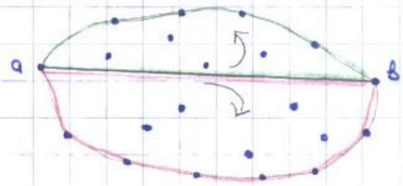
1.3. Algorithmus II: Graham's Scan.

1.3.1. Idee: Berechne "obere" und "untere" Hülle getrennt.

Sei a der linkeste und b der rechteste Pkt von S, dh. a = Minimum bzgl. x-y-Sortierung und b = Maximum bzgl. xy-Sortierung.

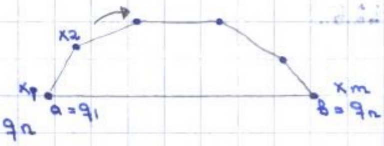
Obere Hülle = Der Teil von $CH(S)$, der oberhalb von $\overline{ab}$ liegt (inclusive a,b). $= CH(S_1)$, $S_1 \subset S$.

Untere Hülle = Der Teil von $CH(S)$, der unterhalb von $\overline{ab}$ liegt (incl. a,b) $= CH(S_2)$, $S_2 \subset S$.



Beobachtung: $S_1 = \{p \in S : \text{orientation}(a,b,p) \geq 0\}$
und $S_2 = \{p \in S : \text{orientation}(a,b,p) \leq 0\}$.

Sei S_1 lexikografisch nach xy-sortiert ($\to n \log n$)
$\leadsto$ sortierte Folge $q_1 \dots q_n = S_1$
$\Rightarrow a = q_1$ und $b = q_n$
Berechne obere Hülle als Teilfolge $x_1 \dots x_m$ von $q_1 \dots q_n$
(dh. im Uhrzeigersinn).



Beobachtung:

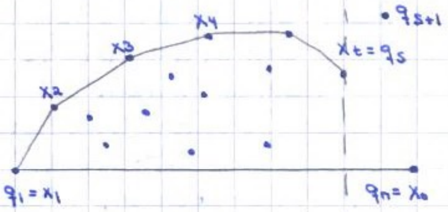
1). Jeweils drei aufeinanderfolgende Ecken x_i, x_{i+1}, x_{i+2} bilden einen Rightturn, dh. $\text{orientation}(x_i, x_{i+1}, x_{i+2}) < 0$.
2). (b,a,p) ist Rightturn $\forall p \in S_1$.
(dh. $\forall p \in S_1$: $\text{orientation}(q_n, q_1, p) < 0$).

1.3.2. Algorithmus:

verwalte stack $x_0, x_1 \dots x_t$ von potenziellen Ecken der oberen Hülle.
Am Ende: genau die Ecken der oberen Hülle.
Initialisierung: stack $S = q_n, q_1, q_2$. Bem: alle drei Pkte wg. lexik. Sort. genau festgelegt und müssen nicht gesucht werden.
Schleife: Betrachte nur $q_3 \dots q_{n-1}$.

Sei q_s aktueller Pkt
Invariante: $x_0, x_1 \dots x_t$ ist Teilfolge von $q_n, q_1 \dots q_s$

mit: 1) $t \geq 2$, $x_0 = q_n$, $x_1 = q_1$, $x_t = q_s$
besser x_1 statt x_0 2) ~~x_0~~ $x_1 \dots x_t$ ist konvexes Polygon
3) obere Hülle von S ist Teilfolge von $x_1 \dots x_t\, q_{s+1} \dots q_n$.



Schritt: $s \mapsto s+1$
while (x_{t-1}, x_t, q_{s+1}) nicht Rightturn do
  pop(x_t)
od
push(q_{s+1}).

Beobachtung: stack speichert ober Hülle der Pkte $q_1 \dots q_s$ (die bis jetzt gewesen).

Die Funktion:

UPPER_HULL($q_1 \dots q_n$)   besser: $q_1 \dots q_m \Rightarrow b = q_m$ (weil $|S| = n$)
(Vorbed. 1). $q_1 \dots q_n$ lexik. nach xy sortiert
2) $q_2 \dots q_{n-1}$ liegen oberhalb $\overline{q_1 q_n}$).
Stack von Pktn. S
(stack <point> S;)
S.push(q_n);
S.push(q_1);
S.push(q_2);
(Ann. $n \geq 2$).

$\rightarrow$

```
for s=3 to n-1 do
    x ← S.top();          gibt das oberste Element an.
    y ← S.top_pred();     eins unter dem obersten auf dem Stack
    while orientation (y, x, q_s) ≥ 0 do
        S.pop();
        x ← y;
        y ← S.top-pred();
    od
    S.push(q_s)
od
return S.
```

While-Schleife terminiert spätestens, wenn S nur noch die Pkte q_n, q_1 enthält. (weil alle Pkte $q_2 .. q_{n-1}$ oberhalb von $\overline{q_1 q_n}$ liegen.).

1.3.3. Beispiel für die Pkt:

Anfang: Stack $q_6 q_1 q_2$.

s = 3

$x = q_2 \wedge y = q_1$

orientation $(q_1 q_2 q_3) = -1$

$\Rightarrow$ Stack $q_6 q_1 q_2 q_3$

s = 4

$x = q_3 \wedge y = q_2$.

orientation $(q_2 q_3, q_4) = -1 \Rightarrow$ Stack $q_6 q_1 q_2 q_3 q_4$

s = 5

$x = q_4 \wedge y = q_3$

orientation $(q_3 q_4 q_5) = +1 \Rightarrow$ stack $q_6 q_1 q_2 q_3$, $x = q_3$, $y = q_2$

orientation $(q_2 q_3 q_5) = -1 \Rightarrow$ stack $q_6 q_1 q_2 q_3 q_5$

Fertig!

1.3.4. Korrektheit.

(1) Invariante ist stets erfüllt.

Zeige mit Induktion, dass Invariante stets erfüllt ist.

Ind.anf: $S = q_n q_1 q_2$

$x_0, x_1, x_2 = q_n q_1 q_2$ mit
1) $t = 2$, $q_n = x_0$, $q_1 = x_1$, $q_2 = x_2$
2) $x_0 x_1 x_2$ konv. Polygon
3). obere Hülle von S ist TF von $x_0 x_1 x_2 q_3 ... q_{n-1}$.

Ind.ann: die Invariante sei für $x_0 x_1 .. x_t$ erfüllt $(t \in \mathbb{N})$.

Indbeh: Sei q_s der nächste Pkt bzgl. der lexikogr. Sortierung.

Wenn orientation $(x_{t-1}, x_t, q_s) = -1$ dann ist die Beh. trivialerweise erfüllt.

Sei also orientation $(x_{t-1}, x_t\, q_s) \in \{1, 0\}$

Z.Z. $x_0 ... x_t$ ist TF von $q_n q_1 .. q_s$ mit
1) $t \geq 2$, $x_0 = q_n$, $x_1 = q_1$, $x_t = q_s$
2) $x_0 .. x_t$ konv. Polygon
3) obere Hülle von S ist TF von $x_1 .. x_t\, q_{s+1} .. q_n$.

$\Rightarrow S = x_0 x_1 .. x_{t-1}$

überall x_1. ... wird so lange gepopt bis orientation$(y, x, q_s) < 0$.

$\Rightarrow S = x_0 x_1 .. x_i$

S.push(q_s) $\Rightarrow S = x_0 x_1 .. x_i x_t$, wobei $x_t := q_s$

$\Rightarrow x_0 x_1 .. x_i x_t$ ist TF von $q_n q_1 ... q_s$

zu 1) $\underbrace{t \geq 2}$, $x_0 = q_n$, $x_1 = q_1$ und $x_t = q_s$ nach Def. $\Rightarrow$ ok.

weil $(x_0 x_1 q_s)$ Rightturn $\Rightarrow i \geq 1 \Rightarrow t \geq 2$.

zu 2). Folge $x_0' .. x_i$ unverändert und (x_{i-1}, x_i, q_s) bilden Rightturn

$\Rightarrow x_0' .. x_i x_t$ auch konvexes Polygon.

zu 3) Pkte $x_{i+1} .. x_{t(alt)}$ wurden entfernt. Sie gehören nicht zu oberen Hülle, da sie rechts von (oder auf) Strecke $\overline{x_i q_s}$ liegen. $\Rightarrow$ ok.

$\Rightarrow$ Beh.

(2). Bei Termination enthält Stack S die obere Hülle.

Voraussetzung: die Invariante ist für den letzten Pkt $(s=n)$ erfüllt.

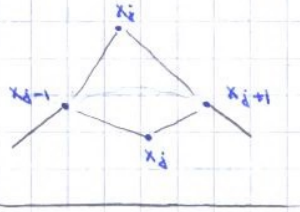
dh. alle Pkte sind abgearbeitet.

Wissen also: $x_1 \dots x_t$ ist TF von $q_1 \dots q_n$ mit bea: läuft ab 1, weil $x_0 = x_t = q_n$.

1) $t \geq 2$, $x_0 = q_n$, $x_1 = q_1$ und $x_t = q_n$

2) $x_1 \dots x_t$ konv. Polygon

3). obere Hülle ist TF von $x_1 \dots x_t$.



Z.Z. obere Hülle $= x_1 \dots x_t$ also nicht echte Teilmenge.

Ann: nein, obere Hülle $\subset x_1 \dots x_t$

$\Rightarrow \exists x_j \in \{x_1 \dots x_t\}$ mit $x_j \notin$ obere Hülle

$\Rightarrow$ 1.Fall: orientation$(x_{j-1}, x_j, x_{j+1}) = +1 \Rightarrow \lightning$ zu 2)

2.Fall: orientation$(x_{j-1}, x_j, x_{j+1}) = -1 \Rightarrow \lightning$ zu 2).

$\Rightarrow$ Ann. falsch

$\Rightarrow$ Beh.

1.3.5. Laufzeit:

Beobachtung: Jeder Pkt kann höchstens einmal aus S entfernt werden.

Rumpf hat Kosten $O(1)$ (orientation, konstante Zahl von stack-Operationen).

Da jeder Pkt höchstens einmal aus S entfernt werden kann $\Rightarrow$

$\Rightarrow$ while-Schleife wird höchstens n mal eingetreten.

In der while-Schleife konst. Operationen, außerhalb auch

$\Rightarrow$ forschleife kostet uns $O(n)$.

$\Rightarrow$ Gesamtkosten der UPPER-Fkt : $O(n)$.

(Analog Berechnet man die untere Hülle).

Die Funktion insgesamt:

CONVEX_HULL(S)		
1. S.sort(xy);	$O(n \log n)$	$(\|S\| = n)$
a ← S.min();	$O(1)$	
b ← S.max();	$O(1)$	
2. forall p ∈ S do eigentlich $S \setminus \{a,b\}$	$O(n)$	(S_1, S_2 Listen $\Rightarrow$ append kostet $O(1)$, $n \times \Rightarrow O(n)$)
3. if (orientation(a,b,p) > 0) S_1.append(p)		
4. if (orientation(a,b,p) < 0) S_2.append(p)		
5. od		
6. S_1.push(a);	$O(n)$	(falls S_1 Liste $\Rightarrow$ einfügen am Ende kostet $O(n)$)
S_1.append(b);	$O(1)$	
7. S_2.push(a);	$O(n)$	
S_2.append(b);	$O(1)$	
8. H_1 ← UPPER-HULL(S_1)	$O(m) = O(n)$	
9. H_2 ← LOWER-HULL(S_2)	$O(n-m) = O(n)$	
10. H_1 umdrehen	$O(n)$	
11. H_2 an H_1 anhängen	$O(1)$	
12 doppeltes Vorkommen von a/b löschen.	$O(n)$	

Zusammenfassung:

CONVEX_HULL(S)	Zeit:
1) Sortiere s lexik.	$O(n \log n)$
2) $a \leftarrow \min \wedge b \leftarrow \max$	$O(1)$
3) $S_1 \leftarrow \{p \in S: b,a,p \text{ Rightturn}\} \cup \{a,b\}$	$O(n)$
4) $S_2 \leftarrow \{p \in S: b,a,p \text{ Leftturn}\} \cup \{a,b\}$	$O(n)$
5) UPPER-HULL(S_1)	$O(n)$
6) LOWER-HULL(S_2)	$O(n)$
7) "Zusammenkleben	$O(n)$

1.3.6. Satz: Sei S Menge von n Pkten im $\mathbb{R}^2$.

a) $CH(S)$ kann in Zeit $O(n \log n)$ berechnet werden (worst case)

b). Falls S lexik. nach xy-Koord. sortiert ist, dann kann $CH(S)$ in Zeit $O(n)$ berechnet worden.

Bew. siehe oben.

1.3.7. Bemerkung:

1) Alg. optimal, da CH-Problem (1.a.) genauso schwierig ist wie sortieren.
   Bew. Übung.

2). obere bzw. untere Hülle sind auch für sich alleine wichtig (für bestimmte Probleme)

3). Optimalität gilt für worst case; es gibt Situationen, in denen Alg. I. besser ist (wenn # Ecken von $CH(S) < \log n$).

1.3.8. Varianten von Graham's Scan:

a) siehe Übung

b). untere u. obere Hülle gleichzeitig berechnen

c). Berechnung der gesamten Hülle in einer Phase.

Idee: 1. Sortiere $S = p_1 .. p_n$.

2. Allgemeiner Schritt:

Bearbeite p_i, $i = 4..n$.

Situation: $C_{i-1} := CH(\{p_1 .. p_{i-1}\})$ ist berechnet.

p_{i-1} ist maximal in $\{p_1 .. p_{i-1}\}$

$\Rightarrow p_{i-1}$ ist die rechteste Ecke von C_{i-1}.

$\Rightarrow \overline{p_{i-1} p_i} \cap C_{i-1} = \{p_{i-1}\}$.

Der eigentliche Schritt:

Berechne Berührpkte t und b der beiden Tangenten von p_i an C_{i-1}.

Genauer:

```
t ← p_{i-1}          d.h. orientation(p_i, t, succ(t)) ≤ 0
while (p_i, t, succ(t)) nicht lefttturn do
    t ← succ(t)
od
b ← p_{i-1}          d.h. orientation(p_i, b, pred(b)) ≥ 0
while (p_i, b, pred(b)) nicht rightturn do
    b ← pred(b)
od
```

Bea: hier ähnliche Argumentation:
die # unterwegs besuchten Pkte kann in $\Omega(n)$ liegen,
aber jeder von ihnen wird anschließend aus der konvexen Hülle entfernt und kann danach nie wieder als Eckpkt in Erscheinung treten
Also wird er auch nie wieder besucht
$\Rightarrow$ Es kann insgesamt höchstens n Besuche geben
$\Rightarrow$ $O(n \log n)$ Laufzeit

Nachdem wir nun b und t berechnet haben:

- Entferne alle Pkte zwischen b und t
- Füge p_i nach C_i ein.

Weitere Implementierungsdetails und Laufzeitanalyse siehe 3. Übung.

Bsp. zu c) siehe Ordner.

1.4. Algorithmus III: Devide & Conquer.

Triangulierung schauen.

1.4.1. Spezialfall: Konvexe Hülle von zwei konvexen Polygonen P und Q.

$P = p_1 .. p_m$, $Q = q_1 .. q_\ell$, Ecken gg. Uhrzeigersinn sortiert.

Aufgabe: Berechne $CH(P \cup Q)$

Triviale Lsg: Graham's Scan auf die Menge aller Ecken. $O(n \log n)$

Bessere Lsg: Ausnutzen der Polygonstruktur. $O(n)$.

1) Finde Sortierung aller Ecken in Zeit $O(n)$
2) Berechne die Hülle in Zeit $O(n)$ (siehe S. 1.3.6. b).)

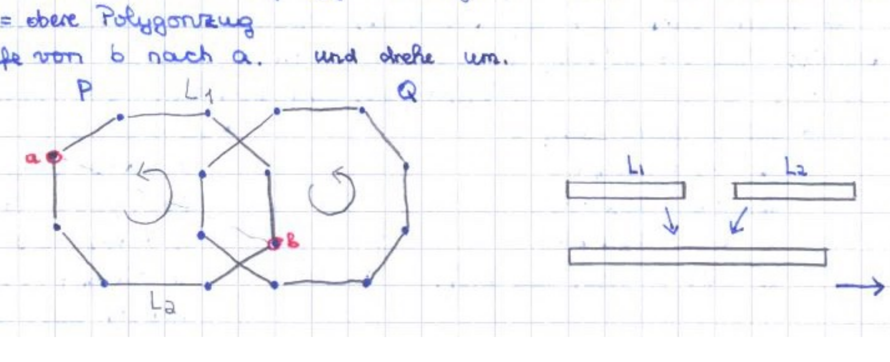
} $O(n)$

zu 1): Betrachte P:

(i) Finde extreme Ecken a und b in Zeit $O(m)$

L_2 := untere Polygonzug
starte bei a und laufe gg. Uhrzeigersinn über P bis b erreicht ist.

L_1 := obere Polygonzug
laufe von b nach a. und drehe um.



(ii) Mische L_1 und L_2 zu einer sortierten Gesamtliste L_P zusammen in Zeit $O(m)$ (→ siehe Mergesort)
Analog: Sortiere Folge L_Q der Ecken von Q in Zeit $O(\ell)$.

(iii) Mische L_P und L_Q in Zeit $O(m+\ell) = O(n)$ (wobei $n := m+\ell$) zu einer sortierten Gesamtfolge zusammen.

→ dann Graham's Scan.

1.4.2. Allgemein: Sei $S \subset \mathbb{R}^2$, $|S| = n$.

```
CONVEX_HULL(S).
if |S| = 1 then
    output S
else
    teile S in zwei möglichst gleich große Teile S1 und S2   } DEVIDE
    (zB. |S1| = ⌈|S|/2⌉ und |S2| = ⌊|S|/2⌋
    P ← CONVEX_HULL(S1)                                      } CONQUER
    Q ← CONVEX_HULL(S2)
    berechne CH(P ∪ Q) wie in letzter Vorlesung gezeigt.     } MISCHSCHRITT
fi.
```

↑ hier wird die eigentliche Rechnung durchgeführt.

1.4.3. Laufzeit:

Teilen: $O(n)$
Mischen: $O(n)$

Merge auf Listen
Graham's Scan (ohne Sortieren).

$$\Rightarrow T(n) = \begin{cases} c_0, & n = 1 \\ c_1 \cdot n + 2 \cdot T(n/2), & n > 1 \end{cases}$$

Das ist also dieselbe Rekursion wie für Mergesort (aber mit anderen Konstanten natürlich).

⇒ Gesamtlaufzeit des Verfahrens ist $O(n \log n)$ (siehe Analyse von Mergesort).

1.5. Eine Anwendung von CONVEX HULL

1.5.1 Problem: Geg. sind n Halbebenen $H_1 \dots H_n$ von $\mathbb{R}^2$

Berechne $P = \bigcap_{i=1}^{n} H_i$

1.5.2. Def: Eine abgeschlossene Halbebene = { alle Pkte auf gleicher Seite einer Geraden ℓ } ∪ ℓ.

1.5.3 Anm: Schnitt P ist konvexes (möglicherweise unbeschränktes) Polygon, da der Schnitt konvexer Mengen wieder konvex ist.

1.5.4. Ziel und Lösungsansatz:

Ziel: Berechne die Folge der Ecken von P gg. den Uhrzeigersinn.

Lösung: Zurückführung auf konvexe Hülle einer geeigneten Pktmenge.
Dazu transformieren wir die definierenden Geraden in "duale" Pkte.

1.5.5. Definition: (Geometr. Transformation)

a). Sei $\ell = \{y : \underbrace{y = ax+b}_{\text{Geradengleichung von } \ell}, x \text{ bel.}, a,b \text{ fest}\}$ eine nicht vertikale Gerade.

Der Punkt $D(\ell) := (a,b)$ heißt der duale Punkt zu ℓ.

b). Sei $p = (a,b) \in \mathbb{R}^2$ ein Pkt. Die Gerade $D(P) = \{y : y = -ax + b, x \text{ Beliebig}\}$ heißt die duale Gerade zu p.

Abbildung D erlaubt die relative Lage von Objekten.

1.5.5 Lemma: Pkt p liegt auf (oberhalb / unterhalb) einer Geraden ℓ
⇔ Gerade $D(P)$ liegt auf (oberhalb / unterhalb) Pkt $D(\ell)$.

Beweis: Sei $p = (p_x, p_y)$, $\ell = \{y \in \mathbb{R} : y = ax+b, x \in \mathbb{R}\}$ (a,b fest).
$\Rightarrow D(\ell) = (a,b)$ und $D(P) = \{y : y = -p_x x + p_y, x \in \mathbb{R}\}$

- sei p gelegen auf ℓ
  $\Leftrightarrow p_y = a \cdot p_x + b$
  $\Leftrightarrow -b = ap_x - p_y$
  $\Leftrightarrow b = -p_x a + p_y$
  $\Leftrightarrow D(P)$ liegt auf $D(\ell)$.

- sei p gelegen oberhalb ℓ (unterhalb analog).
  $\Leftrightarrow ax + b < p_y \ \forall x \in \mathbb{R}$
  Bzw. sogar $ap_x + b < p_y$.
  $\Leftrightarrow -ap_x - b > -p_y$
  $\Leftrightarrow -ap_x + p_y > b$
  $\Leftrightarrow D(P)$ liegt oberhalb $D(\ell)$.

1.3.6. Satz: Sei S Menge von n Pkten im $\mathbb{R}^2$.

a). CH(S) kann in Zeit $O(n \log n)$ berechnet werden (worst case)

b). Falls S lexik. nach xy-Koord. sortiert ist, dann kann CH(S) in Zeit $O(n)$ berechnet werden.

Bew. siehe oben.

1.3.7. Bemerkung:

1) Alg. optimal, da CH-Problem (1.a.) genauso schwierig ist wie sortieren.
Bew. Übung.

2). obere bzw. untere Hülle sind auch für sich alleine wichtig (für bestimmte Probleme)

3). Optimalität gilt für worst case; es gibt Situationen, in denen Alg. I. besser ist (wenn # Ecken von CH(S) $< \log n$).

1.3.8. Varianten von Graham's Scan:

a) siehe Übung

b). untere u. obere Hülle gleichzeitig berechnen

c). Berechnung der gesamten Hülle in einer Phase.

Idee: 1. Sortiere $S = p_1 .. p_n$.

2. Allgemeiner Schritt:

Bearbeite p_i, $i = 4..n$.

Situation: $C_{i-1} := CH(\{p_1 .. p_{i-1}\})$ ist berechnet.

p_{i-1} ist maximal in $\{p_1 .. p_{i-1}\}$

$\Rightarrow p_{i-1}$ ist die rechteste Ecke von C_{i-1}.

$\Rightarrow \overline{p_{i-1} p_i} \cap C_{i-1} = \{p_{i-1}\}$.

Der eigentliche Schritt:

Berechne Berührpkte t und b der beiden Tangenten von p_i an C_{i-1}.

Genauer: $t \leftarrow p_{i-1}$ d.h. orientation$(p_i, t, succ(t)) \leq 0$

while $(p_i, t, succ(t))$ nicht lefttum do
$t \leftarrow succ(t)$
od

$b \leftarrow p_{i-1}$ d.h. orientation$(p_i, b, pred(b)) \geq 0$

while $(p_i, b, pred(b))$ nicht righttum do
$b \leftarrow pred(b)$
od

Bea.: hier ähnliche Argumentation:
die # unterwegs besuchten Pkte kann in $\Omega(n)$ liegen,
aber jeder von ihnen wird anschließend aus der konvexen Hülle entfernt und kann danach nie wieder als Eckpkt in Erscheinung treten
Also wird er auch nie wieder besucht
$\Rightarrow$ Es kann insgesamt höchstens n Besuche geben
$\Rightarrow O(n \log n)$ Laufzeit

Nachdem wir nun b und t berechnet haben:

- Entferne alle Pkte zwischen b und t
- Füge p_i nach C_i ein.

Weitere Implementierungsdetails und Laufzeitanalyse siehe 3. Übung.

Bsp. zu c) siehe Kursordner.

1.4. Algorithmus III: Devide & Conquer.

Triangulierung schauen.

1.4.1. Spezialfall: Konvexe Hülle von zwei konvexen Polygonen P und Q.

$P = p_1 .. p_m$, $Q = q_1 .. q_l$, Ecken gg. Uhrzeigersinn sortiert.

Aufgabe: Berechne $CH(P \cup Q)$

Triviale Lsg: Graham's Scan auf die Menge aller Ecken. $O(n \log n)$

Bessere Lsg: Ausnutzen der Polygonstruktur. $O(n)$.

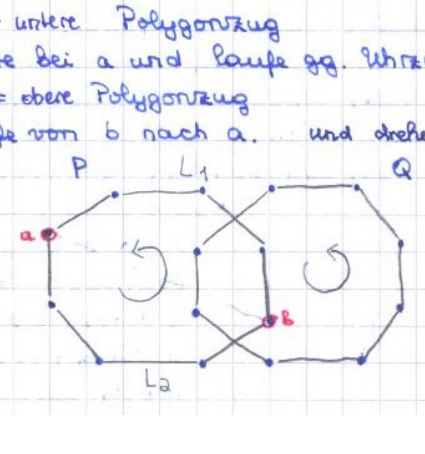
1) Finde Sortierung aller Ecken in Zeit $O(n)$
2) Berechne die Hülle in Zeit $O(n)$ (siehe S. 1.3.6. b).) $\Big\} O(n)$

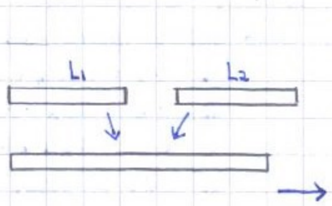
zu 1): Betrachte P:

(i) Finde extreme Ecken a und b in Zeit $O(m)$

L_2 := untere Polygonzug
starte bei a und laufe gg. Uhrzeigersinn über P bis b erreicht ist.

L_1 := obere Polygonzug
laufe von b nach a. und drehe um.





(ii) Mische L_1 und L_2 zu einer sortierten Gesamtliste L_P zusammen in Zeit $O(m)$ (→ siehe Mergesort)
Analog: Sortiere Folge L_Q der Ecken von Q in Zeit $O(l)$.
(iii) Mische L_P und L_Q in Zeit $O(m+l) = O(n)$ (wobei $n := m+l$) zu einer sortierten Gesamtfolge zusammen.
→ dann Graham's Scan.

1.4.2. Allgemein: Sei $S \subset \mathbb{R}^2$, $|S| = n$.

```
CONVEX_HULL(S).
if |S| = 1 then
    output S
else
    teile S in zwei möglichst gleich große Teile S1 und S2   } DEVIDE
    (zB. |S1| = ⌈|S|/2⌉ und |S2| = ⌊|S|/2⌋
    P ← CONVEX_HULL(S1)                                       } CONQUER
    Q ← CONVEX_HULL(S2)
    berechne CH(P ∪ Q) wie in letzter Vorlesung gezeigt.      } MISCHSCHRITT
fi.
```

↑ hier wird die eigentliche Rechnung durchgeführt.

1.4.3. Laufzeit:
Teilen: $O(n)$
Mischen: $O(n)$ }
Merge auf Listen
Graham's Scan (ohne Sortieren).

$$\Rightarrow T(n) = \begin{cases} C_0, & n=1 \\ C_1 \cdot n + 2 \cdot T(n/2), & n>1 \end{cases}$$

Das ist also dieselbe Rekursion wie für Mergesort (aber mit anderen Konstanten natürlich).
⇒ Gesamtlaufzeit des Verfahrens ist $O(n \log n)$ (siehe Analyse von Mergesort).

1.5. Eine Anwendung von CONVEX HULL.

1.5.1 Problem: Geg. sind n Halbebenen $H_1 \dots H_n$ von $\mathbb{R}^2$
Berechne $P = \bigcap_{i=1}^{n} H_i$

1.5.2. Def: Eine abgeschlossene Halbebene = { alle Pkte auf gleicher Seite einer Geraden l } ∪ l.

1.5.3 Anm: Schnitt P ist konvexes (möglicherweise unbeschränktes) Polygon, da der Schnitt konvexer Mengen wieder konvex ist.

1.5.4. Ziel und Lösungsansatz:
Ziel: Berechne die Folge der Ecken von P gg. den Uhrzeigersinn.
Lösung: Zurückführung auf konvexe Hülle einer geeigneten Pktmenge.
Dazu transformieren wir die definierenden Geraden in "duale" Pkte.

1.5.5. Definition: (Geometr. Transformation)
a). Sei $l = \{y : \underbrace{y = ax+b}_{\text{Geradengleichung von } l}, x \text{ bel.}, a,b \text{ fest}\}$ eine nicht vertikale Gerade.
Der Punkt $D(l) := (a,b)$ heißt der duale Punkt zu l.
b). Sei $p = (a,b) \in \mathbb{R}^2$ ein Pkt. Die Gerade $D(p) = \{y : y = -ax + b, x \text{ Beliebig}\}$ heißt die duale Gerade zu p.
Abbildung D erlaubt die relative Lage von Objekten.

1.5.5 Lemma: Pkt p liegt auf (oberhalb / unterhalb) einer Geraden l
⇔ Gerade $D(p)$ liegt auf (oberhalb / unterhalb) Pkt $D(l)$.

Beweis: Sei $p = (p_x, p_y)$, $l = \{y \in \mathbb{R} : y = ax+b, x \in \mathbb{R}\}$ (a,b fest).
$\Rightarrow D(l) = (a,b)$ und $D(p) = \{y : y = -p_x x + p_y, x \in \mathbb{R}\}$

- sei p gelegen auf l
$\Leftrightarrow p_y = a \cdot p_x + b$
$\Leftrightarrow -b = a p_x - p_y$
$\Leftrightarrow b = -p_x a + p_y$
$\Leftrightarrow D(p)$ liegt auf $D(l)$.

- sei p gelegen oberhalb l (unterhalb analog).
$\Leftrightarrow ax + b < p_y \ \forall x \in \mathbb{R}$
Bzw. sogar $a p_x + b < p_y$.
$\Leftrightarrow -a p_x - b > -p_y$
$\Leftrightarrow -a p_x + p_y > b$
$\Leftrightarrow D(p)$ liegt oberhalb $D(l)$.