Dokumentation zum Aufgabenblatt a03

Implementiert wurden die vier Algorithmen wie in der Vorlesung besprochen. Der einfachste, aber auch langsamste, Algorithmus, wo beide Schleifen bis zur uebergebenen Problemgroesse laufen, der etwas schnellere, wo die innere Schleife nur noch bis zum Laufindex der aeusseren Schleife laeuft, der noch weiter verbesserte, wo die aeussere Schleife nur noch ungerade Zahlen beruecksichtigt und die innere nur noch bis zur Wurzel des Laufindex der aeusseren laeuft und das Sieb des Eratosthenes, wo alle vielfachen einer gefundenen Primzahl gestrichen werden.
Weiterhin wurde eine Methode implementiert, die fuer eine uebergebene Zahl prueft, ob diese eine Primzahl ist.

Die geforderte Auswertung und Darstellung der Ergebnisse in Form einer Tabelle und Diagrammen befindet sich im Anschluss an diese Dokumentation.

Martin Slowikowski
Matrikelnummer: 199 91 66

Tell Mueller-Pettenpohl
Matrikelnummer: 198 99 82

Teamname: Tugend und Laster

Quellenangaben

TIB3-AD-skript.pdf
Letzte Änderung 22.03.2011

Auswertung und Darstellung

Tabellarische Darstellung

Um die Tabelle nicht zu ausufernd zu gestalten, wurden die Operatioenn in einer Problemschrittweite von 25 geteilt.
Mit steigender Problemgroeße ist auch ein steigender Aufwand zu erkennen. Man kann bereits anhand der Tabelle das unterschiedliche Steigungsverhalten des Aufwands der einzelnen Algorithmen erkennen.

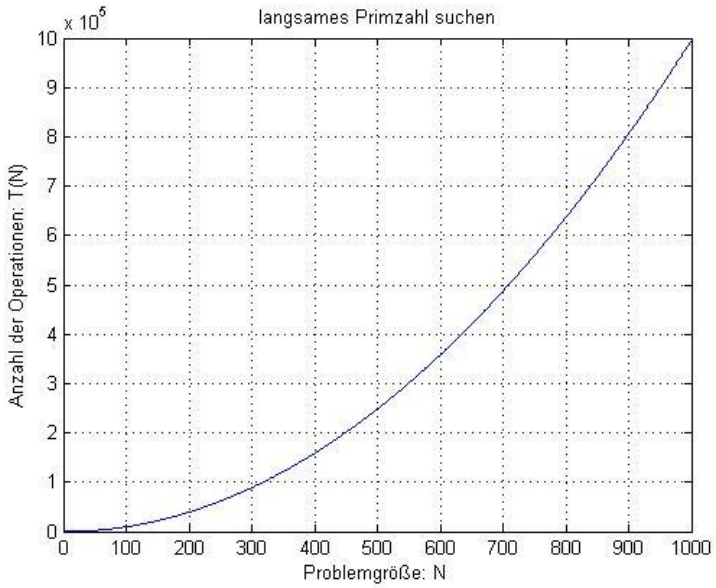
Bei der Primzahlpruefung ist der Aufwand sprunghaft, da in einzelnen Faellen fruehzeitig eine Primzahl erkannt wird (bspw. gerade Zahlen).

Anzahl der Durchläufe / Problemgröße (N)	langsames Primzahl suchen	schnelleres Primzahl suchen	schnelles Primzahl suchen	Sieb des Eratosthenes	Anzahl der Durchläufe / Problemgröße (N)	Primzahl-eigenschaft feststellen
0	0	0	0	0	0	0
25	529	99	21	18	23	3
50	2304	349	65	52	47	5
75	5329	750	126	81	75	2
100	9604	1132	185	111	91	6
125	15129	1696	260	150	125	4
150	21904	2413	332	181	143	10
175	29929	3257	429	225	173	12
200	39204	4423	516	259	200	1
225	49729	4916	605	293	223	13
250	61504	6117	703	326	250	1
275	74529	7459	807	360	271	15
300	88804	8647	906	409	300	1
325	104329	9944	1026	445	323	16
350	121104	11350	1115	478	350	1
375	139129	12847	1242	533	375	2
400	158404	14445	1358	570	397	18
425	178929	16140	1476	605	425	4
450	200704	18377	1593	641	447	2
475	223729	20268	1731	678	475	4
500	248004	22278	1843	715	500	1
525	273529	24371	1980	750	523	21
550	300304	25541	2096	809	550	1
575	328329	27854	2234	848	575	4
600	357604	30264	2353	885	599	23
625	388129	33354	2521	923	625	4

Anzahl der Durchläufe Problemgröße (N)	langsames Primzahl suchen	schnelleres Primzahl suchen	schnelles Primzahl suchen	Sieb des Eratosthenes	Anzahl der Durchläufe Problemgröße (N)	Primzahl-eigenschaft feststellen
650	419904	35968	2655	960	667	22
675	452929	38667	2809	997	675	2
700	487204	40776	2934	1034	700	1
725	522729	42976	3075	1072	723	3
750	559504	45970	3220	1110	750	1
775	597529	49814	3386	1147	775	4
800	636804	51484	3502	1185	799	16
825	677329	54797	3670	1221	825	2
850	719104	57356	3807	1287	850	1
875	762129	60849	3977	1327	873	2
900	806404	64449	4129	1364	900	1
925	851929	67249	4300	1404	925	4
950	898704	71063	4460	1440	947	29
975	946729	74019	4619	1509	975	2
1000	996004	78021	4787	1547	1000	1

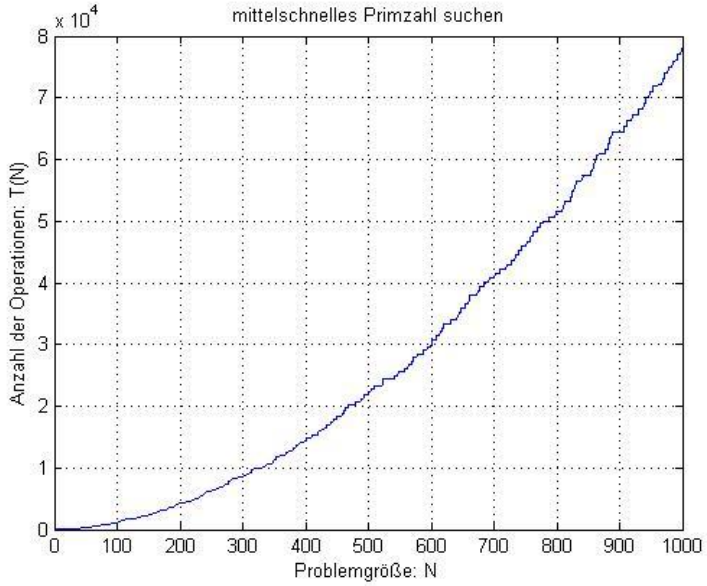
Grafische Darstellung

Im Anschluss an die tabellarische Übersicht folgen nun die grafischen Auswertungen der einzelnen Algorithmen.

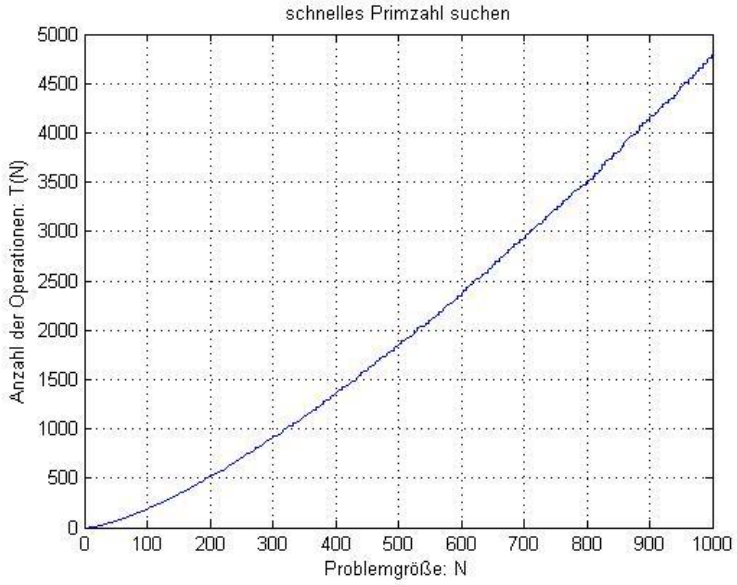


Bei diesem Algorithmus ist bei einer Problemgröße von 1000 ein quadratischer Aufwand zu erkennen, d.h. $O(N^2)$.

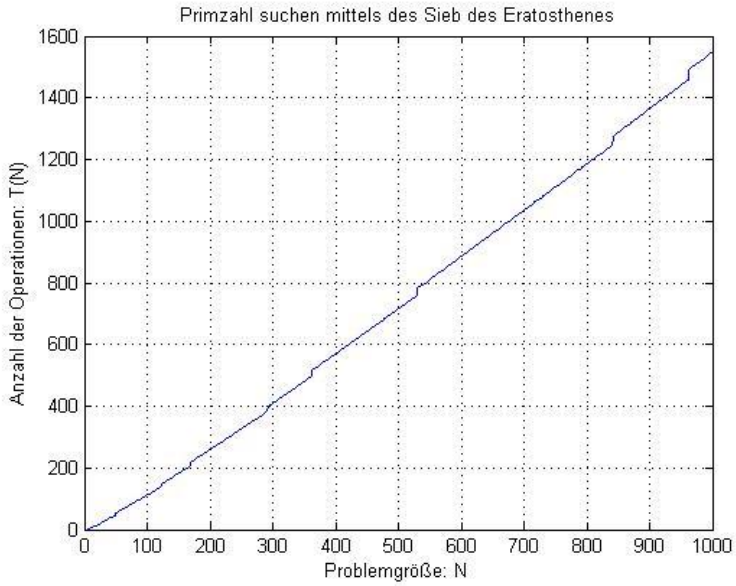
Diese Implementierung ist die einfachste und langsamste um Primzahlen zu ermitteln. Es ist ein „brute-force“-Algorithmus, beide Schleifen werden bis zur Problemgröße durchlaufen.



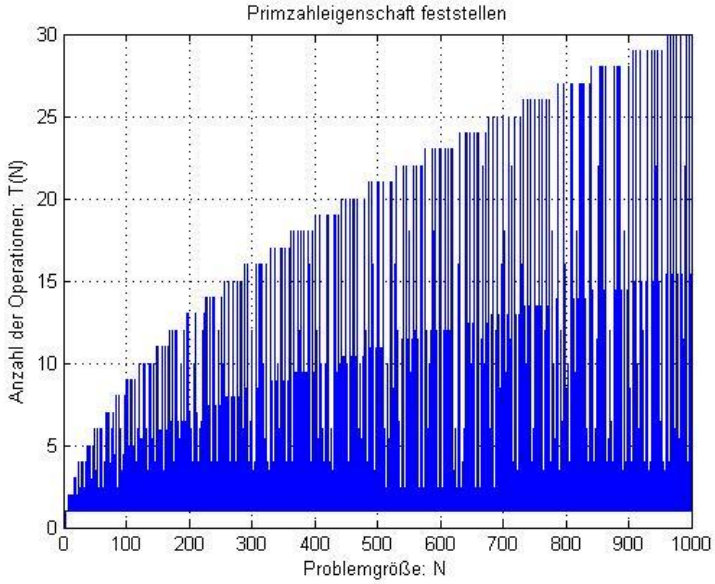
Bei diesem Algorithmus ist bei einer Problemgröße von 1000 ebenfalls ein quadratischer Aufwand zu erkennen, d.h. $O(N^2)$.
Der Algorithmus arbeitet schneller als der vorherige, da die innere Schleife nur noch bis zum Laufindex der äußeren Schleife läuft und abbricht, falls eine Primzahl gefunden wurde.



Bei diesem Algorithmus ist weiterhin bei einer Problemgröße von 1000 ein quadratischer Aufwand zu erkennen, d.h. $O(N^2)$. Allerdings mit einer wesentlich geringeren Steigung, da die innere Schleife nur noch bis zur Wurzel des Laufindex der äußeren läuft und nur ungerade Indizes in der äußeren Schleife geprüft werden.



Bei diesem Algorithmus ist bei einer Problemgröße von 1000 ein superlinearer Aufwand zu erkennen, d.h. O(N log N). Es werden alle Vielfachen der gefundenen Primzahl im Array als nicht-Primzahl (false) gekennzeichnet. Die Innere Schleife läuft bis zum Produkt der beiden Laufindizes, die äußere Schleife bis zur Wurzel der Problemgröße.



Bei diesem Algorithmus ist bei einer Problemgröße von 1000 ist ein der Wurzelfunktion ähnlicher Aufwand zu erkennen, d.h. O(Wurzel von N). In der Grafik ist zu erkennen, dass eine Obergrenze (O(N)) und Untergrenze (Omega(N)) existiert.