**Aufgabe 3**

**a)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | = |  | = |  |
|  |  | = |  | = |  |
|  |  | = |  | = |  |
|  |  | = |  | = |  |

Annahme:

|  |  |  |
| --- | --- | --- |
|  | = |  |

Für n → 0: bereits bewiesen (s.o.).

Für n → n +1:

|  |  |
| --- | --- |
|  |  |
|  |  | = |  |

Ergebnismatrize an der Stelle 2x2 da .

Da und :

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| = |  |  |  |
| = |  |

Da immer ist und die Multiplikation mit dem Vektor immer nur die zweite Spalte ausgibt ist das Ergebnis dieser Formel immer .

**b)**

Eine effizientere Berechnung von Xn lässt sich mit dem folgenden Pseudocode darstellen:

1: **function** Xn(X, n)

2: **if** ((X > 1) & (X % 2 = 0))

3: **return** (Xn(X, n/2))2

4: **else if** ((X > 1) & (X % 2 != 0))

5: **return** (Xn(X, (n-1)/2))2 \* X

6: **else**

7: **return** X

8: **end function**

Diese rekursive Funktion ruft sich immer wieder selber auf bis n = 1 ist. Dabei wird n in jedem Schritt halbiert (Zeile 2, 3). Sollte n nicht ganzzahlig halbierbar sein so wird n-1 halbiert und das fehlende X manuell hinzu multipliziert (Zeile 4, 5). Sobald n den Wert 1 erreicht hat wird X zurückgegeben und der Algorithmus terminiert (Zeile 6, 7, 8).

Da die Laufzeit von n abhängt und dieses sich superlinear verringert, ist die Laufzeit sublinear. Setzt man für n z.B. 32 ein, so wird dieser Algorithmus 6 mal durchlaufen (32, 16, 8, 4, 2, 1). 6 entspricht dabei ln(32). Der Algorithmus wird also ln(n) mal durchlaufen und seine Laufzeit lässt sich mit dem Term Xn(X, n) ϵ O(log n) beschreiben.

**c)**

Das Matrizen-Verfahren lässt sich auch als Pseudocode ausdrücken:

1: **function** MatFib(n)

2: e →

3: **while** (n > 1) )

4: e →

5: n- -

6: **return**

7: **end function**

Diese iterative Berechnung der Fibonaccizahlen führt genau n Rechenoperationen durch   
(n-1 mal die while-Schleife und 1 mal zur Berechnung der Ausgabe). In der while-Schleife werden je Durchlauf 8 Multiplikationen und 4 Additionen durchgeführt ().   
Bei der Ausgabe werden einmalig 4 Multiplikationen und 2 Additionen durchgeführt (). Die Laufzeit lässt sich damit wie folgt berechnen:

Laut Vorlesung kann eine Fibonaccizahl bis zu n Bits beanspruchen. Setzt man n für *l* ein so ergibt sich (abzüglich aller Konstanten) für MatFib(n) ebenfalls eine quadratische Laufzeit. Zu zeigen, dass das Matrizen-Verfahren echt schneller läuft als das in der Vorlesung gezeigte Verfahren ist mir daher nicht möglich.