DSA Blatt 01

Leonard Oertelt 1276156
Julian Opitz 1302082

Aufage 1:

a)

Zur Auswertung des sehr kurzen Array wurde der Algorithmus zum Finden der Teilsumme

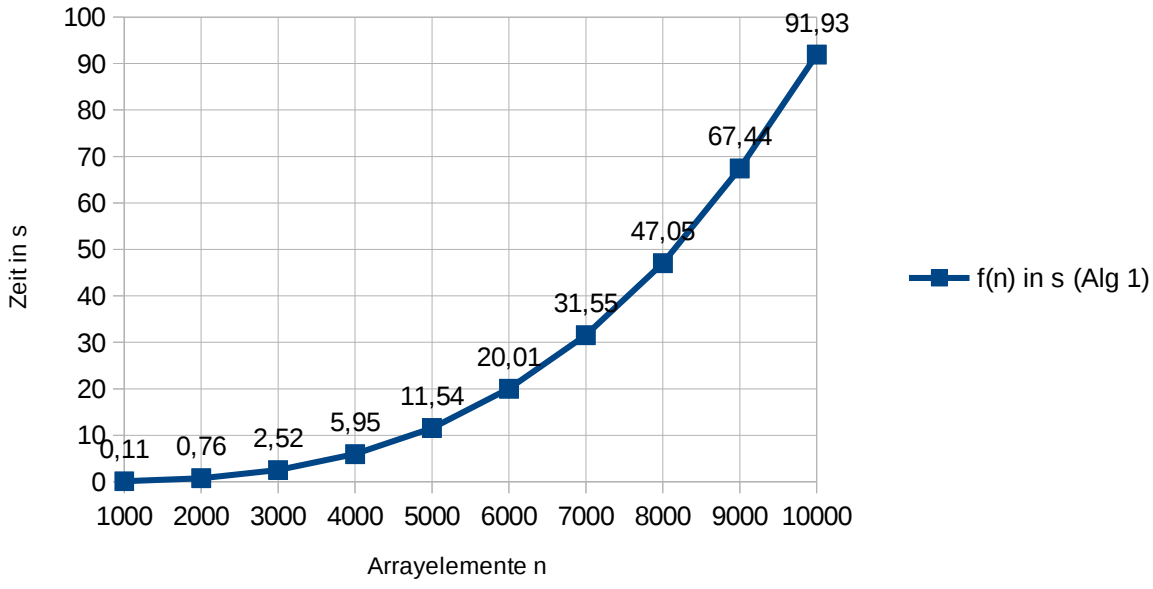
1 Milliarde mal ausgeführt. Die gemessene Zeit dividiert durch die Anzahl der Wiederholungen ergab:

$t \approx 2.0324 * 10^{-7} s$ (ca. 203 ns)

b)

Die Anzahl der Wiederholungen wurde so gewählt, sodass der verwendete Computer jeweils ungefähr eine volle Minute am Rechnen war. Bei einer großen Anzahl von Arrayelementen war es nicht notwendig Wiederholungen zu tätigen da diese bereits sehr viel Rechenzeit in Anspruch nahmen.

Anzahl n Elemente	f(n) in s (Alg 1)
1000	0,11
2000	0,76
3000	2,52
4000	5,95
5000	11,54
6000	20,01
7000	31,55
8000	47,05
9000	67,44
10000	91,93



c)

Mithilfe eines Grafikfähigen Taschenrechners wurde auf die gemessenen Werte eine Power-Regression durchgeführt.

Aus dieser ergibt sich in etwa ein Polynom 3. Grades:

n – Anzahl Arrayelemente

$f(n) = 1.586 * 10^{-10} * n^{2.938}$

Mithilfe der aus der Vorlesung bekannten Laufzeitanalyse (zählen der einfachen Operationen), angewendet auf den Java-Code, ergab sich in unserem Fall folgender Zusammenhang:

$f(n) = 4.5n^3 + 2n^2 + 13n + 7$

Um in etwa die gleichen Zeiten zu bekommen muss ein Faktor verwendet werden, der mit der Funktion multipliziert wird.

In unserem Fall beträgt dieser ca. $2 * 10^{-11}$.:

$t(n) = 2 * 10^{-11} * (4.5n^3 + 2n^2 + 13n + 7)$

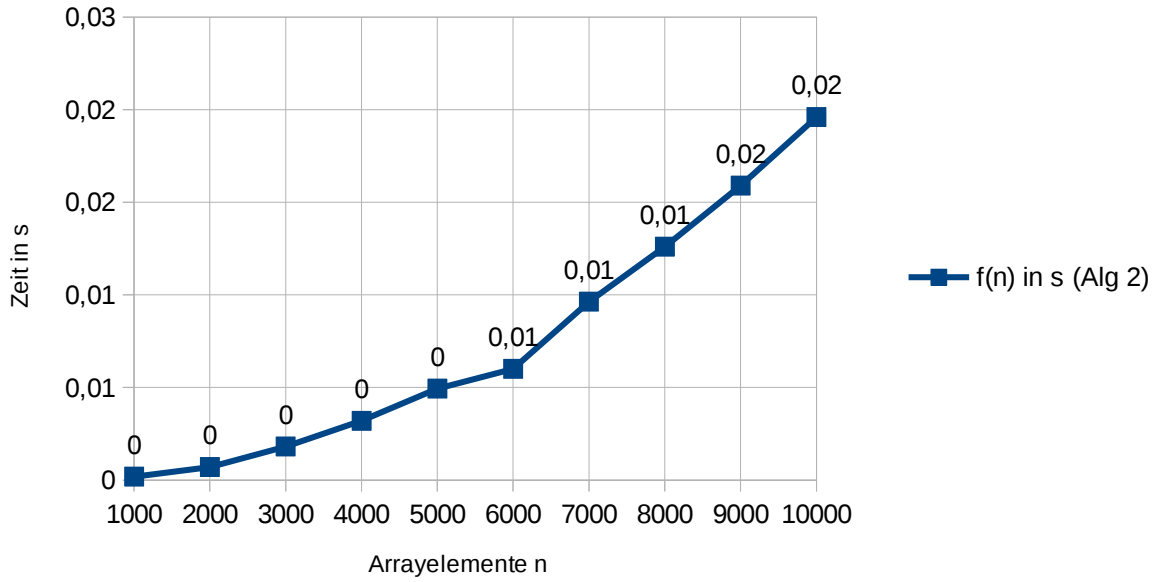
z.B.:

$t(8000) \approx 46.08$ s

Der Faktor ist von Computer zu Computer unterschiedlich.

d)

Anzahl n Elemente	f(n) in s (Alg 2)
1000	0,000184
2000	0,000699
3000	0,00181
4000	0,0032
5000	0,00494
6000	0,006
7000	0,00963
8000	0,0126
9000	0,0159
10000	0,0196



Vorgehen wie bei c)

$f(n) = 1.5 * 10^{-10} * n^{2.03}$

Durch Algorithmusanalyse:

$f(n) = 9n^2 + 13n + 7$

Polynom 2. Grades.

Der Faktor wurde ebenfalls aus c) übernommen:

$t(n) = 2 * 10^{-11} * (9n^2 + 13n + 7)$

z.B.:
t(7000) = 0.0088 s

Im Vergleich zum ersten Agorithmus ist der zweite, verbesserte deutlich schneller, allerdings erhöht sich die Laufzeit immer noch quadratisch.

Aufgabe 2:

a)

1 µs entspricht 10^{-6} s, folglich kann der Rechner folgende Anzahl Anweisungen pro Zeiteinheit ausführen:

t	1s	1h	1Monat	1 Jahrhundert
x	1000000	3600000000	2678400000000	3,2140800E+015

Die Rechnungen:

1. $f(n) = \log_2(n) = x$ umgeformt zu $f(x) = 2^x = n$
2. $f(n) = n = x$ umgeformt zu $f(x) = x = n$
3. $f(n) = n \log_2(n) = x$ lässt sich umformen zu $f(n) = \log_2(n^n) = x$

   Weiter umgeformt ergibt sich $f(x) = 2^x = n^n$

   Dies lässt sich aber nicht ausrechnen, nur näherungsweise bestimmen.

   Durch Recherche im Internet stießen wir auf die Lambertsche W-Funktion

   die wir in der Form

   $$n = e^{w(x * \log(2))}$$

   in Verbindung mit Wolfram Alpha benutzt haben um die entsprechenden Werte für n ausrechnen zu können.
4. $f(n) = n^2 = x$ umgeformt zu $f(x) = \sqrt{x} = n$
5. $f(n) = 2^n = x$ umgeformt zu $f(x) = \log_2(x) = n$

Im Folgenden wurde in die jeweiligen Funktionen das x eingesetzt und ausgerechnet.

Fast alle Lösungen sind dabei Approximierungen, da die Zahlen teilweise sehr groß sind.

f(n)	n bei t=1 s	n bei t= 1h	n bei t = 1Monat	n bei T = 1 Jahrhun
$\log_2(n)$	ca. $10^{300.000}$	2^(36.10^8)	2^(2.678*10^12)	2^(3.21408*10^15)
n	ca. 10^6	36.10^8	2.678*10^12	3.21408*10^15
$n \log_2(n)$	ca. 62746	1.334 * 10^8	7.42*10^10	6.989*10^13
n^2	1000	60000	1636459.6	56692859.7
2^n	19.93	31.75	41.28	51.51

b)

Statt 10^6 Anweisungen pro Sekunde sind es mit dem neuen Super-Rechner 10^9 Anweisungen pro Sekunde.

t	1s	1h	1Monat	1 Jahrhundert
x	1,0E+009	3,600E+012	2678400000000000	3,214080000E+018

Wie bei Aufgabenteil a) wird eingesetzt:

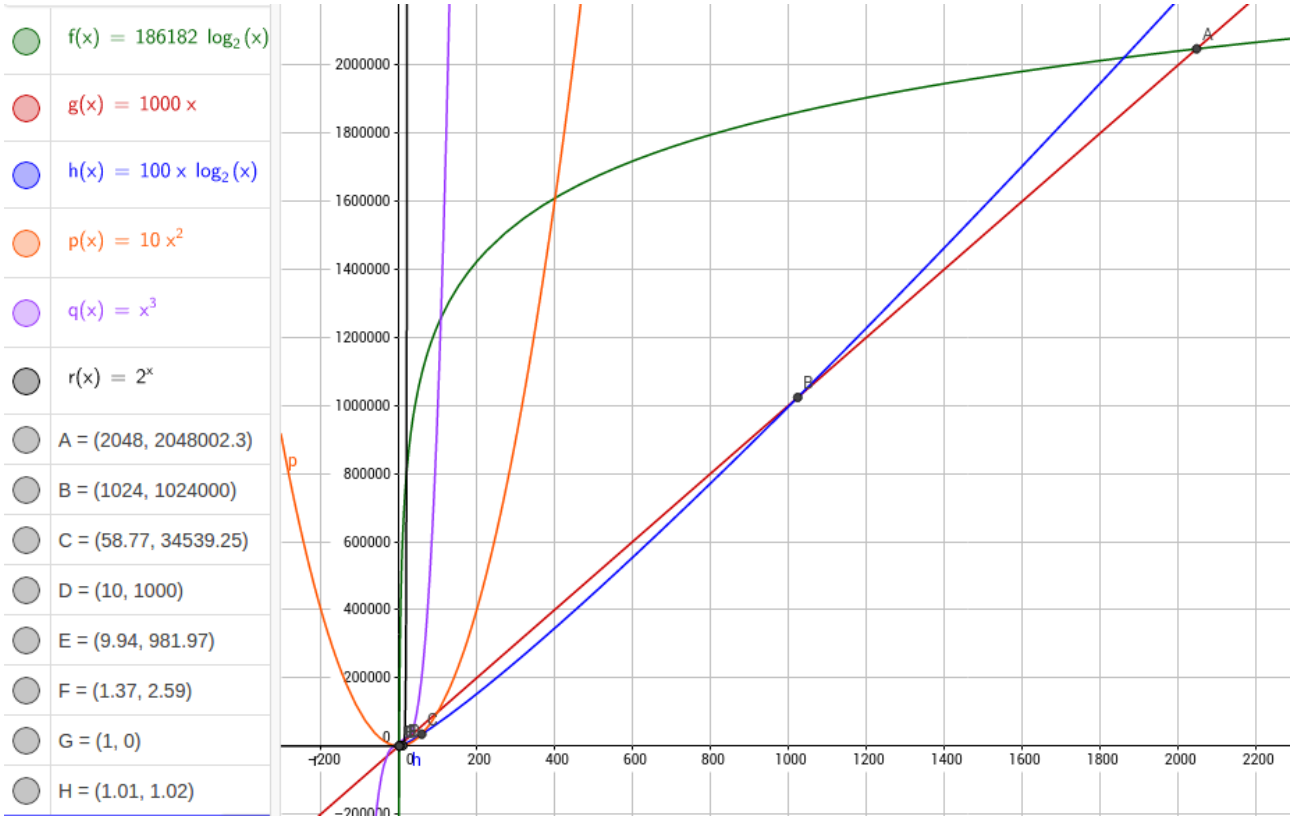
f(n)	n bei t=1 s	n bei t= 1h	n bei t = 1Monat	n bei T = 1 Jahrhundert
n^2	31622.8	1897365.6	51749396.1	1792785542
2^n	29.9	41.71	51.25	61.48

Im Vergleich zu dem Rechner mit 10^6 Anweisungen pro Sekunde

erhöht sich für $f(n) = n^2$ die Problemgröße um den Faktor 31.6 und für $f(n) = 2^n$ erhöht sich die Problemgröße jeweils um etwa 10.

c)

Die Aufgabe wurde mithilfe von Geogebra (Matheplugin für Chromium/Chrome) gelöst.



Alle Funktionen wurden grafisch dargestellt und danach jeweils die Schnittpunkte berechnet.

Da wir annehmen, dass $n \in \mathbb{N}$ ist ergibt sich durch Ablesen:

Für Eingaben der Größe n=1 bis n=10 sollte man 2^n benutzen,
für Eingaben der Größe n=11 bis n=58 sollte man $10 * n^2$ benutzen,
für Eingaben der Größe n=59 bis n=1024 sollte man $100n \log_2(n)$ benutzen,
für Eingaben der Größe n=1025 bis n=2048 sollte man $1000n$ benutzen und für
alle Eingaben der Größe $n \geq 2049$ sollte man $186182 \log_2(n)$ benutzen.

Aufgabe 3:

a)

$100n \in O(n^2)$ mit $f(n) = n^2$ und $g(n) = 100n$

daraus folgt:

1. $100n \leq cn^2$
2. nach n umstellen:

   $100n \leq cn^2$ | :n

   $100 \leq cn$ | :c

   Da n laut Definition mindestens 1 ist (positiv) wird die Richtung des Kleinerzeichens nicht verändert.

   $$\frac{100}{c} \leq n$$

3. Testen ob Behauptung erfüllbar ist:

   $t(n) = n$

   $v(c) = \frac{100}{c}$

   $v(c) \leq t(n)$

   Behauptung ist erfüllbar.

4. c wählen, so dass Ungleichung erfüllt ist:

   $c = 1$

   $$\frac{100}{1} \leq n$$

   $100 \leq n$

5. n_0 finden, so dass Ungleichung für alle $n > n_0$ erfüllt ist.

   $n \geq 100$

   bereits nach n aufgelöst

   $n \geq 100$ gilt für alle $n > 100$, also erfüllt $n_0 = 100$ die Ungleichung

6. Abschluss des Beweises:

   Für alle $n>n_0$ gilt $g(n) \leq c*f(n)$

   mit $n = 100$ und $c = 1$

   Für alle $n>100$ gilt $100n \leq n^2$

   Also gilt für $g(n) = 100n$ und $f(n) = n^2$ die Aussage $g \in O(f)$

- q.e.d.

b)

$\frac{1}{10^5} \cdot n^4 \in O(n^3)$ mit $f(n) = n^3$ und $g(n) = \frac{1}{10^5} \cdot n^4$

daraus folgt:

1. $\frac{1}{10^5} \cdot n^4 \leq cn^3$
2. nach n umstellen:

   $\frac{1}{10^5} \cdot n^4 \leq cn^3 \mid * 10^5$

   $n^4 \leq cn^3 * 10^5 \mid : n^3$

   $n \leq 10^5 * c$
3. Testen ob Behauptung erfüllbar ist:

   $t(n) = n$

   $v(c) = 10^5 * c$

   $t(n) \leq v(c)$

Da t(n) beliebig groß werden kann und v(c) konstant bleibt, ist die Behauptung unmöglich erfüllbar.
Ende des Beweisversuchs.

c)

$n \log_2(n) \in O(n^2)$ mit $f(n) = n^2$ und $g(n) = n \log_2(n)$

daraus folgt:

1. $\log_2(n) \leq cn^2$
2. nach n umstellen:

   $n \log_2(n) \leq cn^2$ | :n

   $\log_2(n) \leq cn$

   $n \leq 2^{cn}$

   alternativ:

   $\log_2(n) \leq cn$ | :c

   $\frac{1}{c} \log_2(n) \leq n$

   $\log_2(n^{1/c}) \leq n$

   $n^{1/c} \leq 2^n$

   $\sqrt[c]{n} \leq 2^n$

   Nach n umstellen lässt sich nicht ohne weiteres bewerkstelligen.

3. Testen ob Behauptung erfüllbar ist:

   $t(n) = 2^n$

   $v(c) = \sqrt[c]{n}$

   $v(c) \leq t(n)$

   Behauptung ist erfüllbar.
4. c wählen, so dass Ungleichung erfüllt ist:

   $c = 1$

   $\sqrt[1]{n} \leq 2^n$

   $n \leq 2^n$

   durch schlaues hingucken sieht man, dass bereits mit $c = 1$ und $n = 1$ die Ungleichung erfüllt ist.
5. n_0 finden, so dass Ungleichung für alle $n>n_0$ erfüllt ist.

   Die Ungleichung lässt sich nicht nach n auflösen.

   $2^n \geq n$ gilt für alle $n > 1$, also erfüllt $n_0 = 1$ die Ungleichung
6. Abschluss des Beweises:

   Für alle $n>n_0$ gilt $g(n) \leq c*f(n)$ mit $n = 1$ und $c = 1$

   Für alle $n>1$ gilt $n \log2(n) \leq n^2$

Also gilt für $g(n) = n \log2(n)$ und $f(n) = n^2$ die Aussage $g \in O(f)$ - q.e.d.