$ is a decreasing sequence of natural numbers and hence (PWO) finite. Let the last element be $\langle l_k - f_k \rangle$, so there is no element $l_{k+1} - f_{k+1}$. This implies that there is no $(k+1)$th iteration of the loop, so the loop must terminate. QED.

Taken together, partial correctness and termination imply that $binSearch2(A, n)$ is correct with respect to its specification. QED.

4. Denote a ternerary digit as a TRIT, and an array of ternary digits as a TRIT CORE. In the (lamentably uncommented) methods below, methods `tritCore.tweak` and `tritCore.groupTweak` are defined, for managing trit cores. Either prove that `tritCore.groupTweak` correctly satisfies its postcondition for every valid input, or provide a counterexample of valid input for which it fails. You may assume, without proof, that `tweak` is correct with respect to its specification.

CLAIM: $P(n)$: "Suppose the precondition of $groupTweak(digit, n, from, to, intermediate)$ is satisfied when it is called. Then it returns, and when it does its postcondition is satisfied," is true for all $n \in \mathbb{N} - \{0\}$.

PROOF (COMPLETE INDUCTION ON n): If $n = 1$, then the "if $(n == 1)$" branch is executed, so $status$ is set to $\texttt{true} \wedge tweak(digit, from, to)$. The precondition for $groupTweak$ when $n = 1$ implies the precondition for $tweak$ (which we are allowed to assume correct with respect to its specification), so $status = true \wedge true = true$, and for some $0 \leq k < digit.length$, $digit$ has $digit[0..k-1] = intermediate$, $digit[k] = to$, and all other elements of $digit$ unchanged, which is what $P(1)$ claims. Thus the base case holds.

INDUCTION STEP: Assume that $P(\{1, \ldots, n-1\})$ is true for some arbitrary natural number $n > 1$. We will show that this implies $P(n)$. When $groupTweak(digit, n, from, to, intermediate)$ is called, $status$ is initialized to $true$, and then the "$(n == 1)$ else" branch is executed. Denote the values passed in to parameters as $f = from$, $t = to$, and $i = intermediate$.

In the first recursive call, $status$ is set to $true \wedge groupTweak(digit, n-1, f, i, t)$. We have already assumed $P(n-1)$, and the precondition for $groupTweak(digit, n-1, f, i, t)$ (substituting f for $from$ and i for to) is guaranteed by the precondition for $groupTweak(digit, n, from, to, intermediate)$.. So $groupTweak(digit, n-1, f, i, t)$ returns $true$ (so $status$ is set to $true$), $digit[0..n-2] == i$, and all other elements are unchanged (this covers the case $n = 2$).

In the second recursive call, $status$ is set to $true \wedge groupTweak(digit, 1, f, t, i)$. We've already assume $P(1)$, and the precondition for $groupTweak(digit, n, from, to, intermediate)$, plus the postcondition of $groupTweak(digit, n-1, f, i, t)$, guarantee that $digit[0..n-2] == i$ and $digit[n-1] == f$, which is exactly the precondition for $groupTweak(digit, 1, f, t, i)$ (with $k = n-1$), so $groupTweak(digit, 1, f, t, i)$ returns $true$ (so $status$ is set to $true$), and when it does $digit[0..n-2] == i$ and $digit[n-1] == t$.

In the third recursive call, $status$ is set to $true \wedge groupTweak(digit, n-1, i, t, f)$. We've already assumed $P(n-1)$,and the postcondition of immediately previous call to $groupTweak$ guarantees that $digit[0..n-2] == i$, which implies the precondition for $groupTweak(digit, n-1, i, t, f)$. Thus, $groupTweak(digit, n-1, i, t, f)$ returns $true$ (so $status$ is set to $true$), and $digit[0..n-2] == t$, and no other elements are changed. Combined with the postcondition of the second call to $groupTweak$, this means that $digit[0..n-1] == t == to$, no other elements of digits are changed, and $status == true$ is returned by $groupTweak(digit, n, from, to, intermediate)$. Thus $P(\{1, \ldots, n-1\}) \Rightarrow P(n)$, as wanted.

I conclude that $P(n)$ holds for all $n \in \mathbb{N} - \{0\}$. QED.

5. Although `Vector` doesn't provide the most efficient implementation, `baseSort` outlines an $O(n)$ (yes, you read that correctly!) sorting algorithm. Either prove that the method `baseSort` correctly satisfies its postcondition whenever its precondition is satisfied, or provide a counter-example.

SOLUTION: First I need a lemma to show that `digitIndex` is extracting digits of integer values in base *base*.

CLAIM 5A: Suppose $m = d_n \cdots d_1 d_0$, written in base b, so $m = \sum_{k=0}^{n} d_k b^k$, where $0 \leq d_k < b$. Then for $0 \leq k \leq n$, digit $d_k = (n \operatorname{div} b^k) \bmod b$.

PROOF: Let $q_k = \sum_{i=k}^{n} d_i b^{i-k}$ and $r_k = \sum_{i=0}^{k-1} d_i b^i$. Then m can be rewritten as:

$$m = b^k q_k + r_k \qquad \text{and, by construction, } 0 \leq r_k < b^k.$$

This means, that by Proposition 1.7, $q_k = n \operatorname{div} b^{k+1}$. Every term of q_k is divisible by b except the first term, so $q_k \bmod b = d^k$, as claimed. QED.

Now in order to prove that the main `while` loop is correct with respect to its specifications, I need to use the properties of three inner `for` loops. In all three `for` loops, termination is trivial (for example, let $\langle base - i \rangle$ be a sequence of natural numbers), so we assume it without proof. Two of the `for` loops are small enough to state their properties without proof: the first one creates array `digit[0..base-1]` of empty `Vectors`, and the third one concatenates `digit[0],...,digit[base-1]` into numList_i. Here is a precondition/postcondition pair for the second `for` loop

```
//sublist by current digit
/**
 * Precondition: sorted is true, and the values in numList are sorted
 *               in non-decreasing order mod magnitude.
 * Postcondition: sorted is true if and only if all values in numList
 *                   are < magnitude * base
 *                numList contains the same elements as the
 *                   concatenation of digit[0]...digit[base-1]
 *                digit[j] < digit[k], mod magnitude * base,
 *                   for 0 <= j < k < base
 *                digit[j] in non-decreasing order mod magnitude
 *
 */
```

CLAIM 5B: $P(i)$ "If the precondition of the `for` loop headed `//sublist by current digit` is satisfied and the loop has i iterations, then at the end of the ith iteration *sorted* is true if and only if all values in `numList[0..i-1]` are less than $magnitude \times base$, `numList[0..i-1]` contains the same elements as the concatenation of `digit[0]...digit[base-1]`, `digit[j]` $<$ `digit[k]` $\bmod$ $magnitude \times base$ for all $0 \leq j < k < base$, and `digit[j]` is in non-decreasing order $\bmod$ $magnitude$ for $0 \leq j < base$, is true for all i in $\mathbb{N}$.

PROOF (INDUCTION ON i): If $i = 0$ then sorted is true (by the precondition) and all values in the empty sublist `numList[0..-1]` are less than $magnitude \times base$, and the empty Vector contains the same elements as the concatenation of the empty Vectors `digit[0]..digit[base-1]`. Since they are empty Vectors, every value in `digit[j]` is less than `digit[k]`, and `digit[j]` is in non-decreasing order $\bmod$ magnitude. So the base case, $P(0)$ holds.

INDUCTION STEP: Assume that $P(i)$ holds for some arbitrary natural number i. If there is no $(i+1)$th iteration, then $P(i+1)$ holds vacuously. Otherwise, numList has an ith element, n_i, whose value mod magnitude is no less than every lower-indexed value in numList (by the precondition).

If $n_i \geq magnitude \times base$, then $n_i/magnitude \geq base$, and *sorted* is set to false, as wanted since there is now at least one element of numlist[0..i] no less than $magnitude \times base$. Otherwise *sorted* is true if and only if all the elements of numList[0..i-1] (and hence of numList[0..i] are less than $magnitude \times base$.

Since the concatenation of digit[0]..digit[base-1] already contained all the elements of numList[0..i-1] (by assumption of $P(i)$), they contain all the elements of numList[0..i] once n_i is added.

Suppose $magnitude = base^h$, for some $h \in \mathbb{N}$ (a very small induction shows this is always so). Let $d_h = (n_i \operatorname{div} magnitude) \bmod base$, then n_i is added to digit[d_h]. Thus, by Claim 5a d_j is the hth digit of n_i in base *base*, and $n_i \bmod magnitude \times base$ is the $(h+1)$-digit number (in base *base*) $d_h d_{h-1} \cdots d_0$, which is less than any $(h+1)$-digit number with its most-significant digit greater than d_h, and less than any h-digit number with its most-significant digit greater than d_h. This preserves the property (assumed by $P(i)$) that digit[j]<digit[k] whenever $0 \leq j < k < base$.

Thus $P(i) \Rightarrow P(i+1)$, as wanted.

I conclude that $P(i)$ is true for all $i \in \mathbb{N}$. QED.

Partial correctness of the loop headed ``//sublist by current digit'' follows by setting $i =$ numList.size().

CLAIM: If m is the maximum integer value in numList and $magnitude_i$ is the value of *magnitude* after the ith loop iteration, then $\langle m/magnitude_i \rangle$ is a strictly-decreasing integer sequence.

PROOF: Let $m = d_n \cdots d_0$ in base *base*. Then $m/magnitude_i$ corresponds to removing the right-most i digits of m (see the proof of Claim 5a). This always yields a natural number, so it only remains to show that the sequence is strictly decreasing.

If there is an $(i+1)$th iteration of the loop then (by the postcondition of the loop beginning ``//sublist by current digit'' there is at least one integer in n in numList with $n/magnitude_i \geq base$. Since m is the largest integer in numList, we must have $m/magnitude_i$ having two or more digits in its base *base* expansion. This, in turn, means that $m/magnitude_{i+1}$ (being one digit shorter) is strictly less than $m/magnitude_i$. QED.

CLAIM (TERMINATION): If the preconditions hold when baseSort(base, numList) is called, then it terminates.

PROOF: Suppose the preconditions hold and baseSort(base, numList) is called. Each iteration of the main while loop is finite (I assume termination of the for loop, which is easy to prove), and associated with the value $m/magnitude_i$, and by the previous claim $\langle m/magnitude_i \rangle$ is a decreasing sequence of natural numbers, and hence (PWO) finite. Denote the last element of the sequence $m/magnitude_k$, so there is no element $m/magnitude_{k+1}$. This implies that there is no $(k+1)$th loop iteration, so the loop terminates. QED.

CLAIM: $P(i)$: "If there is an ith iteration of the loop, then the integer values of numList_i are in non-decreasing order, mod $magnitude_i$ and *sorted* is true if and only if $i > 0$ and all array elements are smaller than $magnitude_i$" is true for all $i \in \mathbb{N}$.

PROOF (INDUCTION ON i): For $i = 0$ you have $magnitude_i = 1$, so every integer value in numList_0 is equal to $0 \bmod 1$, and are thus (trivially) in non-decreasing order, and *sorted* is false. This verifies the base case, $P(0)$.

INDUCTION STEP: Assume that $P(i)$ holds for some arbitrary integer i. I want to show that this implies $P(i+1)$. If there is no $(i+1)$th loop, then $P(i+1)$ holds vacuously. Otherwise, $sorted_i$ is false, and the first statement of the while loop sets *sorted* to true. This satisfies the precondition (stated above, and proved in Claim 5b) of the loop preceded by the "`// sublist by current digit`" comment, hence its postcondition is satisfied: Each `digit[i]` is sorted in non-decreasing order $\mod magnitude_i$, if $0 \leq i < j < base$, then every element of `digit[i]` is less than every element of `digit[j]` $\mod magnitude_i \times base$, *sorted* is true if and only if every value in `numList` is smaller than $magnitude_i \times base$, and

I assume without proof that the loop preceded by the comment "`combine sublists`" concatenates `digit[0]` $\cdots$ `digit[base-1]` into `numList`, so consider two indices $0 \leq j < k < numList.size()$, with $n_j =$ `(numList.elementAt(j)).intValue()` and $n_k =$ `(numList.elementAt(k)).intValue()`. There are two possibilities

CASE 1: n_j and n_k were both concatenated into `numList` from the same sublist, `digit[`d_i`]`, so (by assumption) $n_j \bmod magnitude_i \leq n_k \bmod magnitude_i$ and the ith digit of n_j is d_i, the same as the ith digit of n_k. So, by Claim 5a:

$$\begin{aligned} n_j \bmod magnitude_{i+1} &= d_i \times magnitude_i + n_j \bmod magnitude_i \\ &\leq d_i \times magnitude_i + n_k \bmod magnitude_i \\ &= n_k \bmod magnitude_{i+1}. \end{aligned}$$

CASE 2: n_j was concatenated into `numList` from sublist `digit[`d_i`]` and n_k was concatenated into `numList` from sublist `digit[`d_i'`]`, where $d_i < d_i'$. This means (postcondition of inner loop) that $n_j \bmod magnitude_{i+1} < n_k \bmod magnitude_{i+1}$.

In either case $n_j \leq n_k \bmod magnitude_{i+1}$ and *sorted* is true if and only if the largest array element is no smaller than $magnitude_{i+1}$. Thus $P(i) \Rightarrow P(i+1)$.

I conclude that $P(i)$ holds for all $i \in \mathbb{N}$. QED.

CLAIM (PARTIAL CORRECTNESS): If the preconditions hold, and `baseSort` terminates, then (when it terminates) the postcondition holds.

PROOF: If `baseSort` terminates, then *sorted* is true, and all values are less than $magnitude_i$, and in non-decreasing order *mod* $magnitude_i$. Since each natural number in the range $0, \ldots, magnitude_i - 1$ is equal to itself $\mod magnitude_i$, this means that `numList` contains the same values as it started with, in non-decreasing order. QED.

6. Either prove that the method below satisfies its postcondition whenever its precondition is satisfied, or else exhibit a valid input for which it fails.

CLAIM: $P(b)$: "If $a \in \mathbb{N}$ and MoreEuclid(a, b) is called, then it returns integer array result, where result[0] is the greatest common divisor of a and b, and result[0] = result[1] $\times a$ + result[2] $\times b$," is true for all $b \in \mathbb{N}$.

PROOF (COMPLETE INDUCTION ON b): Suppose MoreEuclid$(a, 0)$ is called, where a is an arbitrary natural number. Then the assignment statement "result = $\{a, 1, 0\}$ is executed," the "if (b !=0)" branch is not executed, and the program returns result. In this case, result[0] $= a$, and a divides both 0 and a, and any natural number that divides both a and 0 divides a, so $a =$ result[0] is the greatest common divisor of 0 and a. Furthermore $a = 1 \times a + 0 \times 0 =$ result[1] $\times a$ + result[2] $\times b$. This verifies that $P(0)$ holds.

INDUCTION STEP: Suppose $P(\{0, \ldots, b-1\})$ all hold, and MoreEuclid(a, b), where a is an arbitrary natural number, and b is an arbitrary natural number greater than 0. Then (by Proposition 1.7 of the course notes) $0 \leq a \bmod b < b$, and (assuming without proof that $a\%b = a \bmod b$ is true for positive integers) our induction hypothesis allows us to assume $P(a\%b)$. Since $b > 0$, the "if (b != 0)" branch is executed, the assignment statement "result = MoreEuclid$(b, a\%b)$," is executed. For notational convenience, denote result $= \{r_0, r_1, r_2\}$ immediately after this statement. By

$P(a\%b)$ we can assume that r_0 is the greatest common divisor of b and $a\%b$, and that $r_0 = r_1 \times b + r_2 \times a\%b$.

The next three assignment statements set

$$\text{result}[1] = r_2 \qquad \text{result}[2] = r_1 - (\text{result}[1] \times (a/b)) = r_1 - (r_2 \times (a/b)),$$

and then MoreEuclid(a, b) returns result. Let q and r be the quotient and remainder defined in Proposition 1.7, so $a = bq + r$, (which implies both $a\%b = r = a - bq$ and $q = a/b$), and apply the induction hypothesis $P(a\%b)$, so at the end of MoreEuclid(a, b)

$$\begin{aligned}
\text{[by IH]} \qquad \text{result}[0] = r_0 &= r_1 b + r_2 a\%b = r_1 b + r_2(a - bq) \\
&= r_2 a + (r_1 - r_2 q)b = r_2 a + (r_1 - r_2(a/b))b \\
\text{[by assignment statements above]} \qquad &= \text{result}[1] \times a + \text{result}[2] \times b.
\end{aligned}$$

This satisfies part of claim $P(b)$. By $P(a\%b)$, $d = \text{result}[0]$ is the greatest common divisor of $(b, a\%b)$. This means there are arbitrary integers h_1 and h_2 such that $b = h_1 d$ and $a\%b = h_2 d$, so (since $a\%b = a - qb$)

$$a = a\%b + qb = d(h_1 q + h_2),$$

and d divides a. Hence d is a natural number that divides both a and b. Let d' be an arbitrary natural number that divides both a and b, in other words there are integers k_1 and k_2 such that $k_1 d' = a$ and $k_2 d' = b$. This means that, by $P(a\%b)$,

$$d = \text{result}[1]a + \text{result}[2]b = d'(\text{result}[1]k_1 + \text{result}[2]k_2),$$

so d' divides d. In other words, d is the greatest common divisor of (a, b), which satisfies the other part of claim $P(b)$. Thus $P(\{0, \ldots, b - 1\})$ implies $P(b)$.

I conclude that $P(b)$ is true for all $b \in \mathbb{N}$. QED.

Predicate $P(b)$ implies that MoreEuclid(a, b) is correct with respect to its specification.