Competitive Programming

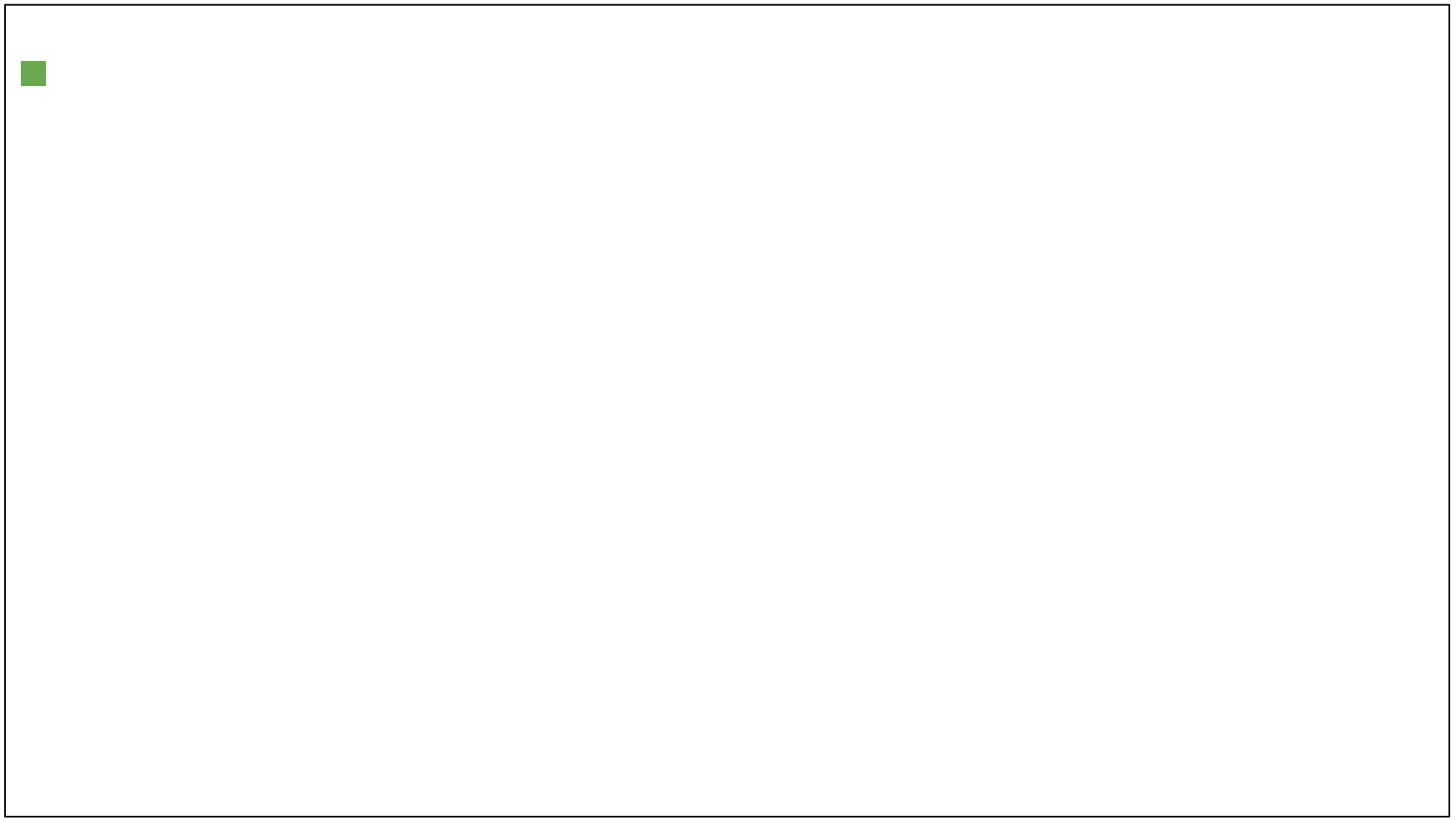
From Problem 2 Solution in O(1)

Computational Geometry

Line Sweep - Rectangles Union

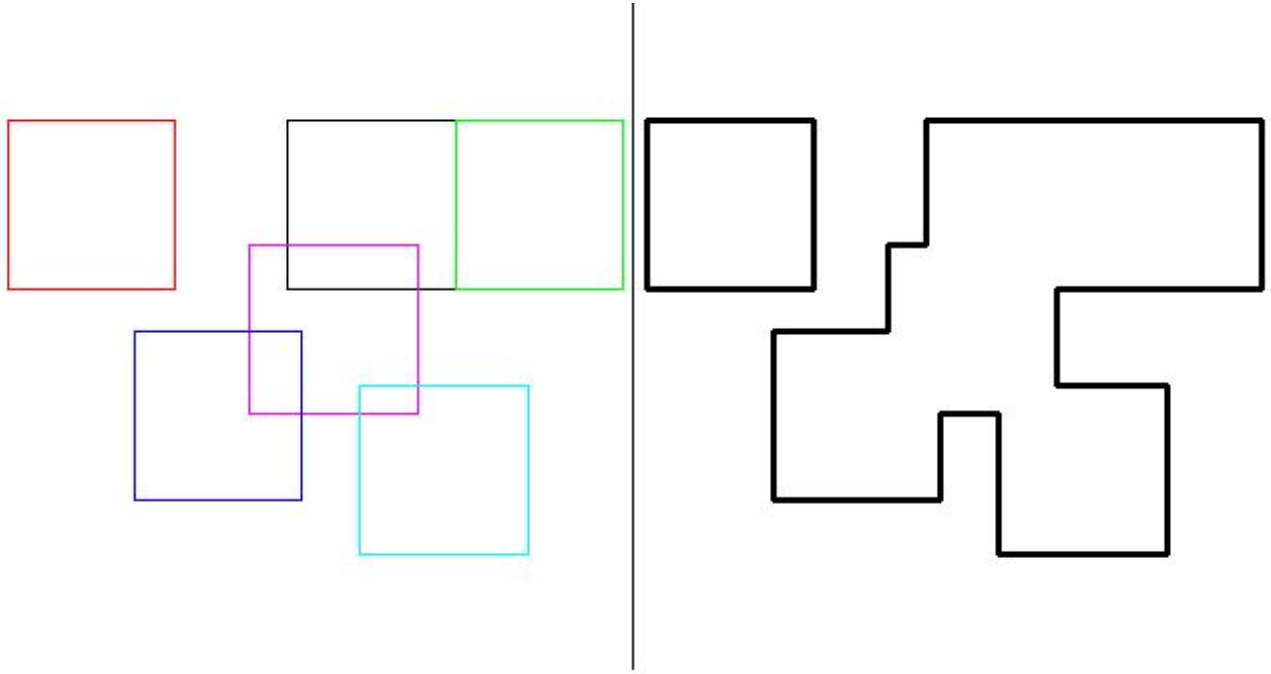
Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University

Rectangles Union



Rectangles Union

- Given set of rectangles, find their union
  - They might intersect. and are axis-aligned

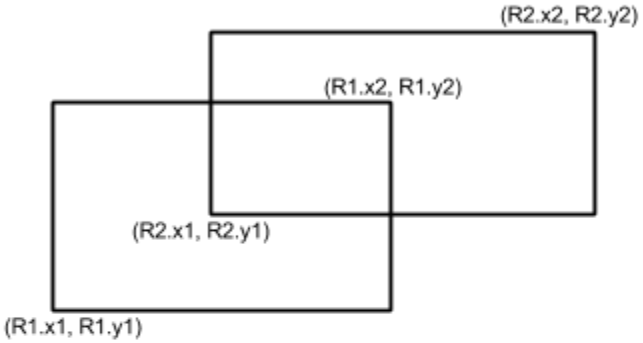


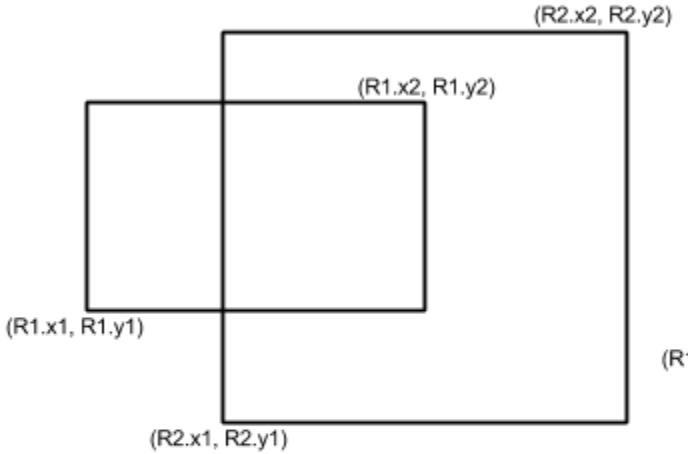
Src: http://cglab.ca/~morin/teaching/2402/notes/planesweep.pdf

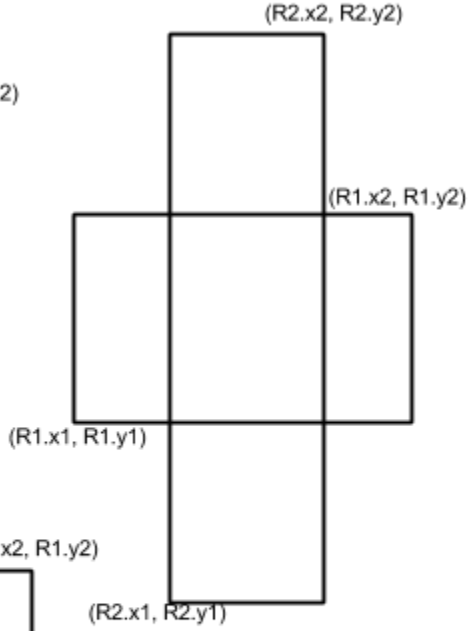
Thinking: Simplification

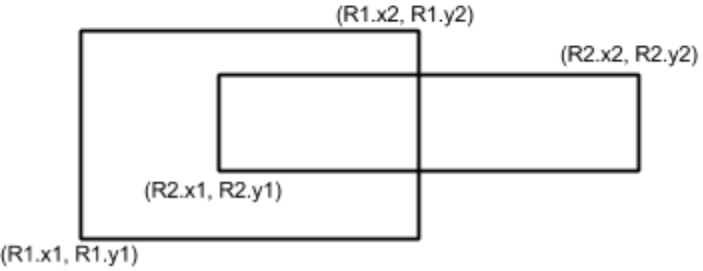
- One might start with N=2 and go up?
  - E.g. Area(r1, r2) = Area(1) + Area(2) - Intersect(r1, r2)
  - This is easy to compute, seems extending in inclusion/exclusion way is not feasible
  - We will show code intersection...study at home
- Restricting the input?
  - Assume all lower edges are fixed on the OX axis
  - Solve this, then extend to general case
  - Seems we can line sweep
  - Events: Each rectangle has 2 vertical edges (in, out)
  - Active area: All rectangles in the current sweep line

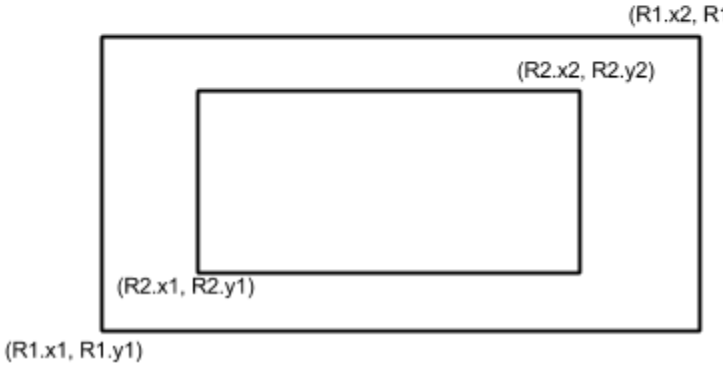
2 Rectangles Intersection











Src: https://discuss.codechef.com/questions/37269/cake1am-editorial

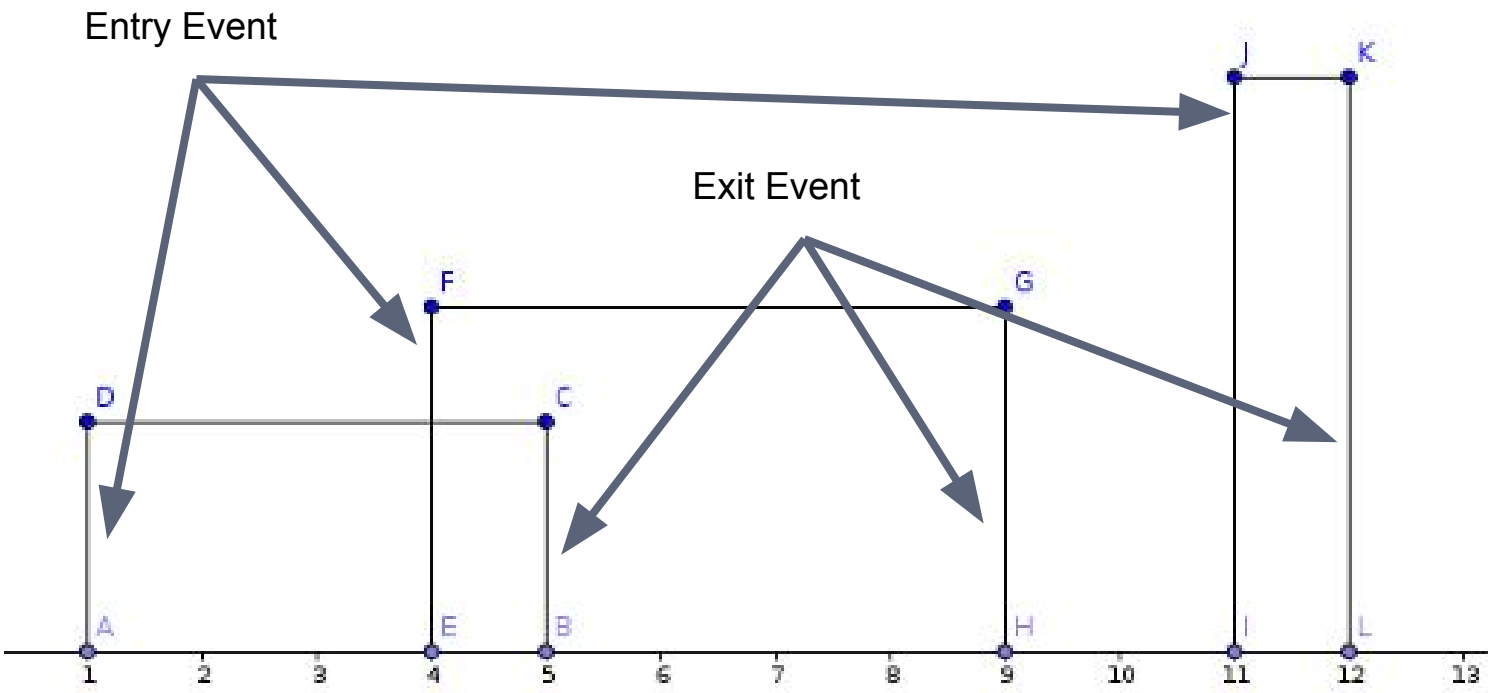
2 Rectangles Intersection

```
struct rect {
  double x1, y1, x2, y2;    // left, bottom, right, and top

  rect(){}
  rect(double x1, double y1, double x2, double y2):
    x1(x1), y1(y1), x2(x2), y2(y2)  { canonicalize(); }

  void canonicalize() {
    if(dcmp(x1, x2) > 0)  swap(x1, x2);
    if(dcmp(y1, y2) > 0)  swap(y1, y2);
  }
};

void intersect(rect a, rect b)
{
    if(b.x2<=a.x1 || b.x1>=a.x2 || b.y2<=a.y1 || b.y1>=a.y2)
    cout<<"No Overlap\n";//No intersection between them
  else
  { //Using the compression method to compress the overlapping area
    if(b.x1 > a.x1)   a.x1 = b.x1;
    if(b.x2 < a.x2)   a.x2 = b.x2;
    if(b.y1 > a.y1)   a.y1 = b.y1;
    if(b.y2 < a.y2)   a.y2 = b.y2;

    cout<<a.x1<<" "<<a.y1<<" "<<a.x2<<" "<<a.y2<<"\n"; //the overlap
  }
}
```

2 Rectangles: Non overlapping

```
// nout = {-2, -1 or CNT} =
//   {no intersection, inside me, or # of extra-sub-rectangles}
rect intersectRectangles(rect a, rect b, rect out[4], int &nout)
{
  nout = -2;
  if(dcmp(b.x2, a.x1) < 0 || dcmp(b.x1, a.x2) > 0
      || dcmp(b.y2, a.y1) < 0 || dcmp(b.y1, a.y2) > 0)
    // Adjust if need boundary edges considered.
    return rect(0, 0, 0, 0); // Do they  intersect?

  nout = -1;
  if(dcmp(b.x1, a.x1) <= 0 && dcmp(b.x2, a.x2) >= 0 &&
      dcmp(b.y1, a.y1) <= 0 && dcmp(b.y2, a.y2)  >= 0)
    return a;      //a Totally inside b

   rect t;
   nout = 0;

  if(dcmp(b.x1, a.x1) > 0)  t = a, t.x2 = b.x1, a.x1 = b.x1;
  out[nout++] = t; // left
  if(dcmp(b.x2, a.x2) < 0)  t = a, t.x1 = b.x2, a.x2 = b.x2;
  out[nout++] = t; // right
  if(dcmp(b.y1, a.y1) > 0)  t = a, t.y2 = b.y1, a.y1 = b.y1;
  out[nout++] = t; // down
  if(dcmp(b.y2, a.y2) < 0)  t = a, t.y1 = b.y2, a.y2 = b.y2;
  out[nout++] = t; // upper
  return a; // a represent the overlapping
}
```

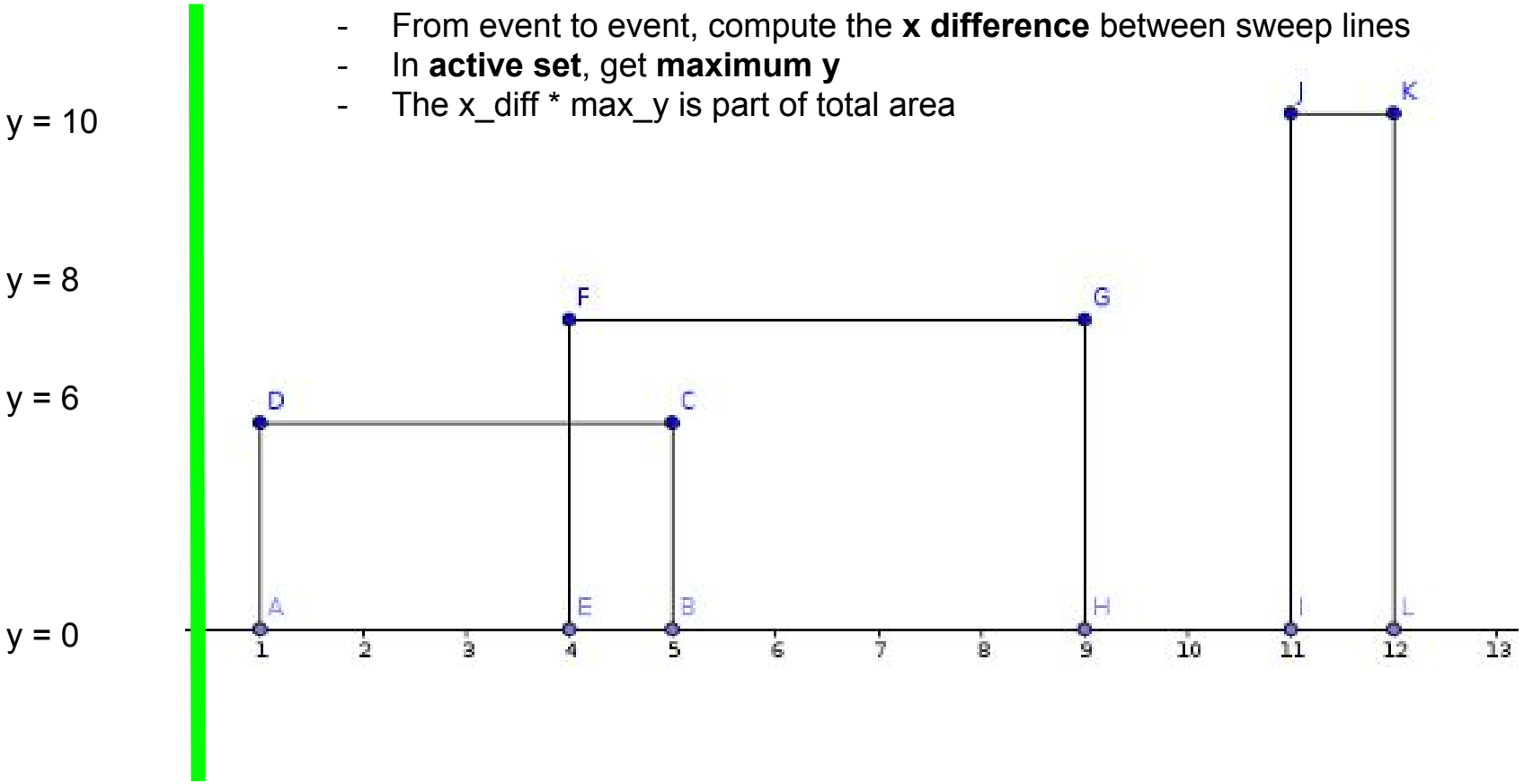
Simplified Rectangles Union



Src: http://codeforces.com/blog/entry/20377

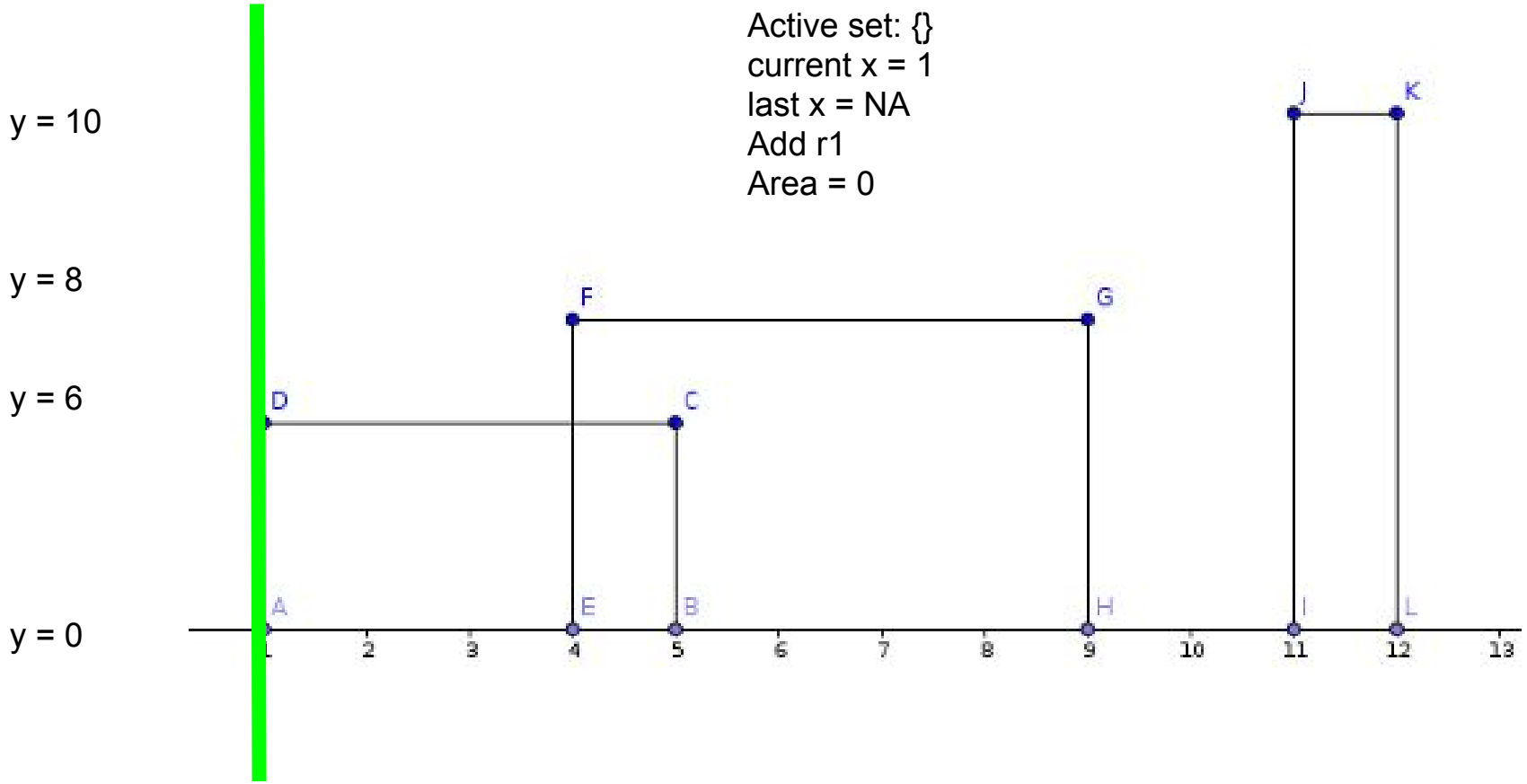
Simplified Rectangles Union

- From event to event, compute the **x difference** between sweep lines
- In **active set**, get **maximum y**
- The x_diff * max_y is part of total area



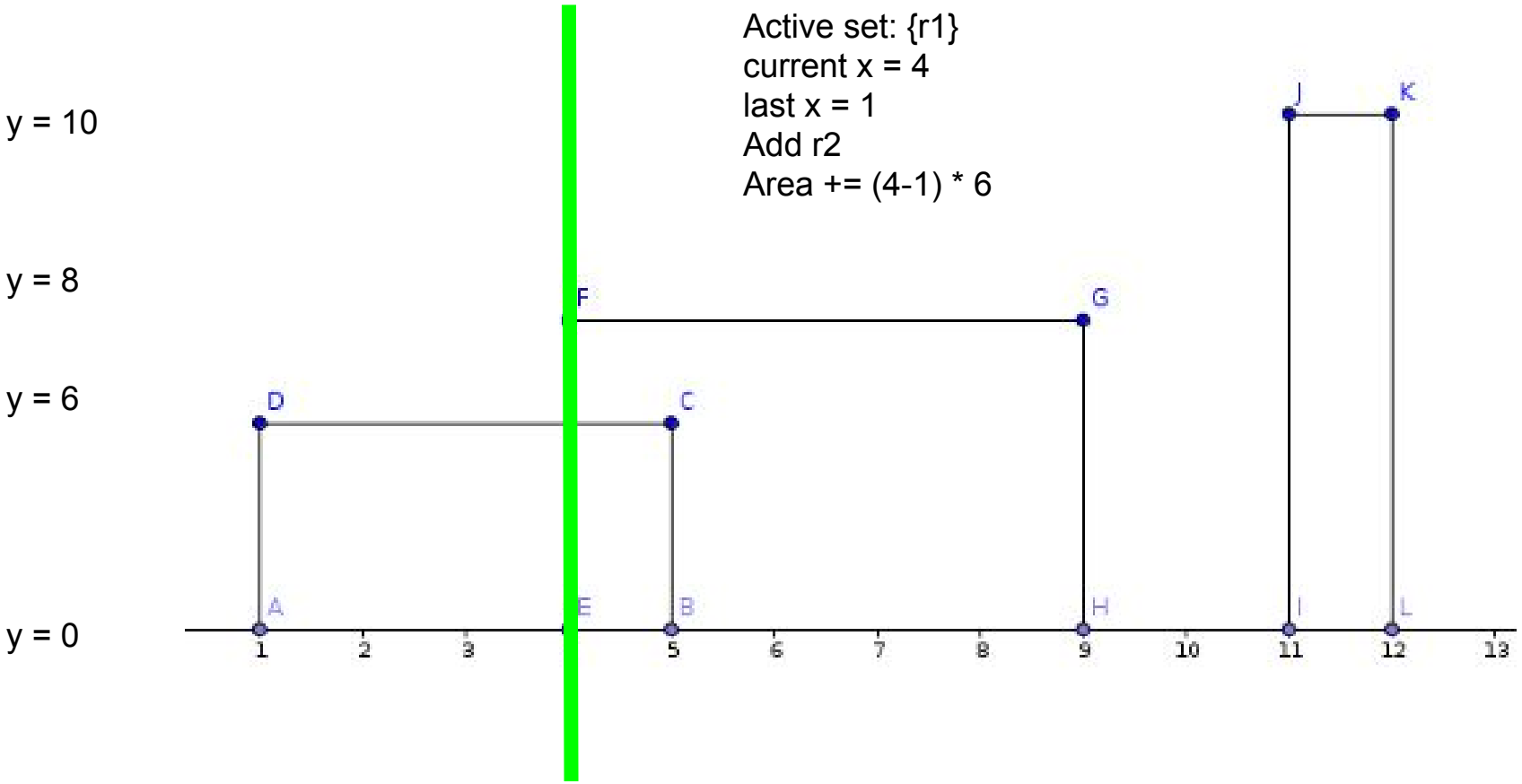
Src: http://codeforces.com/blog/entry/20377

Simplified Rectangles Union



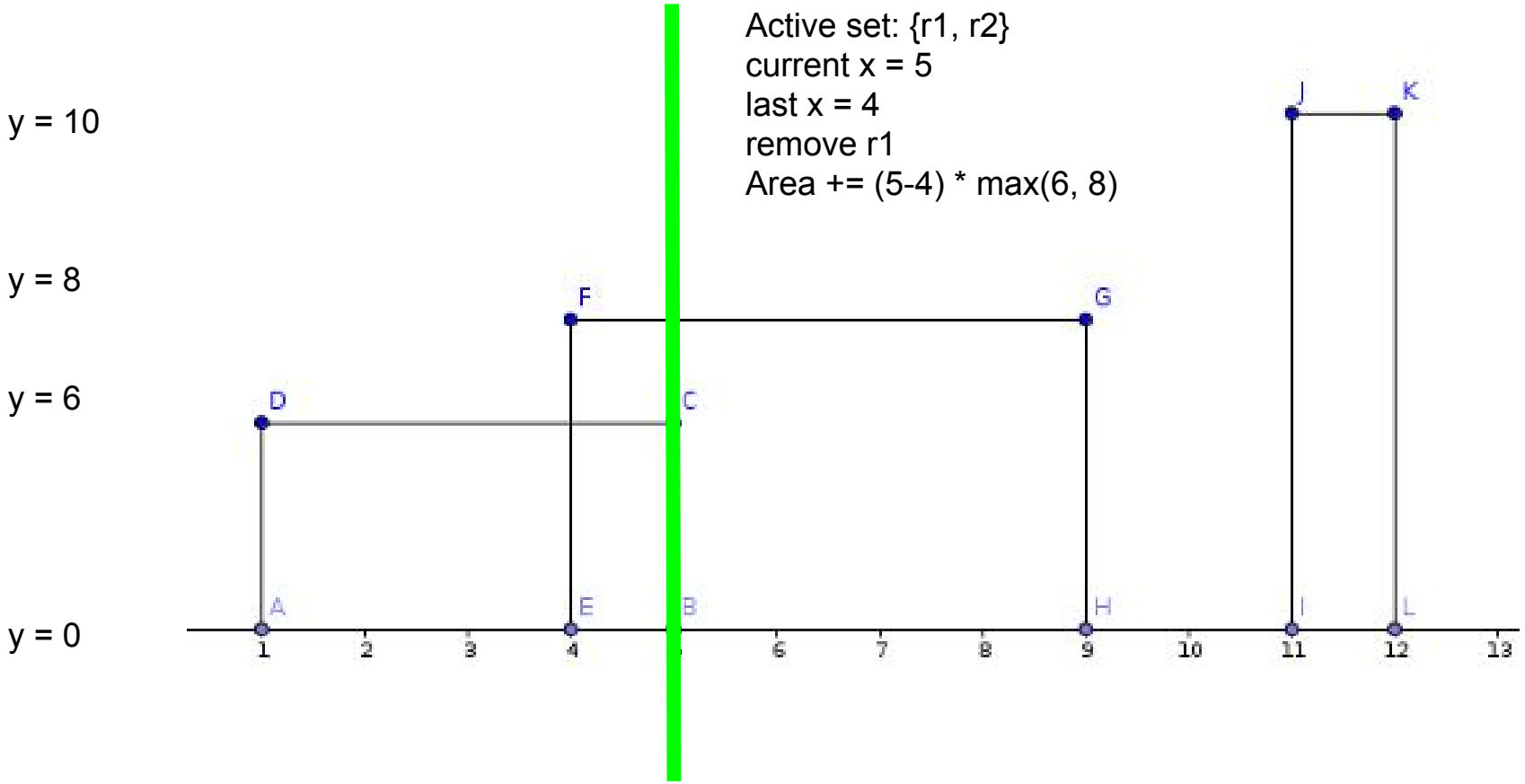
Src: http://codeforces.com/blog/entry/20377

Simplified Rectangles Union



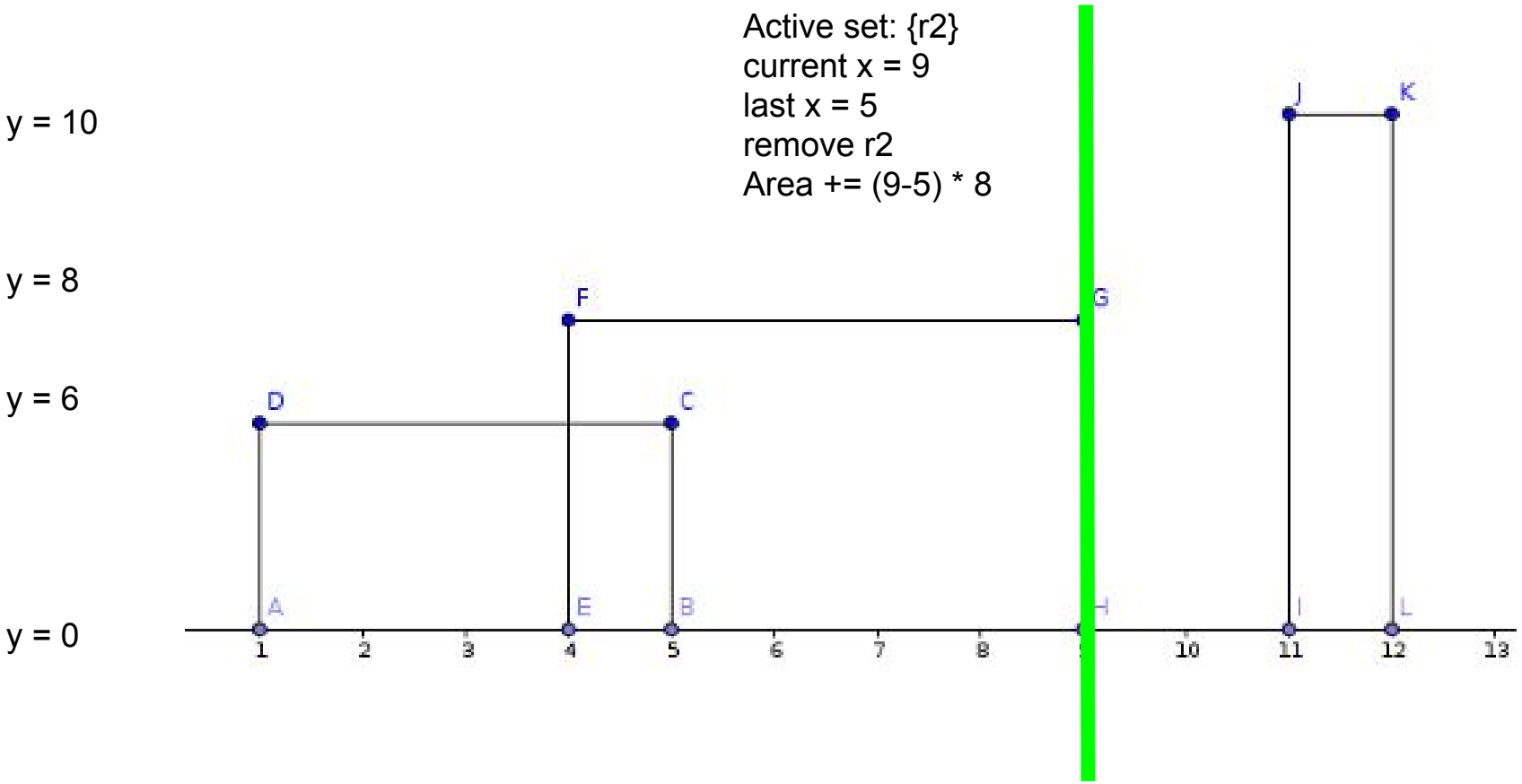
Src: http://codeforces.com/blog/entry/20377

Simplified Rectangles Union



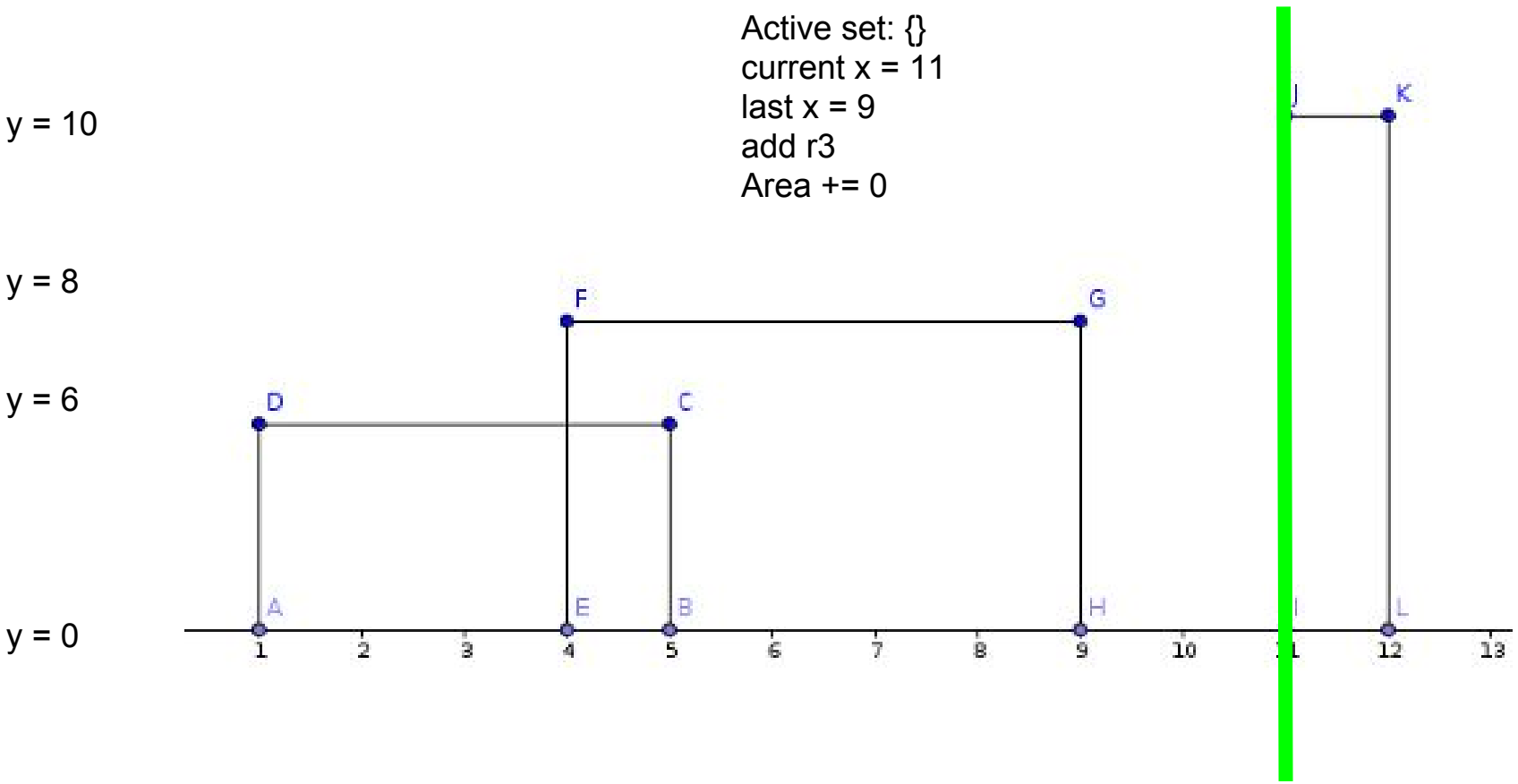
Src: http://codeforces.com/blog/entry/20377

Simplified Rectangles Union



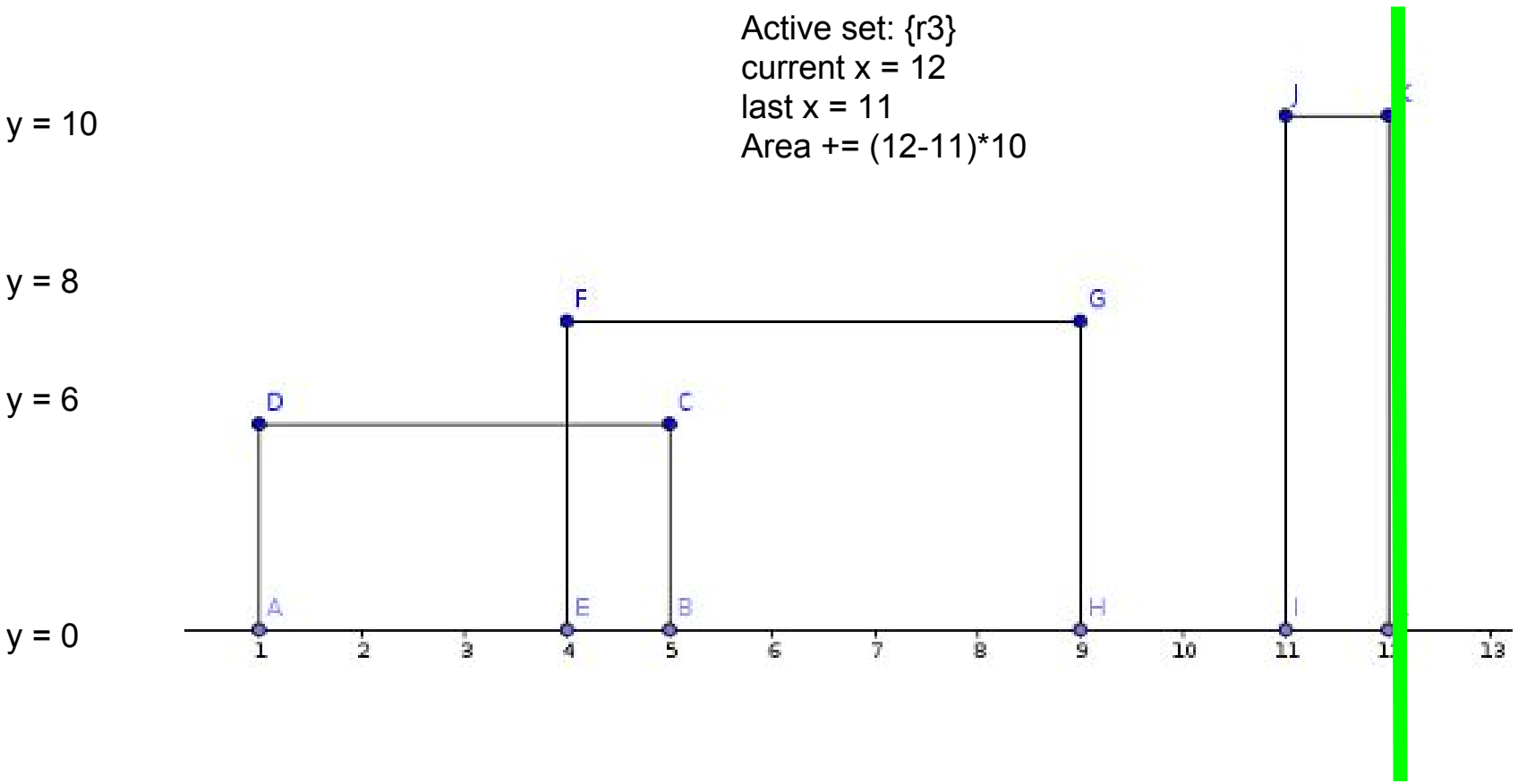
Src: http://codeforces.com/blog/entry/20377

Simplified Rectangles Union



Src: http://codeforces.com/blog/entry/20377

Simplified Rectangles Union

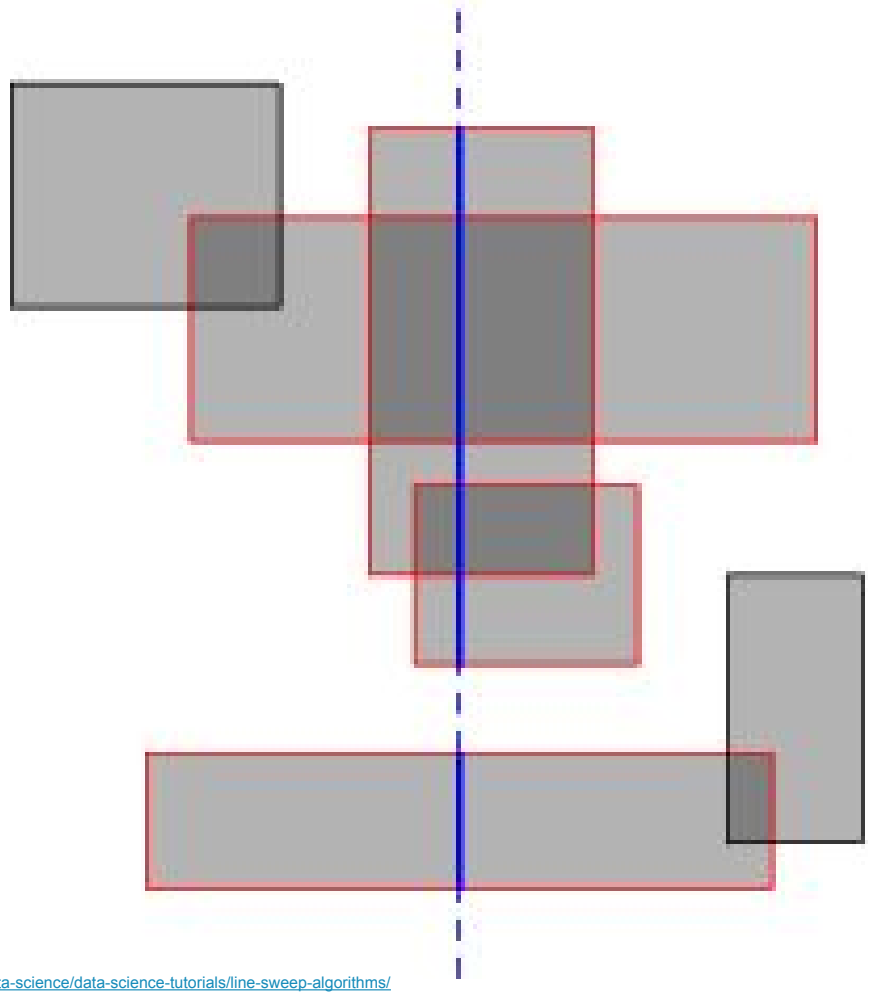


Src: http://codeforces.com/blog/entry/20377

Rectangles Union

- What about the general version?
- We know **x difference**..and the **active rectangles**, but max Y has **more** than the actual **height**
- We know have a new sub-problem. Given set of rectangles, find their covered height!
  - **Horizontal sweep line** running from top to bottom
  - Identify gap by counting how many rectangles are in

Rectangles Union



Src: https://www.topcoder.com/community/data-science/data-science-tutorials/line-sweep-algorithms/

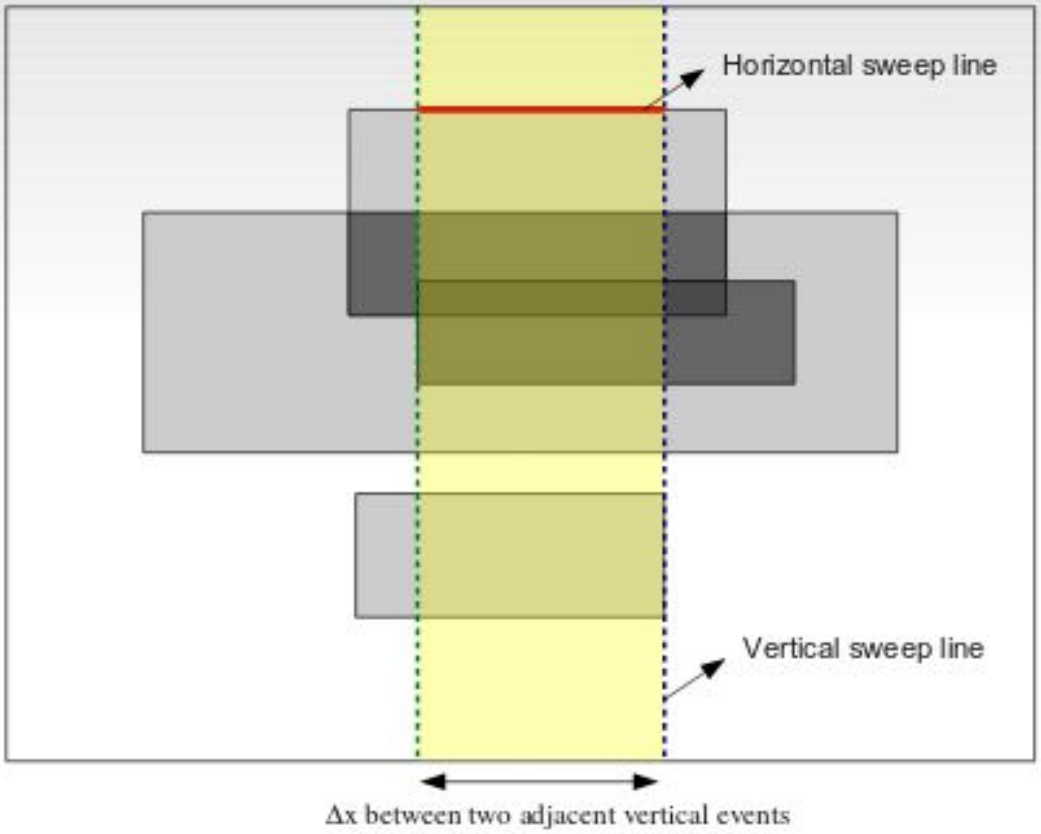
Rectangles Union

- Sweep horizontally and count how many rectangles we have
- New rectangle start => increment counter
  - Remember the Y of the first rectangle (First_Y)
- End of rectangle => decrement counter
- If counter after decrement = zero
  - Then we are about to have gap
  - Y difference = First_Y - Current_Y

Rectangles Union

Remember to keep the first Y

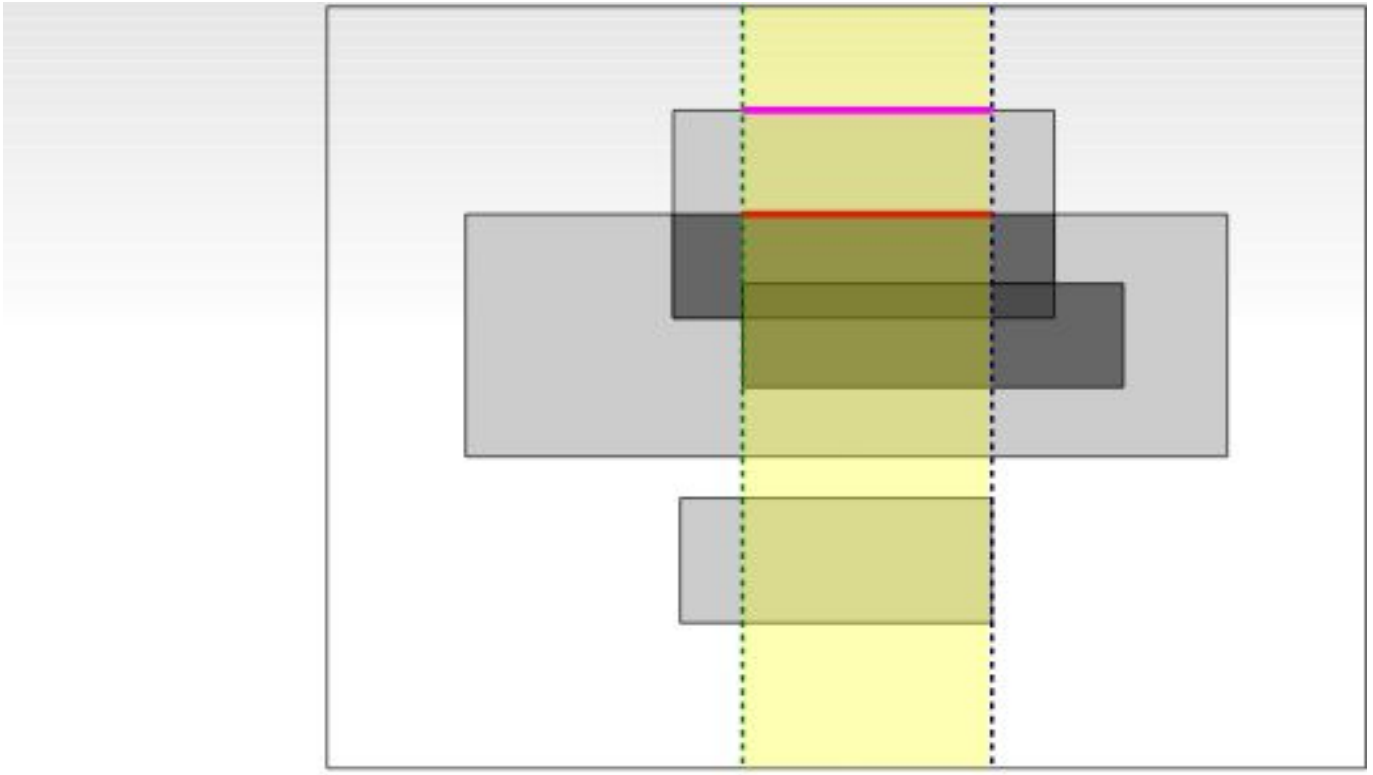
Keep going till meet a gap
e.g. counter = 0



Count: 1

Src: https://olympiad.cs.uct.ac.za/presentations/camp1_2009/linesweep.pdf

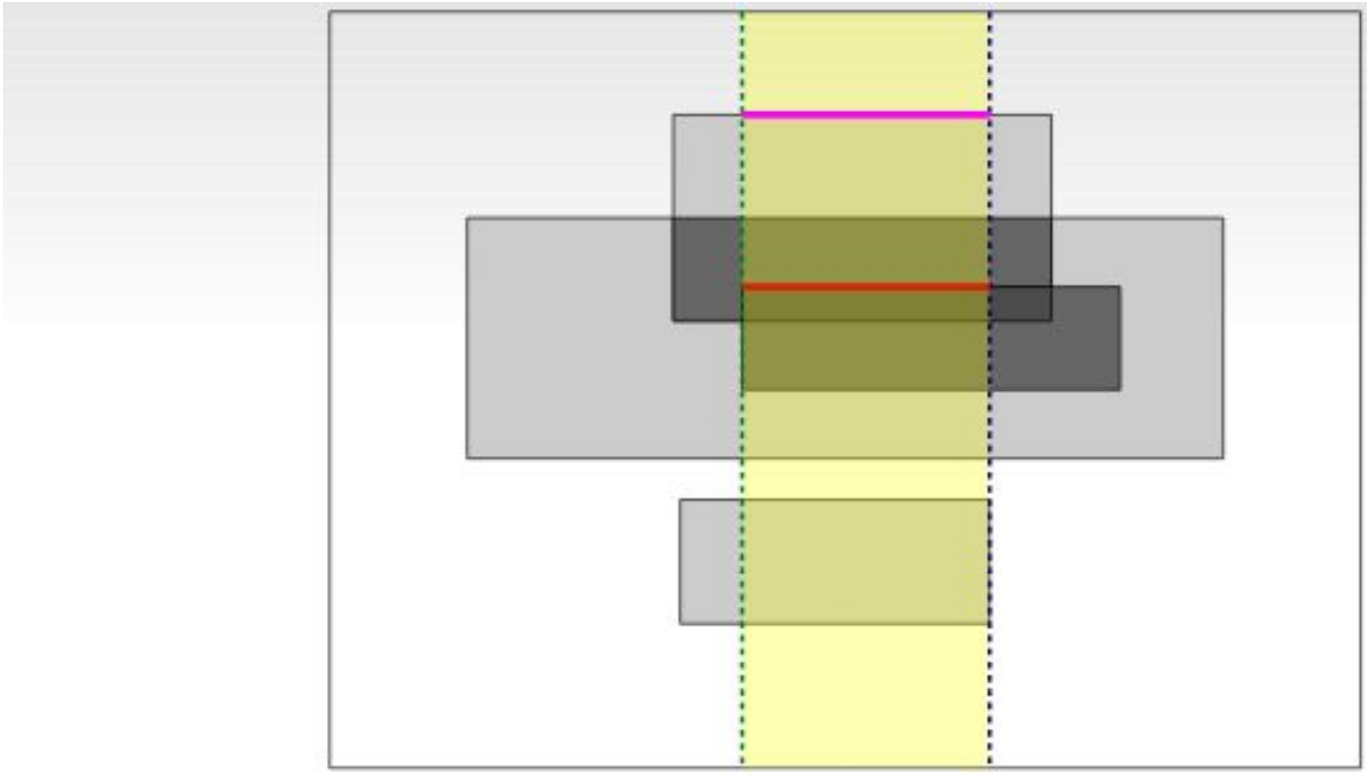
Rectangles Union



Count: 2

Src: https://olympiad.cs.uct.ac.za/presentations/camp1_2009/linesweep.pdf

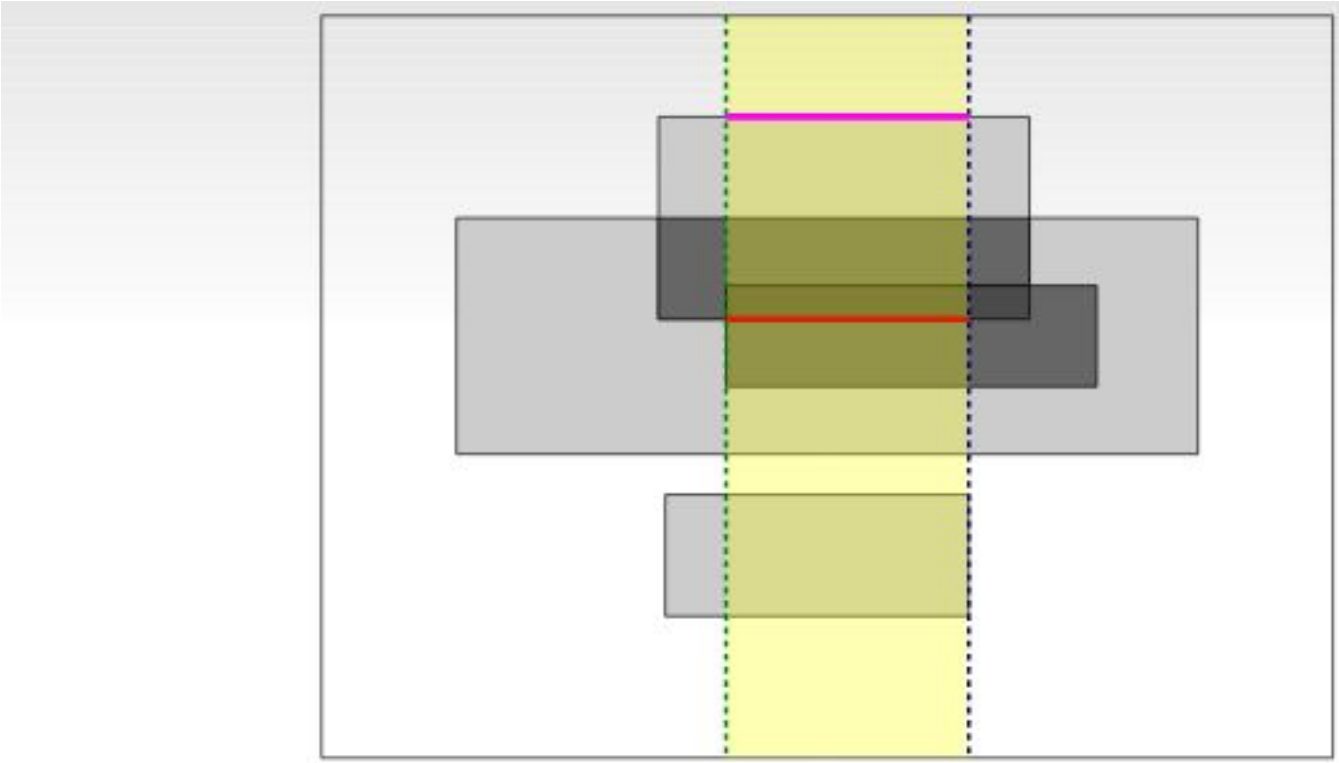
Rectangles Union



Count: 3

Src: https://olympiad.cs.uct.ac.za/presentations/camp1_2009/linesweep.pdf

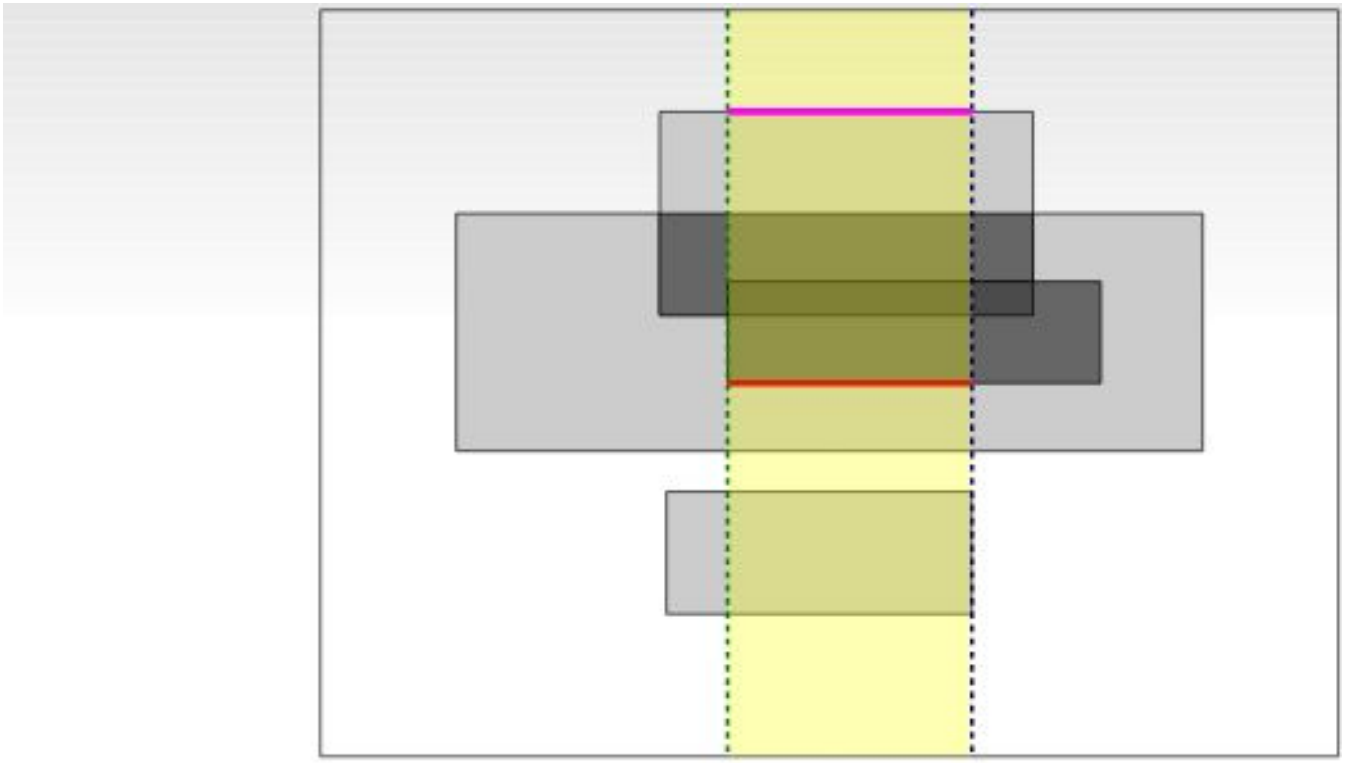
Rectangles Union



Count: 2

Src: https://olympiad.cs.uct.ac.za/presentations/camp1_2009/linesweep.pdf

Rectangles Union



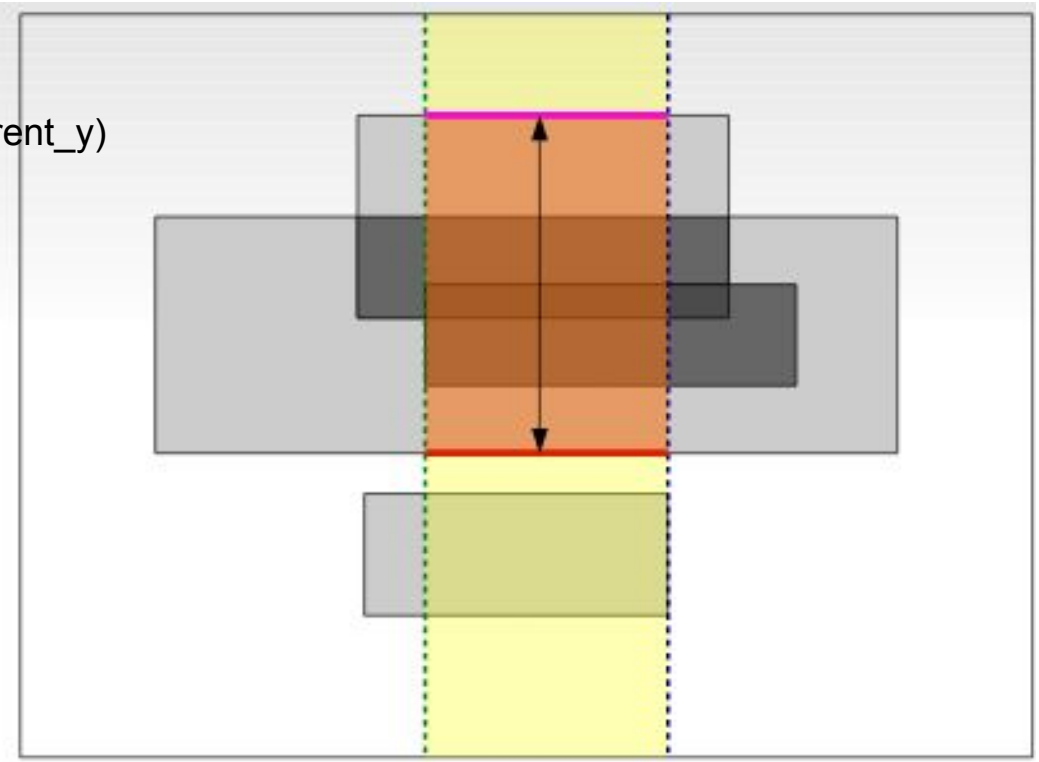
Count: 1

Src: https://olympiad.cs.uct.ac.za/presentations/camp1_2009/linesweep.pdf

Rectangles Union

counter = 0

area += delta_x * (first_y - current_y)



Src: https://olympiad.cs.uct.ac.za/presentations/camp1_2009/linesweep.pdf

Rectangles Union

- So far we have nested line sweep
  - Sorting X events is nlogn [vertical sweep]
  - Iterate on events O(n)
  - Inside it, horizontal sweep O(nlogn) ⇒ O(n^2 logn)
- To improve order
  - Have another copy of events, sorted initially over Y
  - Create a boolean array to mark active rectangles
  - In the horizontal sweep, iterate over all rectangles copy array, check if active or not
  - If active do counter & area computations ⇒ O(n^2)

Implementation: Declarations

```
struct event {
  int ind, type;
  event() {}
  event(int ind, int type) : ind(ind), type(type) {}
};
struct point {  int x, y;   };

const int RECT_MAX = 10000 + 9;
const int ENTRY = 0, EXIT = 1;
point rects[RECT_MAX][2];
bool inActiveSet[RECT_MAX];
event events_v[2 * RECT_MAX], events_h[2 * RECT_MAX];

bool cmpX(event a, event b) {
  return rects[a.ind][a.type].x < rects[b.ind][b.type].x;
}
bool cmpY(event a, event b) {
  return rects[a.ind][a.type].y < rects[b.ind][b.type].y;
}
```

Src: https://olympiad.cs.uct.ac.za/presentations/camp1_2009/linesweep.pdf

Implementation: Read / Sort

```
long long area = 0;
int n = 0, eventsCnt = 0;  // # rectangles, edges

scanf("%d", &n);
for (int i = 0; i < n; ++i) {  // assume rectangle 2 points are ordered
  scanf("%d %d %d %d", &rects[i][0].x, &rects[i][0].y, &rects[i][1].x, &rects[i][1].y);
  events_v[eventsCnt] = event(i, ENTRY), events_v[eventsCnt + 1] = event(i, EXIT);
  events_h[eventsCnt] = event(i, ENTRY), events_h[eventsCnt + 1] = event(i, EXIT);
  eventsCnt += 2;
}
sort(events_v, events_v + eventsCnt, cmpX);
sort(events_h, events_h + eventsCnt, cmpY);
```

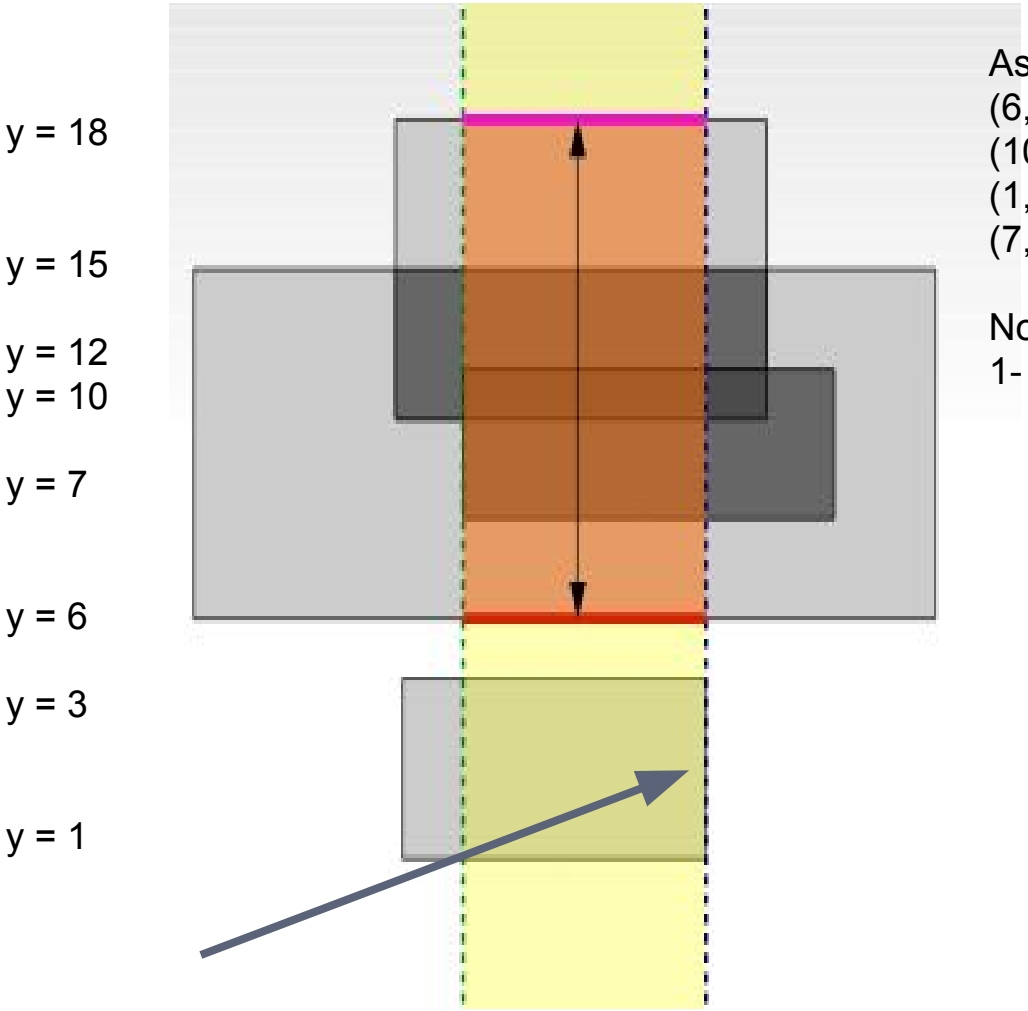
Implementation: Process

```
inActiveSet[events_v[0].ind] = 1;
for (int v = 1; v < eventsCnt; ++v) {  // Vertical sweep
  event c = events_v[v], p = events_v[v - 1];
  int cnt = 0, first_rect, delta_x, delta_y;

  if ((delta_x = rects[c.ind][c.type].x - rects[p.ind][p.type].x) == 0)
    continue;

  for (int h = 0; h < eventsCnt; ++h)
    if (inActiveSet[events_h[h].ind]) {  // Horizontal sweep
      if (events_h[h].type == ENTRY) {
        if (cnt++ == 0)
          first_rect = h;
      } else if (--cnt == 0) {
        delta_y = rects[events_h[h].ind][EXIT].y - rects[events_h[first_rect].ind][ENTRY].y;
        area += delta_x * delta_y;
      }
    }
  inActiveSet[c.ind] = (c.type == ENTRY);
}
```

Segment tree vs horizontal sweep

- In horizontal sweep line, we iterate on rectangle Ys (y_start, y_end = an **interval**)
- Restate problem as **given some intervals** with Add/Remove interval **queries**:
  - Target Query: What is the total covered length?
  - A segment tree problem, but tricky to make it efficient
- One can have 10000 rectangles, but Ys so big
  - Compress the Y's, Remap intervals to the actual Y's
  - E.g. interval (10009, 50000) => (17, 35)

Segment tree



Assume Segment tree has NOW 4 intervals:
(6, 15) = r1
(10, 18) = r2
(1, 3) = r3
(7, 12) = r4

Notes:
1- Covered length: (18-6+1) + (3-1+1) = 16

Segment tree

Assume Segment tree has NOW 4 intervals:
(6, 15) = r1
(10, 18) = r2
(1, 3) = r3
(7, 12) = r4

- See covered nodes below
- 2 important cases
- The first time you add 1 to leaf node (covered+=1)
- The last time you remove 1 to leaf node (covered-=1)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	1	1	1			2	2	2	2	3	3	3	2	2	2	2	2	2	

Segment tree: Efficiency

- Adding/Removing intervals are same
  - Kind of update query with +1 or -1 for interval
- Such query expands **range**, not just **position**
  - O(nlogn) vs O(logn)
  - Trick that reduce time highly: **Lazy propagation**
  - This trick is enough for fast processing
- One more trick, the query is always about whole tree range (all current intervals)
  - Let your call to updates compute also covered length
- O(n log(n) log(n)) code1, code2

Think about

- Given set of rectangles
  - Calculate the perimeter (length of the union boundary)
  - Generate set of non-overlapping rectangles
  - The maximum subset of rectangles intersecting together
- Find union of set of circles
- Further readings
  - link1, link2, link3, link4

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً