Zusammenfassung von Algorithmische Geometrie.

Kapitel 0: Vorbemerkungen:

Inhalt der Vorlesung: Algorithmen und Datenstrukturen zur Lösung geometrischer Probleme.

Beispiele: Robotik / Bewegungsplanung, Computergrafik, CAD, VLSI

Typische Probleme:
- konvexe Hülle
- Zerlegung in einfache konvexe Teile (zB. Triangulierung).
- Schnittprobleme (zB. Segmente, Polygone)
- Suchprobleme (Memberanfrage in Pktmenge, Nearest-Neighbor, Range-Abfragen).
- Bewegungsplanung (Roboter)
- Hidden Line / Surface (Elimination).

Grundlegende Ansätze Bzw. Vorgehensweisen:
- Devide & Conquer
- Plane Sweep.
- Hierarchische Darstellung
- Dualität
- Randomisierte Algorithmen.
- Rundungsfehler u. degenerierte Eingaben.

Kapitel I: Konvexe Hüllen.

1.1 Konstruktion der konvexen Hülle einer Pktmenge im $\mathbb{R}^2$ – Grundlagen.

1.1.1. Def: Sei $S \subset \mathbb{R}^2$ Pktmenge.

S heißt konvex: $\Leftrightarrow \forall\, p, q \in S$ gilt: $\overline{pq} \subset S$.

konvexe Hülle einer Menge $S \subset \mathbb{R}^2$ ($CH(S)$) ist die kleinste konvexe Menge, die S enthält.

Wir betrachten nun endliche Pktmengen, dh. $|S| = n \in \mathbb{N}$.

Anm: Der Rand von $CH(S)$ ist ein konvexes Polygon mit Ecken aus S.
Bew. Übung.

Konvexe Hülle-Problem für endl. Menge $S \subset \mathbb{R}^2$:

Berechne die Folge der Ecken von $CH(S)$ gg. den Uhrzeigersinn (pos. orientiert).
$\rightarrow q_1, \dots q_\ell \in S$. (Anordnung definiert Kanten der Fläche).

Degenerierte Fälle: $\ell = 2$ (alle Pkte sind colinear)
$\ell = 1$ (alle Pkte in S sind gleich).

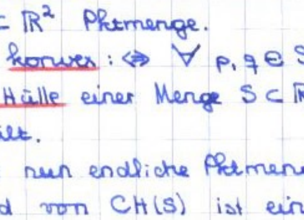
Problem im $\mathbb{R}^3$ analog, $CH(S)$ ist konvexes Polyeder mit Ecken aus S.

1.1.2. Satz: Berechnung der konvexen Hülle von n Pktn. im $\mathbb{R}^2$ ist mind. so schwer wie das Sortieren von n Zahlen.

Bew. siehe Üb. (Idee: Führe Sortieren von $x_1 .. x_n$ zurück auf $CH(\{p_1 .. p_n\})$.).

1.1.3. Korollar: Im Allgemeinen braucht die Berechnung von $CH(S)$ Zeit $\Omega(n \log n)$.

1.1.4. Bsp:



$S = \{1..12\}$, $CH(S) = 8, 7, 3, 1, 4, 9$
(Jeder zyklische Shift)

1.1.5. Notation: Lexikografische Ordnung (siehe Strings) für Pkte im $\mathbb{R}^2$:

Lexikogr. Sortierung nach Kartesischen (x,y)-Koordinaten, dh. für zwei Pkte in der Ebene, also $p, q \in \mathbb{R}^2$ gilt: $p < q :\Leftrightarrow p_x < q_x \vee (p_x = q_x \wedge p_y < q_y)$

DR. Sortierung von links nach rechts Bzw. von unten nach oben (Bei gleicher x-Koordinate).

1.2. Algorithmus I: Gift Wrapping.

Intuitiv: Pktmenge wird mit Geschenkpapier eingewickelt.

Zunächst: $S \subset \mathbb{R}^2$. Voraussetzung: $|S| < \infty$

Ges: $CH(S) = p_1 .. p_h$.

1.2.1. Idee: 1). Startpkt p_1 : Pkt mit der kleinsten y-Koordinate.

(Falls mehrere wähle den linkesten. Dh. Minimum in p_n - lexikogr. Ordnung.

(Übung : p_1 ist Ecke der konvexen Hülle).

2) Wie findet man p_2? (Nachfolger von p_1 auf konv. Hülle).

Betrachte horizontalen Strahl ℓ durch p_1 (nach rechts). Drehen den Strahl gg. den Uhrzeigersinn bis wir einen weiteren Pkt von S treffen. (Ann. $|S| > 1$).

So erhalten wir p_2.

Genauer: $\forall q \in S \setminus \{p_1\}$ sei α_q der Winkel zwischen ℓ und $\overline{p_1 q}$.

Wähle p_2 so, dass α_{p_2} minimal. Falls mehrere Pkte mit minimalen Winkel, wähle den Pkt mit maximaler Entfernung zu p_1.

⇝ lineare Suche in S nach Minimum bzgl. oben definierter Ordnung.

3). Setze Algor. bei p_2 fort, dh. ℓ wird nun der Strahl, der

- in p_2 beginnt.
- in Richtung $p_1 p_2$ zeigt.

⇝ Suche Minimum in $S \setminus \{p_1, p_2\}$.

4) Wiederhole bis p_1 wieder erreicht wird.

1.2.2. Korrektheit : Übung.

1.2.3. Laufzeit: $CH(S) = p_1 .. p_h$, $|S| = n$.

p_1: lineare Suche in S kostet $O(n)$ (lin. Suche nach Min. bzgl. p_y)

$p_2 .. p_h$: lin. Suche in S kostet $O(n)$ (lin. Suche nach Winkelordnung).

⇒ Gesamtlaufzeit: $O(h \cdot n)$.

1.2.4. Satz: Sei S eine Menge von n Pkten im $\mathbb{R}^2$ und h die Zahl der Ecken der konvexen Hülle CH(S). Dann kann CH(S) in Zeit $O(h \cdot n)$ berechnet werden. Bew. siehe oben.

1.2.5. Bem: $h \in \{1 .. n\}$

Worst Case : $h = n$ (zB. alle Pkte aus S liegen auf einem gemeinsamen Kreis, das Innere ist leer). ⇒ Laufzeit: $O(n^2)$.

Bester Fall: $h = \text{const.}$ ⇒ Laufzeit : $O(h \cdot n) = O(n)$.

1.2.6. Details der Implementierung:

1). Wie findet man Pkt q so, dass α_q minimal ist?

⇝ Wir müssen Winkel vergleichen.

```
q ← undefiniert
αq ← ∞
forall p ∈ S \ {p1} do
    if αp < αq then
        q ← p
        αq ← αp
    fi
od
```

Im Allg. 3 Pkte p, q, r → vergleiche Winkel

Naiv: Berechne Winkel aus Koordinaten der Pkte mit trigonometrischen Fktnen.

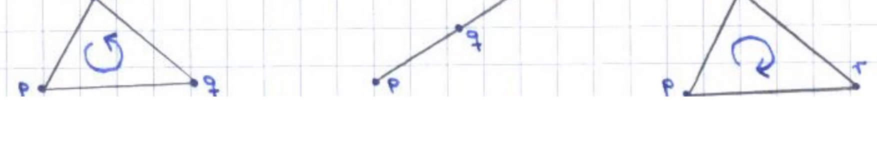
⇒ Nachteile:
- langsam
- Präzisionsprobleme
- Robustheitsproblem (Definitionsbereich der trigonometrischen Fktnen ist eingeschränkt).

⇒ Besser: Andere Betrachtungsweise:

Dazu betrachte Dreieck p, q, r

Drei mögliche Orientierungen:

a) positiv orientiert	b) Orientierung 0	c) negativ orientiert.
(left-turn)	(collinear)	(right-turn)
⇒ $\alpha_q < \alpha_r$	⇒ $\alpha_q = \alpha_r$	⇒ $\alpha_q > \alpha_r$



Allgemein: Seien $p=(p_x,p_y)$, $q=(q_x,q_y)$ und $r=(r_x,r_y)$ drei Pkte im $\mathbb{R}^2$.
Sei $\vec{p}$ der Vektor, der vom Nullpkt ausgeht und in Richtung p zeigt. Und seien $\vec{q}$ und $\vec{r}$ Vektoren, die von p ausgehen und in Richtung q bzw. r zeigen.
Betrachte nun das Parallelogramm, welches durch die Vektoren $\vec{q}$ und $\vec{r}$ aufgespannt wird. Sei $V(P)$ = die Fläche des Parallelogramms.

Beh: $V(P) = |\det A|$, wobei $A = \begin{pmatrix} q_x-p_x & q_y-p_y \\ r_x-p_x & r_y-p_y \end{pmatrix}$

Bem: diese Beh. überträgt sich auch auf $\mathbb{R}^n$.
Man betrachte hierbei $u_1 .. u_n \in \mathbb{R}^n$ und $P=\{u_1a_1+..+u_na_n : 0\leq a_i \leq 1, i\in\{1..n\}\}$.
$\Rightarrow V(P) = |\det A|$, A entsprechend.

Wir betrachten Δpqr:

⟼ **1.2.6.1 Satz**: $p,q,r\in\mathbb{R}^2$, $A = \begin{pmatrix} q_x-p_x & q_y-p_y \\ r_x-p_x & r_y-p_y \end{pmatrix}$.

$\Rightarrow$ (i) $|\det A| = 2\cdot$ Fläche von Δpqr.
(ii) $\operatorname{sign}(\det A)$ gibt die Orientierung an:
- -1: neg. orientiert (right turn)
- 0: collinear.
- $+1$: pos. orientiert (left turn).

Bew. siehe Fernordner.

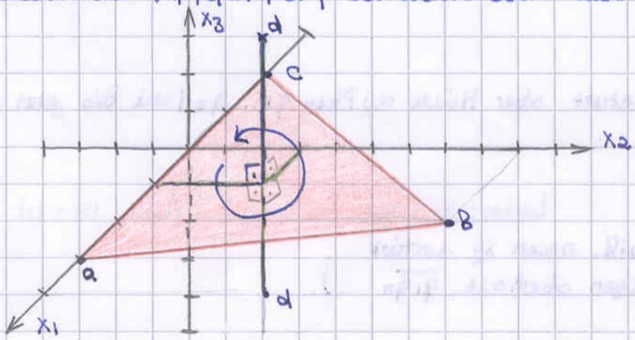
⟼ **1.2.6.2 Def**: Wir definieren das geometrische Prädikat:

$$\text{orientation}(p,q,r) = \operatorname{sign}\begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix} \quad \left(= \operatorname{sign}\left((q_x-p_x)(r_y-p_y)-(r_x-p_x)(q_y-p_y)\right)\right).$$

Dh. orientation(p,q,r) und damit der Test $\alpha_r < \alpha_q$ kann mit zwei Multiplikationen und fünf Subtraktionen berechnet werden.

2) Falls $\alpha_r = \alpha_q$ (dh. orientation(p,q,r) = 0), müssen wir herausfinden, welcher der Pkte q oder r weiter von p entfernt liegt.
Naiv: Berechne Distanzen (mit eukl. Metrik):
$\text{dist}(p,q) = \sqrt{(p_x-q_x)^2+(p_y-q_y)^2}$
$\rightsquigarrow$ Nachteil: Wurzel!
$\Rightarrow$ Besser: Vergleiche Quadrate der Distanzen:
$$(p_x-q_x)^2+(p_y-q_y)^2 \overset{?}{\underset{?}{\lessgtr}} (p_x-r_x)^2+(p_y-r_y)^2 \;?$$
($\rightsquigarrow$ 4 Subtraktionen, 4 Multiplikationen, 2 Additionen).

1.2.7. Übertragung auf höhere Dimensionen:
Das Orientierungsprädikat kann auf höhere Dimensionen verallgemeinert werden.
Insbesondere im $\mathbb{R}^3$: seien $a,b,c,d\in\mathbb{R}^3$
orientation$(a,b,c,d)\in\{-1,0,1\}$
Sei orientation$(a,b,c,d)\neq 0$, betrachte Ebene durch a,b,c.



⟼ **1.2.7.1 Satz**: Für $a,b,c,d\in\mathbb{R}^3$ betrachte Ebene durch a,b,c.
Falls orientation $\neq 0$, so gilt:
(i) orientation$(a,b,c,d) = -1$, wenn von d aus $\Delta a,b,c$ negativ orientiert ist.
orientation$(a,b,c,d) = +1$, wenn von d aus $\Delta a,b,c$ positiv orientiert ist.
(ii) $$\text{orientation}(a,b,c,d) = \operatorname{sign}\begin{vmatrix} a_{x_1} & a_{x_2} & a_{x_3} & 1 \\ b_{x_1} & b_{x_2} & b_{x_3} & 1 \\ c_{x_1} & c_{x_2} & c_{x_3} & 1 \\ d_{x_1} & d_{x_2} & d_{x_3} & 1 \end{vmatrix}$$

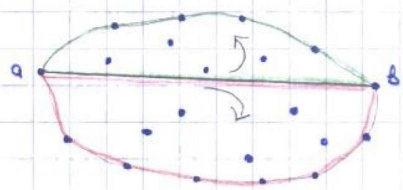
1.3. Algorithmus II: Graham's Scan.

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

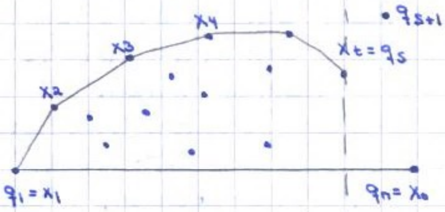
Sei S_1 lexikografisch nach xy-sortiert ($\to n\log n$)
$\leadsto$ sortierte Folge $q_1 \dots q_n = S_1$
$\Rightarrow a = q_1$ und $b = q_n$
Berechne obere Hülle als Teilfolge $x_1 .. x_m$ von $q_1 .. q_n$
(dh. im Uhrzeigersinn).

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

1.3.2. Algorithmus:

verwalte stack $x_0, x_1 .. x_t$ 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 $q_3 .. q_{n-1}$.
Sei q_s aktueller Pkt
Invariante: $x_0, x_1 .. x_t$ ist Teilfolge von $q_n, q_1 .. q_s$

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



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 $q_1 .. q_s$ (die bis jetzt gewesen).

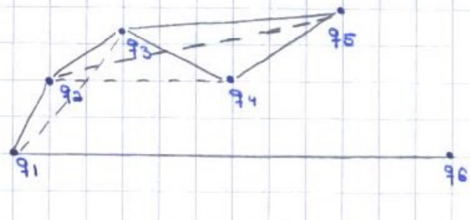
Die Funktion:
UPPER_HULL$(q_1 .. q_n)$ besser: $q_1 \dots q_m \Rightarrow b = q_m$ (weil $|S| = n$)
(Vorbed. 1). $q_1 .. q_n$ lexik. nach xy sortiert
2) $q_2 .. q_{n-1}$ liegen oberhalb $\overline{q_1 q_n}$).
Stack von Pktn. S
(stack<point> S;)
S.push(q_n);
S.push(q_1);
S.push(q_2);
(Ann. $n \geq 2$).

$\rightarrow$

```
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.

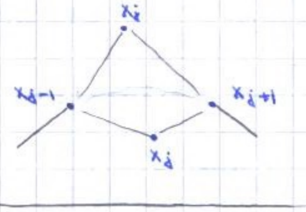
$\Rightarrow$ Beh.

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

Voraussetzung: die Invariante ist für den letzten Pkt ($s=n$) erfüllt.
d.h. alle Pkte sind abgearbeitet.

Wissen also: $x_1 \ldots x_t$ ist TF von $q_1 \ldots q_n$ 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) $x_1 \ldots x_t$ konv. Polygon
3). obere Hülle ist TF von $x_1 \ldots x_t$.



z.z. obere Hülle $= x_1 \ldots x_t$ also nicht echte Teilmenge.

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

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

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

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

$\Bigg\}\Rightarrow$ 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);	$O(n \log n)$	($\lvert S\rvert = n$)
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 ⇒ 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 ⇒ 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 ← min ∧ b ← max	$O(1)$
3) $S_1 \leftarrow \{p \in S:$ b,a,p Rightturn $\} \cup \{a,b\}$	$O(n)$
4) $S_2 \leftarrow \{p \in S:$ b,a,p Leftturn $\} \cup \{a,b\}$	$O(n)$
5) UPPER-HULL(S_1)	$O(n)$
6) LOWER-HULL(S_2)	$O(n)$
7) "Zusammenkleben"	$O(n)$