5.1 a) Idee: Zuerst muss überprüft werden, ob das Array A
6/6 auf- oder absteigend sortiert ist. ← Haha, starke!
Danach wird überprüft, ob für die Mitte des Arrays $A[i]=i$ gilt. Falls ja, wird "true" zurück geben, ansonsten wird überprüft, ob $A[i]>i$ ist. Falls ja, wird der Algorithmus auf der linken Hälfte ausgeführt. Falls nein, auf der rechten Hälfte. Sollte somit kein i für das $A[i]=i$ gilt gefunden werden, wird "false" zurück gegeben.

Pseudocode:

```
boolean fun (Array A){
    int l, r, m;
    l=0;   r= A.length();
if(A[0]<A[1]){while (l≤r){
            m= l+((r-l)/2);
            if(A[m]==m){
                return true;
            } else {
                if(A[m]>m) r=m-1;
                else  l= m+1;
            }
    }}else{
        while(l<=r){
            m = l+((r-l)/2);
            if (A[m]=m) return true;
            else {
                if (A[m]<m) r=m-1;
                else l= m+1;
    }}} return false;}
```

Die Laufzeit ist durch $O(\log(n))$ beschränkt, da sich das zu durchsuchende Array immer halbiert

5.2a) Es sei $L = 35$ und $v_1 = 1$, $v_2 = 7$, $v_3 = 8$

Der Greedy-Ansatz würde das Laub wie folgt aufteilen:

Rest:	35	27	19	11	3	2	1	0
Schubkarre:	v_3	v_3	v_3	v_3	v_3	v_1	v_1	$v_1 = 7$ Schubkarren

Eine optimale Lösung sähe so aus:

Rest	35	28	21	14	7	0 ✓
Schubkarre	v_2	v_2	v_2	v_2	v_2	$v_2 = 5$ Schubkarren

b) Basisfälle: $S_i(0) = 0$

$S_1(\ell) = \ell \Leftarrow S_0(\ell) = \infty$

Rekursionsgleichung: $S_i(\ell) = \text{if } ((\ell - v_i) \geq 0)$

$\min\{S_i(\ell - v_i) + 1, S_{i-1}(\ell)\}$

esle

$S_{i-1}(\ell)$

c) Initialisiere k-viele L-lange Arrays $S_{1\ldots k}$ (k ist die Anzahl der Schubkarren, L die Menge des Laubes)

Setze alle Einträge der Arrays $S_{1\ldots k}$ an der Stelle 0 auf 0.

Setze alle Einträge $S_1(i)$ auf i, für $i = 1 \ldots L$ // Basisfälle

Für $i = 1 \ldots k$ // laufe durch S_i

Für $\ell = 1 \ldots L$ // laufe durch $S[\ell]$

Falls $((\ell - v_i) \geq 0)$

setze $S_i[\ell]$ auf $\min\{S_i[\ell - v_i] + 1, S_{i-1}[\ell]\}$

sonst

schon! setze $S_i[\ell]$ auf $S_{i-1}[\ell]$ // Verfahre nach Rekursionsformel von oben

d) Man benötige i-viele Zähler (für jede Schubkarrengröße einen). Sobald in der Rekursion $S_i[\ell - v_i]$ zu einem kleineren Wert als $S_{i-1}[\ell]$ evaluiert, wird der Zähler für die ...

zu 5.2d)

... v_i-te Schubkarre um 1 erhöht, sollte diese [illegible] zur Lösung führen

Andernfalls bleiben die Zähler unangetastet ✓

e) Der Speicherbedarf sei durch die [illegible] k vielen, d langen Arrays festgelegt. Dieser sei asymptotisch begrenzt durch $O(k \cdot d)$ ✓

5.3 a) Man fragt sich in jedem Teilproblem, ob es eine Zahl $\Sigma - n$ gibt, die sich als Summe $\sum_{i \in S'} x_i = \Sigma - n$ mit $S' = \{1 \dots n-1\}$ darstellen lässt.

b) Basisfälle: $S[0][\text{[illegible]}] =$ True

$S[n][i]$ = True, falls $n == x_i$

$S[n][1]$ = False, falls $n > 1$

Rekursionsgleichung: $S[n][i]$ = True, falls $S[n][i-1] ==$ True

oder $S[n-x_i][i-1] ==$ True

[illegible] und $n - x_i > 0$

False, sonst

nochmal in schön und übersichtlich:

$S[n][i]$ = True, falls $S[n][i-1] ==$ True $\vee$ ($S[n-x_i][i-1] ==$ True

$\wedge\ n - x_i > 0$)

False, sonst

8/8 c) Algo $5c()\{$

Initialisiere Matrix S mit [illegible] Σ-Spalten und n Zeilen.

für $i = 0 \dots \Sigma$

[illegible] setze $S[0][i]$ = True

für $j = 1 \dots n$

setze $[j][i]$ = True, falls $j == x_i$

setze $[j][i]$ = False, falls $i = 1$ // Verfahre nach Fall

für i = 0..S

für j = 1..n

setze S[i][j] auf True, falls S[i][j-1] == True

setze S[i][j] auf True, falls S[i-x_j][j-1] == True
und i - x_j > 0

setze S[i][j] auf False, sollten die beiden oberen
Bedingungen nicht erfüllt sein.

Laufzeit? :)