1 Lösungsidee

Folgend sind die Lösungsideen der Aufgaben 1, 2 der Übung 10 angeführt, wobei die Laufzeitkomplexität zweier Algorithmen berechnet werden soll. Einmal soll die Laufzeitkomplexität eines iterativen und einmal eines rekursiven Algorithmus berechnet werden.

Operationsgewichtung:

Operation	Ausführungszeit
:=	1,4
=, <, >,…	1
AND, OR,…	1
Array[i]	0,4
+, -	0,6
*	3,6
Length(...) / Ord(...)	0
prod (Prozeduraufruf)	16 + 2 * Anzahl der Parameter

Hierbei sollen die einzelnen Operationen der Algorithmen in Abhängigkeit ihrer Aufrufe und Gewichtungen ermittelt werden. Diese Ergebnisse sollen in eine Tabelle übertragen werden. Anschließend sollen die Funktionen für die exakte (Feinanalyse), der geschätzten (Grobanalyse) und der asymptotischen Laufzeitkomplexität ermittelt werden. Mit diesen aufgestellten Funktionen sollen die Laufzeitenkomplexitäten (Funktionswerte in Abhängigkeit von der Zeichenlänge) für die gegeben Inputs berechnet, in einer Excel-Tabelle aufgelistet und über Plot Charts grafisch dargestellt werden.

Ermitteln der exakten Laufzeitkomplexität:

Zuerst sollen die einzelnen Codezeilen analysiert und die Gewichtungen der einzelnen Anweisungen in Abhängigkeit der möglichen Aufrufe (*n*) ermittelt werden. Diese Ermittelten Werte sollen anschließend in eine Tabelle übertragen und entsprechend ihrer Aufrufe (Bsp.: *n*, *n + 1*,…) addiert werden. Mit diesen Ergebnissen soll anschließend die Funktion aufgestellt werden. Hierbei ist auch der Zeichenketteninhalt bekannt.

Ermitteln der geschätzten Laufzeitkomplexität:

Hierbei soll wie bei der exakten Laufzeitkomplexität vorgegangen werden. Lediglich die möglichen Aufrufe, wobei ungültige Zeichen behandelt werden ist mit $^{n}/_{4}$ angenommen werden.

Ermittlung der asymptotischen Laufzeitkomplexität:

Für die Ermittlung der Laufzeitkomplexität müssen folgende Parameter ermittelt werden.

1. n_0: Die untere Grenze der Funktionswerte, ab der $O(n)$ gilt.
2. c: Die Konstante, die mit *g(n)* multipliziert wird.
3. $g(n)$: Die Funktion, die die obere Schranke darstellt.

Für eine asymptotische Laufzeitkomplexität gilt allgemein:

$$f\ ist\ \mathrm{O}(g) \equiv \bigwedge_{c>0} \bigwedge_{n_0 \in N} \bigvee_{n>n_0} f(n) \leq c * g(n)$$

Für die möglichen Aufrufe sollen die folgenden Variablen verwendet werden:

n…. Länge der Zeichenkette.

j…. Anzahl der Zeichen die ganzzahlige Zahlenwerte darstellen.

Des Weiteren sollen die ermittelten Laufzeitkomplexitäten der beiden Algorithmen diskutiert und verglichen werden. Insbesondere soll auf die Unterschiede der Laufzeitkomplexität der iterativen und der rekursiven Implementierung eingegangen werden.

2 Iterativer Algorithmus

Folgend ist Ermittlung der Aufwände in Abhängigkeit ihrer möglichen Aufrufe der Operationen des iterativen Algorithmus angeführt. Innerhalb der Tabelle mit den Programmcodeteil werden die einzelnen Operationen analysiert und dessen mögliche Aufrufe ermittelt. Unterhalb werden innerhalb einer Tabelle die Gesamtgewichtungen ermittelt.

Algorithmus:

```
PROGRAM IterativAlgorithm;

CONST
  Zero = 48;
  Nine = 57;

FUNCTION GetValue(numStr: STRING): LONGINT;
VAR
  i: INTEGER;
  asciiCode: BYTE;
  result: LONGINT;
BEGIN
  result := 0;                                                    1     1. :=
  i := 1;                                                         1     2. :=
  WHILE i <= Length(numStr) DO BEGIN                            n + 1   3. <=
    asciiCode := Ord(numStr[i]);                                  n     4. array[i] + :=
    IF (asciiCode >= Zero) AND (asciiCode <= Nine) THEN           n     5. >= + <= + AND
      result := result * 10 + asciiCode - Zero;                   j     6.* + + + - + :=
    i := i + 1;                                                   n     7. + + :=
  END;
  GetValue := result;                                             1     8. :=
END;

BEGIN
  GetValue(…);
END.
```

Ermitteln der Funktion:

Zeile	Operation	Anzahl	Gewichtung	n	n + 1	j	Konstant
1.	:=	1	1,4				1,4
2.	:=	1	1,4				1,4
3.	<=	n + 1	1		1		
4.1	array[i]	n	0,4	0,4			
4.2	:=	n	1,4	1,4			
5.1	>=	n	1	1			
5.2	<=	n	1	1			
5.3	AND	n	1	1			
6.1	*	j	3,6			3,6	
6.2	+	j	0,6			0,6	
6.3	-	j	0,6			0,6	
6.4	:=	j	1,4			1,4	
7.1	+	n	0,6	0,6			
7.2	:=	n	1,4	1,4			
8.	:=	1	1,4				1,4
				6,8	**1**	**6,2**	**4,2**

2.1 Exakte Laufzeitkomplexität

Diese Funktion wurde aus der oben angeführten Tabelle abgeleitet und kann für die Berechnung der exakten Laufzeitkomplexität des iterativen Algorithmus herangezogen werden, wobei die beiden Parameter *n*, *j* als Übergabeparameter gegeben sein müssen.

$f(n,j) = 6{,}8 * n + 1 * (n+1) + 6{,}2 * j + 4{,}2 =$
$f(n,j) = n * (6{,}8 + 1) + 1 + 6{,}2 * j + 4{,}2$
$f(n,j) = 7{,}8 * n + 6{,}2 * j + 5{,}2$

Folgend sind die Berechnungen der exakten Laufzeitkomplexität des iterativen Algorithmus mit folgenden Inputs unter Abhängigkeit der Zeichenlänge sowie der Zeichen innerhalb der Zeichenkette angeführt.

„1234567“ (n=7, j=7):
$f(n,j) = 7{,}8 * 7 + 6{,}2 * 7 + 5{,}2 = \mathbf{103{,}2}$

„1.234.567“ (n=9 j=7):
$f(n,j) = 7{,}8 * 9 + 6{,}2 * 7 + 5{,}2 = \mathbf{118{,}8}$

„1xv323a#42.83“ (n=13, j=8):
$f(n,j) = 7{,}8 * 13 + 6{,}2 * 8 + 5{,}2 = \mathbf{156{,}2}$

„000001“ (n=6, j=6):
$f(n,j) = 7{,}8 * 6 + 6{,}2 * 6 + 5{,}2 = \mathbf{89{,}2}$

„789“ (n=3, j=3):
$f(n,j) = 7{,}8 * 3 + 6{,}2 * 3 + 5{,}2 = \mathbf{47{,}2}$

„abc“ (n=3, j=0):
$f(n,j) = 7{,}8 * 3 + 6{,}2 * 0 + 5{,}2 = \mathbf{28{,}6}$

„1“ (n=1, j=1):
$f(n,j) = 7{,}8 + 6{,}2 + 5{,}2 = \mathbf{19{,}2}$

„“ (n=0, j=0):
$f(n,j) = 0 + 0 + 5{,}2 = \mathbf{5{,}2}$

2.2 Geschätzte Laufzeitkomplexität

Folgend sind die Berechnung der wahrscheinlichen Laufzeitkomplexität des Algorithmus unter Abhängigkeit der Zeichenlänge angeführt, wobei davon ausgegangen wird, dass ein ungültiges Zeichen (kein gültiger ganzzahliger Zahlenwert) jedes viertes Zeichen auftritt.

Funktion:

Folgend ist die Funktion angeführt, welche sich von der Funktion der exakten Laufzeitkomplexität ableitet aber nicht mehr von den Zeichenketteninhalt abhängt, sondern nur von der Zeichenkettelänge.

$$j = n - \left({}^{6,2}/_{4} * n\right)$$

$$f(n) = 7{,}8 * n + 6{,}2 * n - \left({}^{6,2}/_{4} * n\right) + 5{,}2$$

$$f(n) = 14 * n - 1{,}55 * n + 5{,}2$$

$$f(n) = 12{,}45 * n + 5{,}2$$

Die berechneten Ergebnisse wurden in einer Tabelle aufgelistet und auch grafisch dargestellt.
Siehe angefügte Excel Tabellen am Ende des Dokuments.

2.3 Asymptotische Laufzeitkomplexität

Folgend ist die asymptotische Laufzeitkomplexität angeführt, die sich aus der Funktion der geschätzten Laufzeitkomplexität und dessen berechneten Werten ableitet.

1. n_0 ab $O(n)$ gilt:

n_0 lässt sich aus der Tabelle der Funktionswerte von $f_{grob}(n)$ welche den Funktionswerten von $O(n)$ gegenübergestellt wurden ablesen. n_0 gibt hierbei an ab wann $O(n)$gilt. Davor kann mit $O(n)$ keine Aussage über die Laufzeitkomplexität getroffen werden.

2. Die Konstante c:

Die Ermittlung der Konstante c ergibt sich aus dem Wachstum der Funktionswerte von $f(n)$.

$$c = {}^{f(n)}/_{n}$$

Folgend sind einige Berechnungen angeführt, die c ermitteln. Es werden Funktionswerte aus verschiedenen Bereichen von $0 \leq n \leq 255$ berechnet.

$bei\ n = 10, daraus\ folgt\ {}^{129,7}/_{10} = \mathbf{12,97}$

$bei\ n = 125, daraus\ folgt\ {}^{1561,45}/_{125} = \mathbf{12,49}$

$bei\ n = 255, daraus\ folgt\ {}^{3179,95}/_{255} = \mathbf{12,47}$

Daraus ergibt sich $c = 13$, wobei zu sehen ist, dass die Steigung bei steigenden n abnimmt.

3. $g(n)$ die obere asymptotische Grenze:

Da die Funktion $f(n)$ mit dem Faktor ungefähr 13 in Abhängigkeit von n steigt, ergibt sich daraus $g(n) = 13 * n$. Dies stellt die obere Schranke von $f(n)$ dar.

Aus den vorherigen Berechnungen ergibt sich $f\ ist\ O(n)$:

$$f\ ist\ \mathrm{O}(g) = \bigwedge_{c>0} \bigwedge_{n_{10} \in N} \bigvee_{n>n_{10}} f(n) \leq 13 * g(n)$$

$$f\ ist\ \mathrm{O}(g) = \bigwedge_{c>0} \bigwedge_{n_{10} \in N} \bigvee_{n>n_{10}} 12{,}45 * n + 5{,}2 \leq 13 * n$$

Also ist die Laufzeitkomplexität des Algorithmus $O(13 * n)$, wobei diese Laufzeitkomplexität erst ab $10 * n$ gültig ist.

2.4 Auswertung Laufzeitkomplexität

Folgend ist die Auswertung der beiden ermittelten Laufzeitkomplexitäten (geschätzte und asymptotisch) aufgeführt.

n	f(n)	f(n)/n	O(27*n)
0	20.8	0	0
1	46.4	46.4	27
2	72	36	54
3	97.6	32.53333333	81
4	123.2	30.8	108
5	148.8	29.76	135
6	174.4	29.06666667	162
7	200	28.57142857	189
8	225.6	28.2	216
9	251.2	27.91111111	243

10	276.8	27.68	270
11	302.4	27.49090909	297
12	328	27.33333333	324
13	353.6	27.2	351
14	379.2	27.08571429	378
15	404.8	26.98666667	405
16	430.4	26.9	432
17	456	26.82352941	459
18	481.6	26.75555556	486
19	507.2	26.69473684	513
20	532.8	26.64	540
21	558.4	26.59047619	567
22	584	26.54545455	594
23	609.6	26.50434783	621
24	635.2	26.46666667	648
25	660.8	26.432	675
26	686.4	26.4	702
27	712	26.37037037	729
28	737.6	26.34285714	756
29	763.2	26.31724138	783
30	788.8	26.29333333	810
31	814.4	26.27096774	837
32	840	26.25	864
33	865.6	26.23030303	891
34	891.2	26.21176471	918
35	916.8	26.19428571	945
36	942.4	26.17777778	972
37	968	26.16216216	999
38	993.6	26.14736842	1026
39	1019.2	26.13333333	1053
40	1044.8	26.12	1080
41	1070.4	26.10731707	1107
42	1096	26.0952381	1134
43	1121.6	26.08372093	1161
44	1147.2	26.07272727	1188
45	1172.8	26.06222222	1215
46	1198.4	26.05217391	1242
47	1224	26.04255319	1269
48	1249.6	26.03333333	1296
49	1275.2	26.0244898	1323
50	1300.8	26.016	1350
51	1326.4	26.00784314	1377
52	1352	26	1404
53	1377.6	25.99245283	1431

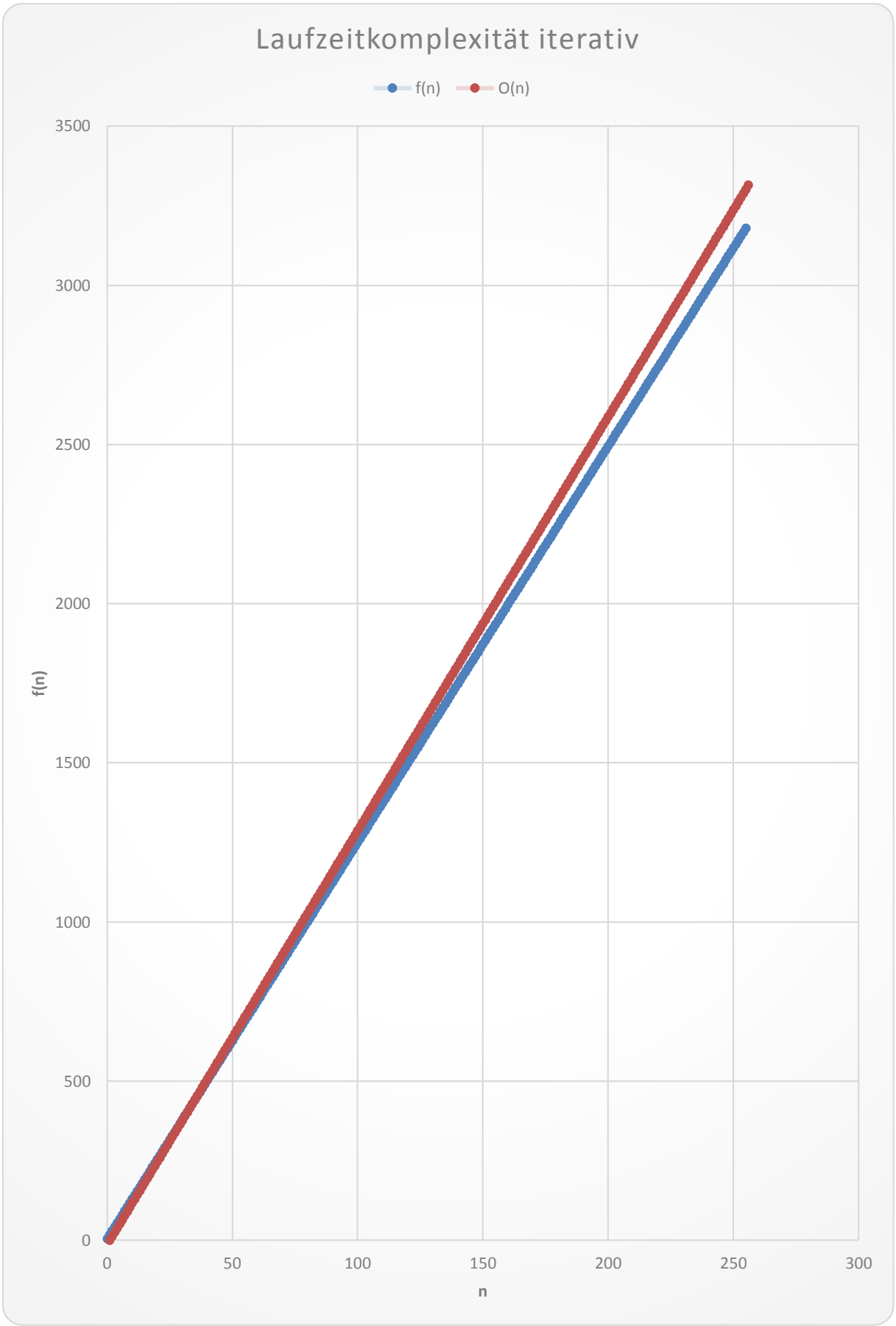
54	1403.2	25.98518519	1458
55	1428.8	25.97818182	1485
56	1454.4	25.97142857	1512
57	1480	25.96491228	1539
58	1505.6	25.95862069	1566
59	1531.2	25.95254237	1593
60	1556.8	25.94666667	1620
61	1582.4	25.94098361	1647
62	1608	25.93548387	1674
63	1633.6	25.93015873	1701
64	1659.2	25.925	1728
65	1684.8	25.92	1755
66	1710.4	25.91515152	1782
67	1736	25.91044776	1809
68	1761.6	25.90588235	1836
69	1787.2	25.90144928	1863
70	1812.8	25.89714286	1890
71	1838.4	25.89295775	1917
72	1864	25.88888889	1944
73	1889.6	25.88493151	1971
74	1915.2	25.88108108	1998
75	1940.8	25.87733333	2025
76	1966.4	25.87368421	2052
77	1992	25.87012987	2079
78	2017.6	25.86666667	2106
79	2043.2	25.86329114	2133
80	2068.8	25.86	2160
81	2094.4	25.85679012	2187
82	2120	25.85365854	2214
83	2145.6	25.85060241	2241
84	2171.2	25.84761905	2268
85	2196.8	25.84470588	2295
86	2222.4	25.84186047	2322
87	2248	25.83908046	2349
88	2273.6	25.83636364	2376
89	2299.2	25.83370787	2403
90	2324.8	25.83111111	2430
91	2350.4	25.82857143	2457
92	2376	25.82608696	2484
93	2401.6	25.82365591	2511
94	2427.2	25.8212766	2538
95	2452.8	25.81894737	2565
96	2478.4	25.81666667	2592
97	2504	25.81443299	2619

98	2529.6	25.8122449	2646
99	2555.2	25.81010101	2673
100	2580.8	25.808	2700
101	2606.4	25.80594059	2727
102	2632	25.80392157	2754
103	2657.6	25.80194175	2781
104	2683.2	25.8	2808
105	2708.8	25.79809524	2835
106	2734.4	25.79622642	2862
107	2760	25.79439252	2889
108	2785.6	25.79259259	2916
109	2811.2	25.79082569	2943
110	2836.8	25.78909091	2970
111	2862.4	25.78738739	2997
112	2888	25.78571429	3024
113	2913.6	25.7840708	3051
114	2939.2	25.78245614	3078
115	2964.8	25.78086957	3105
116	2990.4	25.77931034	3132
117	3016	25.77777778	3159
118	3041.6	25.77627119	3186
119	3067.2	25.77478992	3213
120	3092.8	25.77333333	3240
121	3118.4	25.77190083	3267
122	3144	25.7704918	3294
123	3169.6	25.76910569	3321
124	3195.2	25.76774194	3348
125	3220.8	25.7664	3375
126	3246.4	25.76507937	3402
127	3272	25.76377953	3429
128	3297.6	25.7625	3456
129	3323.2	25.76124031	3483
130	3348.8	25.76	3510
131	3374.4	25.75877863	3537
132	3400	25.75757576	3564
133	3425.6	25.75639098	3591
134	3451.2	25.75522388	3618
135	3476.8	25.75407407	3645
136	3502.4	25.75294118	3672
137	3528	25.75182482	3699
138	3553.6	25.75072464	3726
139	3579.2	25.74964029	3753
140	3604.8	25.74857143	3780
141	3630.4	25.74751773	3807

142	3656	25.74647887	3834
143	3681.6	25.74545455	3861
144	3707.2	25.74444444	3888
145	3732.8	25.74344828	3915
146	3758.4	25.74246575	3942
147	3784	25.7414966	3969
148	3809.6	25.74054054	3996
149	3835.2	25.73959732	4023
150	3860.8	25.73866667	4050
151	3886.4	25.73774834	4077
152	3912	25.73684211	4104
153	3937.6	25.73594771	4131
154	3963.2	25.73506494	4158
155	3988.8	25.73419355	4185
156	4014.4	25.73333333	4212
157	4040	25.73248408	4239
158	4065.6	25.73164557	4266
159	4091.2	25.73081761	4293
160	4116.8	25.73	4320
161	4142.4	25.72919255	4347
162	4168	25.72839506	4374
163	4193.6	25.72760736	4401
164	4219.2	25.72682927	4428
165	4244.8	25.72606061	4455
166	4270.4	25.7253012	4482
167	4296	25.7245509	4509
168	4321.6	25.72380952	4536
169	4347.2	25.72307692	4563
170	4372.8	25.72235294	4590
171	4398.4	25.72163743	4617
172	4424	25.72093023	4644
173	4449.6	25.72023121	4671
174	4475.2	25.71954023	4698
175	4500.8	25.71885714	4725
176	4526.4	25.71818182	4752
177	4552	25.71751412	4779
178	4577.6	25.71685393	4806
179	4603.2	25.71620112	4833
180	4628.8	25.71555556	4860
181	4654.4	25.71491713	4887
182	4680	25.71428571	4914
183	4705.6	25.7136612	4941
184	4731.2	25.71304348	4968
185	4756.8	25.71243243	4995

186	4782.4	25.71182796	5022
187	4808	25.71122995	5049
188	4833.6	25.7106383	5076
189	4859.2	25.71005291	5103
190	4884.8	25.70947368	5130
191	4910.4	25.70890052	5157
192	4936	25.70833333	5184
193	4961.6	25.70777202	5211
194	4987.2	25.70721649	5238
195	5012.8	25.70666667	5265
196	5038.4	25.70612245	5292
197	5064	25.70558376	5319
198	5089.6	25.70505051	5346
199	5115.2	25.70452261	5373
200	5140.8	25.704	5400
201	5166.4	25.70348259	5427
202	5192	25.7029703	5454
203	5217.6	25.70246305	5481
204	5243.2	25.70196078	5508
205	5268.8	25.70146341	5535
206	5294.4	25.70097087	5562
207	5320	25.70048309	5589
208	5345.6	25.7	5616
209	5371.2	25.69952153	5643
210	5396.8	25.69904762	5670
211	5422.4	25.6985782	5697
212	5448	25.69811321	5724
213	5473.6	25.69765258	5751
214	5499.2	25.69719626	5778
215	5524.8	25.69674419	5805
216	5550.4	25.6962963	5832
217	5576	25.69585253	5859
218	5601.6	25.69541284	5886
219	5627.2	25.69497717	5913
220	5652.8	25.69454545	5940
221	5678.4	25.69411765	5967
222	5704	25.69369369	5994
223	5729.6	25.69327354	6021
224	5755.2	25.69285714	6048
225	5780.8	25.69244444	6075
226	5806.4	25.6920354	6102
227	5832	25.69162996	6129
228	5857.6	25.69122807	6156
229	5883.2	25.69082969	6183

230	5908.8	25.69043478	6210
231	5934.4	25.69004329	6237
232	5960	25.68965517	6264
233	5985.6	25.68927039	6291
234	6011.2	25.68888889	6318
235	6036.8	25.68851064	6345
236	6062.4	25.68813559	6372
237	6088	25.68776371	6399
238	6113.6	25.68739496	6426
239	6139.2	25.68702929	6453
240	6164.8	25.68666667	6480
241	6190.4	25.68630705	6507
242	6216	25.68595041	6534
243	6241.6	25.68559671	6561
244	6267.2	25.6852459	6588
245	6292.8	25.68489796	6615
246	6318.4	25.68455285	6642
247	6344	25.68421053	6669
248	6369.6	25.68387097	6696
249	6395.2	25.68353414	6723
250	6420.8	25.6832	6750
251	6446.4	25.68286853	6777
252	6472	25.68253968	6804
253	6497.6	25.68221344	6831
254	6523.2	25.68188976	6858
255	6548.8	25.68156863	6885


Laufzeitkomplexität iterativ
f(n)
O(n)
3500
3000
2500
2000
1500
1000
500
0
f(n)
0
50
100
150
200
250
300
n

3 Rekursiver Algorithmus

Folgend ist Ermittlung der Aufwände in Abhängigkeit ihrer möglichen Aufrufe der Operationen des rekursiven Algorithmus angeführt. Innerhalb der Tabelle mit den Programmcodeteil werden die einzelnen Operationen analysiert und dessen mögliche Aufrufe ermittelt. Unterhalb werden innerhalb einer Tabelle die Gesamtgewichtungen ermittelt.

Algorithmus:

```
PROGRAM RecursiveAlgorithm;

FUNCTION GetValue2(numStr: STRING): LONGINT;
  FUNCTION GetValueInternal(pos: INTEGER): LONGINT;
  VAR
    asciiCode: BYTE;
  BEGIN
    ////////////////////////////////////////////////
    Zweig 1
    ////////////////////////////////////////////////
    IF pos < 1 THEN                                          n + 1    1. <
      GetValueInternal := 0                                    1      2. :=
    ////////////////////////////////////////////////
    Zweig 2
    ////////////////////////////////////////////////
    ELSE BEGIN
      asciiCode := Ord(numStr[pos]);                           n      1. array[i] + :=
      IF (asciiCode >= Zero) AND (asciiCode <= Nine) THEN      n      2. >= + <= + AND
        GetValueInternal := asciiCode - Zero +                 j      3. := + - + +
        10 * GetValueInternal(pos - 1)                         j         * + - + prod(1)
      ELSE
        GetValueInternal := GetValueInternal(pos - 1);       n - j    4. - + prod(1) + :=
    END;
    ////////////////////////////////////////////////
  END;

BEGIN
  ////////////////////////////////////////////////
  Erstaufruf
  ////////////////////////////////////////////////
  GetValue2 := GetValueInternal(Length(numStr));               1      8. prod(1) + :=
END;

BEGIN
  GetValue2(...);
END.
```

Erstaufruf:

Zeile	Operation	Anzahl	Gewichtung	n	n + 1	j	n - j	Konstant
1.1	prod(1)	1	16 + (2 * 1)					18
1.2	:=	1	1,4					1,4
								19,4

$f_1(n,j) = 19{,}4$

Zweig 1:

Zeile	Operation	Anzahl	Gewichtung	n	n + 1	j	n - j	Konstant
1.	<	n + 1	1		1			
2.	:=	1	1,4					1,4
					1			**1,4**

$f_2(n,j) = 1 * (\mathrm{n} + 1) + 1{,}4$

$= 1 * n + 1 + 1{,}4$

$= 1 * n + 2{,}4$

Zweig 2:

Zeile	Operation	Anzahl	Gewichtung	n	n + 1	j	n - j	Konstant
1.1	array[i]	n	0,4	0,4				
1.2	:=	n	1,4	1,4				
2.1	>=	n	1	1				
2.2	<=	n	1	1				
2.3	AND	n	1	1				
3.1	-	j	0,6			0,6		
3.2	prod(1)	j	16 + (2 * 1)			18		
3.3	*	j	3,6			3,6		
3.4	+	j	0,6			0,6		
3.5	-	j	0,6			0,6		
3.6	:=	j	1,4			1,4		
4.1	-	n – j	0,6				0,6	
4.2	prod(1)	n – j	16 + (2 * 1)				18	
4.3	:=	n – j	1,4				1,4	
				4,8		**24,8**	**20**	

$f_3(n,j) \coloneqq 4{,}8 * \mathrm{n} + 24{,}8 * \mathrm{j} + 20 * (\mathrm{n} - \mathrm{j})$

$= 4{,}8 * n + 24{,}8 * j + 20 * n - 20 * j$

$= n * (4{,}8 + 20) + j * (24{,}8 - 20)$

$= 24{,}8 * n + 4{,}8 * j$

Funktion gesamt:

$f_{gesamt}(n,j) \coloneqq f_1(n,j) + f_2(n,j) + f_3(n,j)$

$= 19{,}4 + 1 * \text{n} + 2{,}4 + 24{,}8 * n + 4{,}8 * j$

$= n * (1 + 24{,}8) + 4{,}8 * j + 19{,}4 + 2{,}4$

$= 25{,}8 * n + 4{,}8 * j + 20{,}8$

3.1 Exakte Laufzeitkomplexität

Diese Funktion wurde aus der oben angeführten Tabelle abgeleitet und kann für die Berechnung der exakten Laufzeitkomplexität des rekursiven Algorithmus herangezogen werden, wobei die beiden Parameter *n*, *j* als Übergabeparameter gegeben sein müssen.

$f(n,j) = 25{,}8 * n + 4{,}8 * j + 20{,}8$

Folgend sind die Berechnungen der exakten Laufzeitkomplexität des rekursiven Algorithmus mit folgenden Inputs unter Abhängigkeit der Zeichenlänge sowie der Zeichen innerhalb der Zeichenkette angeführt.

„1234567“ (n=7, j=7):

$f_{gesamt}(n,j) = 25{,}8 * 7 + 4{,}8 * 7 + 20{,}8 = \mathbf{235}$

„1.234.567“ (n=9 j=7):

$f(n,j) = 25{,}8 * 9 + 4{,}8 * 7 + 20{,}8 = \mathbf{286{,}6}$

„1xv323a#42.83“ (n=13, j=8):

$f(n,j) = 25{,}8 * 13 + 4{,}8 * 8 + 20{,}8 = \mathbf{394{,}6}$

„000001“ (n=6, j=6):

$f(n,j) = 25{,}8 * 6 + 4{,}8 * 6 + 20{,}8 = \mathbf{204{,}4}$

„789“ (n=3, j=3):

$f(n,j) = 25{,}8 * 3 + 4{,}8 * 3 + 20{,}8 = \mathbf{112{,}6}$

„abc“ (n=3, j=0):

$f(n,j) = 25{,}8 * 3 + 4{,}8 * 0 + 20{,}8 = \mathbf{98{,}2}$

„1“ (n=1, j=1):

$f(n,j) = 25{,}8 * 1 + 4{,}8 * 1 + 20{,}8 = \mathbf{51{,}4}$

„“ (n=0, j=0):

$f(n,j) = 25{,}8 * 0 + 4{,}8 * 0 + 20{,}8 = \mathbf{20{,}8}$

3.2 Geschätzte Laufzeitkomplexität

Folgend sind die Berechnung der wahrscheinlichen Laufzeitkomplexität des rekursiven Algorithmus unter Abhängigkeit der Zeichenlänge angeführt, wobei davon ausgegangen wird, dass ein ungültiges Zeichen (kein gültiger ganzzahliger Zahlenwert) jedes viertes Zeichen auftritt.

Funktion:

Folgend ist die Funktion angeführt, welche sich von der Funktion der exakten Laufzeitkomplexität ableitet aber nicht mehr von den Zeichenketteninhalt abhängt, sondern nur von der Zeichenkettelänge.

$j = n - \left({}^{20}/_{4} * n\right)$

$f(n) = 25{,}8 * n + 4{,}8 * n - \left({}^{20}/_{4} * n\right) + 20{,}8$

$f(n) = 25{,}8 * n + 4{,}8 * n - 5 * n + 20{,}8$

$f(n) = 25{,}6 * n + 20{,}8$

3.3 Asymptotische Laufzeitkomplexität

Folgend ist die asymptotische Laufzeitkomplexität angeführt, die sich aus der Funktion der geschätzten Laufzeitkomplexität und dessen berechneten Werten ableitet.

1. n_0 ab $O(n)$ gilt:

n_0 lässt sich aus der Tabelle der Funktionswerte von $f_{grob}(n)$ welche den Funktionswerten von $O(n)$ gegenübergestellt wurden ablesen. n_0 gibt hierbei an ab wann $O(n)$gilt. Davor kann mit $O(n)$ keine Aussage über die Laufzeitkomplexität getroffen werden.

2. Die Konstante c:

Die Ermittlung der Konstante c ergibt sich aus dem Wachstum der Funktionswerte von $f(n)$.

$c = {}^{f(n)}/_{n}$

Folgend sind einige Berechnungen angeführt, die c ermitteln. Es werden Funktionswerte aus verschiedenen Bereichen von $0 \leq n \leq 255$ berechnet.

$bei\ n = 10, daraus\ folgt\ {}^{276,8}/_{10} = \mathbf{27,68}$

$bei\ n = 10, daraus\ folgt\ {}^{660,8}/_{25} = \mathbf{26,43}$

$bei\ n = 125, daraus\ folgt\ {}^{3220,8}/_{125} = \mathbf{25,76}$

$bei\ n = 255, daraus\ folgt\ {}^{6548,8}/_{255} = \mathbf{25,68}$

Daraus ergibt sich $c = 27$.

3. $g(n)$ die obere asymptotische Grenze:

Da die Funktion $f(n)$ mit dem Faktor 27 in Abhängigkeit von n steigt, ergibt sich daraus $g(n) = 27 * n$. Dies stellt die obere Schranke von $f(n)$ dar.

Aus den vorherigen Berechnungen ergibt sich $f\ ist\ O(n)$:

$$f\ ist\ \mathrm{O}(g) = \bigwedge_{c>0} \bigwedge_{n_9 \in N} \bigvee_{n>n_9} f(n) \leq 27 * g(n)$$

$$f\ ist\ \mathrm{O}(g) = \bigwedge_{c>0} \bigwedge_{n_9 \in N} \bigvee_{n>n_9} 25{,}6 * n + 20{,}8 \leq 27 * n$$

Also ist die Laufzeitkomplexität des Algorithmus $O(27 * n)$, wobei diese Laufzeitkomplexität erst ab $9 * n$ gültig ist.

3.4 Auswertung Laufzeitkomplexität

Folgend ist die Auswertung der beiden ermittelten Laufzeitkomplexitäten (geschätzte und asymptotisch) aufgeführt.

n	f(n)	f(n)/n	O(27*n)
0	20.8	0	0
1	46.4	46.4	27
2	72	36	54
3	97.6	32.53333333	81
4	123.2	30.8	108
5	148.8	29.76	135
6	174.4	29.06666667	162
7	200	28.57142857	189
8	225.6	28.2	216
9	251.2	27.91111111	243
10	276.8	27.68	270
11	302.4	27.49090909	297
12	328	27.33333333	324
13	353.6	27.2	351
14	379.2	27.08571429	378
15	404.8	26.98666667	405
16	430.4	26.9	432
17	456	26.82352941	459
18	481.6	26.75555556	486
19	507.2	26.69473684	513
20	532.8	26.64	540
21	558.4	26.59047619	567
22	584	26.54545455	594
23	609.6	26.50434783	621
24	635.2	26.46666667	648
25	660.8	26.432	675
26	686.4	26.4	702
27	712	26.37037037	729
28	737.6	26.34285714	756
29	763.2	26.31724138	783
30	788.8	26.29333333	810
31	814.4	26.27096774	837
32	840	26.25	864
33	865.6	26.23030303	891
34	891.2	26.21176471	918
35	916.8	26.19428571	945
36	942.4	26.17777778	972
37	968	26.16216216	999
38	993.6	26.14736842	1026
39	1019.2	26.13333333	1053

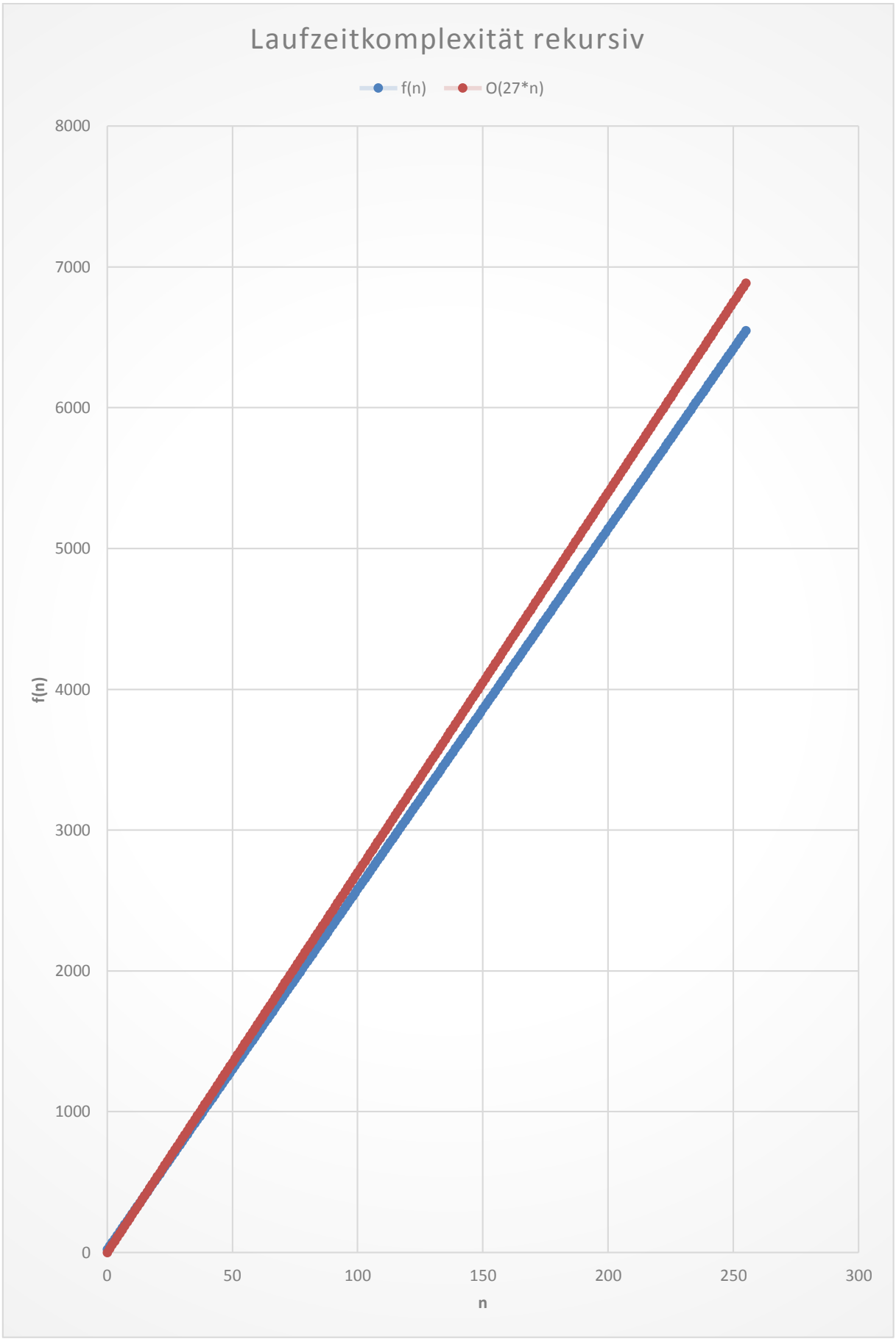
40	1044.8	26.12	1080
41	1070.4	26.10731707	1107
42	1096	26.0952381	1134
43	1121.6	26.08372093	1161
44	1147.2	26.07272727	1188
45	1172.8	26.06222222	1215
46	1198.4	26.05217391	1242
47	1224	26.04255319	1269
48	1249.6	26.03333333	1296
49	1275.2	26.0244898	1323
50	1300.8	26.016	1350
51	1326.4	26.00784314	1377
52	1352	26	1404
53	1377.6	25.99245283	1431
54	1403.2	25.98518519	1458
55	1428.8	25.97818182	1485
56	1454.4	25.97142857	1512
57	1480	25.96491228	1539
58	1505.6	25.95862069	1566
59	1531.2	25.95254237	1593
60	1556.8	25.94666667	1620
61	1582.4	25.94098361	1647
62	1608	25.93548387	1674
63	1633.6	25.93015873	1701
64	1659.2	25.925	1728
65	1684.8	25.92	1755
66	1710.4	25.91515152	1782
67	1736	25.91044776	1809
68	1761.6	25.90588235	1836
69	1787.2	25.90144928	1863
70	1812.8	25.89714286	1890
71	1838.4	25.89295775	1917
72	1864	25.88888889	1944
73	1889.6	25.88493151	1971
74	1915.2	25.88108108	1998
75	1940.8	25.87733333	2025
76	1966.4	25.87368421	2052
77	1992	25.87012987	2079
78	2017.6	25.86666667	2106
79	2043.2	25.86329114	2133
80	2068.8	25.86	2160
81	2094.4	25.85679012	2187
82	2120	25.85365854	2214
83	2145.6	25.85060241	2241

84	2171.2	25.84761905	2268
85	2196.8	25.84470588	2295
86	2222.4	25.84186047	2322
87	2248	25.83908046	2349
88	2273.6	25.83636364	2376
89	2299.2	25.83370787	2403
90	2324.8	25.83111111	2430
91	2350.4	25.82857143	2457
92	2376	25.82608696	2484
93	2401.6	25.82365591	2511
94	2427.2	25.8212766	2538
95	2452.8	25.81894737	2565
96	2478.4	25.81666667	2592
97	2504	25.81443299	2619
98	2529.6	25.8122449	2646
99	2555.2	25.81010101	2673
100	2580.8	25.808	2700
101	2606.4	25.80594059	2727
102	2632	25.80392157	2754
103	2657.6	25.80194175	2781
104	2683.2	25.8	2808
105	2708.8	25.79809524	2835
106	2734.4	25.79622642	2862
107	2760	25.79439252	2889
108	2785.6	25.79259259	2916
109	2811.2	25.79082569	2943
110	2836.8	25.78909091	2970
111	2862.4	25.78738739	2997
112	2888	25.78571429	3024
113	2913.6	25.7840708	3051
114	2939.2	25.78245614	3078
115	2964.8	25.78086957	3105
116	2990.4	25.77931034	3132
117	3016	25.77777778	3159
118	3041.6	25.77627119	3186
119	3067.2	25.77478992	3213
120	3092.8	25.77333333	3240
121	3118.4	25.77190083	3267
122	3144	25.7704918	3294
123	3169.6	25.76910569	3321
124	3195.2	25.76774194	3348
125	3220.8	25.7664	3375
126	3246.4	25.76507937	3402
127	3272	25.76377953	3429

128	3297.6	25.7625	3456
129	3323.2	25.76124031	3483
130	3348.8	25.76	3510
131	3374.4	25.75877863	3537
132	3400	25.75757576	3564
133	3425.6	25.75639098	3591
134	3451.2	25.75522388	3618
135	3476.8	25.75407407	3645
136	3502.4	25.75294118	3672
137	3528	25.75182482	3699
138	3553.6	25.75072464	3726
139	3579.2	25.74964029	3753
140	3604.8	25.74857143	3780
141	3630.4	25.74751773	3807
142	3656	25.74647887	3834
143	3681.6	25.74545455	3861
144	3707.2	25.74444444	3888
145	3732.8	25.74344828	3915
146	3758.4	25.74246575	3942
147	3784	25.7414966	3969
148	3809.6	25.74054054	3996
149	3835.2	25.73959732	4023
150	3860.8	25.73866667	4050
151	3886.4	25.73774834	4077
152	3912	25.73684211	4104
153	3937.6	25.73594771	4131
154	3963.2	25.73506494	4158
155	3988.8	25.73419355	4185
156	4014.4	25.73333333	4212
157	4040	25.73248408	4239
158	4065.6	25.73164557	4266
159	4091.2	25.73081761	4293
160	4116.8	25.73	4320
161	4142.4	25.72919255	4347
162	4168	25.72839506	4374
163	4193.6	25.72760736	4401
164	4219.2	25.72682927	4428
165	4244.8	25.72606061	4455
166	4270.4	25.7253012	4482
167	4296	25.7245509	4509
168	4321.6	25.72380952	4536
169	4347.2	25.72307692	4563
170	4372.8	25.72235294	4590
171	4398.4	25.72163743	4617

172	4424	25.72093023	4644
173	4449.6	25.72023121	4671
174	4475.2	25.71954023	4698
175	4500.8	25.71885714	4725
176	4526.4	25.71818182	4752
177	4552	25.71751412	4779
178	4577.6	25.71685393	4806
179	4603.2	25.71620112	4833
180	4628.8	25.71555556	4860
181	4654.4	25.71491713	4887
182	4680	25.71428571	4914
183	4705.6	25.7136612	4941
184	4731.2	25.71304348	4968
185	4756.8	25.71243243	4995
186	4782.4	25.71182796	5022
187	4808	25.71122995	5049
188	4833.6	25.7106383	5076
189	4859.2	25.71005291	5103
190	4884.8	25.70947368	5130
191	4910.4	25.70890052	5157
192	4936	25.70833333	5184
193	4961.6	25.70777202	5211
194	4987.2	25.70721649	5238
195	5012.8	25.70666667	5265
196	5038.4	25.70612245	5292
197	5064	25.70558376	5319
198	5089.6	25.70505051	5346
199	5115.2	25.70452261	5373
200	5140.8	25.704	5400
201	5166.4	25.70348259	5427
202	5192	25.7029703	5454
203	5217.6	25.70246305	5481
204	5243.2	25.70196078	5508
205	5268.8	25.70146341	5535
206	5294.4	25.70097087	5562
207	5320	25.70048309	5589
208	5345.6	25.7	5616
209	5371.2	25.69952153	5643
210	5396.8	25.69904762	5670
211	5422.4	25.6985782	5697
212	5448	25.69811321	5724
213	5473.6	25.69765258	5751
214	5499.2	25.69719626	5778
215	5524.8	25.69674419	5805

216	5550.4	25.6962963	5832
217	5576	25.69585253	5859
218	5601.6	25.69541284	5886
219	5627.2	25.69497717	5913
220	5652.8	25.69454545	5940
221	5678.4	25.69411765	5967
222	5704	25.69369369	5994
223	5729.6	25.69327354	6021
224	5755.2	25.69285714	6048
225	5780.8	25.69244444	6075
226	5806.4	25.6920354	6102
227	5832	25.69162996	6129
228	5857.6	25.69122807	6156
229	5883.2	25.69082969	6183
230	5908.8	25.69043478	6210
231	5934.4	25.69004329	6237
232	5960	25.68965517	6264
233	5985.6	25.68927039	6291
234	6011.2	25.68888889	6318
235	6036.8	25.68851064	6345
236	6062.4	25.68813559	6372
237	6088	25.68776371	6399
238	6113.6	25.68739496	6426
239	6139.2	25.68702929	6453
240	6164.8	25.68666667	6480
241	6190.4	25.68630705	6507
242	6216	25.68595041	6534
243	6241.6	25.68559671	6561
244	6267.2	25.6852459	6588
245	6292.8	25.68489796	6615
246	6318.4	25.68455285	6642
247	6344	25.68421053	6669
248	6369.6	25.68387097	6696
249	6395.2	25.68353414	6723
250	6420.8	25.6832	6750
251	6446.4	25.68286853	6777
252	6472	25.68253968	6804
253	6497.6	25.68221344	6831
254	6523.2	25.68188976	6858
255	6548.8	25.68156863	6885


Laufzeitkomplexität rekursiv
f(n)
O(27*n)
8000
7000
6000
5000
4000
3000
2000
1000
0
f(n)
0
50
100
150
200
250
300
n

4 Diskussion

Folgend ist die Diskussion der berechneten Laufzeitkomplexitäten angeführt, wobei diese gegeneinander Verglichen und analysiert werden.

Feinanalyse:

Die Feinanalysen der beiden Algorithmen erlauben die Berechnung der exakte Laufzeitkomplexität. Hierbei muss allerdings nicht nur die Zeichenlänge sondern auch der Zeicheninhalt bekannt sein, was in der Realität kaum möglich sein wird, da der Zeicheninhalt und auch die exakte Zeichenlänge nicht bekannt sein werden, sondern von der jeweiligen Applikation und dessen Nutzen abhängen. Es zeigt aber den Aufwand verschiedene bekannte Inputs verursachen.

Grobanalyse:

Die Grobanalysen haben keine Abhängigkeit mehr auf den Zeichenketteninhalt nur mehr auf die Zeichenlänge. Es wird angenommen, das jedes vierte Zeichen ein ungültiges Zeichen ist, daher ergab sich der Faktor $^{n}/_{4}$, wobei hier anzumerken ist, dass hierbei bei jeder Iteration bereits ein Aufwand von $^{n}/_{4}$ für den Zweig der die ungültigen Zeichen in der Zeichenkette behandelt, hinzugezählt wird, was eine Ungenauigkeit mit sich bringt. Aber dennoch kann die Laufzeitkomplexität gut abgeschätzt werden. Mit den aufgestellten Funktionen kann man ohne Kenntnis des konkreten Inhalts bereits die Laufzeitkomplexität bzw. den Aufwand des Algorithmus abschätzen, wobei man sich bewusst sein muss, das jedes vierte Zeichen als ungültiges Zeichen innerhalb der Berechnung angesehen wird. Sollte sich die Zeichenkette bei großen n stark von der angenommen Zeichenkette unterscheiden, also viel mehr oder viel weniger als angenommene ungültige Zeichen in der Zeichenkette vorkommen, so wird auch die Ungenauigkeit des berechneten Aufwands entweder in den positiven oder negativen Teil abdriften.

Asymptotische Laufzeit:

Die asymptotische Laufzeitkomplexität gibt die Steigerung des Aufwandes in Abhängigkeit der Länge der Zeichenkette an, die niemals die durch $O(n)$ definierte Steigerung überschreiten wird. $O(n)$ definiert also die obere Schranke, die niemals überschritten wird. Hierbei ist aber anzumerken, dass diese Schranke erst ab einen gewiesen n_0 gilt. Davor kann keine Aussage über den Aufwand gemacht werden. Diese Art der Laufzeitkomplexität ist zwar ungenauer, aber jedoch zeigt sie gut die Aufwandssteigerung eines Algorithmus, denn meistens ist es nicht erforderlich den exakten Aufwand für einen bestimmten zu kennen, aber die Art wie sich der Algorithmus bei steigender Anzahl des Inputs verhält.

Vergleich iterativ vs. rekursiv:

Bei den beiden Analysen zeigte sich das der rekursive Algorithmus eine weitaus schlechtere Laufzeitkomplexität aufweist, was vor allem darin begründet ist, dass der rekursive Algorithmus viele Prozeduren Aufrufe hat, die mit einer Gewichtung von 18 einen sehr hohen Aufwand haben (Stack, Aktivierungssätze). Daher sollten rekursive Algorithmen nur dort verwendet werden, wo sie wirklich Sinn machen und man mit den Aufwand leben kann.

Schon bei $n = 0$ (Grobanalyse) ist zu sehen, dass der iterative Algorithmus mit einem Funktionswert von 5,2 weitaus weniger Aufwand produziert als der rekursive Algorithmus mit 20,8. Der rekursive Algorithmus produziert also um den Faktor 4 mehr Aufwand und das schon bei einer Zeichenlänge von 0.