Übung 10 Lösung

Aufgabe 2

(a) Die Funktion uberpüft, ob das Produkt zweier Vektorelemente gleich der Summe aller Elemente ist:

Die erste *for-Schleife* addiert alle Elemente des Vektors, daher hat dieser Abschnitt eine Laufzeit von **n**. Als nächstes wird erneut der Vektor durchlaufen, und dies zwei Mal (ersichtlich an den Iteratorvariablen i und j. Innerhalb dieser 2 Schleifen wird nun geschaut, ob das Produkt der zwei Einträge *v[i]* und *v[j]* gleich der in der vorherigen Schleife errrechneten Summe ist. Wurden diese 2 Einträge gefunden, so brechen die 2 Schleifen ab und die Funktion gibt *true* zurück, andernfalls haben die zwei Schleifen eine Laufzeit von $\mathbf{n^2}$.

Dieser Algorithmus terminiert also spätestens nach n^2+n Schleifendurchäufen und hat damit eine Laufzeit von $n^2 + n \in O(n^2)$, jedoch ist der Algorithmus nicht korrekt, da in den beiden Schleifen nicht geprüft wird, ob *v[i] == v[j]* ist, sprich, ob ein Vektorelement mit sich selbst multipliziert wird.

(b) Die Funktion addiert die Elemente an geraden (even) Index-Positionen in sum_even_odd[0] und die Elemente an ungeraden (odd) Index-Positionen in sum_even_odd[1]:

Zu Beginn wird ein Vektor mit 2 Einträgen, welche mit 0 initialisiert werden, erstellt. Für die Bestimmung der Laufzeit ist dieser Befehl uninteressant. Als nächstes folgen 2 *for-Schleifen*, wobei der Iterator **i** der äußeren Schleife die Werte 0 und 1 annimmt. In der inneren Schleife wird der Iterator **j** immer um 2 erhöht, was zur Folge hat, dass im 1. äußeren Schleifendurchlauf zunächst alle geraden Einträge durch v[i + j] (also v[0 + j]) addiert werden (i = 0) und im 2. Durchlauf (i = 1) alle ungeraden Einträge (v[1 + j]) miteinander addiert werden.

Der Algorithmus funktioniert gemäß seiner Beschreibung und hat eine Laufzeit (terminiert spätestens in) von $2 \cdot \frac{n}{2} \in O(n)$.

(c) Die Funktion hangt für jedes Element im Vektor die Zahl mit umgedrehten Vorzeichen an:

Interessant an diesem Algorithmus ist, dass die Abbruchbedingung das Ende des Vektors ist, diese aber nicht vorher in einer seperaten Variable festgehalten wird, sondern sich nach jedem Schleifendurchlauf neu berechnet, was zur Folge hat, dass dieser Algorithmus nicht terminieren kann, da mit jedem Schleifenduchlauf der Vektor größer wird und das Ende nie erreicht werden kann.

(d) Die Funktion berechnet die Summe der Elemente an den Index-Positionen, die eine Zweierpotenz sind: $2^0, 2^1, 2^2, 2^3, ...$ (sofern dies ein gultiger Index, also kleiner als die Länge des Vektors ist)

Weil die *for-Schleife* lediglich Index-Positionen abruft, welche eine Zweierpotenz ist, terminiert der Algorithmus spätestens nach $0(\log_2(n))$ Schritten. Allerdings ist der Algorithmus fehlerhaft, da die Einträge von Vektoren mit 0 (v[0]) und nicht 1 (v[1]) beginnen. Zudem müsste i mit i = 1 initalisiert werden, um den 2^0.ten Eintrag zu addieren; es müsste also *sum += v[i - 1]* in der 8. Zeile stehen.

(e) Die Funktion bestimmt die Menge aller Teilmengen der n Elemente im Vektor v (Beispiel: v={1,2,3} ⇒ { {},{1},{2},{3},{1,2},{1,3},{2,3},{1,2,3} })

Da die Potenzmenge $\mathcal{P}(v)$ gebildet wird, werden $(2^n)-1$ Elemente erzeugt, was eine gleiche Laufzeit von $2^n \in O(2^n)$ zur Folge hat, was man bei der Betrachtung des Algorithmus nachvollziehen kann:
Zu Beginn wird die leere Menge zu *subsets* hinzugefügt, was zur Folge hat, dass in der inneren *for-Schleife* das 1. Element des Vektors v als Teilmenge hinzugefügt wird. Im nächsten äußeren Schleifendurchlauf
(i = 1) ist die *subset.size()* nun 2, d.h. die innere Schleife wird 2 mal durchlaufen, wobei die j-te Teilmenge durch *single_set = subsets[j]* (also single_set = {} im ersten Durchlauf und single_set = {1}) in *single_set* geschrieben wird, bevor der *i-te* Eintrag (2) in v hinzugefügt und das *single_set* dem *subsets* hinzugefügt wird. Für die nächsten Einträge läuft der Algorithmus analog.