AD PR3

Aufgabe3.1. 1:  
Bestimme die Anzahl der Operationen, die der folgende Algorithmus ausführt:  
  
1: x = 0

2: for i = 1 to n do

3: x = x + A[i]

4: end for

5: return x

In Zeile 1 wird eine Zuweisung gemacht. (1)  
In Zeile 2 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. (n)  
In Zeile 3 wird eine Addition und eine Zuweisung durchgeführt. Allerdings passiert das in einem Assemblertakt, sodass wir dies nur als eine Operation werten. Trotzdem wird diese Operation n mal durchgeführt. (n)  
In Zeile 5 wird eine Rückgabe gemacht. (1)

Zusammen beträgt die Laufzeit T(2n + 2)

Aufgabe 3.1.2:

Bestimme die Anzahl der Operationen, die der folgende Algorithmus ausführt:

1: for i = 1 to n do

2: A[i] = i

3: end for

4: for i = 1 to n do

5: C[i] = 0

6: for j = n downto 1 do

7: if A[j] > C[i] then

8: C[i] = A[j]

9: end if

10: end for

11: end for

12: return C

In Zeile 1 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. (n)

In Zeile 2 wird eine Zuweisung gemacht. Diese findet n mal statt. (n)

In Zeile 4 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. (n)  
In Zeile 5 wird eine Zuweisung gemacht. Diese findet n mal statt. (n)  
In Zeile 6 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. Dies wird n mal gemacht (n²)  
In Zeile 7 wird ein Vergleich durchgeführt. Dies findet n² mal statt. (n²)  
In Zeile 8 findet eine Zuweisung statt. Diese kann maximal n² mal statt finden. (n²)

In Zeile12 findet eine Rückgabe statt. (1)

Zusammen beträgt die Laufzeit T(3n² + 4n + 1)

Aufgabe 3.1.3:

Bestimme die Anzahl der Operationen, die der folgende Algorithmus ausführt:

1: for i = 1 to n do

2: for j = 1 to n do

3: C[i][j] = 0

4: for k = 1 to n do

5: C[i][j] = A[i][k] \_ B[k][j]

6: end for

7: end for

8: end for

9: return C

In Zeile 1 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. (n)

In Zeile 2 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. Dies wird n mal gemacht (n²)  
In Zeile 3 findet eine Zuweisung statt. Dieses passiert n² mal. (n²)

In Zeile 4 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. Dies wird n² mal gemacht (n³)

In Zeile 5 findet eine Multiplikation und eine Zuweisung statt. Da dies in einem Assemblertakt passiert, werten wir dies als eine Operation. Das findet n³ mal statt. (n³)

In Zeile 9 findet eine Rückgabe statt. (1)

Zusammen beträgt die Laufzeit T(2n³ + 2n² + n + 1)

Aufgabe 3.1.4

Bestimme die Anzahl der Operationen, die der folgende Algorithmus ausführt:

1: for i = 1 to n do

2: for j = 1 to i do

3: x = x + A[i][j]

4: end for

5: end for

6: return x

In Zeile 1 wird erst eine Zuweisung gemacht und in jedem darauf folgenden Durchlauf i inkrementiert. (n)

In Zeile 2 wird erst eine zuweisung gemacht und in jedem darauf folgenden Durchlauf I inkrementiert. Dies findet n mal statt, allerdings veringert sich die Menge der Inkrementationen bei jedem Durchlauf um 1. Daher lässt sich der Aufwand mit der Gaußschen Summenformel beschreiben. ((n² + n) / 2)

In Zeile 3 findet eine Addition und eine Zuweisung statt. Da dies in einem Prozessortakt statt findet werten wir dies als eine Operation. Diese findet (n² + n) / 2 mal statt. ((n² + n) / 2)

In Zeile 6 findet eine Rückgabe statt. (1)

Zusammen beträgt die Laufzeit T(n² + 2n + 1)

Aufgabe 3.2.1

Vergleichen Sie die Laufzeit der Algotihmen in Abhängigkeit von k! Machen Sie geeignete

Experiemente und stellen Sie die beiden Laufzeiten graphisch dar.

Für welche Werte von x und k kommen die Implementation an ihre Grenzen?

Die Obere Grenze von expOpt liegt bei k = 30 für n = 2.

Die Obere Grenze von exp liegt ebenso bei k = 30 für n = 2.

Die Obere Grenze von expOpt liegt bei n = 46340 für k = 2.

Die Obere Grenze von exp liegt bei n = 46340 für k = 2.

Daraus ist zu entnehmen, dass die optimierte Version von expOpt keine höheren Zahlen benötigt, als exp und so die selben Grenzen hat wie exp, daür aber wesentlich schneller ist.

Aufgabe 3.2.4

Beide Implementation können auch dann eingesetzt werden, wenn x keine Zahl, sondern

eine Matrix ist.

Vergelichen Sie die Laufzeiten, indem Sie wieder quadratische

(geeignet große) Zufallsmatrizen erzeugen und diese Potenzieren.

3.3.1

Beweisen sie:

Zeigen durch:

Einsetzen

Da 0 offensichtlich kleiner als Unendlich ist, ist die Aussage wahr.

3.3.2

Beweisen sie:

Zeigen durch:

Einsetzen

Da das Ergebnis nicht < unendlich ist, ist die Aussage wahr.

3.3.3.1

Wir wählen

Da der höchste Exponent von f kleiner als der von g ist, ist es offensichtlich, dass

Ab einem n0 von 0 gilt, dass

Wie durch die Konstante +3 in zu sehen ist.

3.3.3.2

Wir wählen

Da f und g bis auf die Konstanten identisch sind gilt

Ab einem n0 von 0 gilt, dass

Wie durch die größere Konstante +4 in und sonstige Gleichheit von f und g zu sehen ist.

3.3.4

Da keiner der Summanden 0 sein kann ( durch ) ist der einzig relevante Summand

Daher müssen wir nur zeigen, dass die beiden Summanden mit dem größten Exponenten in der selben Komplexitätsklasse sind.

Weil f und g den selben Grad haben, gilt

Wir wählen und

Daher gilt:

Und somit ist

3.3.5

Da bei einer Summation von Polynomen mit natürlichzahligen Koeffizienten nur derjenige für die Komplexität der Funktion f relevant ist, der den größten Exponenten hat, können wir sagen, dass

Also reicht es zu zeigen, dass

Was offensichtlich wahr ist