Übung 02: Divide & Conquer

Aufgabe 1: Maximum Subarray

Gegeben sei das folgende Array:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Wert	13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

In der Vorlesung wurde der rekursive Algorithmus (Version 2.0, Divide-and-Conquer) zur Bestimmung des Maximum Subarray Problems besprochen, siehe Pseudocode.

```
FIND-MAXIMUM-SUBARRAY(A, low, high)  // Version 2.0
1          if high == low
2               return (low, high, A[low]) // only 1 element
3          else mid = ⌊(low+high)/2⌋
4               (leftLow, leftHigh, leftSum) =
                     FIND-MAXIMUM-SUBARRAY(A, low, mid)
5               (rightLow, rightHigh, rightSum)=
                     FIND-MAXIMUM-SUBARRAY(A, mid+1, high)
6               (crossLow, crossHigh, crossSum) =
                     FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
7               if leftSum ≥ rightSum and leftSum ≥ crossSum
8                    return  (leftLow, leftHigh, leftSum)
9               elseif rightSum ≥ leftSum and rightSum ≥ crossSum
10                   return (rightLow, rightHigh, rightSum)
11              else return (crossLow, crossHigh, crossSum)
```

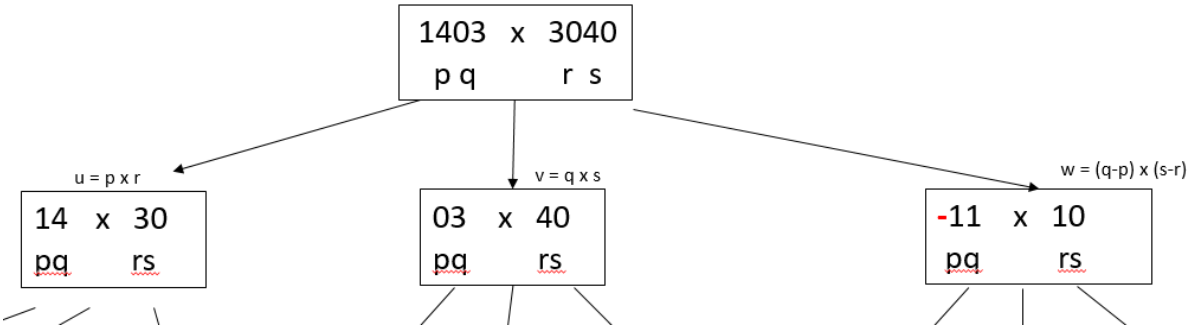
a) Wenden Sie den Pseudocode für das ***komplette (!)*** Array an.
- Wie lauten die Parameter für den initialen Aufruf der obigen Funktion?
- Welche 3 Teilarrays ergeben sich bei den Aufrufen von `FIND-MAXIMUM-SUBARRAY` bzw. `FIND-MAX-CROSSING-SUBARRAY` in Zeile 4, 5, und 6?
- Wie lautet das Ergebnis für das komplette Array?

b) Wie passiert, falls ***alle*** Elemente der Eingabe negativ sind? Was ist das Ergebnis?

Aufgabe 2: Multiplikation großer Zahlen, Karatsuba

Multiplizieren Sie den linken Faktor a =1403 mit dem rechten Faktor b =3040 nach dem Algorithmus von Karatsuba! Ergänzen Sie hierzu ***die folgende Abbildung***!

Mehrstellige Multiplikationen müssen ***rekursiv*** auf ***einstellige*** Multiplikationen zurückgeführt werden. Anschließend müssen Sie die Teilergebnisse wieder korrekt ***kombinieren***. Das alles soll aus Ihrer Lösung ersichtlich sein.



Aufgabe 3: Maximum Subarray - Implementierung
(Modifikation des *Bundeswettbewerb für Informatik* 1985)

Ein Wirtschaftsmagazin will seinen Lesern eine Analyse der Börsenentwicklung der letzten 17 Tage für eine bestimmte Aktie präsentieren. Für diese Aktie soll nachträglich ein bester Einkaufs- und Verkaufstag festgestellt werden.
Der beste Einkaufs- bzw. Verkaufstag für eine Aktie wäre derjenige gewesen, der den ***höchsten absoluten Gewinn*** geliefert hätte. Der absolute Preis der Aktie spielt keine Rolle, es geht nur darum, wo der Anleger absolut am meisten verdienen kann.
Dabei wird angenommen, dass ein Kapitalanleger die Aktie nur ***einmal* kaufen** darf und dass er spätestens zum betrachten Zeitraum die Aktie wieder verkaufen muss. Ferner kauft der Kapitalanleger von jeder Aktie ***nur 1 Stück***.
Das Wirtschaftsmagazin hat Informationen über die (absoluten) ***Kursänderungen*** der Aktie für die letzten 17 Börsentage. Letztendlich hat er also eine Zahlenfolge aus 16 Kursänderungen zwischen den Tagen. Die *n*.-te Zahl gibt die Kursänderung zwischen Tag *n* und Tag *n*+1 an. Man kann nicht am gleichen Tag kaufen und verkaufen.

Unterstützen Sie die Kursanalyse durch Schreiben eines Programms, das für eine Aktie aus der gegebenen Zahlenfolge nachträglich ***einen besten Einkaufstag, einen besten Verkaufstag und den dabei höchsten erzielbaren absoluten Gewinn*** ermittelt.

Beachten Sie folgende Hinweise:

- Das Ziel ist ***lineare*** Laufzeit $\Theta(n)$ bei *n* Börsentagen. Setzen Sie die in der Vorlesung besprochene Idee des **`MAX-SUBARRAY-SCANLINE-`**Algorithmus um ***(Version 3.0)!***
- Verwenden Sie den Coderahmen aus der Datei `Aktienkurse.java` verwenden. Dort ist bereits ein Zahlenbeispiel vorgegeben.
- Zur Kontrolle: Im vorgegebenen Beispiel ist der optimale Einkaufstag der 6. Tag und der optimale Verkaufstag der 11. Tag.
- Verwenden Sie Debugging zur Fehlersuche!