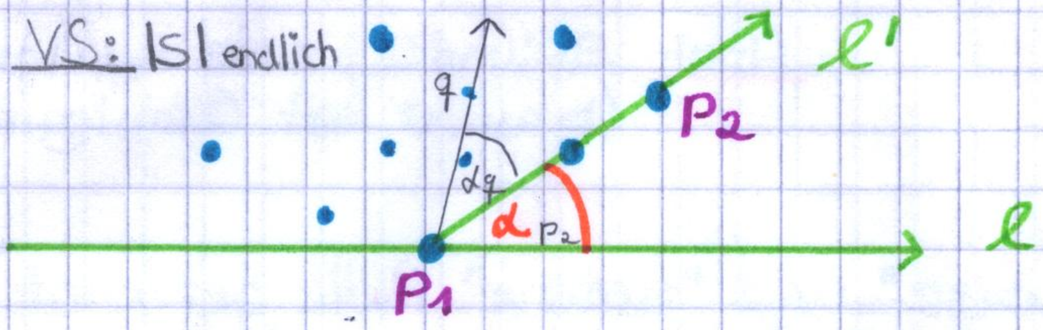
- **Korollar:**
  J.A. braucht Berechnung von $CH(S)$ Zeit $\Omega(n \log n)$.

1.1.1 Algorithmus I: Gift wrapping



- **Algorithmus:**
  - Startpkt P_1, Pkt mit kleinster y-Koord
    Falls mehrere ⇒ wähle linkosten
    - horizontaler Strahl ℓ durch P_1 solange gegen UZS drehen bis man nächsten Pkt trifft
      $\forall q \in S \setminus \{P_1\}$ sei α_q Winkel zwischen ℓ und $\overrightarrow{pq}$
      wähle P_2 so, dass α_{P_2} minimal
      Falls mehrere ⇒ wähle den Pkt mit maximaler Entfernung zu P_1
      ⇒ lineare Suche in S nach Min bzgl Winkelordnung
    - Alg bei P_2 fortsetzen mit Strahl $\ell = \overrightarrow{P_1P_2}$
    - Wiederholen bis P_1 wieder erreicht wird
- Korrektheit: siehe Übung 1.1 a)
- **Laufzeit:** Algorithmus ist output-sensitiv
  $|S| = n$, Anzahl Ecken $= h$
  P_1: lineare Suche nach Min bzgl. yx-Ordn in S kostet $O(n)$
  $P_2, \dots, P_h$: ——"—— Winkelordn ——"——
  ⇒ **insgesamt: $O(h \cdot n)$** nicht so gut, da im schlechtesten Fall $O(n^2)$
  worst-case: $h = n \Rightarrow O(n^2)$
  bester Fall: $h = const \Rightarrow O(h \cdot n) = O(n)$

- **Details der Implementierung:**
  - **Winkelberechnung:**
    2 Nachteile: langsam, ungenau $\left(\alpha_q = \arctan \frac{q_y - p_y}{q_x - p_x}\right)$ da trigonometr Fktn
    ⇒ **besser: orientation:**

right-turn	im UZS	neg. orient	orientation < 0
left-turn	gegen UZS	pos. orient	orientation > 0
colinear	/	Orient $= 0$	orientation $= 0$

$$\begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix} = (q_x - p_x)\cdot(r_y - p_y) - (q_y - p_y)\cdot(r_x - p_x)$$

⇒ 2 Multiplikationen und 5 Subtraktionen
Dies ist viel einfacher, schneller und genauer als die Winkelberechnung

geht auch im $\mathbb{R}^3$ ↓

$$\text{orientation}(p,q,r) = \text{sign} \begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix} = \begin{cases} -1 & \text{neg. orient} & \text{right-turn} \\ 0 & & \text{colinear} \\ +1 & \text{pos. orient} & \text{left-turn} \end{cases}$$

  - **Maximumssuche, falls $\alpha_r = \alpha_q$:** d.h. orientation $(p,q,r) = 0$
    Naiv: $dist_q = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$, $dist_r = \sqrt{\dots}$
    2 Nachteile: langsam, ungenau bezogen auf Wurzel
    ⇒ **besser: Vergleiche Quadrate der Distanzen:**
    $(p_x - q_x)^2 + (p_y - q_y)^2 \lessgtr (p_x - r_x)^2 + (p_y - r_y)^2$

    Hier nur 4 Subtraktionen, 4 Multiplikationen und 2 Additionen

    → orientation $(a,b,c,d) = \text{sign} \begin{vmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ d_x & d_y & d_z & 1 \end{vmatrix} = \begin{cases} -1 & \text{von d aus ist das } \Delta abc \text{ neg orient} \\ 0 & \text{Pkte in einer Ebene} \\ +1 & \text{von d aus ist das } \Delta abc \text{ pos orient} \end{cases}$