1. (12 points)

c, a, d, b, e

e, a, d, b, c

b, d, a, e or b, d, a, c
Full credit if somehow indicated that (c) is neither stronger, nor weaker than (e), and had the rest of the ordering correct.

2. (8 points)
b, c, e, f

3. (10 points)

a) Not valid; x = 0, y = -1 => z = 0 is a counter example
b) Not valid; x = 5, y = 0 => z = 5 is a counter example
c) Not valid; z = -1 => x = -3 is a counter example
d) Valid
e) Valid

4. (10 points)

```
P: { _(false && x == 0) || (|x| > 1 && x ≠ 0)_ } => { |x| > 1}
if ( x == 0 ) {
   { ___false____ }
   rez = -1;
   { __ rez < -1___ }
}
else {
   { _(x > 1 && x > 0) || (x < -1 && x ≤ 0)_ } =>
                              {x > 1 || x < -1} => { |x| > 1 }
   if ( x > 0 ) {
       { ___ x > 1_____ }
       x = -x
       { ___ x < -1____ }
   }
   else {
       { ___ x < -1___ }
       // Do nothing
       { ___ x < -1____ }
   }
   { ___x < -1_____ }
   rez = x*x*x;
    { __ rez < -1__ }
 }
    Q: { rez < -1 }
```

5. (9 points)

a)

```
{ |x| > 5 } = {|x_0| > 5 } = { x_0 > 5 || x_0 < -5 }
y = 5;
{ _( x_0 > 5 || x_0 < -5 ) && y == 5_ }
could be written { _( x > 5 || x < -5 ) && y == 5_ }

x = x * 3;
{ _( x_0 > 5 || x_0 < -5 ) && y == 5 && x_1 == x_0 * 3_ } =
{ _( x_1 > 15 || x_1 < -15 ) && y == 5 && x_1 % 3 == 0_ }
could be written {(x > 15 || x < -15 ) && y == 5 && x % 3 == 0}
Note: x > 15 && x % 3 == 0 => x > 16 (or > 17, or >= 18!) && x % 3 == 0.
Similarly, for x < -15.

x = x - 2;
{ _( x_1 > 15 || x_1 < -15 ) && y == 5 && x_1 % 3 == 0 && x_2 == x_1 - 2_ } =
{ _(x_2 > 13 || x_2 < -17) && y == 5 && (x_2 + 2) % 3 == 0_}
could be written { _(x > 13 || x < -17) && y == 5 && (x + 2) % 3 == 0_ }
```

b)

```
{ x > 0 && z > 4 }
if (x < 5 && z < x) {
      { _false_ }
      since {0 < x < 5 && z > 4 && z < x = 0 < x < 5 && false = false}

      y = z + x;
      { _false_ }
      since { y == z + x && false = false}

}
{ _x > 0 && z > 4_ }
since (false ||(x > 0 && z > 4) && !(x < 5 && z < x)) = (x > 0 && z > 4) &&
(x >= 5 || z >= x) = (x > 0 && z >= 5) && (x >= 5 || z >= x) = (x > 0 && z >=
5 && x >= 5) || (x > 0 && z >= 5 && z >= x) = (x >= 5 && z >= 5) || (x > 0 &&
z >= 5 && z >= x) = (x >= 5 && z >= 5) || (x > 0 && x < 5 && z >= 5 && z >=
x) || (x >= 5 && z >= 5 && z >= x) = (x >= 5 && z >= 5) || (x > 0 && x < 5 &&
z >= 5) || (x >= 5 && z >= 5 && z >= x) = (x > 0 && x < 5 && z >= 5) || ((x
>= 5 && z >= 5) && true) || ((x >= 5 && z >= 5) && z >= x) = (x > 0 && x < 5
&& z >= 5) || ((x >= 5 && z >= 5) && (true || z >= x)) = (x > 0 && x < 5 && z
>= 5) || (x >= 5 && z >= 5) = (z >= 5) && ((x > 0 && x < 5) || (x >= 5)) = (x
> 0) && (z >= 5) = x > 0 && z > 4
```

Alternatively, can show that if A && B == false then A && !B == A.

6. (16 points)

Proofs below are very step-by-step and formal. We do not expect this level of rigor from the students. Their proofs have to be strict enough but not necessarily need to spell out all the finer details.

a) (2 points)

LI: `(c + g * (m + a + b) == a0 + b0) && a >=0 && b >= 0`

There can be variations on this, but must include a, b, a0, b0, c, and m.

b) (2 points)
Initially, a == a0, b == b0, m == 0, g == 1, c == 0, n == 0
LI: `(c + g * (m + a + b) == a0 + b0) && a >=0 && b >= 0`
a > = 0 since a == a0 and a0 >= 0 by the precondition
b > = 0 since b == b0 and b0 >= 0 by the precondition
Substituting, (0 + 1 * (0 + a0 + b0) ==1 * (a0 + b0) == a0 + b0

c) (8 points)

z_k means z at step k.

Assume at step k: (c_k + g_k * (m_k + a_k + b_k) == a0 + b0) && a_k >= 0 && b_k >= 0

n_{k+1} = m_k + a_k % 2 + b_k % 2

a_{k+1} = a_k / 2, since it's integer division, a_{k+1} * 2 + a_k % 2 = a_k, a_{k+1} = (a_k - a_k % 2) / 2

b_{k+1} = b_k / 2, since it's integer division, b_{k+1} * 2 + b_k % 2 = b_k, b_{k+1} = (b_k - b_k % 2) / 2

c_{k+1} = c_k + (n_{k+1} % 2) * g_k

g_{k+1} = g_k * 2

m_{k+1} = n_{k+1} / 2, since it's integer division, m_{k+1} * 2 + n_{k+1} % 2 = n_{k+1}, m_{k+1} = (n_{k+1} - n_{k+1} % 2) / 2

Then at step (k + 1): c_{k+1} + g_{k+1} * (m_{k+1} + a_{k+1} + b_{k+1}) == c_k + (n_{k+1} % 2) * g_k + g_k * 2 * ((n_{k+1} - n_{k+1} % 2) / 2 + (a_k - a_k % 2) / 2 + (b_k - b_k % 2) / 2) = c_k + (n_{k+1} % 2) * g_k + g_k * 2 * (n_{k+1} - n_{k+1} % 2 + a_k - a_k % 2 + b_k - b_k % 2) / 2 = c_k + (n_{k+1} % 2) * g_k + g_k * (m_k + a_k % 2 + b_k % 2- n_{k+1} % 2 + a_k - a_k % 2 + b_k - b_k % 2) = c_k + (n_{k+1} % 2) * g_k + g_k * (m_k- n_{k+1} % 2+ a_k+ b_k) = c_k + (n_{k+1} % 2) * g_k + g_k * m_k - g_k * n_{k+1} % 2 + g_k * a_k + g_k * b_k = c_k + g_k * m_k + g_k * a_k + g_k * b_k = c_k + g_k * (m_k + a_k + b_k) which is a0 + b0 by inductive hypothesis.

At iteration k, a_k > 0 || b_k > 0 because otherwise we would not have entered iteration k + 1. So, there are the following possibilities to consider:
a_k > 0 && b_k > 0
a_k > 0 && b_k == 0
a_k == 0 && b_k > 0

Suppose, a_k > 0. By definition of the modulo operation on two non-negative integers, a_k % 2 is either 0 or 1. By definition of integer division, a_{k+1} * 2 + a_k % 2 = a_k. Then there are two cases:

- a_k % 2 == 0, then a_{k+1} * 2 = a_k; a_{k+1} * 2 > 0; and a_{k+1} > 0.
- a_k % 2 == 1, then a_{k+1} * 2 + 1 = a_k; a_{k+1} * 2 + 1 > 0; a_{k+1} * 2 > -1; a_{k+1} > -1/2 but a_{k+1} is an integer, therefore a_{k+1} >= 0.

Combining two cases yields a_{k+1} >= 0.
Proof that when b_k > 0, (b_{k+1} >= 0) is similar. Thus, when (a_k > 0 && b_k > 0), (a_{k+1} >= 0 && b_{k+1} >= 0).
When (a_k > 0 && b_k == 0), b_{k+1} == 0 >= 0. Therefore, (a_{k+1} >= 0 && b_{k+1} >= 0). A proof that when (a_k == 0 && b_k > 0), (a_{k+1} >= 0 && b_{k+1} >= 0) is similar.

d) (2 points)

The postconidition for the entire bitwise method is c == a0 + b0. We can use backward reasoning to determine the postcondition for the loop: c + m * g == a0 + b0.

!(a > 0 || b > 0) && (c + g * (m + a + b) == a0 + b0) && a >= 0 && b >= 0

= a <= 0 && b <= 0 && (c + g * (m + a + b) == a0 + b0) && a >= 0 && b >= 0

= a==0 && b == 0 && (c + g * (m + a + b) == a0 + b0)

= c + g * m == a0 + b0

e) (2 points)

D = a + b

- Initially, a == a0 >= 0 and b == b0 >= 0 by the precondition and initial assignment statements. Therefore, a + b >= 0
- At iteration k, a_k > 0 || b_k > 0 because otherwise we would not have entered iteration k + 1.

  Similarly to the inductive proof in c), there are the following possibilities to consider:
  - a_k > 0 && b_k > 0
  - a_k > 0 && b_k == 0
  - a_k == 0 && b_k > 0

  Suppose, a_k > 0. By definition of the modulo operation on two non-negative integers, a_k % 2 is either 0 or 1. By definition of integer division, a_{k+1} * 2 + a_k % 2 = a_k. Then, there are two cases:
  - a_k % 2 == 0, then a_{k+1} * 2 = a_k = 2 * p for some integer p. Then a_{k+1} = p, a_{k+1} - a_k == p – 2 * p = -p. But since a_k > 0, 2 * p > 0; p > 0; p >= 1; -p <= -1. Therefore, a_{k+1} - a_k <= -1; a_{k+1} <= a_k – 1; a_{k+1} < a_k.
  - a_k % 2 == 1, then a_{k+1} * 2 + 1 = a_k= 2 * p + 1 for some integer p. Then a_{k+1} = p, a_{k+1} - a_k == p – 2 * p - 1 = -p - 1. But since a_k > 0, 2 * p + 1 > 0; p > -1/2. p is an integer, therefore p >= 0. Therefore, a_{k+1} - a_k = -p - 1 <= -1; a_{k+1} <= a_k – 1; a_{k+1} < a_k.
  - Combining two cases yields a_{k+1} < a_k.

  Proof that when b_k > 0, b_{k+1} < b_k is similar. Thus, when (a_k > 0 && b_k > 0), (a_{k+1} < a_k && b_{k+1} < b_k). D_{k+1} = a_{k+1} + b_{k+1} < a_k + b_k.

  When (a_k > 0 && b_k == 0), a_{k+1} < a_k and b_{k+1} == b_k == 0, so b_{k+1} - b_k == 0. In other words, a decreases but b does not change. D_{k+1} = a_{k+1} + b_{k+1} < a_k + b_k

  A proof that when (a_k == 0 && b_k > 0), D_{k+1} = a_{k+1} + b_{k+1} < a_k + b_k is similar.

  Thus, we proved that in all cases D_{k+1} = a_{k+1} + b_{k+1} < a_k + b_k == D_k.
- When D == 0 => a + b == 0; a == -b; b == -a

  We previously showed that the following part of LI holds at the end of every iteration: a >= 0 && b >= 0. Therefore, when D == 0 => (a >= 0 && -a >= 0) => (a >= 0 && a <= 0) => a == 0. Similarly, when D == 0 => (-b >= 0 && b >= 0) => (b <= 0 && b >= 0) => b == 0.

  Loop exit condition is !(a > 0 || b > 0).

  Thus, D == 0 => (a == 0 && b == 0) => (a < =0 && b <= 0) = !(a > 0 || b > 0)

7. (7 points)

a) false
b) false
c) false
d) false
e) false
f) true
g) false

8. (12 points)

a) Neither
b) A
c) Neither
d) B
e) Neither
f) D

9. (10 points)
Requires:
None
Modifies:
“this” or this queue
Effects:
Same as paragraph “Inserts the specified element …”
Returns:
True.
Throws:
Same clause as Throws in JavaDoc above.

10. (6 points)

Either of the following is OK for E:

E: removes duplicate elements from lst, maintains order.

E, more formally: (forall j :: 0 <= j < lst_{pre}.Length ==> ∃ k :: 0 <= k < lst_{post}.Length && lst_{pre}[j] == lst_{post}[k]) && (forall j, k :: 0 <= j < k < lst_{post}.Length ==> lst_{post}[j] != lst_{post}[k]) && (forall j, k :: 0 <= j < k < lst_{pre}.Length && lst_{pre}[j] != lst_{pre}[k] ==> ∃ m :: 0 <= m < lst_{post}.Length && ∃ n :: 0 <= n < lst_{post}.Length && lst_{pre}[j] == lst_{post}[m] && lst_{pre}[k] == lst_{post}[n] && m < n)

lst is non-null => (E && (∃ m :: 0 <= m < lst_{post}.Length, forall j :: 0 <= j < lst_{pre}.Length && (j <= m ==> lst_{pre}[j] == lst_{post}[j] && m < j < lst_{post}.Length ==> lst_{pre}[j] != lst_{post}[j]))
It’s OK to express the last part in words, i.e., “nothing else is modified” or “nothing but lst is modified”.