Primzahlen, Primzahltests, Primfaktorzerlegung Ac 2013-2018

Eine natürliche Zahl heißt **Primzahl**, wenn sie durch genau 2 Zahlen teilbar ist, nämlich durch 1 und durch sich selbst. Demnach ist 1 keine Primzahl ! Ist n Primzahl, so sagt man auch „**n ist prim**“.

Es gibt **unendlich viele Primzahlen**, was sich indirekt beweisen lässt.
Man nimmt an, dass es k Primzahlen $p_1, p_2, p_3, \dots, p_k$ gibt, also endlich viele.
Die Zahl $n = p_1 \cdot p_2 \cdot p_3 \cdot \dots \cdot p_k + 1$ ist jedoch größer als alle vorher bekannten Primzahlen und durch keine davon teilbar, weil immer der Rest 1 bleibt. Also muss entweder n eine noch größere Primzahl sein als die vorher als bekannt angenommenen oder n ist durch mindestens eine größere als die k bekannten Primzahlen teilbar. Die Annahme war also falsch ! Es muss daher unendlich viele Primzahlen geben.

Die ersten Primzahlen sind:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241
251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383
389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523
541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673
677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829
839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997
1009 1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093 1097 1103 1109
1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213 1217 1223 1229 1231 1237 1249
1259 1277 1279 1283 1289 1291 1297 1301 1303 1307 1319 1321 1327 1361 1367 1373 1381 1399
1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471 1481 1483 1487 1489 1493 1499 1511
1523 1531 1543 1549 1553 1559 1567 1571 1579 1583 1597 1601 1607 1609 1613 1619 1621 1627
1637 1657 1663 1667 1669 1693 1697 1699 1709 1721 1723 1733 1741 1747 1753 1759 1777 1783
1787 1789 1801 1811 1823 1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931
1933 1949 1951 1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063 2069
2081 2083 2087 2089 2099 2111 2113 2129 2131 2137 2141 2143 2153 2161 2179 2203 2207 2213
2221 2237 2239 2243 2251 2267 2269 2273 2281 2287 2293 2297 2309 2311 2333 2339 2341 2347
2351 2357 2371 2377 2381 2383 2389 2393 2399 2411 2417 2423 2437 2441 2447 2459 2467 2473
2477 2503 2521 2531 2539 2543 2549 2551 2557 2579 2591 2593 2609 2617 2621 2633 2647 2657
2659 2663 2671 2677 2683 2687 2689 2693 2699 2707 2711 2713 2719 2729 2731 2741 2749 2753
2767 2777 2789 2791 2797 2801 2803 2819 2833 2837 2843 2851 2857 2861 2879 2887 2897 2903
2909 2917 2927 2939 2953 2957 2963 2969 2971 2999 3001 3011 3019 3023 3037 3041 3049 3061
3067 3079 3083 3089 3109 3119 3121 3137 3163 3167 3169 3181 3187 3191 3203 3209 3217 3221
3229 3251 3253 3257 3259 3271 3299 3301 3307 3313 3319 3323 3329 3331 3343 3347 3359 3361
3371 3373 3389 3391 3407 3413 3433 3449 3457 3461 3463 3467 3469 3491 3499 3511 3517 3527
3529 3533 3539 3541 3547 3557 3559 3571 3581 3583 3593 3607 3613 3617 3623 3631 3637 3643
3659 3671 3673 3677 3691 3697 3701 3709 3719 3727 3733 3739 3761 3767 3769 3779 3793 3797
3803 3821 3823 3833 3847 3851 3853 3863 3877 3881 3889 3907 3911 3917 3919 3923 3929 3931
3943 3947 3967 3989 4001 4003 4007 4013 4019 4021 4027 4049 4051 4057 4073 4079 4091 4093
4099 4111 4127 4129 4133 4139 4153 4157 4159 4177 4201 4211 4217 4219 4229 4231 4241 4243
4253 4259 4261 4271 4273 4283 4289 4297 4327 4337 4339 4349 4357 4363 4373 4391 4397 4409
4421 4423 4441 4447 4451 4457 4463 4481 4483 4493 4507 4513 4517 4519 4523 4547 4549 4561
4567 4583 4591 4597 4603 4621 4637 4639 4643 4649 4651 4657 4663 4673 4679 4691 4703 4721
4723 4729 4733 4751 4759 4783 4787 4789 4793 4799 4801 4813 4817 4831 4861 4871 4877 4889
4903 4909 4919 4931 4933 4937 4943 4951 4957 4967 4969 4973 4987 4993 4999
```

Ist eine natürliche Zahl **nicht prim**, so nennt man sie **zerlegbar** bzw.
zusammengesetzt (engl.: composite) .

Jede zusammengesetzte Zahl n lässt sich als Produkt von Primzahlen eindeutig darstellen, es gilt also
$n = p_1^{t1} \cdot p_2^{t2} \cdot p_3^{t3} \cdot \dots \cdot p_k^{tk}$ ($t_i \in \mathbb{N}$) Beispiel: $420 = 2^2 \cdot 3 \cdot 5 \cdot 7$

Es gibt eine Reihe von Algorithmen, mit denen eine natürliche Zahl n auf „**Primalität“** geprüft werden kann (Primzahltests).

Auch für eine eventuell existierende Primfaktorzerlegung von n gibt es Algorithmen.

Primzahltests:

Die einfachste (sehr langsame) Methode ist die sog. „Holzhammermethode“ bzw. brute-force-Methode namens **Probedivision** (trial division) :
Die zu testende Zahl n wird durch t = 2; 3; 4; 5; 6; …; n-1 geteilt .
Geht die Division bei einem der Teiler t auf (Rest=0), so ist n keine Primzahl und man kann die Untersuchung beenden. Andernfalls ist n Primzahl.

Verbesserung1:
Es genügt, ab 3 lediglich ungerade Teiler zu betrachten, denn wenn 2 schon kein Teiler war, können 4; 6; 8; … auch keine Teiler sein.

Verbesserung2:
Es genügt, lediglich bis zur Quadratwurzel von n zu teilen, denn falls t ein Teiler von n ist, so ist auch n/t ein Teiler von n.
Beispiel: p=1000 10 teilt 1000 und 100 teilt 1000 ; es genügt also, bis 31 zu teilen .

Verbesserung 3:
Je Primzahl größer als 3 ist von der Form 6i-1 oder 6i+1 (ab i = 1). Daher braucht man lediglich t=2; t=3; t=5; zu testen und dann den Teiler t abwechselnd um 2 bzw. 4 erhöhen.

Hierzu folgende JAVA-Methode:

```
public static boolean istPrimBruteForce(long n) {
	if (n < 7)
		return (n == 2 || n == 3 || n == 5);
	if (n % 2 == 0 || n % 3 == 0 || n % 5 == 0)
		return false;
	long max = (long) Math.sqrt(n), teiler = 7, increment = 4;
	while (teiler <= max) {
		if (n % teiler == 0)
			return false;
		else {
			teiler += increment;
			increment = 6 - increment;
		}
	}
	return true;
}
```

Probabilistische Primzahltests:

Dies sind Tests, bei denen Wahrscheinlichkeiten eine Rolle spielen.

Erkennt ein solcher Test eine Zahl als zusammengesetzt, so ist sie mit Sicherheit zusammengesetzt.
Erkennt der Test eine Zahl als Primzahl, so irrt er mit einer gewissen Wahrscheinlichkeit.

Primzahltest von **Fermat**:

Grundlage ist der „kleine Satz von Fermat“:

Wenn n eine Primzahl ist, dann gilt für alle a aus der Menge { 1 ; 2; … ; n-1 }: a^{n-1} mod n = 1

Dies bedeutet: a^{n-1} lässt bei Division durch n den ganzzahligen Rest 1 .

Beispiel: n = 7: $1^6 \bmod 7 = 1 \bmod 7 = 1$ $2^6 \bmod 7 = 64 \bmod 7 = 1$ $3^6 \bmod 7 = 729 \bmod 7 = 1$
$4^6 \bmod 7 = 4096 \bmod 7 = 1$ $5^6 \bmod 7 = 15625 \bmod 7 = 1$ $6^6 \bmod 7 = 46656 \bmod 7 = 1$

Der Satz ist nur von links nach rechts gültig (wenn... dann...) und daher nicht umkehrbar , d.h. :
Ist das Ergebnis von $\mathbf{a^{n-1} \bmod n}$ ungleich 1, war n auf alle Fälle keine Primzahl.
Ist das Ergebnis von $\mathbf{a^{n-1} \bmod n}$ gleich 1, könnte n eine Primzahl sein. Dies ist aber nicht sicher !

Gegenbeispiel 1: **n=42** :
Für a=2 gilt: $2^{41} = 2199023255552$; $2199023255552 \bmod 42 = 32$; also 42 nicht prim.

Gegenbeispiel 2: **n=341** (im folgenden wird mit dem unten erläuterten PowerMod gerechnet) :
Für a=2 gilt: $2^{340} \bmod 341 = 1$.
Für a = 3 gilt aber: $3^{340} \bmod 341 = 56$; also 341 nicht prim.
Man nennt die Zahl 341 **Fermatsche Pseudoprimzahl** zur Basis 2 und
die Zahl 2 einen **Lügner** für die Primalität von 341 .
Anmerkung: Unter anderem sind auch 561 und 645 Fermatsche Pseudoprimzahlen zur Basis 2.

Es gibt sogar Zahlen, die für alle Basen a aus { 1; 2; … ; n-1 } Pseudoprimzahlen sind. Das Ergebnis von $\mathbf{a^{n-1} \bmod n}$ ist bei diesen Zahlen immer 1, obwohl es sich **nicht** um eine Primzahl handelt.
Man nennt diese Zahlen **Carmichael-Zahlen** (R.D.Carmichael entdeckte sie 1910).
Das kleinste Beispiel für Carmichael-Zahlen ist die Zahl 561.
Bei den Carmichael-Zahlen versagt der Primzahltest von Fermat völlig !!

Beispielrechnungen für n=561 (Carmichael-Zahl):
Für a=2: $2^{560} \bmod 561 = 1$ Für a=3: $3^{560} \bmod 561 = 1$ Für a=4: $4^{560} \bmod 561 = 1$
Für a=5: $5^{560} \bmod 561 = 1$ Für a=500: $500^{560} \bmod 561 = 1$ Für a=560: $4^{560} \bmod 561 = 1$
Jedoch ist 561 keine (!) Primzahl, denn $561 = 3 \cdot 11 \cdot 17$

Seit 1992 weiß man, dass es unendlich viele Carmichael-Zahlen gibt, sie sind aber dünn gesät.
Die ersten Carmichael-Zahlen sind :
561 1105 1729 2465 2821 6601 8911 10585 ...

PowerMod :

Die Berechnung von von $\mathbf{a^{n-1} \bmod n}$ kann auch ohne riesige Zwischenergebnisse vorgenommen werden, und zwar mit der Modulo-Division **PowerMod** . Diese basiert auf der folgenden Identität:

$$a^n \bmod p = \begin{cases} (a^{n/2} \bmod p)^2 \bmod p \ ; \ n \ gerade \\ (a^{n-1} \bmod p) \cdot a \bmod p \ ; \ n \ ungerade \end{cases}$$

Beispiel: $3^9 \bmod 17 = (3^8 \bmod 17) \cdot 3 \bmod 17 = ((3^4 \bmod 17)^2 \bmod 17) \cdot 3 \bmod 17 =$
$(((3^2 \bmod 17)^2 \bmod 17)^2 \bmod 17) \cdot 3 \bmod 17 =$
$(((9 \bmod 17)^2 \bmod 17)^2 \bmod 17) \cdot 3 \bmod 17 =$
$((9^2 \bmod 17)^2 \bmod 17) \cdot 3 \bmod 17 =$
$((81 \bmod 17)^2 \bmod 17) \cdot 3 \bmod 17 =$
$(13^2 \bmod 17) \cdot 3 \bmod 17 =$
$(169 \bmod 17) \cdot 3 \bmod 17 =$
$16 \cdot 3 \bmod 17 =$
$48 \bmod 17 = 14$

Wie man sieht, kommen keine riesigen Zahlen vor.

Ein effizienter Algorithmus ist der folgende:

Algorithmus PowerMod (basis, expo, m) :
(berechnet basis^expo mod m)

```
tmp = 1
solange expo > 0
     falls expo ungerade dann tmp = tmp·basis mod m
     expo = expo div 2
     basis = basis² mod m
ergebnis = tmp
```

Test des Algorithmus für obiges Beispiel: $3^9 \bmod 17$:

basis=3 expo=9 m=17
tmp=1
tmp=1*3 mod 17 = 3
expo = 9 div 2 = 4
basis = 3^2 mod 17 = 9
expo = 4 div 2 = 2
basis = 9^2 mod 17 = 13
expo = 2 div 2 = 1
basis = 13^2 mod 17 = 16
tmp=3*16 mod 17 = 14
expo = 1 div 2 = 0
basis = 16^2 mod 17 = 1
ergebnis=14

Primzahltest von **Miller** und **Rabin**

Grundlage ist auch hier der bereits oben erwähnte „kleine Satz von Fermat“. Jedoch umgeht der Test von Miller und Rabin das Problem mit den Carmichael-Zahlen. Entscheidend ist folgender Satz:

Ist n eine Primzahl und $n-1 = 2^S \cdot d$ mit ungeradem d , dann gilt für alle a $<$ n mit ggT(a, n) = 1 :

Entweder $\boxed{a^d \bmod n = 1}$ oder $\boxed{a^{2^r \cdot d} \bmod n = (n-1)}$ für ein r aus der Menge { 0; 1; … ; s-1 }

Außerdem: Wenn $a^d \bmod n \neq 1 \wedge a^d \bmod n \neq n-1 \wedge a^{2^r \cdot d} \bmod n = 1;\ r > 0$, dann ist n nicht prim !

Der Primzahl-Test besteht nun darin, zu der zu prüfenden Zahl n eine teilerfremde Zahl a zu finden, die keine der ersten beiden Aussagen oder aber die 3. erfüllt. In diesem Fall wäre n keine Primzahl .
Da mehr als ¾ der Zahlen a aus der Menge {2; 3; … ; n-2} die Eigenschaft (beide Aussagen nicht erfüllt) besitzen, ist die Wahrscheinlichkeit für die Nicht-Primalität (Zerlegbarkeit) von n bei einem einmaligen Miller-Rabin-Test größer als ¾ .

Umgekehrt ist die Wahrscheinlichkeit dafür, die Primalität von n zu bestätigen, kleiner als ¼ .
Jargon: „**Man findet keinen Zeugen (engl: witness) gegen die Primalität von n**“ .
Durch Wiederholung des Tests mit anderem a-Wert kann man die Wahrscheinlichkeit verkleinern.
Bei 10-facher Durchführung ist die Wahrscheinlichkeit, nicht prim zu sein, kleiner als $0{,}25^{10} = 9{,}5 \cdot 10^{-7}$
Hat man dann immer noch keinen Zeugen gegen die Primalität von n gefunden, so ist n mit hoher Wahrscheinlichkeit prim !

Zu prüfen ist für jedes ausgewählte a:

- Berechne $x = a^d \bmod n$; falls x = 1 oder x = n-1, dann untersuche ein neues a (n evtl. prim)
- Andernfalls berechne $x = a^{2^r \cdot d} \bmod n$ für r aus { 1; 2; ... ; s-1 }
  - Falls x = 1, dann ist n auf jeden Fall zusammengesetzt (nicht prim)
  - Falls x = n-1, dann untersuche ein neues a (n evtl. prim)
- Falls x ≠ n-1, dann ist n zusammengesetzt, sonst n prim

Beispiel 1: **n = 97**. Dann ist $96 = 2^5 \cdot 3$. Also d = 3 und s = 5 .

Wähle **a = 2** < n und berechne erst $a^d \bmod n$, also $2^3 \bmod 97 = 8$.
Berechne nun $a^{2^r \cdot d} \bmod n$ für r = 1; 2; 3; 4 , also
$2^6 \bmod 97 = 64$ $2^{12} \bmod 97 = 22$ $2^{24} \bmod 97 = 96 = 97\text{-}1$
Daher spricht a = 2 nicht gegen die Primalität von 97.

Wähle **a = 37** < n und berechne erst $a^d \bmod n$, also $37^3 \bmod 97 = 19$.
Berechne nun $a^{2^r \cdot d} \bmod n$ für r = 1; 2; 3; 4.
$37^6 \bmod 97 = 70$ $37^{12} \bmod 97 = 50$ $37^{24} \bmod 97 = 75$ $37^{48} \bmod 97 = 96 = 97\text{-}1$!!
Daher spricht auch a = 37 nicht gegen die Primalität von 97.

Wähle **a = 96** < n und berechne erst $a^d \bmod n$, also $96^3 \bmod 97 = 96 = 97\text{-}1$!!
Daher spricht auch a = 96 nicht gegen die Primalität von 97.

Inzwischen liegt die Wahrscheinlichkeit für die Zerlegbarkeit von 97 bei $0{,}25^3 \approx 1{,}5\%$.
Für n = 97 findet man auch bei weiterer Suche kein a, das gegen die Primalität von 97 spricht .

Beispiel 2: **n = 561**. Dann ist $560 = 2^4 \cdot 35$. Also d = 35 und s = 4 .

Wähle **a = 2** < n und berechne erst $a^d \bmod n$, also $2^{35} \bmod 561 = 1$.
Dieses Ergebnis spricht nicht gegen die Primalität von 561.

Wähle **a = 3** < n und berechne erst $a^d \bmod n$, also $3^{35} \bmod 561 = 78$.

Berechne nun $a^{2^r \cdot d} \bmod n$ für r = 1; 2; 3 .
$3^{70} \bmod 561 = 474$ $3^{140} \bmod 561 = 276$ $3^{280} \bmod 561 = 441$.
Da keines der Ergebnisse 560 ist, spricht a = 3 gegen die Primalität von 561.
Daher ist 561 zusammengesetzt !

Beispiel 3: **n = 1729**. Dann ist $1728 = 2^6 \cdot 27$. Also d = 27 und s = 6 .

Wähle **a = 2** < n und berechne erst $a^d \bmod n$, also $2^{27} \bmod 1729 = 645$.

Berechne nun $a^{2^r \cdot d} \bmod n$ für r = 1; 2; 3; 4; 5 .
$2^{54} \bmod 1729 = 1065$ $2^{108} \bmod 1729 = 1$.
Dieses Ergebnis spricht gegen die Primalität von 1729.
Daher ist 1729 zusammengesetzt !

Aus "Wikipedia":

Es genügt, a aus der folgenden Zahlenmenge auszuwählen: $a \in \{2, 3, \ldots, \min(n-1, 2 \cdot (\ln n)^2)\}$

Außerdem ist bekannt, dass für kleine n eine viel kleinere Anzahl ausreicht, um auf Primalität zu testen:

n < 1.373.653 : es genügt, a= 2 und 3 zu testen,
n < 9.080.191 : es genügt, a = 31 und 73 zu testen,
n < 4.759.123.141 : es genügt, a = 2, 7, und 61 zu testen,
n < 2.152.302.898.747 : es genügt, a = 2, 3, 5, 7, und 11 zu testen,
n < 3.474.749.660.383 : es genügt, a = 2, 3, 5, 7, 11, und 13 zu testen,
n < 341.550.071.728.321 : es genügt, a = 2, 3, 5, 7, 11, 13, und 17 zu testen.
n < 3.825.123.056.546.413.051 : es genügt, a = 2, 3, 5, 7, 11, 13, 17, 19, und 23 zu testen.
n < 318.665.857.834.031.151.167.461 : es genügt, a = 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 zu testen.

Diese Tests sind sogar deterministisch !

In der folgenden JAVA-Methode für den Datentyp **long** wollen wir jedoch lediglich auf obige Menge mit $2(\ln n)^2$ zurückgreifen.

Man beachte aber, dass beim Datentyp **long** die Gefahr eines Überlaufs besteht, wenn Rechenoperationen wie Potenzierung vorgenommen werden.

Beim Quadrieren einer natürlichen Zahl a darf a^2 die Zahl

Long.MAX_VALUE = $9223372036854775807 = 2^{63}-1$ nicht überschreiten .

Daher muss gelten: a < 3037000500
Die zu untersuchende Zahl n darf somit höchstens 3037000500 betragen !

Das beim Miller-Rabin-Test notwendige Potenzieren (bei $a^d \bmod n$) birgt noch eine andere Gefahr, nämlich dass Potenzen extrem groß werden. Zur Vermeidung dieses Problems gibt es die so genannte Methode powerMod() , die bereits weiter oben erläutert wurde und die als höchste Potenz das Quadrat verwendet. Die entsprechende Java-Methode ist unten angegeben:

```java
public static long powerMod(long basis, long expo, long m) { // berechnet basis^expo mod m
   long erg = 1;
   while (expo > 0) {
      if (expo % 2 == 1)
         erg = (erg * basis) % m;
      expo = expo / 2;
      basis = (basis * basis) % m;
   }
   return erg;
}

public static long zufZahl(long von, long bis) { // Hilfsmethode
   Random rand = new Random();
   long zuf = Math.abs(rand.nextLong()) % (bis - von + 1);  // zuf in [0..bis-von]
   return zuf + von;  // Ergebnis in [von..bis]
}

public static boolean istPrimMillerRabin(long n) { // testet probabilistisch, ob n prim ist
   if (n == 2) return true;
   if (n < 2 || n % 2 == 0) return false;
   // Erzeuge ungerades d mit:   2^s * d = n-1
   long d = n - 1; // d zunächst gerade
   int s = 0;
   while (d % 2 == 0) {
      d = d / 2;
      s = s + 1;
   }
   long k = 20; // k ist die Anzahl der Durchläufe ; typisch k = 20
   long a, x;
   long aMin = 2;
   long aMax = Math.min(n-1, (long) (Math.log(n)*Math.log(n)*2.0));
   for (long i = 1; i <= k; i++) {
      a = zufZahl(aMin, aMax);  // a = random(2, min(n-1, 2*(ln n)²))
      // Berechnung von x = a^d mod n
      x = powerMod(a, d, n);
      if (x == 1 || x == n - 1)
         continue; // evtl. n prim; nächsten Durchlauf starten
      // Berechnung von x = a^(2^r*d) mod n
      for (int r = 1; r < s; r++) {
         x = (x * x) % n; // evtl. hier "Long-Überlauf" ?
         if (x == 1)
            return false; // n ist zusammengesetzt
         if (x == n - 1)
            break; // evtl. n prim; nächsten Durchlauf starten
      }
      if (x != n - 1)
         return false; // n ist zusammengesetzt
   } // Ende for long i
   return true; // sehr wahrscheinlich ist n prim
}

public static void main(String[] args) {
   String sEingabe = JOptionPane.showInputDialog("Miller-Rabin: LongZahl n = ", 982609483L);
   long n = Long.parseLong(sEingabe);
   if (n > 3037000500)
      System.out.println("Zahl " + n + " außerhalb des gültigen Bereichs !");
   else {
      if (istPrimMillerRabin(n))
         System.out.println(n + " ist vermutlich Primzahl ");
      else
         System.out.println(n + " ist keine Primzahl.");
   }
}
```

Anmerkung: In der BigInteger-Klasse von Java ist ein Primzahltest bereits eingebaut , nämlich die Methode "isProbablePrime()" .

Primfaktorzerlegung:

Die Primfaktorzerlegung hat heutzutage eine große Bedeutung erlangt, weil man damit Verschlüsselungen realisieren kann (**Kryptologie**). Es ist nämlich für große Zahlen mit mehreren 100 Stellen sehr schwer, sie in ihre Primfaktoren zu zerlegen. Auch Supercomputer beißen sich da die Zähne aus. Es wundert daher nicht, dass die Bestimmung der Primfaktorzerlegung einer natürlichen Zahl n deutlich aufwendiger ist als der bloße Primzahltest.

Brute-force-Methode:

Man prüft für alle Primzahlen (2;3;5;...), ob sie n teilen. Ist dies für eine Primzahl p der Fall, so wird n ersetzt durch n div p (div ist die ganzzahlige Division). Das Ergebnis n: = n div p wird weiter durch p dividiert. Geht die Division auf, so wird wieder n durch n div p ersetzt usw. Andernfalls wird die nächste Primzahl ausprobiert. Dies führt man durch bis zur Quadratwurzel des ursprünglichen n (entsprechend der Vorgehensweise beim Primzahltest) oder bis n=1 ist. Die Primzahleigenschaft p=6i-1 oder p=6i+1 ab i=1 sollte hier aus Geschwindigkeitsgründen ebenfalls Anwendung finden.

Beispiel: n = 12
12 div 2 = 6, also ist **2** ein Primfaktor von 12. Ersetze 12 durch 12 div 2 = 6.
6 div 2 = 3, also ist **2** ein weiterer Primfaktor von 12. Ersetze 6 durch 6 div 2 = 3.
3 div 2 = 1 Rest 1; 2 ist kein weiterer Primfaktor von 12.
Die nächste zu überprüfende Primzahl ist 3:
3 div 3 = 1, also ist **3** ein Primfaktor von 12. Ersetze 3 durch 3 div 3 = 1.
Die Zahl 1 enthält keine weiteren Primfaktoren, daher erfolgt nun der Programmabbruch !
Ergebnis: $12 = 2 \cdot 2 \cdot 3$

JAVA-Methode (optimiert):

```
public static String primfaktorZerlegung(long n) {
	String s = "";
	long max = (long) Math.sqrt(n);
	while (n % 2 == 0) {
		s = s + "2";
		n = n / 2;
		if (n > 1) s = s + "*";
		else return s;
	}
	while (n % 3 == 0) {
		s = s + "3";
		n = n / 3;
		if (n > 1) s = s + "*";
		else return s;
	}
	while (n % 5 == 0) {
		s = s + "5";
		n = n / 5;
		if (n > 1) s = s + "*";
		else return s;
	}
	long primfaktor = 7, increment = 4, letzteZahl = n;
	boolean letzteZahlprim = true;
	while (n != 1 && primfaktor <= max) {
		if (n % primfaktor == 0) {
			s = s + primfaktor;
			n = n / primfaktor;
			letzteZahlprim = false;
			letzteZahl = n;
			if (n > 1) s = s + "*";
		} else {
			primfaktor = primfaktor + increment;
			increment = 6 - increment;
			letzteZahlprim = true;
		}
	}
	if (letzteZahlprim) s = s + letzteZahl;
	return s;
}
```

Eine andere Methode zur Faktorzerlegung ist das

Verfahren von **Fermat** :

Dieses Verfahren wurde 1643 von FERMAT am Beispiel `2027651281 = 44021·46061` beschrieben.

Es ist Grundlage für das wesentlich leistungsfähigere Zerlegungsverfahren **„Quadratisches Sieb“** .

Zugrunde liegende Idee:

Wenn man die zu zerlegende Zahl n als Differenz zweier Quadrate [$n = a^2 - b^2$] darstellen kann, erhält man mithilfe der 3. Binomischen Formel: $n = a^2 - b^2 = (a + b)(a - b)$. Somit hat man n faktorisiert !

Wie findet man die ganzen Zahlen a und b ?

Zunächst wird $n = a^2 - b^2$ umgestellt zu $b^2 = a^2 - n$.
Man sucht also ganzzahlige a derart, dass $a^2 - n$ ein Quadrat b^2 ergeben (Probiermethode).

Genauer:
Ausgehend von $a_0 = \lceil \sqrt{n} \rceil$ berechnet man nacheinander die Quadrate der Zahlen
a_0 , a_0+1 , a_0+2 , … , und subtrahiert davon n, bis man als Ergebnis wieder ein Quadrat hat !

Ohne Beweis: Eine Obergrenze für die $a_0 + k$ ist $(n+9)/6$.

Beispiel: $n = 561 = 17 \cdot 33$

Die Quadratwurzel aus 493 ist $a_0 = 23$. Wir beginnen also mit 24 !
$24^2 - 561 = 15$ (kein Quadrat)
$25^2 - 561 = 64 = 8^2$ Somit ist eine Zerlegung gefunden, nämlich $561 = (25 - 8)(25 + 8) = 17 \cdot 33$.

Achtung: Ersichtlich müssen die beiden gefundenen Faktoren nicht zwingend prim sein !

JAVA-Methode :

```
public static long faktorZerlegungFermat(long n) {
   long grenze = (long) (n+9)/6;
   long a0 = (long)(ceil)Math.sqrt(n);
   long wurzel;
   for (long a = a0; a < grenze; a++) {
      wurzel = (long) Math.sqrt(a*a - n);
      if (wurzel*wurzel == a*a - n)
         return a - wurzel;   // bzw. a + wurzel
   }
   return 0L;  // nichts gefunden      }
```

Nachteil der FERMAT-Methode:

Bei weit voneinander entfernt liegenden Faktoren dauert das Verfahren sehr lange, weil viele Schritte durchzuführen sind !

Eine bessere Methode zur Faktorzerlegung ist die

Rho-Methode von Pollard :

Ein Beispiel: n = 143 = 11·13

Die Rho-Methode wendet eine Funktion f der Form $f(x) = (x^2 + c) \bmod n$ an, die eine Pseudozufallszahl liefert. Durch fortwährende Anwendung der Funktion f auf ihre Ergebnisse $f(x_{n+1}) = f(f(x_n))$ wird eine ganze Folge von Zufallszahlen generiert, die wegen der Modulo-Division eine Periode haben muss.

Für n = 143 und z.B. x0=5 c=37 ergibt sich eine Periodenlänge von 6 (Vorperiodenlänge = 3):
`xk = 5 62 20 **8 101 85 112 140 46** 8 101 85 112 140 46 8 101 85 112 140`

Wendet man die gleiche Funktion f auf einen der Primfaktoren, etwa p = 11 an, so erhält man:
`ak = 5 7 9 **8 2** 8 2 8 2 8 2 8 2 8 2 8`
Die Periodenlänge beträgt hier **2** (bei gleicher Vorperiodenlänge 3).

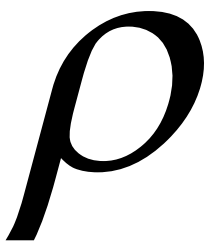
Aus der letztgenannten Tatsache folgert man, dass $x_{k+2} - x_k$ den Faktor p=11 enthält.
In der Tat findet man $x_5 - x_3 = 85 - 8 = 77 = 7 \cdot 11$. Entsprechend $x_6 - x_4 = 112 - 101 = 11 = 1 \cdot 11$.
Für jedes dieser Beispiele gilt folglich auch $ggT(x_{k+2} - x_k , n) = 11$.
Für x_k-Paare mit anderem Index-Unterschied gilt übrigens immer $ggT(x_k - x_i , n) = 1$.

Aus der x_k-Folge findet man die Paare mit $ggT(x_k - x_i , n) \neq 1$ mit einem Trick (Floyd):
Man berechnet neben der x_k-Folge noch eine zweite Folge x_{2k} , d.h. es wird nur jeder zweite Wert der Folge x_k berechnet. Somit ist die zweite Folge schneller als die erste, und man muss nur immer wieder $ggT(x_{2k} - x_k , n)$ berechnen. Irgendwann ist $x_{2k} = x_{k+pL}$ (pL=Periodenlänge der p-Folge) und der ggT ist gleich einem der gesuchten Primfaktoren. Dabei muss nicht unbedingt der kleinste der Primfaktoren als erster gefunden werde, ja es muss nicht einmal ein Primfaktor sein, sondern es kann auch nur irgend ein Teiler von n sein .

Für das obige Beispiel mit n = 143 läuft das Verfahren folgendermaßen ab:

```
xk  =  5 62  20   8 101  85 112 140 46 8 101
x2k =  5 20 101 112  46 101
```
Es gilt ggT(112-8,143) = ggT(104,143) = 13 .
Also hat man hier den größeren der beiden Primfaktoren gefunden !

Anmerkung:
Das periodische Verhalten der Folge gab der Rho-Methode ihren Namen, da man sich die Periode wie einen Kreis vorstellen kann und die Folgenglieder am Anfang wie einen Stängel, der in den Kreis hineinführt. Graphisch sieht das aus wie der griechische Buchstabe ρ.



JAVA-Methode (vorausgesetzt: n sei ungerade) :

```
public static long rho(long N) {
	long divisor;
	long c = (long) (Math.random() * (N - 1) + 1.0);
	long x = (long) (Math.random() * (N - 1) + 1.0);
	long xx = x;
	do {  x = (x * x + c) % N;
		xx = (xx * xx + c) % N;
		xx = (xx * xx + c) % N;
		divisor = ggT(Math.abs(x - xx), N);
	} while (divisor == 1);
	return divisor;
}
```

Pollard-Rho: Beispielrechnung für n = 341:

```
c= 22  x= 252
x= 100  xx= 133
divisor= 11
Primfaktorzerlegung von 341 = 11*31
```

Pollard-Rho: Beispielrechnung für n = 1073741823:

```
c= 394261293  x= 682806759
x= 473509515  xx= 240021111     divisor= 279
c= 76  x= 234
x= 148  xx= 218                 divisor= 1
x= 218  xx= 239                 divisor= 3
c= 13684341  x= 132608568
x= 276515673  xx= 225268035     divisor= 3
c= 70884621  x= 58719382
x= 33848693  xx= 23944366       divisor= 1
x= 23944366  xx= 2618003        divisor= 1
x= 55615084  xx= 119069118      divisor= 7
c= 13583729  x= 2535019
x= 7146956  xx= 4344211         divisor= 11
c= 413404  x= 635240
x= 120753  xx= 193492           divisor= 1
x= 193492  xx= 421148           divisor= 1
x= 1149475  xx= 808968          divisor= 1
x= 421148  xx= 901630           divisor= 151
c= 10084  x= 5584
x= 7961  xx= 5408               divisor= 1
x= 5408  xx= 7934               divisor= 1
x= 2437  xx= 1449               divisor= 1
x= 7934  xx= 781                divisor= 1
x= 7205  xx= 9195               divisor= 1
x= 1449  xx= 6988               divisor= 1
x= 6180  xx= 6986               divisor= 31
Primfaktorzerlegung von 1073741823 = 3*3*7*11*31*151*331
```

Primzahlerzeugung:

Der bekannteste Primzahlerzeuger ist das aus der Antike bekannte **„Sieb des Eratosthenes“** (auch als „sieve“ bekannt; ca. 275 -194 v.Chr.), mit dem man eine Tabelle aller Primzahlen von 2 bis zu einem vorgegebenen n bestimmt:

Will man die Primzahlen im Bereich von 2 bis n bestimmen, so legt man zunächst eine Tabelle mit allen natürlichen Zahlen von 2 bis n an. Jetzt streicht man aus dieser Tabelle zunächst alle Vielfachen von 2, anschließend alle Vielfachen von 3. Zu diesem Zeitpunkt ist die 4 bereits gestrichen, es müssen also keine Vielfachen davon beachtet werden. Als nächstes sind die Vielfachen der 5 dran, dann die von 7, usw. .
Hat man die Quadratwurzel von n überschritten, so findet man keine Vielfachen mehr, denn wenn t ein Teiler von n ist, dann ist es auch n / t . Man ist fertig.
Die nicht gestrichenen Zahlen sind die Primzahlen.

Nachteil: Für große n ergibt sich ein hoher Rechen- und Speicheraufwand.

JAVA-Methode:

```
public static ArrayList <Integer> eratosthenesListe(int nmax) {
   // Sieb des Eratosthenes; ArrayList wird erstellt von 2 bis zu <= nmax
   // nmax <= Integer.MAX_VALUE = 2^31-1 = 2.147.483.647   beachten !!
   nmax++;
   final int maxprim = (int)Math.sqrt(nmax)+2;
   boolean[] boolFeld = new boolean[nmax]; // alle Zahlen von 0 bis nmax = false
   for (int i = 0; i < nmax; i++)
      boolFeld[i] = i % 2 == 1; // alle ungeraden Zahlen = true

   // der eigentliche Algorithmus; boolsches Feld aufbauen:
   for (int prim = 3; prim < maxprim; prim += 2) // nur die ungeraden Zahlen
      // (die geraden Zahlen wurden schon gestrichen)
      if (boolFeld[prim]) { // noch nicht gestrichen, d.h. prim
         for (int i = prim; i <= nmax / prim; i++) {
            final int zahl = i * prim;
            if (zahl < nmax) // Überlauf verhindern
               boolFeld[zahl] = false; // zahl = i*prim streichen
         }
      }
   // ArrayListe aufbauen
   ArrayList <Integer> primZahlenListe = new ArrayList <Integer> () ;
   // Primzahl 2 am Anfang einfügen, da 2 nicht im boolFeld !
   primZahlenListe.add(2);
   for (int i = 3; i < nmax; i++)
      if (boolFeld[i])
         primZahlenListe.add(i);
   return primZahlenListe;
}
```

Wie erzeugt man **große** Primzahlen ?

Es gibt eine Reihe von Formeln, welche zu Primzahlen führen. Allerdings muss geprüft werden, ob es sich tatsächlich um eine Primzahl handelt !

Zunächst wird eine Methode vorgestellt, die auf bereits bekannte Primzahlen zurückgreift :

Kennt man die ersten k Primzahlen p_i (i=1 bis k), so lässt sich eine neue Zahl n auf folgende Weise erzeugen: $\mathbf{n = p_1 \cdot p_2 \cdot p_3 \cdot \ldots \cdot p_k + 1}$
n ist entweder **prim** oder
n besitzt **mindestens zwei Primfaktoren**, die von den bekannten k Primzahlen verschieden sind.

Man findet also auf jeden Fall mindestens eine Primzahl oberhalb von p_k !

Beispiele:
$n = 2 \cdot 3 + 1 = 7$ (prim)
$n = 2 \cdot 3 \cdot 5 + 1 = 31$ (prim)
$n = 2 \cdot 3 \cdot 5 \cdot 7 + 1 = 211$ (prim)
$n = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 + 1 = 2311$ (prim)
$n = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 + 1 = 30031 = 59 \cdot 509$ (alle Faktoren sind prim)
$n = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 + 1 = 510511 = 19 \cdot 97 \cdot 277$ (alle Faktoren sind prim)
$n = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 + 1 = 9699691 = 347 \cdot 27953$ (alle Faktoren sind prim)
$n = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 + 1 = 223092871 = 317 \cdot 703763$ (alle Faktoren sind prim)
$n = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 + 1 = 6469693231 = 331 \cdot 571 \cdot 34231$ (alle Faktoren sind prim)
$n = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 + 1 = 200560490131$ (prim)

Die so erzeugte Zahl n ist prim für k = 2, 3, 5, 7, 11, 31, 379, 1019, 1021, ...

Weitere Formeln:

$n! - 1$ ist prim für n = 3, 4, 6, 7, 12, 14, 30, 32, 33, 38, 94, 166, ...

$n! + 1$ ist prim für n = 1, 2, 3, 11, 27, 37, 41, 73, 77, 116, 154, …

$kgV(1, \ldots, n) + 1$ liefert die Primzahlen 2, 3, 7, 13, 61, 421, 2521, 232792561, …

Drei sehr bekannte Generatoren gehen auf die bekannten Mathematiker
Leonhard **Euler** (Schweiz) und
Pierre **de Fermat** (Frankreich) sowie auf
den franz. Mönch Marin **Mersenne** zurück:

Einige **Euler-Zahlen** ; das sind Zahlen der Form $E_n = n^2 + n + 41;\ n \in \mathbb{N}$

Die ersten 39 Euler-Zahlen sind Primzahlen.

$0^2 + 0 + 41 = 41$ (prim)
$1^2 + 1 + 41 = 43$ (prim)
$2^2 + 2 + 41 = 47$ (prim)
$3^2 + 3 + 41 = 53$ (prim)
...
...
$38^2 + 38 + 41 = 1523$ (prim)
$39^2 + 39 + 41 = 1601$ (prim)

$40^2 + 40 + 41 = 1681 = 41 \cdot 41$
$41^2 + 41 + 41 = 1763 = 41 \cdot 43$
...
$100^2 + 100 + 41 = 10141 =$ (prim)
...
$100000^2 + 100000 + 41 = 10000100041 = 163 \cdot 199 \cdot 308293$
...
$100000000^2 + 100000000 + 41 = 10000000100000041 =$ (prim)

Einige **Fermat-Zahlen** ; Zahlen der Form $F_m = 2^{2^m} + 1;\ m \in \mathbb{N}$)

Pierre de Fermat (1601-1665) vermutete, dass diese Zahlen immer Primzahlen seien, jedoch zeigte der schweizer Mathematiker Leonhard Euler bereits 1732 , dass **F5 zerlegbar** ist.

$F_0 = 2^{2^0} + 1 = 2^1 + 1 = 3$ (prim)
$F_1 = 2^{2^1} + 1 = 2^2 + 1 = 5$ (prim)
$F_2 = 2^{2^2} + 1 = 2^4 + 1 = 17$ (prim)
$F_3 = 2^{2^3} + 1 = 2^8 + 1 = 257$ (prim)
$F_4 = 2^{2^4} + 1 = 2^{16} + 1 = 65537$ (prim; größte bekannte Fermatsche Primzahl !)
$F_5 = 2^{2^5} + 1 = 2^{32} + 1 = 4294967297 = 641 \cdot 6700417$
$F_6 = 2^{2^6} + 1 = 2^{64} + 1 = 18446744073709551617 = 274177 \cdot 67280421310721$
$F_7 = 2^{2^7} + 1 = 2^{128} + 1 = 340282366920938463463374607431768211457 = 59649589127497217 \cdot 5704689200685129054721$

Bisher unbewiesene Vermutung: $2^{2^m} + 1$ ist für m > 4 stets zerlegbar !

Weitere Zerlegungen siehe Tabelle weiter unten !

Einige **Mersenne-Zahlen** ; Zahlen der Form $E_n = 2^n - 1;\ n \in \mathbb{N}^*$

Mersenne-Zahlen haben die Binärdarstellung 111 ... 111 .
Mersenne-Zahlen M_n können nur dann Primzahlen sein, wenn auch n prim ist.
Aber nicht jede Mersenne-Zahl mit n=prim ist eine Primzahl (z.B. M_{11}).
Bisher (26. Dez 2017) sind erst 50 Mersenne-Primzahlen bekannt !

$2^1-1 = 1$
2^2-1 = 3 (prim)
2^3-1 = 7 (prim)
$2^4-1 = 15 = 3 \cdot 5$
2^5-1 = 31 (prim)
$2^6-1 = 63 = 3 \cdot 3 \cdot 7$
2^7-1 = 127 (prim)
$2^8-1 = 255 = 3 \cdot 5 \cdot 17$
$2^9-1 = 511 = 7 \cdot 73$
$2^{10}-1 = 1023 = 3 \cdot 11 \cdot 31$
$2^{11}-1 = 2047 = 23 \cdot 89$
$2^{13}-1$ = 8191 (prim)
$2^{17}-1$ = 131071 (prim)
$2^{19}-1$ = 524287 (prim)
$2^{23}-1 = 8388607 = 47 \cdot 178481$
$2^{29}-1 = 536870911 = 233 \cdot 1103 \cdot 2089$
$2^{30}-1 = 1073741823 = 3^2 \cdot 7 \cdot 11 \cdot 31 \cdot 151 \cdot 331$
$2^{31}-1$ = 2147483647 = 2147483647 (prim; Euler)
$2^{41}-1 = 2199023255551 = 13367 \cdot 164511353$
$2^{61}-1$ = 2305843009213693951 (prim)
$2^{60}-1 = 1152921504606846975 = 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 31 \cdot 41 \cdot 61 \cdot 151 \cdot 331 \cdot 1321$ 19 Stellen
$2^{70}-1 = 1180591620717411303423 = 3 \cdot 11 \cdot 31 \cdot 43 \cdot 71 \cdot 127 \cdot 281 \cdot 86171 \cdot 122921$ 22 Stellen
$2^{89}-1$ = 618970019642690137449562111 (prim) 27 Stellen
$2^{97}-1 = 158456325028528675187087900671 = 11447 \cdot 13842607235828485645766393$
$2^{107}-1$ = 162259276829213363391578010288127 (prim) 33 Stellen
$2^{127}-1$ = 170141183460469231731687303715884105727 (prim; Lucas) 39 Stellen
$2^{137}-1 = 174224571863520493293247799005065324265471 =$
$32032215596496435569 \cdot 5439042183600204290159$ 42 Stellen
$2^{140}-1 = 1393796574908163946345982392040522594123775 =$
$3 \cdot 5^2 \cdot 11 \cdot 29 \cdot 31 \cdot 41 \cdot 43 \cdot 71 \cdot 113 \cdot 127 \cdot 281 \cdot 86171 \cdot 122921 \cdot 7416361 \cdot 47392381$ 43 Stellen
$2^{147}-1 = 178405961588244985132285746181186892047843327 =$
$7^3 \cdot 127 \cdot 337 \cdot 4432676798593 \cdot 2741672362528725535068727$ 45 Stellen
$2^{149}-1 = 713623846352979940529142984724747568191373311 =$
$86656268566282183151 \cdot 8235109336690846723986161$ 45 Stellen
$2^{151}-1 = 285449538541191976211657193889899027276549324 7 =$
$18121 \cdot 55871 \cdot 165799 \cdot 2332951 \cdot 7289088383388253664437433$ 46 Stellen
$2^{157}-1 = 182687704666362864775460604089535377456991567871 =$
$852133201 \cdot 60726444167 \cdot 1654058017289 \cdot 2134387368610417$ 48 Stellen
$2^{257}-1 = 231584178474632390847141970017375815706539969331281128078915168015826259279871 =$
$535006138814359 \cdot 1155685395246619182673033 \cdot 374550598501810936581776630096313181393$ 78 Stellen

$2^{521}-1$ = 647976601306097149819007990813932172694353001433054093944634591855431833976560521225596406614545549772963113914808580371219879997166438125740282911150571 51 (prim) 157 Stellen

$2^{607}-1$ = 531137992816767098689588206552468627329593117727031923199444138200403559860852242739162502265229285668889329486246501015346579337652707239409519978766587351943831270835393219031728127 (prim) 183 Stellen

$2^{1039}-1 =$
$589068086431683676644738724917747624711938696459815017753575689937658432079465555993259138490065014034006389161562581754376322314451080388584562460719428810761069833174599222153387113189363201210623862217392146903328852155899782370013718480620182690736866953411252382072659135491210334387684956209126576528293887$
$= 5080711 \cdot 558536666199362912607492046583159449686465270184886376480100523463198532883747 53 \cdot$
$207581819464423827645704813703594695162939708007395209881208387037927290903246793823431438841448348825340533447691122230281583276965253760914101891052419938993341097116243589620659721674811617490048036597355734092532054255236 89$ zerlegt von Thorsten Kleinjung; 2007 313 Stellen

$2^{1279}-1$ =

104079321946643990819252403273640855386152622472667048053191123504036080596733602980122394417323241848424216139542810077913835662483234649081399066056773207629241295093892203457731833496615835504729594205476898112116936771475484788669625013844382602917323488853111608285384165850282556046662248318909188018470682222031405210266984354887329580288780508697361869007147207105557031687290870 (prim) 386 Stellen

$2^{2203}-1$ =

147597991521418023508489862273738173631206614533316977514777121647857029787807894937740733704938928938274850753149648047728126483876025919181446336533026954049696120111343015690239609398909022625932693502528140961498349938822283144859860183431853623092377264139020949023183644689960821079548296376309423663094541083279376990539998245718632294472963641889062337217172374210563644036821845964963294853869690587265048691443463745750728044182367681351785209934866084717257940842231667809767022401199028017047489448742692474210882353680848507250224051945258754287534997655857267022963396257521263747789778550155264652260998886991401354048380986568125041949768669779710071 (prim) 664 Stellen

$2^{2281}-1$ =

446087557183758429571151706402101809886208632412859901111991219963404685792820473369112545269003989026153245931124316702395758705693679364790903497461147071065254193353938124978226307947312410798874869040070279328428810311754844108094878252494866760969586998128982645877596028979171536962503068429617331702184750324583009171832104916050157628886606372145501702225925125224076829605427173573964812995250569412480720738476855293681666712844831190877620606786663862190240118570736831901886479225810414714078935386562497968178729127629594924411960961386713946279899275006954917139758796061223803393537381034666494402951052059047968693255388647930440925104186817009640171764133172418132836351 (prim) 687 Stellen

Weitere riesige Mersenne-**Prim**zahlen

18. $2^{3217}-1$ = (prim) 969 Stellen
19. $2^{4253}-1$ = (prim) 1281 Stellen
20. $2^{4423}-1$ = (prim) 1332 Stellen
21. $2^{9689}-1$ = (prim) 2917 Stellen
22. $2^{9941}-1$ = (prim) 2993 Stellen
23. $2^{11213}-1$ = (prim) 3376 Stellen
24. $2^{19937}-1$ = (prim) 6002 Stellen
25. $2^{21701}-1$ = (prim) 6533 Stellen
26. $2^{23209}-1$ = (prim) 6987 Stellen
27. $2^{44497}-1$ = (prim) 13395 Stellen
28. $2^{86243}-1$ = (prim) 25962 Stellen
29. $2^{110503}-1$ = (prim) 33265 Stellen
30. $2^{132049}-1$ = (prim) 39751 Stellen
31. $2^{216091}-1$ = (prim) 65050 Stellen
32. $2^{756839}-1$ = (prim) 227832 Stellen
33. $2^{859433}-1$ = (prim) 258716 Stellen
34. $2^{1257787}-1$ = (prim) 378632 Stellen
35. $2^{1398269}-1$ = (prim) 420921 Stellen
36. $2^{2976221}-1$ = (prim) 895932 Stellen
37. $2^{3021377}-1$ = (prim) 909526 Stellen
38. $2^{6972593}-1$ = (prim) 2098960 Stellen
39. $2^{13466917}-1$ = (prim) 4053946 Stellen
40. $2^{20996011}-1$ = (prim) 6320430 Stellen
41. $2^{24036583}-1$ = (prim) 7235733 Stellen
42.
43.
44.
45.
46. $2^{42643801}-1$ = (prim) 12837064 Stellen ; 2009 gefunden
47. $2^{43112609}-1$ = (prim) 12978189 Stellen ; 2008 gefunden
48. $2^{57885161}-1$ = (prim) 17425170 Stellen ; 2013 gefunden
49. $2^{74207281}-1$ = (prim) 22338618 Stellen ; 2016 gefunden
50. $2^{77232917}-1$ = (prim) 23249425 Stellen ; Dez. 2017 gefunden (von Jon Pace)
größte bisher bekannte Primzahl (Stand: 26.12.2017 !)

RepUnit - Zahlen R_n

Das sind Zahlen, die ausschließlich die Ziffer 1 enthalten !
R_n kann nur dann prim sein, wenn n prim ist (notwendig, nicht hinreichend !)
Gemessen an den zerlegbaren RepUnit-Zahlen gibt es nur wenige prime (s.u.), man vermutet dennoch, dass es unendlich viele prime RepUnit-Zahlen gibt.

Beispiele:

R2 = 11 prim;
R3 = 111 = 3·37
R7 = 1111111 = 239·4649
R11 = 11111111111 = 21649·513239
R17 = 11111111111111111 = 2071723·5363222357
R19 = 1111111111111111111 prim;
R23 = 11111111111111111111111 prim;
R29 = 11111111111111111111111111111 = 3191·16763·43037·62003·77843839397
R32 = 11111111111111111111111111111111 = 11·17·73·101·137·353·449·641·1409·69857·5882353

R41 = 11111111111111111111111111111111111111111 =
83·1231·538987·20176370990032280374865794236 1

R47 = 11111111111111111111111111111111111111111111111 =
35121409·316362908763458525001406154038726382279

R53 = 11111111111111111111111111111111111111111111111111111 =
107·1659431·1325815267337711173·47198858799491425660200071

R71 = 11111111111111111111111111111111111111111111111111111111111111111111111 =
241573142393627673576957439049·459948113478868463102217288952230343018 39 Yafu: 29 s

R97 = 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111 =
12004721·109399846855370537540339266842070119107662296580348039·846035731396919233767211537899097169 Yafu: 1042 s

R131 = 1111...11 = 80173·109517·141811693·446790173·7370364319027·15594845538029429933·
731772397003105767769 3·131758351065116151205213·18022206228783402545124708 1 yafu: 4 s

R200 = 1111...11 = 11·41·73·101·137·251·271·401·1201·1601·3541·5051·9091·21401·25601·27961·
60101·1676321·7019801·5964848081·182521213001·14103673319201·78875943472201·1680588011350901·
12969441902905775055138577118456427449907570094765675782153729152719 6801

Desweiteren sind auch **R317** und **R1031** prim !

Wahrscheinlich prim sind:
R49081 (09-1999; Dubner), **R86453** (10-2000; Baxter), **R109297** (03-2007; Bourdelais-Dubner),
R270343 (07-2007; Voznyy-Budnyy).

RepUnit-Zahlen haben die Form: $\mathbf{R_n = (10^n - 1) / 9}$, da $9 \cdot R_n = 10^n - 1$ gilt !
Beispiel: R_5 (= 11111) = $(10^5 - 1) / 9 = 99999 / 9 = 11111$

Die Faktoren von RepUnit-Zahlen haben die Form: **faktor = 2 · k · n + 1**

Algorithmus zum Suchen von RepUnit-Zerlegungen:

```
Eingabe(n)
rn = (10^n - 1) / 9
für k=1 bis 1000000
   factor = 2*k*n+1
   rest = rn % factor
   falls rest = 0
      dann ausgabe(factor)
```

Weitere Zahlen und ihre Faktorisierung :

Stellen	Zahl n =	Zeit / s
9	138909787 = 11783·11789	
10	8616460799 = 89681·96079	
10	4294967297 = 641·6700417 (Fermat **F5 = 2^32+1**)	
10	2^{32} + 2768 = 4294970064 = 2·2·2·2·3·3·7·11·19·19·29·37	
11	17543877193 = 52579·333667	
12	241873694197 = 15619·15485863	
13	4156338464051 = 2038717·2038703	
14	18128486155487 = 4257749·4257763	
14	31415926535897 = 163·44777·4304347	
15	341550071728318 = 2·53·1194883·2696641	
15	341550071728319 = 4185971·81593989	
15	239812014798221 = 15485863·15485867	
15	191427922719503 = 13835747·13835749	
16	1001001000000007 = 64513·15516268039	
17	68442389367182561 = 15485863·4419669047	
17	10000000000000331 = 31·322580645161301	
18	352523661753384896 = 2^6·83·2603401·25491133	
18	965521343020615207 = 982609429·982609483	
19	4160426670634335197 = 2039712397·2039712401	
19	4342808478641644163 = 982609429·4419669047	
19	3825123056546413051 = 149491·747451·34233211	
20	18446744073709551617 = 274177·67280421310721 (Fermat **F6 = 2^64+1**)	
21	109890109889010989011 = 599144041·183411838171	
21	415758465533848642967 = 11110693267·37419669101	
24	662835905978993515936337 = 724170440549·915303730813	
25	5018557517866741394046901 = 1481124532001·3388342714901	
26	41615325283752522089299429 = 16927^5·29947	
26	33333332222222222218888889 = 29999999·1111111111111111111	
28	5313531353135313531353135313 = 3·7·11·23·29·239·281·4649·909091·121499449	
32	79798272259984334113870248395353 = 1303·6763·10627·16927·29947·157543·10670053	
33	100000000000000000000000000000001 = 19841·976193·6187457·834427406578561	
34	1240819002867598361280210027487189 = 538119463428139·2305843009213693951	brent: 16
37	2310000000000000000000000000000000000 = 2^{34}·3·5^{34}·7·11	
39	340282366920938463463374607431768211457 = 59649589127497217·5704689200685129054721 (Fermat **F7 = 2^128+1**)	brent: 468
42	281728347264723747372617263727162736372615 = 3·5·11^2·1673043·26741327·4972638385453668937186 61	
45	100000000000000000000000000000000000000000001 = 3·137·617·1620583484601296758492708265640210 6953	
46	142724769270595988043931594750096198971949 0561 = 2305843009213693951·6189700196426901374495621 11 = M61· M89	
48	465235439986679108204139705633885130899652358311 = 2305843009213693951·2017637099003228037486579423 61 = M61· ?	rho: 81:19 m brent: 16:50 m
49	857791229326544540316236146216299722004310287619 9 = 127647878463584414 71·671998030559713968361666935769 ("mathematik-alpha")	yafu: 0,4
51	314159265358979323846264338327950288419716939937510 = 2·5·1657·2767321·685121861329964043983664837678339854 6983	

55	675062234896414305195630546932696211776378888978198538 7 = 7·97·997·9973·99991·999983·9999991·99999989·999999937·9999999967 dies ist das Produkt der jeweils größten ein-, zwei-, drei- bis zehnstelligen Primzahlen	
63	85338001347199411325839928623044853896918013840619601452909774 3 = 525936039003934785796425018980 9·162259276829213363391578010288127	yafu: 6 ct2: 4
71	27606985387162255149739023449107931668458716142620601169954803000803329 = 162259276829213363391578010288127·170141183460469231731687303715884105727 = M107· M127	yafu: 32
78	11579208923731619542357098500868790785326998466564056403945758400791312963993 7 = 1238926361552897·93461639715357977769163558199606896584051237541638188580280321 (Fermat **F8 = 2^256+1**)	brent: 62 rho: 826 ct2: 39 yafu: 0,2
80	25389739913858345524208216151645759882986445317021182277812631082013053557679073 = 57841295759118288267473253993921326751 37· 43895524089907382652596083010742568073 29	yafu: 131 ct2: 89
84	23986136625956052485865148735412958823329151363927624346327130843836030789585592 1843 = 489756435648946347624324859824637632984391· 48975643564894634762432485982463763298437 3	yafu: 0,03
91	10185179881672430431342228442046890805257341968329681253180702246771906498816683 53091698687 = 47·599·8191·178481·9341359·14718679249·13444476836590589479· 51441563151591093599·260242449712509916159	
121	14248424502937046318559413786173650827928703629619394683997793538001378025398313 94422161828003733369548864158809441716321 = 523·2417·1523·32497·11491·4639·196717·48397·100673·993683·489553·1126847·2299159·9316273·17094767·7504421·29592773689035264646070825945299759722 1	
155	13407807929942597099574024998205846127479365820592393377723561443721764030073546976801874298166903427690031858186486050853753882811946569946433649006084097 = 2424833·74556028256478842083373957362004549187833663426 57·74164006262753080152478714190193747405994078109751902390582131614441575950470500809281871169394073 7 (Fermat **F9 = 2^512+1**)	yafu: Abbruch
287	2^{953} + 1 = 761352657140624928152607999052749086787205223249609963036555191969761616466127039255126177022032051501976946580578802201832747645718991722388905257524426048400881070309099455458655441835160441277562468240563935849506687337921474605136434657458512938502167093992521683365767858593118289 93 = 3·1907·425796183929·16247002794788943855987796558425843 77· 38023067385494413244321390912718281 21·128064886830166671444802576129115872060027·33884958374667213943683932046721815228158303686049930480849258405552811 77· 11658823406671259903148376558383270818131012258146392600439520994131344334162924536139	yafu: einige Tage
309	17976931348623159077293051907890247336179769789423065727343008115773267580550096313270847732240753602112011387987139335765878976881441662249284743063947412437776789342486548527630221960124609411945308295208500576883815068234246288147391311054082723716335051068458629823994724593847971630483535632962422413721 7 = 45592577·6487031809·46597757852200185432645607430767781928 97·1304398744054881897274847687965099039466085308416118921868952957768324162514718635741402279775731048958987839288429238448311490329137987290886016179460941194490105959067101305319061710183544916096191939124885381160807122996723228062178207531270144245 77 (Fermat **F10 = 2^1024+1**)	
617	32317006071311007300714876688669951960444102669715484032130345427524655138867890893197201411522913463688717960921898019494119559150490921095088152386448283120630877367300996091750197750389652106796057638384067568276792218642619756161838094338476170470581645852036305042887575891541065808607552399123930385521914333389668342420684974786564569494856176035326322058077805659331026192708460314150258592864177116725943603718461857357598351152301645904403697613233287231227125684710820209725157101726931323469678542580656697935045997268352998638215525166389437335543602135433229604645318478604952148193555853611059596230657 = 319489·974849·167988556341760475137·3560841906445833920513· 17346244717914755543025897086430977837742184472366408464934701906136357919287910885759103833040883717798381086845154642194071297830613418986428082601454275870858924387368556397311894886939915854550661114742021613255701726056413939436694579322096866510895968548270538807264582855415193640191246493118254609287981573305779557335850498227928009094287256759151891211862275171431922978810097925103603549691727991266352735878323664719315477709142774537703829458491891759032511093938132248604429857397165071105924446217754254070691304703466464360349138244172330659883 4177 (Fermat **F11 = 2^2048+1**)	

Die Faktorisierungen wurden durchgeführt auf einem Intel i7-6700 .
yafu: meint yafu-x64 .
ct2: meint CrypTool2 .
rho: bzw. brent: meint die jeweilige Berechnung mit jLangzahlRechner .

Programme zur Faktorisierung mit Ermittelung der Berechnungszeit

1) CrypTool2 (mit GUI)
2) Yafu-x64 (Konsolenprogramm) ; Aufruf z.B. yafu-x64 factor(1001)
3) jlangzahlrechner (jar executable GUI von Ac)
4) GeoGebra(CAS-Modul; GUI)
5) wxMaxima(CAS; GUI)

RSA – Zahlen

Es handelt sich hier um sog. Semiprimzahlen, also Produkte aus genau 2 Primzahlen .

Diese Zahlen sind schwer zu faktorisieren, wenn sie genügend groß sind (etwa 100 Ziffern und mehr !) . Daher werden sie in der **Kryptographie** verwendet.

Beispiel 1: `7136238463529799405291429847247475681913733 11` besteht aus 45 Ziffern

Diese Zahl lässt sich zerlegen in `86656268566282183151·823510933669084672398616 1`

Beispiel 2: RSA-100 = `1522605027922533360535618378132637429718068114961380688657908494580122963258952897654000350692006139` besteht aus 100 Ziffern

Diese Zahl lässt sich zerlegen in `37975227936943673922808872755445627854565536638199·`
`4009469095092088103068373529276146838921489972406 1`

Anmerkung: CrypTool2 zerlegt RSA-100 in 1 Stunde 40 min (!) ; Prozessor Intel i6700

Beispiel aus „Spektrum der Wissenschaft“:

Die **129-stellige** Zahl „**RSA-129**“ =

`114 381 625 757 888 867 669 235 779 976 146 612 010 218 296 721 242 362 562 561 842 935 706 935 245 733 897 830 597 123 563 958 705 058 989 075 147 599 290 026 879 543 541`

ist Produkt zweier Primzahlen. Wie lauten diese Faktoren?

Diese Frage stellte **Martin Gardner** (1914 - 2010) den Lesern des Scientific American im August 1977 in seiner Kolumne "Mathematical Recreations".

Im Gegensatz zu den Rätseln, die Gardner sonst aufzugeben pflegte, musste dieses ungewöhnlich lange auf eine Lösung warten. Erst mehr als 16 Jahre später, im April 1994, präsentierten Paul Leyland von der Universität Oxford, Michael Graff von der Universität von Iowa in Iowa City und Derek Atkins vom Massachusetts Institute of Technology in Cambridge die 64- bzw. 65-stelligen Primfaktoren

`3490 529 510 847 650 949 147 849 619 903 898 133 417 764 638 493 387 843 990 820 577`

sowie

`32769 132 993 266 709 549 961 988 190 834 461 413 177 642 967 992 942 539 798 288 533`

Weitere Beispiele für RSA-Zahlen

RSA 576

Die Primfaktorzerlegung dieser 174-stelligen Zahl wurde im Dezember 2003 von Jens Franke und Thorsten Kleinjung vom Mathematischen Institut in Bonn und dem Institut für Experimentelle Mathematik in Essen gefunden. Das Preisgeld lag bei 10.000 US$.

```
RSA-576 =
188198812920607963838697239461650439807163563379417382700763356422988859715234665485319060606504743045317388011303396716199692321205734031879550656996221305168759307650257059
```

wird zerlegt in

```
39807508642406493739712550055038649119906436234252670840638518957594638895726176858331 7 · 472772146107435302536223071973048224632914695302097116459852171130520711256363590397527
```

RSA 640

Die Faktoren dieser 193-stelligen Zahl wurden im November 2005 von F. Bahr, M. Boehm, J. Franke, T. Kleinjung gefunden, die zuvor schon RSA 200 faktorisiert hatten. Das Preisgeld lag bei 20.000 US$.

```
RSA-640 =
3107418240490043721350750035888567930037346022842727545720161948823206440518081504556346829671723286782437916272838033415471073108501919548529007337724822783525742386454014691736602477652346609
```

wird zerlegt in

```
163473364580925384844313388386509085984178367003309231218111085238933310010450815121211816751157 9 · 190087128166482211312685157393541397547189678996851549366663853908802710380210449895719126146557 1
```

RSA 768

Die Faktorisierung dieser 232-stelligen Zahl wurde am 12. Dezember 2009 von Thorsten Kleinjung et al. vollendet.[1] Der RSA Factoring Challenge war zu dieser Zeit schon beendet, sodass kein Preisgeld ausgezahlt wurde.

```
RSA-768 =
123018668453011775513049495838496272077285356959533479219732245215172640050726365751874520219978646938995647494277406384592519255732630345373154826850791702612214291346167042921431160222124047927473779408066535141959745985690214341 3
```

wird zerlegt in

```
33478071698956898786044169848212690817704794983713768568912431388982883793878002287614711652531743087737814467999489 · 367460436667995904282446337996279526322791581643430876426760322838157396665112792333734171433968102700927987363089177
```