```
Algorithm sortRR(L).
    if L.isEmpty() then
        return L.

    cur ← L.first().

    while !L.isLast(cur) do
        cur ← L.after(cur).
        b ← cur
        while !L.isFirst(b). do {
            c ← L.before(b).
            if c.color = RED ∧ b.color = BLUE then.
                L.swapElement(c, b).
2           b ← c
        }
```

```
Algorithm sortRGB(L)
    if L.isEmpty() then
        return L.
    cur ← L.first()
    while !L.isLast(cur) do
        cur ← L.after(cur).
        b ← (cur)
        while !L.isFirst(b)
            c ← L.before(b).
            if b.color() != c.color() ∧ b.color() = RED then
                L.swapElement(b, c)
            if b.color() = BLUE ∧ c.color() = GREEN,
                L.swapElemnt(b, c).
            b ← c.
```

∧ and. conjunction ∩ intersection.

∨ or disjunction ∪ union

```
Algorithm noduplicate(A, B)
    PQ ← new priority queue using C.
    while ! A.isEmpty() do.
               e
        PQ. ~~insert~~( A.remove(A.first()); e).
            insertLast
    While ! B.isEmpty() do.
        ~~[illegible]~~      e
        PQ.insertLast(B.remove(B.first()), ~~B~~ ).
    current ← ~~[illegible]~~ PQ.remove min()
                                   → A.InsertLast.(current)
    while ! PQ.~~insertLast~~.isEmpty().do.
        ~~[illegible]~~ e ← PQ.removeMin()
        if current != e then.
            A.insertLast(e)
    return A._
```