

Aufgabe 5.1

```
[ > restart:
[ > zuffi := rand(2..2^29):
[ > for i from 1 to 10 do
  n := 2*zuffi()-1; # => n gleichverteilt in den ungeraden Zahlen von 3 bis 2^30-1.
  for e from 0 do
    if 2^e > n then
      printf("Vermutung ist falsch für %d.\n",n);
      break;
    fi;
    p := n - 2^e;
    if isprime( p ) then
      printf("Vermutung ist korrekt für %d = 2^%d + %d.\n",n,e,p);
      break;
    fi;
  od;
od:
Vermutung ist korrekt für 36843657 = 2^3 + 36843649.
Vermutung ist korrekt für 148871517 = 2^10 + 148870493.
Vermutung ist korrekt für 254269625 = 2^10 + 254268601.
Vermutung ist korrekt für 448954285 = 2^3 + 448954277.
Vermutung ist falsch für 426949031.
Vermutung ist korrekt für 987911609 = 2^24 + 971134393.
Vermutung ist korrekt für 377199185 = 2^10 + 377198161.
Vermutung ist falsch für 268443113.
Vermutung ist korrekt für 744762743 = 2^12 + 744758647.
Vermutung ist falsch für 899697463.
```

Mit Hilfe der Funktion 'zuffi' werden gleichverteilt ungerade Zufallszahlen n zwischen 3 und $2^{30}-1$ erzeugt. Danach werden dann die Zweierpotenzen bis zu einer Abbruchbedingung hochgezählt. Wenn eine Zweierpotenz erreicht wird, die größer ist als die zufällig erzeugte Zahl, ist die Vermutung widerlegt und die Berechnung wird abgebrochen. Solange das nicht geschieht, wird überprüft, ob die Differenz Zufallszahl - Zweierpotenz eine Primzahl ergibt. Wenn das der Fall ist, wird ebenfalls abgebrochen und die Zerlegung der Zahl in Primzahl und Zweierpotenz ausgegeben. Wie leicht zu sehen ist, ist die vorgegebene Vermutung falsch. Wenn die Zerlegung existiert, ist sie im allgemeinen nicht eindeutig.

Bsp: $23 = 2^2 + 19 = 2^4 + 7$

Aufgabe 5.3

```
[ (i)
[ > N := 64:
[ > st := time():
  P := NULL:
  for n from 2 to N by 2 do
    p := 3;
    q := 5;
    while n <> q - p do
      p := q;
      q := nextprime(q);
    od;
    printf("%d = %d - %d\n",n,p,q);
  od;
  printf("Laufzeit = %f Sekunden",time() - st);
2 = 3 - 5
4 = 7 - 11
6 = 23 - 29
8 = 89 - 97
10 = 139 - 149
12 = 199 - 211
14 = 113 - 127
16 = 1831 - 1847
18 = 523 - 541
20 = 887 - 907
22 = 1129 - 1151
24 = 1669 - 1693
26 = 2477 - 2503
28 = 2971 - 2999
30 = 4297 - 4327
32 = 5591 - 5623
34 = 1327 - 1361
36 = 9551 - 9587
38 = 30593 - 30631
```

```

40 = 19333 - 19373
42 = 16141 - 16183
44 = 15683 - 15727
46 = 81463 - 81509
48 = 28229 - 28277
50 = 31907 - 31957
52 = 19609 - 19661
54 = 35617 - 35671
56 = 82073 - 82129
58 = 44293 - 44351
60 = 43331 - 43391
62 = 34061 - 34123
64 = 89689 - 89753
Laufzeit = 6.827000 Sekunden

```

Mit der Schleife oben suchen wir einfach für jede gerade Zahl von 2 bis 64 nach dem passenden Primzahlpärchen.

Für jedes n werden am Anfang der Schleife p und q auf 3 und 5 gesetzt und dann immer jeweils eine Primzahl weiter'geschoben', bis als Differenz n erreicht wird. Bis n = 64 geht das auch mit diesem Ansatz noch ziemlich schnell. Die Laufzeit kann aber ohne großen Aufwand erheblich verbessert werden. Leicht zu sehen ist z.B., daß bei jedem Durchlauf wieder die Differenzen 5-3 = 2, 7-5 = 2, 11-7 = 4, 13 - 11 = 2 usw. getestet werden. Wir können also leicht bei den Vergleichen und Subtraktionen sparen:

```

> N:=64:
> st := time():
  hits := 0:
  P := proc(n) option remember; false; end:
  p := 3:
  q := 5:
  while hits < N/2 do
    n := q - p;
    if P(n) = false then
      P(n) := p;
      if n <= N then hits := hits + 1; fi;
    fi;
    p := q;
    q := nextprime(p);
  od:
  for n from 2 to N by 2 do
    printf("%d = %d - %d\n",n,P(n)+n,P(n));
  od;
  printf("Laufzeit = %f Sekunden",time() - st);

```

```

2 = 5 - 3
4 = 11 - 7
6 = 29 - 23
8 = 97 - 89
10 = 149 - 139
12 = 211 - 199
14 = 127 - 113
16 = 1847 - 1831
18 = 541 - 523
20 = 907 - 887
40 = 19373 - 19333
42 = 16183 - 16141
44 = 15727 - 15683
46 = 81509 - 81463
48 = 28277 - 28229
50 = 31957 - 31907
52 = 19661 - 19609
54 = 35671 - 35617
56 = 82129 - 82073
58 = 44351 - 44293
60 = 43391 - 43331
62 = 34123 - 34061
64 = 89753 - 89689
Laufzeit = 1.188000 Sekunden

```

Die Laufzeitverbesserung ist offensichtlich. Was haben wir geändert?

P ist jetzt eine Prozedur, die für jede Eingabe n den Wert 'false' zurückliefert, sofern n kein anderer Rückgabewert zugewiesen wurde. Dann setzen wir wie vorher für p und q die Startwerte 3 und 5 ein, außerdem benötigen wir noch eine Zählvariable ('hits'). Solange diese Zählvariable jetzt kleiner als N/2 ist (in unserem Beispiel ist N = 64, also N/2 = 32) wird folgendes getan:

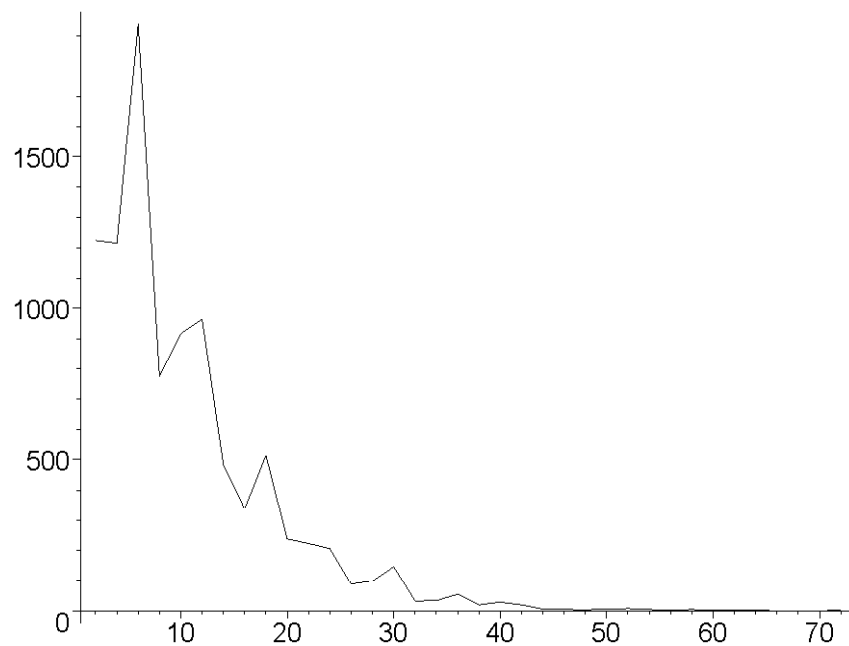
1. n wird die Differenz von q und p zugewiesen,
2. wenn die Funktion P für n noch den Wert 'false' zurückliefert, weise diesem n als Ausgabewert p zu (q ist dann einfach zu berechnen mit p+n!) und
3. zähle 'hits' einen hoch, für den Fall, dass n kleiner als N (also hier 64) ist.

Damit wird jetzt die Aufgabe mit ungefähr dem Aufwand gelöst, mit dem die erste Schleife das Problem für N löst! (plus einige Zuweisungen und N/2-1 Additionen)

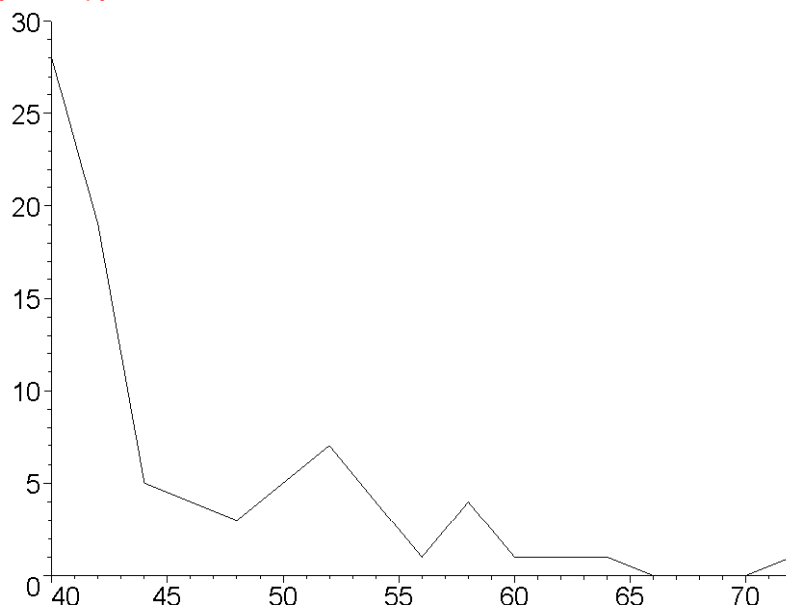
Ihr könnt ja mal in der ersten Schleife n von 64 an laufen lassen (also die ganze Schleife nur für 64 ausführen) und die Laufzeiten vergleichen!

(ii)

```
> N := 100000:
> hit := proc(n) option remember; 0; end:
> p := 3:
  q := 5:
  MAX := 0:
  while q < N do
    n := q-p;
    if n > MAX then MAX := n; fi;
    hit(n) := hit(n) + 1;
    p := q;
    q := nextprime(q);
  od:
> L := [seq([2*n, hit(2*n)], n=1..MAX/2)]:
> plot(L);
```



```
> plot(L, 40..MAX, 0..30);
```



Diese Teilaufgabe ist in etwa genauso gelöst worden, wie (i). Die Funktion 'hit' zählt für jedes n mit, wie oft die jeweilige Differenz von aufeinanderfolgenden Primzahlen bis N vorgekommen ist. Das zu plottende L ist deshalb von der Form $(2*n, \text{hit}(2*n))$, wobei n bis MAX/2 läuft, weil sonst die 0 mal vorkommenden ungeraden Zahlen mitgeplottet würden.

Aufgabe 5.4

(i)

```
> N := 10:
> for n from 1 to N do
  fund := false;
  for p from n^2+1 to (n+1)^2-1 while not fund do
    if isprime(p) then
      printf("%d^2 = %d < %d < %d = %d^2.\n",n,n^2,p,(n+1)^2,n+1);
      fund := true;
    fi;
  od;
  if not fund then
    printf("Behauptung ist falsch für n = %d.\n",n);
    break;
  fi;
od:
1^2 = 1 < 2 < 4 = 2^2.
2^2 = 4 < 5 < 9 = 3^2.
3^2 = 9 < 11 < 16 = 4^2.
4^2 = 16 < 17 < 25 = 5^2.
5^2 = 25 < 29 < 36 = 6^2.
6^2 = 36 < 37 < 49 = 7^2.
7^2 = 49 < 53 < 64 = 8^2.
8^2 = 64 < 67 < 81 = 9^2.
9^2 = 81 < 83 < 100 = 10^2.
10^2 = 100 < 101 < 121 = 11^2.
```

Die Prozedur sucht einfach von der kleineren Quadratzahl ausgehend nach der nächsten Primzahl. Sie würde falsch zurückgeben, wenn sie einmal keine Primzahl bis zur folgenden Quadratzahl fände.

Das wird nicht passieren, da man mit dem Primzahlsatz leicht nachweisen kann, dass zwischen den zwei Quadraten n^2 und $(n+1)^2$ (n aus den natürlichen Zahlen und n größer als 59) immer mindestens eine Primzahl liegen muss und bis 59 haben wir ja hier nachgeprüft.

(ii)

```
> N := 100:
> p := 5:
  while p < N do
    if p-1 mod 6 = 0 then
      printf("%d = 6 * %d + 1\n",p,(p-1)/6);
    elif p+1 mod 6 = 0 then
      printf("%d = 6 * %d - 1\n",p,(p+1)/6);
    else
      printf("Behauptung ist falsch für p = %d.\n",p);
      break;
    fi;
    p := nextprime(p);
  od:
5 = 6 * 1 - 1
7 = 6 * 1 + 1
11 = 6 * 2 - 1
13 = 6 * 2 + 1
17 = 6 * 3 - 1
19 = 6 * 3 + 1
23 = 6 * 4 - 1
29 = 6 * 5 - 1
31 = 6 * 5 + 1
37 = 6 * 6 + 1
41 = 6 * 7 - 1
43 = 6 * 7 + 1
47 = 6 * 8 - 1
53 = 6 * 9 - 1
59 = 6 * 10 - 1
61 = 6 * 10 + 1
67 = 6 * 11 + 1
71 = 6 * 12 - 1
73 = 6 * 12 + 1
79 = 6 * 13 + 1
83 = 6 * 14 - 1
89 = 6 * 15 - 1
97 = 6 * 16 + 1
```

Unsere Prozedur beginnt mit $p = 5$ (kleinste Primzahl größer 3). Dann wird für 5 und jede folgende Primzahl p bis N (hier als

Beispiel 100) überprüft, ob $p-1$ (bzw. $p+1$) modulo 6 gleich Null ist. Das bedeutet nur, daß getestet wird, ob $p-1 = n*6$ (n aus \mathbb{N}), also $p = 6*n + 1$. Das gleiche gilt für $p+1$. Bei dieser Behauptung ist relativ leicht zu überlegen, dass sie richtig ist. Schreiben wir mal die Aussage anders hin:

Jede Primzahl ist 1 oder 5 modulo 6. Gilt $p = 0 \pmod 6$, ist p durch 6 teilbar und sicherlich nicht prim. Bei $p = 2 \pmod 6$, ist $p = 6*n + 2 = 2 * (3*n + 1)$, also durch 2 teilbar. Einen analogen Schluss kann man für Rest 3 machen: $p = 6*n + 3 = 3 * (2*n + 1)$. Bleibt noch $p = 6*n + 4 = 2 * (3*n + 2)$, p wäre also auch durch 2 teilbar. Es folgt, dass unsere Behauptung richtig sein muß.

(iii)

```
> N := 100:
> p := 5:
  while p < N do
    fund := false;
    for m from 0 while 2*m^2 <= p and not fund do
      n := sqrt(p-m^2);
      if type(n,integer) then
        printf("%d = %d^2 + %d^2\n",p,m,n);
        fund := true;
      fi;
    od;
    if not fund then
      printf("Behauptung ist falsch für p = %d.\n",p);
    fi;
    p := nextprime(p);
    while p mod 4 <> 1 do
      p := nextprime(p);
    od;
  od:
5 = 1^2 + 2^2
13 = 2^2 + 3^2
17 = 1^2 + 4^2
29 = 2^2 + 5^2
37 = 1^2 + 6^2
41 = 4^2 + 5^2
53 = 2^2 + 7^2
61 = 5^2 + 6^2
73 = 3^2 + 8^2
89 = 5^2 + 8^2
97 = 4^2 + 9^2
```

Die äußere Schleife durchläuft alle Primzahlen der Form $4n+1$ bis zu der Schranke N und sucht dann nach zwei Quadratzahlen, deren Summe der Primzahl entspricht.

Die 'Suchschleife' läuft für jede Primzahl von 0 los und bricht ab, wenn der Laufindex m die Bedingung $2*m^2 \leq p$ verletzt, wenn also $p/2 < m^2$ und somit keine passenden Quadratzahlen mehr gefunden werden können. Ansonsten wird getestet, ob die Wurzel aus Primzahl - m^2 ein Integerwert ist. Ist das der Fall, ist eine Zerlegung in Quadratzahlen gefunden, andernfalls wird m einen hochgezählt. Wenn eine Zerlegung gefunden ist oder festgestellt wurde, dass die Behauptung falsch ist, wird die nächste Primzahl nach p gesucht, die die Form $4n+1$ hat und p auf diesen Wert gesetzt.

(iv)

```
> N := 100:
> p := 3:
  while p < N do
    fund := false;
    for m from 0 while 2*m^2 <= p and not fund do
      n := sqrt(p-m^2);
      if type(n,integer) then
        printf("Behauptung ist falsch für p = %d = %d^2 + %d^2\n",p,m,n);
        fund := true;
      fi;
    od;
    if not fund then
      printf("Behauptung ist wahr für p = %d.\n",p);
    fi;
    p := nextprime(p);
    while p mod 4 <> 3 do
      p := nextprime(p);
    od;
  od:
Behauptung ist wahr für p = 3.
```

Behauptung ist wahr für $p = 7$.
Behauptung ist wahr für $p = 11$.
Behauptung ist wahr für $p = 19$.
Behauptung ist wahr für $p = 23$.
Behauptung ist wahr für $p = 31$.
Behauptung ist wahr für $p = 43$.
Behauptung ist wahr für $p = 47$.
Behauptung ist wahr für $p = 59$.
Behauptung ist wahr für $p = 67$.
Behauptung ist wahr für $p = 71$.
Behauptung ist wahr für $p = 79$.
Behauptung ist wahr für $p = 83$.

Es handelt sich bis auf die Form der Primzahlen um die gleiche Schleife wie in (iii).

Die Behauptungen (iii) und (iv) sind richtig und liefern zusammen für prime p die Äquivalenz: $p \bmod 4 = 1 \Leftrightarrow$ es existieren m, n in \mathbb{N} mit $p = m^2 + n^2$.