AlgoDat Übung 07: Laufzeitkomplexität

Aufgabe **1 (35%)**: **Laufzeitkomplexität ermitteln**

Gegeben ist folgender Algorithmus zur Umwandlung einer als Zeichenkette (String) gegebenen Binärzahl in einen Integer-Wert.

```
FUNCTION IntOf(dual: STRING): INTEGER;
VAR
   result, i: INTEGER;
BEGIN
   result := 0;
   i := 1;
  WHILE i <= Length(dual) DO BEGIN
      result := result * 2;
      IF dual[i] = '1' THEN
         result := result + 1;
      i := i + 1;
   END; (*WHILE*)
   IntOf := result;
END; (*IntOf*)
```

a) Entwickeln Sie unter Verwendung der nachstehenden Tabelle eine Formel, welche für die Länge einer Binärzahl und die Anzahl der darin enthaltenen Einsen die exakte Laufzeit des Algorithmus angibt. Berechnen Sie damit die Laufzeit für die Binärzahlen 100100, 100001, 110100, 1111, 0000, 1 und 0.

Operation	**Ausführungszeit**
Wertzuweisung	1
Vergleich	1
Indizierung	0,5
Addition, Subtraktion	0,5
Multiplikation	3
Prozeduraufruf	16 + 2 * Anzahl der Parameter

b) Entwickeln Sie nun unter der Annahme, dass Einsen und Nullen in einer Binärzahl etwa gleich oft vorkommen, eine neue Formel, die auf Basis der Länge einer beliebigen Binärzahl die (angenäherte) Laufzeit des Algorithmus angibt. Erstellen Sie damit eine Tabelle, welche die Laufzeiten für Binärzahlen der Länge 1 bis 20 sowie der Längen 50, 100 und 200 darstellt.

c) Bestimmen die asymptotische Laufzeitkomplexität für den gegebenen Algorithmus in Abhängigkeit von der Länge der Binärzahl. (Die Anzahl der Einsen und Nullen können Sie wieder als gleichmäßig verteilt annehmen.) Begründen Sie, wie Sie auf Ihre Lösung gekommen sind. Wie steht diese in Zusammenhang mit der in b) erstellten Tabelle?

Aufgabe 2 **(35%): Laufzeitkomplexität und Rekursion**

Gegeben ist der folgende rekursive Algorithmus, der wiederum für eine Binärzahl in Form einer Zeichenkette den Integer-Wert liefert.

```
FUNCTION IntOf2(dual: STRING): INTEGER;
   FUNCTION IORec(pos: INTEGER): INTEGER;
   BEGIN
      IF pos = 0 THEN
         IORec := 0
      ELSE IF dual[pos] = '1' THEN
         IORec:= IORec(pos - 1) * 2 + 1
      ELSE
         IORec:= IORec(pos - 1) * 2;
   END; (*IORec*)
BEGIN (*IntOf2*)
   IntOf2 := IORec(Length(dual));
END; (*IntOf2*)
```

a) Entwickeln Sie wieder eine Formel für die Laufzeit auf Basis von Länge und Anzahl der Einsen in der Binärzahl auf und bestimmen Sie „exakten" Laufzeiten für die Beispiele aus Beispiel 1a).

Vorgehensweise: Bestimmen Sie zunächst getrennt für jeden der (drei) möglichen Zweige der inneren Funktion IORec die Laufzeit eines einzelnen Durchlaufs und überlegen Sie dann (z.B. an Hand eines Beispiels), wie oft jeder der Zweige durchlaufen wird. Durch einfaches Multiplizieren und Addieren erhalten Sie dann die Gesamtlaufzeit.

b) Finden Sie auch hier wieder eine Formel, welche von einer gleichmäßigen Verteilung Einsen und Nullen ausgeht, und stellen Sie die gleiche Tabelle wie Beispiel 1b) auf.

c) Bestimmen Sie auch für den rekursiven die asymptotische Laufzeitkomplexität. Betrachten Sie dazu am Besten wieder die Daten der unter b) erstellten Tabelle.

d) Vergleichen Sie nun Ihre Analysen des iterativen (Beispiel 1) und den rekursiven Algorithmus (Beispiel 2). Wie ist jeweils die asymptotische Laufzeitkomplexität? Was zeigen (im Gegensatz dazu) die Grob- bzw. Feinanalyse? Was schließen Sie daraus in Bezug auf die Güte der beiden Lösungen?

Aufgabe 3 **(30%): Laufzeitkomplexität**

Gegeben sind wieder drei Algorithmen (A1, A2 und A3), die alle das gleiche Problem lösen (Problemgröße ist n). Hier die Anzahl der Berechnungsschritte, die jeder Algorithmus braucht, um ein Problem "der Größe n" zu lösen:

- A1 benötigt $1000 \cdot n$ Schritte,
- A2 benötigt $50 \cdot n^2$ Schritte und
- A3 benötigt 2^n Schritte.

Für welche Werte von n ist welcher Algorithmus am schnellsten?