```
"""
Functions related to the congruent number problem for elliptic curves.

AUTHOR: William Stein, 2005-10-17

EXAMPLES:
    sage: attach "cong.sage"
    sage: cong_number_sets(101)
    ([], [])
    sage: is_conj_congruent_number(101)
    True
    sage: is_congruent_number(101)
    True
    sage: print conj_congruent_number_list(30)
    [5, 6, 7, 13, 14, 15, 20, 21, 22, 23, 24, 28, 29, 30]
    sage: cong_number_curve(101)
    Elliptic Curve defined by y^2  = x^3 - 10201*x over Rational Field
    sage: congruent_curve_gens(101)
    Entering qsieve::search: y^2 = 1x^3 + 0x^2 + -10201x^1 + 0
     ...
    sage: P = _[0]
    sage: congruent_triangle(P)
    (44538033219/1326635050,
     267980280100/44538033219,
     20152424629497600019 61/5908571592638972595 0)
    sage: congruent_triangle(2*P)
    (-38084245742706258219731099054869086738455 21/23814408737729498353819656789984450517 5900,
     -481051056502135866747157067157685900455318 00/380842457427062582197310990548690867384 5521,
     1848262862847386282568218740622471373193310336920681489039410834684959818722637176 1441/906953794584941364348309838871418957694907021279988390325859242498283526441532143900)
"""

def cong_number_sets(n):
    """
    Given a positive integer n, returns the two sets appearing in the
    conjectural criterion for when a number is congruent.
    """
    n = ZZ(n)
    n = ZZ(prod([p for p, e in n.factor() if e%2 == 1]))

    if n % 2 == 0:   # even case
        E = []       # with c even
        O = []       # with c odd
        a_bound = (sqrt(n/8) + 1).integer_part()
        c_bound = (sqrt(n)/4 + 1).integer_part()
        half_n = n//2
```

```
        for c in range(-c_bound, c_bound+1):
            c_square = c^2
            for a in range(-a_bound, a_bound+1):
                a_square = a^2
                z = half_n - 4*a_square - 8*c_square
                try:
                    b = z.square_root()
                    if b.denominator() == 1:
                        b = b.numerator()
                        assert 4*a^2 + b^2 + 8*c^2 == n/2
                        if c % 2 == 0:
                            E.append((a,b,c))
                        else:
                            O.append((a,b,c))
                except ValueError:
                    pass
    else:
        E = []       # with c even
        O = []       # with c odd
        a_bound = (sqrt(n/2)+1).integer_part()
        c_bound = (sqrt(n/8) + 1).integer_part()
        for c in range(-c_bound, c_bound+1):
            c_square = c^2
            for a in range(-a_bound, a_bound+1):
                a_square = a^2
                z = n - 2*a_square - 8*c_square
                try:
                    b = z.square_root()
                    if b.denominator() == 1:
                        b = b.numerator()
                        assert 2*a^2 + b^2 + 8*c^2 == n
                        if c % 2 == 0:
                            E.append((a,b,c))
                        else:
                            O.append((a,b,c))
                except ValueError:
                    pass
    return E, O

def is_conj_congruent_number(n):
    """
    Returns True if n is conjecturally a congruent number, according
    to the Birch and Swinnerton-Dyer conjecture.
    """
    E, O = cong_number_sets(n)
```

```
    return len(E) == len(O)

def is_congruent_number(n):
    """
    Returns True if n is provably a congruent number.  This computes
    the rank of the corresponding elliptic curve.
    """
    return cong_number_curve(n).rank() > 0

def conj_congruent_number_list(bound):
    """
    Returns the conjectural congruent numbers up to a given bound.
    """
    return [n for n in range(1,bound+1) if is_conj_congruent_number(n)]

def cong_number_curve(n):
    """
    Returns the elliptic curve corresponding to the integer n.  This
    curve has positive rank if and only if n is a congruent number.
    """
    return EllipticCurve([-n^2,0])

def congruent_curve_gens(n, verbose=True):
    """
    Returns generators for the Mordell-Weil group of the elliptic
    curve corresponding to n.
    """
    E = cong_number_curve(n)
    return E.gens(verbose=verbose)

def congruent_triangle(P):
    """
    Returns the triangle corresponding to the point P on a congruent
    number elliptic curve.  The point P must have nonzero y coordinate.
    """
    m = P.curve().a_invariants()[3]
    n = (-m).square_root()
    (x, y) = (P[0],P[1])
    if y == 0:
        raise ArithmeticError, "point does not define a rational right triangle"
    T = ((n^2-x^2)/y, -2*x*n/y, (n^2+x^2)/y)
    assert (T[0]^2 + T[1]^2 == T[2]^2), "bug in program."
    return T
```