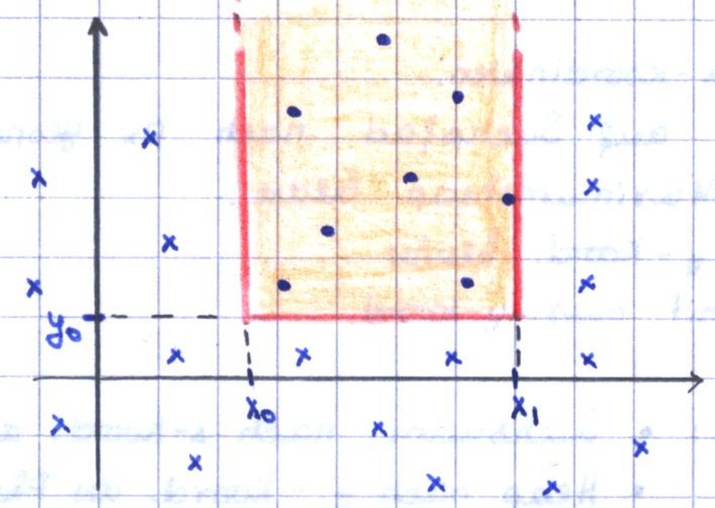
5.3.5 Problem 2 + Lösung:

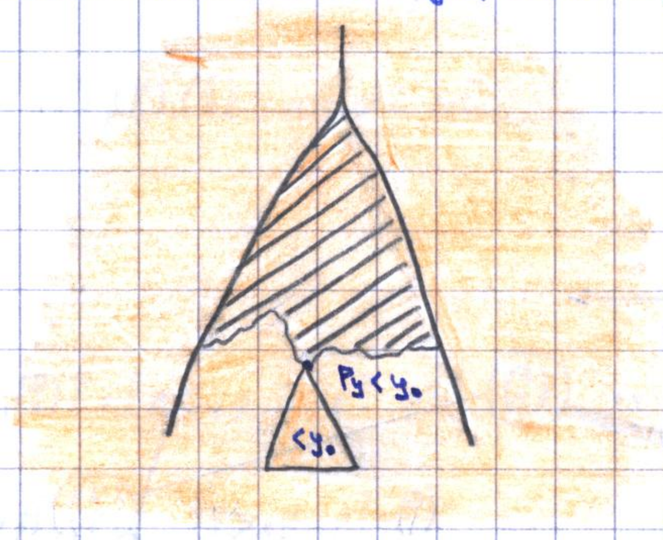
Wozu Heap-Eigenschaft nach y-Koordinaten?
→ 1½ - dimensionale Range Abfragen
(Halb offene oder 3 seitige).

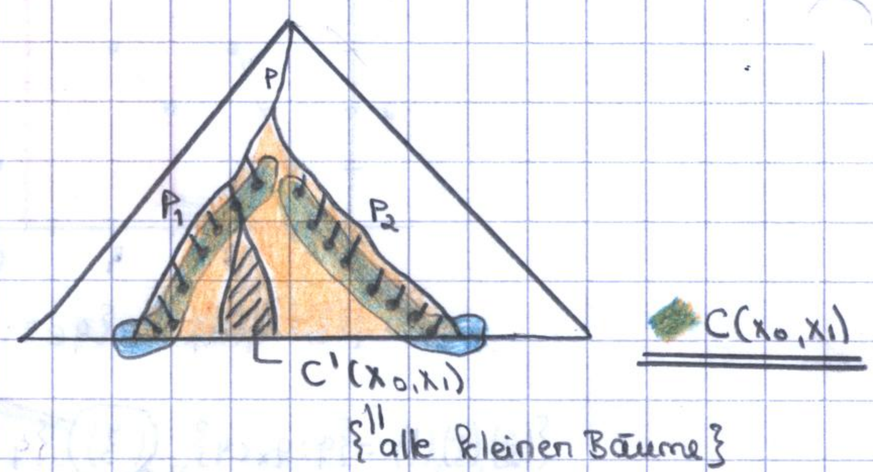
Geg: $x_0, x_1, y_0, \ldots, S \subset \mathbb{R}^2$

Ges: alle $p \in S$ mit
$x_0 \leq p_x \leq x_1$ und
$p_y \geq y_0$



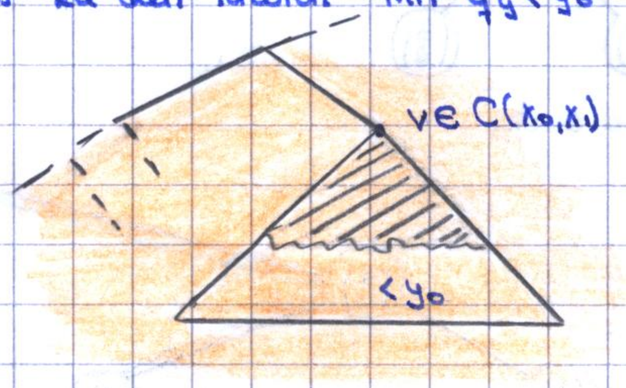
Lösung:
- Filtern des Pfade: $\forall$ Knoten $v \in P_1 \cup P_2 \cup P$ gib enthaltenen Pkt p aus, falls $x_0 \leq p_x \leq x_2 \wedge p_y \geq y_0$
  → kostet Zeit $O(\log n)$. Mit Ausgabe kostet dies $O(\log n + k)$.
- Rest der Ausgabe (normalerweise müssen alle $v \in P_1 \cup P_2 \cup P$ und zwischen P_1 und P_2 betrachtet werden. Oben nun $v \in P_1 \cup P_2 \cup P$, hier also die restlichen, dh. also: v zwischen P_1 und P_2) ist im oberen Bereich von $C'(x_0 x_1)$ gespeichert (wg. Heapeigenschaft)





{"alle kleinen Bäume"}

Auflisten dieser Pkte:
Starte in den Knoten von $C(x_0, x_1)$.
Scanne den jeweiligen Unterbaum bis zu den Knoten mit $q_y < y_0$
→ kostet $O(\text{Beitrag zur Ausg} + 1)$.



Laufzeit: $\forall v \in C(x_0, x_1)$: $\sum_{v \in C(x_0,x_1)} O(\text{Beitrag zur Ausg} + 1) =$

$= O(\log n + K')$, wobei K' = Beitrag zur Gesamtlösung von $C'(x_0 x_1)$

$\text{Höhe}(v) + 1 + O(\text{Beitr. zu Ausg}(v_1) + 1) + 1 + O(\ldots) + \ldots + \text{Höhe}(v) + 1 + O(\ldots) + \ldots = 2 \cdot \log n + \underbrace{\sum_{v \in C(x_0,x_1)} O(\text{Beitr. zu Ausg} + 1)}_{=: K'} = O(\log n + K')$

⇒ Gesamtlaufzeit: $a \log n + k + O(\log n + K') = \underline{O(\log n + \tilde{K})}$ $\tilde{K} = k + K'$, # Ausgaben.

5.3.6 Satz (Zusammenfassung):

Der Priority-Search Tree verwaltet eine Menge von n Pkten im $\mathbb{R}^2$ unter folgenden Operationen und Laufzeiten:
- Insert / Delete: $O(\log n)$
- Drei-seitige Bereichsabfrage: $O(\log n + K)$, K = # Ausgaben.

und benötigt Speicherplatz: $O(n)$.

5.3.7 Anwendung: