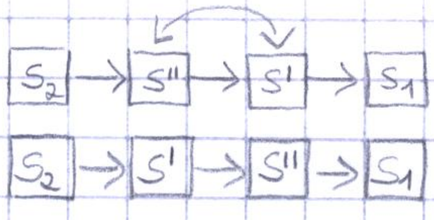
$S_1 \leftarrow Y.succ(s')$
$S_2 \leftarrow Y.pred(s'')$
$Y.swap(s', s'')$
$X.delete(S_1 \cap s')$
$X.delete(S_2 \cap s'')$
$X.insert(S_1 \cap s'')$
$X.insert(S_2 \cap s')$
Ausgabe: "$p = s' \cap s$" "



} end switch } X.delete(p)
end while }

- **Annahmen:**
  - Alle x-Koordinaten von Segment-Anf-, -End- und -Schnittpktn sind paarweise verschieden (d.h. in einem Pkt schneiden sich höchstens 2 Segmente)
  - $\nexists$ vertikalen Segmente

- **Laufzeit:**
  - Alle Operationen auf X und Y benötigen höchstens Zeit $O(\log n)$ (siehe: Operationen auf X bzw Y-Struktur, 2 Seiten vorher)
  - Die Initialisierung vor der while-Schleife benötigt Zeit $O(n \log n)$ Einfügen von $2n$ Anfangs- bzw Endpktn
  - Die while-Schleife wird $2n+s$ $s \hat{=}$ # Schnittpkte mal ausgeführt. Jeder Schleifendurchlauf kostet $O(\log n)$ da in jedem Fall konstant viele Operationen auf X und Y und konstant viele Schnitttests ausgeführt werden

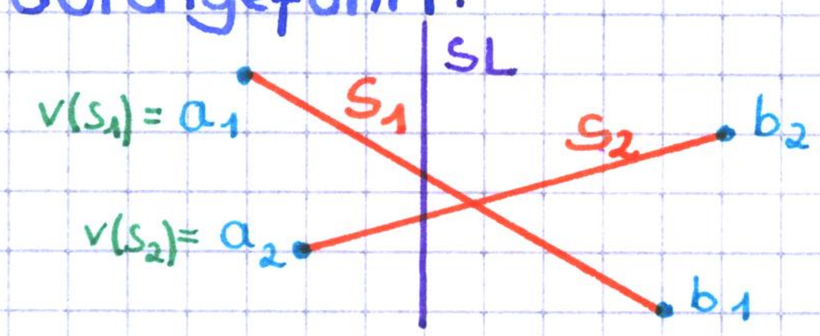
⇒ **insgesamt:** $O((n+s)\log n) = O(n \cdot \log n + s \cdot \log n)$

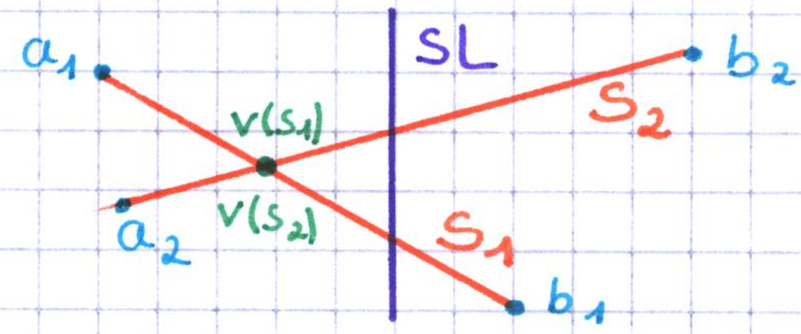
Dies ist wie gewünscht output-sensitiv !!

3.2.1 1. Modifikation

orientation-Tests statt "compare" Fkt

- Zur Definition der linearen Ordnung in der Y-Struktur wurde im obigen Algorithmus die "compare"-Fkt verwendet. Anstelle dieser Fkt werden jetzt nur noch orientation-Tests durchgeführt.





Für jedes Segment s in Y werden der letzte Schnittpkt bzw der linke Endpkt als v(s) gespeichert.

1. Fall: orientation $(v(S_1), b_1, v(S_2)) > 0$
⇒ $S_1 < S_2$

2. Fall: orientation $(v(S_1), b_1, v(S_2)) < 0$
⇒ $S_1 > S_2$

3. Fall: orientation $(v(S_1), b_1, v(S_2)) = 0$
⇒ $v(S_1) = v(S_2)$

⇒ orientation $(v(S_2), b_1, b_2)$ $< 0 \Rightarrow S_1 > S_2$; $= 0 \Rightarrow$ collinear; $> 0 \Rightarrow S_1 < S_2$