Algorithm 1: Find candidate disks with plane sweeping technique

Input: $\mathcal{T}[t_i]$: positions in timestamp t_i, sorted by x-axis values, ϵ: flock diameter, μ: minimum size of flock
Output: $\mathcal{C}$: candidate disks for timestamp t_i, $\mathcal{B}$: active boxes in timestamp t_i

$\mathcal{C} \leftarrow \emptyset$, $\mathcal{B} \leftarrow \emptyset$
foreach $p_r \in \mathcal{T}[t_i]$ **do** // analyze elements in increasing x-values
  $\mathcal{P} \leftarrow \emptyset$ // list of elements of current box defined by p_r
  foreach $p_s \in \mathcal{T}[t_i] : \mid p_s.x - p_r.x \mid \leqslant \epsilon$ **do** // test only elements inside 2ϵ x-band
    if $|p_s.y - p_r.y| \leqslant \epsilon$ **then** // check if p_s is inside 2ϵ y-band
      $\mathcal{P} \leftarrow \mathcal{P} \cup p_s$ // add element p_s to box
  foreach $p \in \mathcal{P} : p.x \geqslant p_r.x$ **do** (// elements inside right half of box
    if $dist(p_r, p) \leqslant \epsilon$ **then** // calculate pair distance
      let $\{c_1, c_2\}$ be disks defined by $\{p_r, p\}$ and radius $\epsilon/2$
      foreach $c \in \{c_1, c_2\}$ **do**
        if $|c \cap \mathcal{P}| \geqslant \mu$ **then** // check the number of entries in disk
          $\mathcal{C} \leftarrow \mathcal{C} \cup c$ // add c to candidate disks
          $\mathcal{B} \leftarrow \mathcal{B} \cup box(p_r)$ // add box to active boxes
return $\mathcal{C}, \mathcal{B}$

Algorithm 2: Filter out disks which are subsets

```
1  Algorithm FilterCandidates(B)
       Input: B: active boxes of timestamp t_i, sorted by x-axis values
       Output: C: final set of disks for timestamp t_i
2      C ← ∅
3      for j ← 0 to j ⩽ |B| do
4          for k ← j + 1 to k ⩽ |B| do
5              if IntersectsWith(B[j], B[k]) then
6                  foreach c ∈ B[j].disks do
7                      C ← InsertDisk (C, c)
8              else // No intersection.
9                  break

10 Procedure InsertDisk(C, c)
       Input: C: set of disks, c: new disk
11     foreach d ∈ C do
12         if c.sign ∧ d.sign = c.sign && dist(c, d) ⩽ ϵ then // c can be a subset of d
13             if d ∩ c = c then // Remove chance of false-positive
14                 return C // No need to insert c
15         else if c.sign ∧ d.sign = d.sign then // d can be a subset of c
16             if c ∩ d = d then // Remove chance of false-positive
17                 C ← C \ d // Remove d
18     return C ∪ c
```