```
/*
 * Dynamic version of data structure
 * to be used in dynamic programming optimisation
 * called "Convex Hull trick"
 * 'Dynamic' means that there is no restriction on adding lines order
 */
class ConvexHullDynamic
{
    typedef long long coef_t;
    typedef long long coord_t;
    typedef long long val_t;

    /*
     * Line 'y=a*x+b' represented by 2 coefficients 'a' and 'b'
     * and 'xLeft' which is intersection with previous line in hull(first line has -INF)
     */
private:
    struct Line
    {
        coef_t a, b;
        double xLeft;

        enum Type {line, maxQuery, minQuery} type;
        coord_t val;

        explicit Line(coef_t aa=0, coef_t bb=0) : a(aa), b(bb), xLeft(-INFINITY), type(Type::line), val(0) {}
        val_t valueAt(coord_t x) const { return a*x+b; }
        friend bool areParallel(const Line& l1, const Line& l2) { return l1.a==l2.a; }
        friend double intersectX(const Line& l1, const Line& l2) { return areParallel(l1,l2)?INFINITY:1.0*(l2.b-l1.b)/(l1.a-l2.a); }
        bool operator<(const Line& l2) const
        {
            if (l2.type == line)
                return this->a     < l2.a;
            if (l2.type == maxQuery)
                return this->xLeft < l2.val;
            if (l2.type == minQuery)
                return this->xLeft > l2.val;
        }
    };

private:
    bool            isMax; //whether or not saved envelope is top(search of max value)
    std::set<Line>  hull;  //envelope itself

private:
    /*
     * INFO:        Check position in hull by iterator
     * COMPLEXITY:  O(1)
     */
    bool hasPrev(std::set<Line>::iterator it) { return it!=hull.begin(); }
    bool hasNext(std::set<Line>::iterator it) { return it!=hull.end() && std::next(it)!=hull.end(); }

    /*
     * INFO:        Check whether line l2 is irrelevant
     * NOTE:        Following positioning in hull must be true
     *              l1 is next left to l2
     *              l2 is right between l1 and l3
     *              l3 is next right to l2
     * COMPLEXITY:  O(1)
     */
    bool irrelevant(const Line& l1, const Line& l2, const Line& l3) { return intersectX(l1,l3) <= intersectX(l1,l2); }
    bool irrelevant(std::set<Line>::iterator it)
    {
```

```
64          return hasPrev(it) && hasNext(it)
65                 && (    isMax && irrelevant(*std::prev(it), *it, *std::next(it))
66                         || !isMax && irrelevant(*std::next(it), *it,
                           *std::prev(it)) );
67      }
68
69      /*
70       * INFO:        Updates 'xValue' of line pointed by iterator 'it'
71       * COMPLEXITY:  O(1)
72       */
73      std::set<Line>::iterator updateLeftBorder(std::set<Line>::iterator it)
74      {
75          if (isMax && !hasPrev(it) || !isMax && !hasNext(it))
76              return it;
77
78          double val = intersectX(*it, isMax?*std::prev(it):*std::next(it));
79          Line buf(*it);
80          it = hull.erase(it);
81          buf.xLeft = val;
82          it = hull.insert(it, buf);
83          return it;
84      }
85
86  public:
87      explicit ConvexHullDynamic(bool isMax): isMax(isMax) {}
88
89      /*
90       * INFO:        Adding line to the envelope
91       *              Line is of type 'y=a*x+b' represented by 2 coefficients 'a'
       and 'b'
92       * COMPLEXITY:  Adding N lines(N calls of function) takes O(N*log N) time
93       */
94      void addLine(coef_t a, coef_t b)
95      {
96          //find the place where line will be inserted in set
97          Line l3 = Line(a, b);
98          auto it = hull.lower_bound(l3);
99
100         //if parallel line is already in set, one of them becomes irrelevant
101         if (it!=hull.end() && areParallel(*it, l3))
102         {
103             if (isMax && it->b < b || !isMax && it->b > b)
104                 it = hull.erase(it);
105             else
106                 return;
107         }
108
109         //try to insert
110         it = hull.insert(it, l3);
111         if (irrelevant(it)) { hull.erase(it); return; }
112
113         //remove lines which became irrelevant after inserting line
114         while (hasPrev(it) && irrelevant(std::prev(it))) hull.erase(std::prev(it));
115         while (hasNext(it) && irrelevant(std::next(it))) hull.erase(std::next(it));
116
117         //refresh 'xLine'
118         it = updateLeftBorder(it);
119         if (hasPrev(it))
120             updateLeftBorder(std::prev(it));
121         if (hasNext(it))
122             updateLeftBorder(std::next(it));
123     }
124     /*
125      * INFO:        Query, which returns max/min(depends on hull type - see more
       info above) value in point with abscissa 'x'
126      * COMPLEXITY:  O(log N), N-amount of lines in hull
127      */
128     val t getBest(coord t x) const
129     {
130         Line q;
```

```
131             q.val = x;
132             q.type = isMax ? Line::Type::maxQuery : Line::Type::minQuery;
133
134             auto bestLine = hull.lower bound(q);
135             if (isMax) --bestLine;
136             return bestLine->valueAt(x);
137         }
138     };
139
140
141     int main() {
142         NSYNC;
143         ConvexHullDynamic hull(true);
144         int n,q;
145         cin >> n >> q;
146         bool has ins = false;
147         unordered map<int, pair<ll,ll>> mp;
148         while (q--) {
149             int tp;
150             cin >> tp;
151             if (tp==1) {
152                 has ins = true;
153                 ll t, m;
154                 int id;
155                 cin >> t >> id >> m;
156                 ll y0 = (mp.count(id) ? mp[id].first*t + mp[id].second : 0);
157                 ll b = y0 - m*t;
158                 mp[id] = {m,b};
159                 hull.addLine(m,b);
160             }
161             else {
162                 ll t;
163                 cin >> t;
164                 cout << (has ins ? hull.getBest(t) : 0) << endl;
165             }
166         }
167         return 0;
168     }
169
```