In[ ]:=
```
(*Сложение двух точек эллиптической кривой*)
EllipticAdd[p_, a_, b_, c_, P_List, Q_List] := Module[{lam, x3, y3, P3},
                                                 программный модуль
  Which[
  условный оператор с множественными ветвями
   P == {O}, Q,
        O большое
   Q == {O}, P,
        O большое
   P[[1]] ≠ Q[[1]],
       lam = Mod[(Q[[2]] - P[[2]]) PowerMod[Q[[1]] - P[[1]], p - 2, p], p];
             остаток от деления          степень по модулю
       x3 = Mod[lam^2 - a - P[[1]] - Q[[1]], p];
            остаток от деления
       y3 = Mod[- (lam (x3 - P[[1]]) + P[[2]]), p];
            остаток от деления
       {x3, y3},
   (P == Q) ∧ (P[[2]] == 0), {O},
                             O большое
   (P == Q) ∧ (P ≠ {O}),
                  O большое
       lam = Mod[(3 * P[[1]]^2 + 2 a * P[[1]] + b) PowerMod[2 P[[2]], p - 2, p], p];
             остаток от деления                    степень по модулю
       x3 = Mod[lam^2 - a - P[[1]] - Q[[1]], p];
            остаток от деления
       y3 = Mod[- (lam (x3 - P[[1]]) + P[[2]]), p];
            остаток от деления
       {x3, y3},
   (P[[1]] == Q[[1]]) ∧ (P[[2]] ≠ Q[[2]]), {O}
                                           O большое
  ]
 ]
EllipticMult[p0_, a0_, b0_, c0_, pointP0_, n0_] := Module[
                                                   программный модуль
  {pointP = pointP0, n = n0, p = p0, a = a0, b = b0, c = c0, pointQ = {0, 0}, binN},
  binN = IntegerDigits[n, 2];
         цифры целого числа
  Do[
  оператор цикла
   If[binN[[i]] == 0,
   условный оператор
    pointQ = EllipticAdd[p, a, b, c, pointQ, pointQ]
    , pointQ = EllipticAdd[p, a, b, c, EllipticAdd[
        p, a, b, c, pointQ, pointQ
       ], pointP]
   ]
   , {i, 1, Length[binN]}];
           длина
  pointQ]
```

In[ ]:=
```
(*Нахождение порядка эллиптической кривой*)
EllipticRank[p_, a_, b_, c_, pointP0_] := Module[
                                          программный модуль
  {pointP = pointP0, pointQ = {O}
                              O большое
   , maxPossibleRank
   , n
   , table
   , alf
   , gam
   , j, i = 0
   , rank},
  n = p + 1 + 2 * Sqrt[p];
              квадратный корень
  maxPossibleRank = Ceiling[Sqrt[n]];
                    округле··· квадратный корень
  table = {};

  Do[AppendTo[table, pointQ = EllipticAdd[p, a, b, c, pointQ, pointP]],
  ··· добавить в конец к
    {k, 1, maxPossibleRank}];
  If[table[[Length[table]]] == {O}, rank = Length[table]; Return[rank]];
  условный ··· длина          O большое  длина          вернуть управление
  alf = Mod[{table[[Length[table]]][[1]], -table[[Length[table]]][[2]]}, p];
        остаток от де··· длина                          длина
  gam = {O};
         O большое
  i = 1;
  While[! MemberQ[table, gam = EllipticAdd[p, a, b, c, gam, alf] ] &&
  цикл-п··· элемент списка?
    i < maxPossibleRank, i++];
  If[(j = FirstPosition[table, gam]) ≠ {}, rank = maxPossibleRank * i + j, rank = -1];
  услов··· позиция первого по образцу
  rank[[1]]
 ]
```

In[ ]:=
```
(*Нахождение базовой точки с условиями*)
SearchPoint[p_, a_, b_, c_] := Module[
                               программный модуль
 {minXStart = Floor[p / 2],
             округление вверх
  minRankConstr = Floor[2 * p / 3], foundFlg = False, i = 0, xOffsetMax = 5,
                  округление вверх                  ложь
  x1, y1, xOffset, rank, solveOutput},

 While[! foundFlg && i < 100,
 цикл–пока
  If[Mod[4 * b^3 + 27 * (c + i)^2, p] ≠ 0,
  у··· остаток от деления
   xOffset = 0;
   While[(xOffset < xOffsetMax) && (! foundFlg),
   цикл–пока
    x1 = minXStart + xOffset;
    If[Solve[y^2 == x1^3 + a * x1^2 + b * x1 + c + i, {y}, Modulus → p] ≠ {},
       решить уравнения                                  модуль
     y1 = y /. Flatten[Solve[y^2 == x1^3 + a * x1^2 + b * x1 + c + i, {y}, Modulus → p], 1];
               уплостить решить уравнения                                  модуль

     rank = EllipticRank[p, a, b, c + i, {x1, y1}];
     If[rank > minRankConstr && PrimeQ[rank],
                                простое число?
      solveOutput = {{x1, y1}, i};
      foundFlg = True;
                 истина
      , x1++;
      xOffset++;
     ],
     x1++;
     xOffset++;
    ];
   ];
  ];
  Print["=> ", i];
  печатать
  i++;
 ];
 solveOutput
]
```