[Jupyter Notebook](http://localhost:8888/tree?token=bd0fd89648d97bd7ac12834cd2e722a88854ceaf05269d3e)

Project\_Euler\_Julia(autosaved)Logout

Julia 0.6.1

Not Trusted

* [File](http://localhost:8888/notebooks/Downloads/Project_Euler_Julia.ipynb)
* [Edit](http://localhost:8888/notebooks/Downloads/Project_Euler_Julia.ipynb)
* [View](http://localhost:8888/notebooks/Downloads/Project_Euler_Julia.ipynb)
* [Insert](http://localhost:8888/notebooks/Downloads/Project_Euler_Julia.ipynb)
* [Cell](http://localhost:8888/notebooks/Downloads/Project_Euler_Julia.ipynb)
* [Kernel](http://localhost:8888/notebooks/Downloads/Project_Euler_Julia.ipynb)
* [Widgets](http://localhost:8888/notebooks/Downloads/Project_Euler_Julia.ipynb)
* [Help](http://localhost:8888/notebooks/Downloads/Project_Euler_Julia.ipynb)

Run

#Project Euler

**Multiples of 3 and 5   
Problem 1**

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Find the sum of all the multiples of 3 or 5 below 1000.

In [146]:

**function** Mult3or5\_below(num)

sum\_all **=** 0

**for** i **=** 1**:**num

**if** ((i**%**5 **==** 0) **||** (i**%**3 **==** 0))

sum\_all **+=** i

**end**

**end**

**return** sum\_all

**end**

Out[146]:

Mult3or5\_below (generic function with 1 method)

In [148]:

Mult3or5\_below(999)

Out[148]:

233168

**Even Fibonacci numbers   
Problem 2**

Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.

In [65]:

**function** SumFibEvens\_below(num)

fib\_num\_prev **=** 1

fib\_num **=** 1

fib\_sum **=** 0

answer **=** 0

**while** fib\_sum **<** num

fib\_sum **=** fib\_num\_prev **+** fib\_num

fib\_num\_prev **=** fib\_num

fib\_num **=** fib\_sum

**if** fib\_sum**%**2 **==** 0

answer **+=** fib\_sum

**end**

**end**

**return** answer

**end**

Out[65]:

SumFibEvens\_below (generic function with 1 method)

In [66]:

SumFibEvens\_below(4000000)

Out[66]:

4613732

**Largest prime factor   
Problem 3**

The prime factors of 13195 are 5, 7, 13 and 29.

What is the largest prime factor of the number 600851475143 ?

In [84]:

**function** LargestPrimeFactor(num)

factor\_dict **=** factor(num)

**return** maximum(factor\_dict)[1]

**end**

Out[84]:

LargestPrimeFactor (generic function with 1 method)

In [83]:

LargestPrimeFactor(600851475143)

Out[83]:

6857

**Largest palindrome product   
Problem 4**

A palindromic number reads the same both ways. The largest palindrome made from the product of two 2-digit numbers is 9009 = 91 × 99.

Find the largest palindrome made from the product of two 3-digit numbers.

In [39]:

**function** IsPalindrome(num1, num2)

product **=** string(num1 **\*** num2)

**return** product **==** reverse(product)

**end**

Out[39]:

IsPalindrome (generic function with 1 method)

In [40]:

IsPalindrome(91,99)

Out[40]:

true

In [41]:

**function** MaxPalindrome\_digits(num)

min\_num **=** 10**^**(num**-**1)

max\_num **=** int(repeat("9",num))

product **=** 0

answer **=** 0

**for** x **=** min\_num**:**max\_num

**for** y **=** min\_num**:**max\_num

product **=** x **\*** y

**if** IsPalindrome(x,y) **&&** product **>** answer

answer **=** product

**end**

**end**

**end**

**return** answer

**end**

Out[41]:

MaxPalindrome\_digits (generic function with 1 method)

In [42]:

MaxPalindrome\_digits(3)

Out[42]:

906609

**Smallest multiple   
Problem 5**

2520 is the smallest number that can be divided by each of the numbers from 1 to 10 without any remainder.

What is the smallest positive number that is evenly divisible by all of the numbers from 1 to 20?

In [101]:

**function** SmallestMult(num)

range\_list **=** Int64[]

**for** i **=** 1**:**num

push!(range\_list,i)

**end**

answer **=** lcm(range\_list**...**)

**return** answer

**end**

Out[101]:

SmallestMult (generic function with 1 method)

In [102]:

SmallestMult(20)

Out[102]:

232792560

In [176]:

lcm(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20)

Out[176]:

232792560

**Sum square difference   
Problem 6**

The sum of the squares of the first ten natural numbers is,

12 + 22 + ... + 102 = 385 The square of the sum of the first ten natural numbers is,

(1 + 2 + ... + 10)2 = 552 = 3025 Hence the difference between the sum of the squares of the first ten natural numbers and the square of the sum is 3025 − 385 = 2640.

Find the difference between the sum of the squares of the first one hundred natural numbers and the square of the sum.

In [218]:

**function** SumSqDif(num)

sumsquares **=** 0

runningsum **=** 0

**for** i **=** 1**:**num

sumsquares **+=** i**^**2

runningsum **+=** i

**end**

squaresum **=** runningsum**^**2

answer **=** squaresum **-** sumsquares

**return** answer

**end**

Out[218]:

SumSqDif (generic function with 1 method)

In [219]:

SumSqDif(100)

Out[219]:

25164150

**10001st prime   
Problem 7**

By listing the first six prime numbers: 2, 3, 5, 7, 11, and 13, we can see that the 6th prime is 13.

What is the 10001st prime number?

In [22]:

**function** Prime(n)

x **=** 2

i **=** 0

answer **=** 0

**while** i **<** n

**if** isprime(x)

answer **=** x

i **=** i **+** 1

**end**

x **=** x **+** 1

**end**

**return** answer

**end**

Out[22]:

Prime (generic function with 1 method)

In [23]:

Prime(10001)

Out[23]:

104743

**Largest product in a series   
Problem 8**

Find the greatest product of five consecutive digits in the 1000-digit number.

73167176531330624919225119674426574742355349194934 96983520312774506326239578318016984801869478851843 85861560789112949495459501737958331952853208805511 12540698747158523863050715693290963295227443043557 66896648950445244523161731856403098711121722383113 62229893423380308135336276614282806444486645238749 30358907296290491560440772390713810515859307960866 70172427121883998797908792274921901699720888093776 65727333001053367881220235421809751254540594752243 52584907711670556013604839586446706324415722155397 53697817977846174064955149290862569321978468622482 83972241375657056057490261407972968652414535100474 82166370484403199890008895243450658541227588666881 16427171479924442928230863465674813919123162824586 17866458359124566529476545682848912883142607690042 24219022671055626321111109370544217506941658960408 07198403850962455444362981230987879927244284909188 84580156166097919133875499200524063689912560717606 05886116467109405077541002256983155200055935729725 71636269561882670428252483600823257530420752963450

In [249]:

x **=** "7316717653133062491922511967442657474235534919493496983520312774506326239578318016984801869478851843858615607891129494954595017379583319528532088055111254069874715852386305071569329096329522744304355766896648950445244523161731856403098711121722383113622298934233803081353362766142828064444866452387493035890729629049156044077239071381051585930796086670172427121883998797908792274921901699720888093776657273330010533678812202354218097512545405947522435258490771167055601360483958644670632441572215539753697817977846174064955149290862569321978468622482839722413756570560574902614079729686524145351004748216637048440319989000889524345065854122758866688116427171479924442928230863465674813919123162824586178664583591245665294765456828489128831426076900422421902267105562632111110937054421750694165896040807198403850962455444362981230987879927244284909188845801561660979191338754992005240636899125607176060588611646710940507754100225698315520005593572972571636269561882670428252483600823257530420752963450"

Out[249]:

"7316717653133062491922511967442657474235534919493496983520312774506326239578318016984801869478851843858615607891129494954595017379583319528532088055111254069874715852386305071569329096329522744304355766896648950445244523161731856403098711121722383113622298934233803081353362766142828064444866452387493035890729629049156044077239071381051585930796086670172427121883998797908792274921901699720888093776657273330010533678812202354218097512545405947522435258490771167055601360483958644670632441572215539753697817977846174064955149290862569321978468622482839722413756570560574902614079729686524145351004748216637048440319989000889524345065854122758866688116427171479924442928230863465674813919123162824586178664583591245665294765456828489128831426076900422421902267105562632111110937054421750694165896040807198403850962455444362981230987879927244284909188845801561660979191338754992005240636899125607176060588611646710940507754100225698315520005593572972571636269561882670428252483600823257530420752963450"

In [318]:

**function** GrtstProdofFive(input\_string)

answer **=** 0

**for** i **=** 1**:**length(input\_string)**-**4

product **=** input\_string[i] **-** '0' *# subtract '0' to convert char into int*

**for** k **=** 1**:**4

product **=** product **\*** (input\_string[i**+**k] **-** '0')

**end**

**if** product **>** answer

answer **=** product

**end**

**end**

**return** answer

**end**

Out[318]:

GrtstProdofFive (generic function with 1 method)

In [320]:

GrtstProdofFive(x)

Out[320]:

40824

**Special Pythagorean triplet   
Problem 9**

A Pythagorean triplet is a set of three natural numbers, a < b < c, for which,

*a2 + b2 = c2*

For example, 32 + 42 = 9 + 16 = 25 = 52.

There exists exactly one Pythagorean triplet for which a + b + c = 1000. Find the product *abc*.

In [361]:

**function** PythagTriple()

answer **=** 0

**for** a **=** 1**:**1000

**for** b **=** a**:**1000

c **=** sqrt(a**^**2 **+** b**^**2)

**if** (a**+**b**+**c) **==** 1000

answer **=** a**\***b**\***c

**return** int(answer)

**end**

**end**

**end**

**end**

Out[361]:

PythagTriple (generic function with 1 method)

In [362]:

PythagTriple()

Out[362]:

31875000

**Summation of primes   
Problem 10**

The sum of the primes below 10 is 2 + 3 + 5 + 7 = 17.

Find the sum of all the primes below two million.

In [381]:

**function** Prime2(n)

x **=** 1

answer **=** 2

**while** x **<** n

**if** isprime(x)

answer **+=** x

**end**

x **=** x **+** 2

**end**

**return** answer

**end**

Out[381]:

Prime2 (generic function with 1 method)

In [384]:

Prime2(2000000)

Out[384]:

142913828922

**Largest product in a grid   
Problem 11**

In the 20×20 grid below, four numbers along a diagonal line have been marked in red.

08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08   
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00   
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65   
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91   
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80   
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50   
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70   
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21   
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72   
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95   
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92   
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57   
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58   
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40   
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66   
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69   
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36   
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16   
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54   
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48

The product of these numbers is 26 × 63 × 78 × 14 = 1788696.

What is the greatest product of four adjacent numbers in the same direction (up, down, left, right, or diagonally) in the 20×20 grid?

In [84]:

My2DArray **=** [[08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08

49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00

81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65

52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91

22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80

24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50

32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70

67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21

24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72

21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95

78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92

16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57

86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58

19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40

04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66

88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69

04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36

20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16

20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54

01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48]]

​

Out[84]:

20x20 Array{Int64,2}:

8 2 22 97 38 15 0 40 0 … 5 7 78 52 12 50 77 91 8

49 49 99 40 17 81 18 57 60 40 98 43 69 48 4 56 62 0

81 49 31 73 55 79 14 29 93 67 53 88 30 3 49 13 36 65

52 70 95 23 4 60 11 42 69 56 1 32 56 71 37 2 36 91

22 31 16 71 51 67 63 89 41 54 22 40 40 28 66 33 13 80

24 47 32 60 99 3 45 2 44 … 53 78 36 84 20 35 17 12 50

32 98 81 28 64 23 67 10 26 67 59 54 70 66 18 38 64 70

67 26 20 68 2 62 12 20 95 39 63 8 40 91 66 49 94 21

24 55 58 5 66 73 99 26 97 78 96 83 14 88 34 89 63 72

21 36 23 9 75 0 76 44 20 14 0 61 33 97 34 31 33 95

78 17 53 28 22 75 31 67 15 … 80 4 62 16 14 9 53 56 92

16 39 5 42 96 35 31 47 55 24 0 17 54 24 36 29 85 57

86 56 0 48 35 71 89 7 5 37 44 60 21 58 51 54 17 58

19 80 81 68 5 94 47 69 28 13 86 52 17 77 4 89 55 40

4 52 8 83 97 35 99 16 7 32 16 26 26 79 33 27 98 66

88 36 68 87 57 62 20 72 3 … 67 46 55 12 32 63 93 53 69

4 42 16 73 38 25 39 11 24 18 8 46 29 32 40 62 76 36

20 69 36 41 72 30 23 88 34 69 82 67 59 85 74 4 36 16

20 73 35 29 78 31 90 1 74 71 48 86 81 16 23 57 5 54

1 70 54 71 83 51 54 69 16 48 61 43 52 1 89 19 67 48

In [82]:

**function** FourProd(array\_2d)

enum **=** size(array\_2d)

transpose\_array **=** array\_2d**'**

rotate\_array **=** rotl90(array\_2d)

answer **=** 0

**for** x **=** 1**:**enum[1]**-**4

**for** y **=** 1**:**enum[2]**-**4

product1 **=** array\_2d[x,y]

product2 **=** transpose\_array[x,y]

product3 **=** array\_2d[x,y]

product4 **=** rotate\_array[x,y]

**for** z **=** 1**:**3

h **=** x **+** z

v **=** y **+** z

product1 **\*=** array\_2d[h,y] *# search for up/down product*

product2 **\*=** transpose\_array[h,y] *# search for left/right product*

product3 **\*=** array\_2d[h,v] *# search for diagonal \ product*

product4 **\*=** rotate\_array[h,v] *# search for diagonal / product*

**end**

max\_product **=** max(product1, product2, product3, product4)

**if** max\_product **>** answer

answer **=** max\_product

**end**

**end**

**end**

**return** answer

**end**

Out[82]:

FourProd (generic function with 1 method)

In [85]:

FourProd(My2DArray)

Out[85]:

70600674

**Highly divisible triangular number   
Problem 12**

The sequence of triangle numbers is generated by adding the natural numbers. So the 7th triangle number would be 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28. The first ten terms would be:

1, 3, 6, 10, 15, 21, 28, 36, 45, 55, ...

Let us list the factors of the first seven triangle numbers:

1: 1   
3: 1,3   
6: 1,2,3,6   
10: 1,2,5,10   
15: 1,3,5,15   
21: 1,3,7,21   
28: 1,2,4,7,14,28

We can see that 28 is the first triangle number to have over five divisors.

What is the value of the first triangle number to have over five hundred divisors?

In [179]:

**function** NumDivisors(n)

answer **=** 0

x **=** floor(sqrt(n))

**for** i**=**1**:**x

**if** n**%**i **==** 0

answer **+=** 2

**end**

**end**

**if** n**/**x **==** x

answer **-=** 1

**end**

**return** answer

**end**

​

**function** TriangleDivisors(n)

count\_div **=** 0

counter **=** 1

answer **=** 0

**while** count\_div **<** n

answer **+=** counter

count\_div **=** NumDivisors(answer)

counter **+=** 1

**end**

**return** answer

**end**

Out[179]:

TriangleDivisors (generic function with 1 method)

In [182]:

TriangleDivisors(500)

Out[182]:

76576500

**Large sum   
Problem 13**

Work out the first ten digits of the sum of the following one-hundred 50-digit numbers.

37107287533902102798797998220837590246510135740250 46376937677490009712648124896970078050417018260538 74324986199524741059474233309513058123726617309629 91942213363574161572522430563301811072406154908250 23067588207539346171171980310421047513778063246676 89261670696623633820136378418383684178734361726757 28112879812849979408065481931592621691275889832738 44274228917432520321923589422876796487670272189318 47451445736001306439091167216856844588711603153276 70386486105843025439939619828917593665686757934951 62176457141856560629502157223196586755079324193331 64906352462741904929101432445813822663347944758178 92575867718337217661963751590579239728245598838407 58203565325359399008402633568948830189458628227828 80181199384826282014278194139940567587151170094390 35398664372827112653829987240784473053190104293586 86515506006295864861532075273371959191420517255829 71693888707715466499115593487603532921714970056938 54370070576826684624621495650076471787294438377604 53282654108756828443191190634694037855217779295145 36123272525000296071075082563815656710885258350721 45876576172410976447339110607218265236877223636045 17423706905851860660448207621209813287860733969412 81142660418086830619328460811191061556940512689692 51934325451728388641918047049293215058642563049483 62467221648435076201727918039944693004732956340691 15732444386908125794514089057706229429197107928209 55037687525678773091862540744969844508330393682126 18336384825330154686196124348767681297534375946515 80386287592878490201521685554828717201219257766954 78182833757993103614740356856449095527097864797581 16726320100436897842553539920931837441497806860984 48403098129077791799088218795327364475675590848030 87086987551392711854517078544161852424320693150332 59959406895756536782107074926966537676326235447210 69793950679652694742597709739166693763042633987085 41052684708299085211399427365734116182760315001271 65378607361501080857009149939512557028198746004375 35829035317434717326932123578154982629742552737307 94953759765105305946966067683156574377167401875275 88902802571733229619176668713819931811048770190271 25267680276078003013678680992525463401061632866526 36270218540497705585629946580636237993140746255962 24074486908231174977792365466257246923322810917141 91430288197103288597806669760892938638285025333403 34413065578016127815921815005561868836468420090470 23053081172816430487623791969842487255036638784583 11487696932154902810424020138335124462181441773470 63783299490636259666498587618221225225512486764533 67720186971698544312419572409913959008952310058822 95548255300263520781532296796249481641953868218774 76085327132285723110424803456124867697064507995236 37774242535411291684276865538926205024910326572967 23701913275725675285653248258265463092207058596522 29798860272258331913126375147341994889534765745501 18495701454879288984856827726077713721403798879715 38298203783031473527721580348144513491373226651381 34829543829199918180278916522431027392251122869539 40957953066405232632538044100059654939159879593635 29746152185502371307642255121183693803580388584903 41698116222072977186158236678424689157993532961922 62467957194401269043877107275048102390895523597457 23189706772547915061505504953922979530901129967519 86188088225875314529584099251203829009407770775672 11306739708304724483816533873502340845647058077308 82959174767140363198008187129011875491310547126581 97623331044818386269515456334926366572897563400500 42846280183517070527831839425882145521227251250327 55121603546981200581762165212827652751691296897789 32238195734329339946437501907836945765883352399886 75506164965184775180738168837861091527357929701337 62177842752192623401942399639168044983993173312731 32924185707147349566916674687634660915035914677504 99518671430235219628894890102423325116913619626622 73267460800591547471830798392868535206946944540724 76841822524674417161514036427982273348055556214818 97142617910342598647204516893989422179826088076852 87783646182799346313767754307809363333018982642090 10848802521674670883215120185883543223812876952786 71329612474782464538636993009049310363619763878039 62184073572399794223406235393808339651327408011116 66627891981488087797941876876144230030984490851411 60661826293682836764744779239180335110989069790714 85786944089552990653640447425576083659976645795096 66024396409905389607120198219976047599490197230297 64913982680032973156037120041377903785566085089252 16730939319872750275468906903707539413042652315011 94809377245048795150954100921645863754710598436791 78639167021187492431995700641917969777599028300699 15368713711936614952811305876380278410754449733078 40789923115535562561142322423255033685442488917353 44889911501440648020369068063960672322193204149535 41503128880339536053299340368006977710650566631954 81234880673210146739058568557934581403627822703280 82616570773948327592232845941706525094512325230608 22918802058777319719839450180888072429661980811197 77158542502016545090413245809786882778948721859617 72107838435069186155435662884062257473692284509516 20849603980134001723930671666823555245252804609722 53503534226472524250874054075591789781264330331690

In [243]:

LargeNumString **=** "37107287533902102798797998220837590246510135740250

46376937677490009712648124896970078050417018260538

74324986199524741059474233309513058123726617309629

91942213363574161572522430563301811072406154908250

23067588207539346171171980310421047513778063246676

89261670696623633820136378418383684178734361726757

28112879812849979408065481931592621691275889832738

44274228917432520321923589422876796487670272189318

47451445736001306439091167216856844588711603153276

70386486105843025439939619828917593665686757934951

62176457141856560629502157223196586755079324193331

64906352462741904929101432445813822663347944758178

92575867718337217661963751590579239728245598838407

58203565325359399008402633568948830189458628227828

80181199384826282014278194139940567587151170094390

35398664372827112653829987240784473053190104293586

86515506006295864861532075273371959191420517255829

71693888707715466499115593487603532921714970056938

54370070576826684624621495650076471787294438377604

53282654108756828443191190634694037855217779295145

36123272525000296071075082563815656710885258350721

45876576172410976447339110607218265236877223636045

17423706905851860660448207621209813287860733969412

81142660418086830619328460811191061556940512689692

51934325451728388641918047049293215058642563049483

62467221648435076201727918039944693004732956340691

15732444386908125794514089057706229429197107928209

55037687525678773091862540744969844508330393682126

18336384825330154686196124348767681297534375946515

80386287592878490201521685554828717201219257766954

78182833757993103614740356856449095527097864797581

16726320100436897842553539920931837441497806860984

48403098129077791799088218795327364475675590848030

87086987551392711854517078544161852424320693150332

59959406895756536782107074926966537676326235447210

69793950679652694742597709739166693763042633987085

41052684708299085211399427365734116182760315001271

65378607361501080857009149939512557028198746004375

35829035317434717326932123578154982629742552737307

94953759765105305946966067683156574377167401875275

88902802571733229619176668713819931811048770190271

25267680276078003013678680992525463401061632866526

36270218540497705585629946580636237993140746255962

24074486908231174977792365466257246923322810917141

91430288197103288597806669760892938638285025333403

34413065578016127815921815005561868836468420090470

23053081172816430487623791969842487255036638784583

11487696932154902810424020138335124462181441773470

63783299490636259666498587618221225225512486764533

67720186971698544312419572409913959008952310058822

95548255300263520781532296796249481641953868218774

76085327132285723110424803456124867697064507995236

37774242535411291684276865538926205024910326572967

23701913275725675285653248258265463092207058596522

29798860272258331913126375147341994889534765745501

18495701454879288984856827726077713721403798879715

38298203783031473527721580348144513491373226651381

34829543829199918180278916522431027392251122869539

40957953066405232632538044100059654939159879593635

29746152185502371307642255121183693803580388584903

41698116222072977186158236678424689157993532961922

62467957194401269043877107275048102390895523597457

23189706772547915061505504953922979530901129967519

86188088225875314529584099251203829009407770775672

11306739708304724483816533873502340845647058077308

82959174767140363198008187129011875491310547126581

97623331044818386269515456334926366572897563400500

42846280183517070527831839425882145521227251250327

55121603546981200581762165212827652751691296897789

32238195734329339946437501907836945765883352399886

75506164965184775180738168837861091527357929701337

62177842752192623401942399639168044983993173312731

32924185707147349566916674687634660915035914677504

99518671430235219628894890102423325116913619626622

73267460800591547471830798392868535206946944540724

76841822524674417161514036427982273348055556214818

97142617910342598647204516893989422179826088076852

87783646182799346313767754307809363333018982642090

10848802521674670883215120185883543223812876952786

71329612474782464538636993009049310363619763878039

62184073572399794223406235393808339651327408011116

66627891981488087797941876876144230030984490851411

60661826293682836764744779239180335110989069790714

85786944089552990653640447425576083659976645795096

66024396409905389607120198219976047599490197230297

64913982680032973156037120041377903785566085089252

16730939319872750275468906903707539413042652315011

94809377245048795150954100921645863754710598436791

78639167021187492431995700641917969777599028300699

15368713711936614952811305876380278410754449733078

40789923115535562561142322423255033685442488917353

44889911501440648020369068063960672322193204149535

41503128880339536053299340368006977710650566631954

81234880673210146739058568557934581403627822703280

82616570773948327592232845941706525094512325230608

22918802058777319719839450180888072429661980811197

77158542502016545090413245809786882778948721859617

72107838435069186155435662884062257473692284509516

20849603980134001723930671666823555245252804609722

53503534226472524250874054075591789781264330331690"

Out[243]:

"37107287533902102798797998220837590246510135740250\n46376937677490009712648124896970078050417018260538\n74324986199524741059474233309513058123726617309629\n91942213363574161572522430563301811072406154908250\n23067588207539346171171980310421047513778063246676\n89261670696623633820136378418383684178734361726757\n28112879812849979408065481931592621691275889832738\n44274228917432520321923589422876796487670272189318\n47451445736001306439091167216856844588711603153276\n70386486105843025439939619828917593665686757934951\n62176457141856560629502157223196586755079324193331\n64906352462741904929101432445813822663347944758178\n92575867718337217661963751590579239728245598838407\n58203565325359399008402633568948830189458628227828\n80181199384826282014278194139940567587151170094390\n35398664372827112653829987240784473053190104293586\n86515506006295864861532075273371959191420517255829\n71693888707715466499115593487603532921714970056938\n54370070576826684624621495650076471787294438377604\n53282654108756828443191190634694037855217779295145\n36123272525000296071075082563815656710885258350721\n45876576172410976447339110607218265236877223636045\n17423706905851860660448207621209813287860733969412\n81142660418086830619328460811191061556940512689692\n51934325451728388641918047049293215058642563049483\n62467221648435076201727918039944693004732956340691\n15732444386908125794514089057706229429197107928209\n55037687525678773091862540744969844508330393682126\n18336384825330154686196124348767681297534375946515\n80386287592878490201521685554828717201219257766954\n78182833757993103614740356856449095527097864797581\n16726320100436897842553539920931837441497806860984\n48403098129077791799088218795327364475675590848030\n87086987551392711854517078544161852424320693150332\n59959406895756536782107074926966537676326235447210\n69793950679652694742597709739166693763042633987085\n41052684708299085211399427365734116182760315001271\n65378607361501080857009149939512557028198746004375\n35829035317434717326932123578154982629742552737307\n94953759765105305946966067683156574377167401875275\n88902802571733229619176668713819931811048770190271\n25267680276078003013678680992525463401061632866526\n36270218540497705585629946580636237993140746255962\n24074486908231174977792365466257246923322810917141\n91430288197103288597806669760892938638285025333403\n34413065578016127815921815005561868836468420090470\n23053081172816430487623791969842487255036638784583\n11487696932154902810424020138335124462181441773470\n63783299490636259666498587618221225225512486764533\n67720186971698544312419572409913959008952310058822\n95548255300263520781532296796249481641953868218774\n76085327132285723110424803456124867697064507995236\n37774242535411291684276865538926205024910326572967\n23701913275725675285653248258265463092207058596522\n29798860272258331913126375147341994889534765745501\n18495701454879288984856827726077713721403798879715\n38298203783031473527721580348144513491373226651381\n34829543829199918180278916522431027392251122869539\n40957953066405232632538044100059654939159879593635\n29746152185502371307642255121183693803580388584903\n41698116222072977186158236678424689157993532961922\n62467957194401269043877107275048102390895523597457\n23189706772547915061505504953922979530901129967519\n86188088225875314529584099251203829009407770775672\n11306739708304724483816533873502340845647058077308\n82959174767140363198008187129011875491310547126581\n97623331044818386269515456334926366572897563400500\n42846280183517070527831839425882145521227251250327\n55121603546981200581762165212827652751691296897789\n32238195734329339946437501907836945765883352399886\n75506164965184775180738168837861091527357929701337\n62177842752192623401942399639168044983993173312731\n32924185707147349566916674687634660915035914677504\n99518671430235219628894890102423325116913619626622\n73267460800591547471830798392868535206946944540724\n76841822524674417161514036427982273348055556214818\n97142617910342598647204516893989422179826088076852\n87783646182799346313767754307809363333018982642090\n10848802521674670883215120185883543223812876952786\n71329612474782464538636993009049310363619763878039\n62184073572399794223406235393808339651327408011116\n66627891981488087797941876876144230030984490851411\n60661826293682836764744779239180335110989069790714\n85786944089552990653640447425576083659976645795096\n66024396409905389607120198219976047599490197230297\n64913982680032973156037120041377903785566085089252\n16730939319872750275468906903707539413042652315011\n94809377245048795150954100921645863754710598436791\n78639167021187492431995700641917969777599028300699\n15368713711936614952811305876380278410754449733078\n40789923115535562561142322423255033685442488917353\n44889911501440648020369068063960672322193204149535\n41503128880339536053299340368006977710650566631954\n81234880673210146739058568557934581403627822703280\n82616570773948327592232845941706525094512325230608\n22918802058777319719839450180888072429661980811197\n77158542502016545090413245809786882778948721859617\n72107838435069186155435662884062257473692284509516\n20849603980134001723930671666823555245252804609722\n53503534226472524250874054075591789781264330331690"

In [245]:

num\_array **=** split(LargeNumString, "\n")

A **=** [ float(num\_array[i]) **for** i**=**1**:**length(num\_array) ]

answer **=** sum(A)

Out[245]:

5.537376230390877e51

**Longest Collatz sequence   
Problem 14**

The following iterative sequence is defined for the set of positive integers:

n → n/2 (n is even)  
n → 3n + 1 (n is odd)

Using the rule above and starting with 13, we generate the following sequence:

13 → 40 → 20 → 10 → 5 → 16 → 8 → 4 → 2 → 1

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not been proved yet (Collatz Problem), it is thought that all starting numbers finish at 1.

Which starting number, under one million, produces the longest chain?

**NOTE: Once the chain starts the terms are allowed to go above one million.**

In [28]:

**function** CollatzSeq(n)

A **=** [n]

**while** n **>** 1

n **=** iseven(n)**?** int(n**/**2) **:** 3**\***n **+** 1

push!(A, n)

**end**

**return** A

**end**

Out[28]:

CollatzSeq (generic function with 1 method)

In [29]:

begin\_time **=** time()

​

answer **=** []

seq **=** []

​

**for** i **=** 700000**:**1000000

seq **=** CollatzSeq(i)

length(seq) **>=** length(answer)**?** answer **=** seq **:** pass

**end**

​

print("runtime: ", time() **-** begin\_time)

**return** answer[1]

runtime: 6.568037033081055

Out[29]:

837799

In [30]:

begin\_time **=** time()

​

seqs **=** map((x) **->** length(CollatzSeq(x)), 700000**:**1000000)

​

print("runtime: ",time() **-** begin\_time)

**return** 700000**+**indmax(seqs)**-**1

runtime: 3.7657041549682617

Out[30]:

837799

**Lattice paths   
Problem 15**

Starting in the top left corner of a 2×2 grid, and only being able to move to the right and down, there are exactly 6 routes to the bottom right corner.

How many such routes are there through a 20×20 grid?

In [231]:

**function** LatticeRecursive(x, y)

**if** x **==** 1.0 **&&** y **==** 1.0

**return** 2.0

**elseif** x **==** 1.0

**return** 1.0 **+** LatticeRecursive(x, y**-**1.0)

**elseif** y **==** 1.0

**return** 1.0 **+** LatticeRecursive(x**-**1.0, y)

**else**

**return** LatticeRecursive(x**-**1.0, y) **+** LatticeRecursive(x, y**-**1.0)

**end**

**end**

Out[231]:

LatticeRecursive (generic function with 1 method)

In [222]:

**function** Lattice(x, y)

answer **=** zeros(x**+**1, y**+**1)

**for** i **=** 1**:**y**+**1

**for** j **=** 1**:**x**+**1

**if** i **==** 1.0 **||** j **==** 1.0

answer[i, j] **=** 1.0

**else**

answer[i, j] **=** answer[i**-**1, j] **+** answer[i, j**-**1]

**end**

**end**

**end**

**return** answer[x**+**1, y**+**1]

**end**

Out[222]:

Lattice (generic function with 1 method)

In [246]:

big(int((Lattice(20,20))))

Out[246]:

137846528820

**Power digit sum   
Problem 16**

215 = 32768 and the sum of its digits is 3 + 2 + 7 + 6 + 8 = 26.

What is the sum of the digits of the number 21000?

In [1]:

**function** PowerDigit(x)

n **=** string(big(2)**^**x)

answer **=** 0

**for** i**=** 1**:**length(n)

answer **+=** int(n[i] **-** '0') *# subtract '0' to convert char into int*

**end**

**return** answer

**end**

Out[1]:

PowerDigit (generic function with 1 method)

In [2]:

PowerDigit(1000)

Out[2]:

1366

**Number letter counts   
Problem 17**

If the numbers 1 to 5 are written out in words: one, two, three, four, five, then there are 3 + 3 + 5 + 4 + 4 = 19 letters used in total.

If all the numbers from 1 to 1000 (one thousand) inclusive were written out in words, how many letters would be used?

*NOTE:* Do not count spaces or hyphens. For example, 342 (three hundred and forty-two) contains 23 letters and 115 (one hundred and fifteen) contains 20 letters. The use of "and" when writing out numbers is in compliance with British usage.

In [96]:

**function** LetterCounts(n)

l **=** {

1 **=>** "one",

2 **=>** "two",

3 **=>** "three",

4 **=>** "four",

5 **=>** "five",

6 **=>** "six",

7 **=>** "seven",

8 **=>** "eight",

9 **=>** "nine",

10 **=>** "ten",

11 **=>** "eleven",

12 **=>** "twelve",

13 **=>** "thirteen",

14 **=>** "fourteen",

15 **=>** "fifteen",

16 **=>** "sixteen",

17 **=>** "seventeen",

18 **=>** "eighteen",

19 **=>** "nineteen",

20 **=>** "twenty",

30 **=>** "thirty",

40 **=>** "forty",

50 **=>** "fifty",

60 **=>** "sixty",

70 **=>** "seventy",

80 **=>** "eighty",

90 **=>** "ninety",

1000 **=>** "onethousand"

}

*# adds:*

*# 100, 200, 300...*

*# 101, 102, 103...*

*# 201, 202, 203...*

*# 110, 120, 130...*

**for** h**=**1**:**9

l[(h**\***100)] **=** l[h] **\*** "hundred"

**for** ones**=**1**:**9

l[(h**\***100) **+** ones] **=** l[h**\***100] **\*** "and" **\*** l[ones]

l[(h**\***100) **+** (ones**\***10)] **=** l[h**\***100] **\*** "and" **\*** l[ones**\***10]

**end**

**end**

​

*# adds:*

*# 21, 22, 23...*

*# 31, 32, 33...*

*# 111, 112, 113...*

*# 121, 122, 123...*

*# 231, 232, 233...*

**for** ones**=**1**:**9

**for** tens**=**2**:**9

l[(tens**\***10) **+** ones] **=** l[tens**\***10] **\*** l[ones]

**for** h**=**1**:**9

l[(h**\***100) **+** (tens**\***10) **+** ones] **=** l[h**\***100] **\*** "and" **\*** l[tens**\***10] **\*** l[ones]

l[(h**\***100) **+** (10) **+** ones] **=** l[h**\***100] **\*** "and" **\*** l[10 **+** ones]

**end**

**end**

**end**

**return** length(l[n])

**end**

​

answer **=** 0

​

**for** x**=**1**:**1000

answer **+=** LetterCounts(x)

**end**

​

**return** answer

Out[96]:

21124

**Maximum path sum I   
Problem 18**

By starting at the top of the triangle below and moving to adjacent numbers on the row below, the maximum total from top to bottom is 23.

3

7 4

2 4 6

8 5 9 3

That is, 3 + 7 + 4 + 9 = 23.

Find the maximum total from top to bottom of the triangle below:

75

95 64

17 47 82

18 35 87 10

20 04 82 47 65

19 01 23 75 03 34

88 02 77 73 07 63 67

99 65 04 28 06 16 70 92

41 41 26 56 83 40 80 70 33

41 48 72 33 47 32 37 16 94 29

53 71 44 65 25 43 91 52 97 51 14

70 11 33 28 77 73 17 78 39 68 17 57

91 71 52 38 17 14 91 43 58 50 27 29 48

63 66 04 68 89 53 67 30 73 16 69 87 40 31

04 62 98 27 23 09 70 98 73 93 38 53 60 04 23

*NOTE:* As there are only 16384 routes, it is possible to solve this problem by trying every route. However, Problem 67, is the same challenge with a triangle containing one-hundred rows; it cannot be solved by brute force, and requires a clever method! ;o)

In [247]:

**function** MakeTriangle()

triangle **=** zeros(Int64, 15, 15)

triangle[1,1] **=** 75

triangle[2,1**:**2] **=** [95 64]

triangle[3,1**:**3] **=** [17 47 82]

triangle[4,1**:**4] **=** [18 35 87 10]

triangle[5,1**:**5] **=** [20 04 82 47 65]

triangle[6,1**:**6] **=** [19 01 23 75 03 34]

triangle[7,1**:**7] **=** [88 02 77 73 07 63 67]

triangle[8,1**:**8] **=** [99 65 04 28 06 16 70 92]

triangle[9,1**:**9] **=** [41 41 26 56 83 40 80 70 33]

triangle[10,1**:**10] **=** [41 48 72 33 47 32 37 16 94 29]

triangle[11,1**:**11] **=** [53 71 44 65 25 43 91 52 97 51 14]

triangle[12,1**:**12] **=** [70 11 33 28 77 73 17 78 39 68 17 57]

triangle[13,1**:**13] **=** [91 71 52 38 17 14 91 43 58 50 27 29 48]

triangle[14,1**:**14] **=** [63 66 04 68 89 53 67 30 73 16 69 87 40 31]

triangle[15,1**:**15] **=** [04 62 98 27 23 09 70 98 73 93 38 53 60 04 23]

**return** triangle

**end**

​

**function** MaxPathSum(triangle)

sums **=** zeros(Int64, size(triangle))

sums[1,1] **=** triangle[1,1]

rows, cols **=** size(triangle)

**for** row **=** 2**:**rows

**for** col **=** 1**:**row

val **=** triangle[row,col]

**if** col **==** 1

sums[row,col] **=** val **+** sums[row**-**1,col]

**else**

sums[row,col] **=** val **+** max(sums[row**-**1,col], sums[row**-**1,col**-**1])

**end**

**end**

**end**

**return** maximum(sums[**end**,1**:end**])

**end**

Out[247]:

MaxPathSum (generic function with 2 methods)

In [248]:

MaxPathSum(MakeTriangle())

Out[248]:

1074

**Maximum path sum II   
Problem 67**

By starting at the top of the triangle below and moving to adjacent numbers on the row below, the maximum total from top to bottom is 23.

3

7 4

2 4 6

8 5 9 3

That is, 3 + 7 + 4 + 9 = 23.

Find the maximum total from top to bottom in triangle.txt, a 15K text file containing a triangle with one-hundred rows.

*NOTE:* This is a much more difficult version of Problem 18. It is not possible to try every route to solve this problem, as there are 299 altogether! If you could check one trillion (1012) routes every second it would take over twenty billion years to check them all. There is an efficient algorithm to solve it. ;o)

In [385]:

**function** MakeBigTriangle()

big\_triangle **=** open("triangle.txt")

lines **=** readlines(big\_triangle)

rows **=** length(lines)

triangle **=** zeros(Int64, rows, rows)

**for** x**=**1**:**rows

triangle[x,1**:**x] **=** [int(split(lines[x], ' '))]

**end**

**return** triangle

**end**

Out[385]:

MakeBigTriangle (generic function with 1 method)

In [386]:

MaxPathSum(MakeBigTriangle())

Out[386]:

7273

**Counting Sundays   
Problem 19**

You are given the following information, but you may prefer to do some research for yourself.

1 Jan 1900 was a Monday.

* Thirty days has September,
* April, June and November.
* All the rest have thirty-one,
* Saving February alone,
* Which has twenty-eight, rain or shine.
* And on leap years, twenty-nine.

A leap year occurs on any year evenly divisible by 4, but not on a century unless it is divisible by 400. How many Sundays fell on the first of the month during the twentieth century (1 Jan 1901 to 31 Dec 2000)?

In [44]:

first\_sundays **=** 0

months **=** [

31, *#jan*

28, *#feb*

31, *#mar*

30, *#apr*

31, *#may*

30, *#jun*

31, *#jul*

31, *#aug*

30, *#sep*

31, *#oct*

30, *#nov*

31 *#dec*

]

day **=** 3 *# jan 1, 1901 is a tue (3)*

m **=** 1 *# m = month in year*

**for** m\_range **=** 1**:**1200 *# m\_range = total months elapsed*

yr **=** floor((m\_range **-** 1) **/** 12) **+** 1901

**if** day **==** 1

*# print("$m/$yr: $day\n")*

first\_sundays **+=** 1

**end**

**if** (m **==** 2) **&&** (yr **%** 4 **==** 0) **&&** ((yr **%** 100 **!=** 0) **||** (yr **%** 400 **==** 0))

day **+=** 1 *# leap year*

**end**

drift **=** months[m] **-** 28

day **=** ((day **+** drift) **%** 7 **==** 0) **?** 7 **:** (day **+** drift) **%** 7

m **=** (m\_range **%** 12 **==** 0) **?** 1 **:** m **+** 1

**end**

**return** first\_sundays

Out[44]:

171

**Factorial digit sum   
Problem 20**

n! means n × (n − 1) × ... × 3 × 2 × 1

For example, 10! = 10 × 9 × ... × 3 × 2 × 1 = 3628800, and the sum of the digits in the number 10! is 3 + 6 + 2 + 8 + 8 + 0 + 0 = 27.

Find the sum of the digits in the number 100!

In [45]:

sum(digits(factorial(big(100)))) *# digits does not work on BigInt*

no method convert(Type{Unsigned},BigInt)

at In[45]:1

in digits at intfuncs.jl:289

in digits at intfuncs.jl:288

In [48]:

**function** FactDigitSum(n)

fact **=** string(reduce(**\***, 1**:**big(n)))

answer **=** 0

**for** i**=** 1**:**length(fact)

answer **+=** int(fact[i] **-** '0') *# subtract '0' to convert char into int*

**end**

**return** answer

**end**

Out[48]:

FactDigitSum (generic function with 1 method)

In [49]:

FactDigitSum(100)

Out[49]:

648

**Amicable numbers   
Problem 21**

Let d(n) be defined as the sum of proper divisors of n (numbers less than n which divide evenly into n). If d(a) = b and d(b) = a, where a ≠ b, then a and b are an amicable pair and each of a and b are called amicable numbers.

For example, the proper divisors of 220 are 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 and 110; therefore d(220) = 284. The proper divisors of 284 are 1, 2, 4, 71 and 142; so d(284) = 220.

Evaluate the sum of all the amicable numbers under 10000.

In [27]:

**function** ProperDivisor(n)

divisors **=** [1]

n\_sqrt **=** floor(sqrt(n))

**for** i **=** 2**:**n\_sqrt

**if** n **%** i **==** 0

**if** n**/**i **==** i

push!(divisors, i)

**else**

push!(divisors, i)

push!(divisors, n**/**i)

**end**

**end**

**end**

**return** divisors

**end**

​

**function** Amicable(n)

sum1 **=** sum(ProperDivisor(n))

sum2 **=** sum(ProperDivisor(sum1))

**return** sum2 **==** n **&&** sum1 **!=** sum2

**end**

​

amicables **=** Int64[]

**for** num **=** 2**:**10000

**if** Amicable(num)

print(num, " ")

push!(amicables, num)

**end**

**end**

​

**return** sum(amicables)

220 284 1184 1210 2620 2924 5020 5564 6232 6368

Out[27]:

31626

**Names scores   
Problem 22**

Using names.txt, a 46K text file containing over five-thousand first names, begin by sorting it into alphabetical order. Then working out the alphabetical value for each name, multiply this value by its alphabetical position in the list to obtain a name score.

For example, when the list is sorted into alphabetical order, COLIN, which is worth 3 + 15 + 12 + 9 + 14 = 53, is the 938th name in the list. So, COLIN would obtain a score of 938 × 53 = 49714.

What is the total of all the name scores in the file?

In [26]:

alpha\_score **=** Dict{Char,Int64}()

**for** character **=** 1**:**26

alpha\_score[(char(character **+** 64))] **=** character

**end**

​

name\_list **=** open(readall, "names.txt")

names **=** sort(split(replace(name\_list, '"',""), ","))

​

score **=** 0

**for** name **=** 1**:**length(names)

name\_score **=** 0

**for** letter **=** 1**:**length(names[name])

name\_score **+=** alpha\_score[names[name][letter]]

**end**

name\_score **\*=** name

score **+=** name\_score

**end**

​

**return** score

Out[26]:

871198282

**Non-abundant sums   
Problem 23**

A perfect number is a number for which the sum of its proper divisors is exactly equal to the number. For example, the sum of the proper divisors of 28 would be 1 + 2 + 4 + 7 + 14 = 28, which means that 28 is a perfect number.

A number n is called deficient if the sum of its proper divisors is less than n and it is called abundant if this sum exceeds n.

As 12 is the smallest abundant number, 1 + 2 + 3 + 4 + 6 = 16, the smallest number that can be written as the sum of two abundant numbers is 24. By mathematical analysis, it can be shown that all integers greater than 28123 can be written as the sum of two abundant numbers. However, this upper limit cannot be reduced any further by analysis even though it is known that the greatest number that cannot be expressed as the sum of two abundant numbers is less than this limit.

Find the sum of all the positive integers which cannot be written as the sum of two abundant numbers.

In [32]:

begin\_time **=** time()

​

abundant\_nums **=** [12]

**for** n **=** 13**:**20161

**if** sum(ProperDivisor(n)) **>** n

push!(abundant\_nums, n)

**end**

**end**

​

answer **=** [1**:**23]

**for** i **=** 24**:**20161

found **=** false

**for** num **in** abundant\_nums

**if** (i **-** num) **in** abundant\_nums

found **=** true

**break**

**end**

**end**

**if** found **==** false

answer **=** push!(answer, i)

**end**

**end**

​

print("runtime: ", time() **-** begin\_time)

**return** sum(answer)

runtime: 129.338219165802

Out[32]:

4179871

In [31]:

*# scratchpad*

ab\_num\_sums **=** Array(Int64, 1)

**for** num **in** abundant\_nums

sums\_array **=** pmap((x) **->** x**+**num, abundant\_nums)

ab\_num\_sums **=** cat(1, ab\_num\_sums, sums\_array)

**end**

​

answer **=** [1]

**for** i **=** 2**:**28123

answer **=** sum(push!(i **in** ab\_num\_sums**?** i **:** pass))

**end**

**return** answer

abundant\_nums not defined

at In[31]:6

in anonymous at no file

**Lexicographic permutations   
Problem 24**

A permutation is an ordered arrangement of objects. For example, 3124 is one possible permutation of the digits 1, 2, 3 and 4. If all of the permutations are listed numerically or alphabetically, we call it lexicographic order. The lexicographic permutations of 0, 1 and 2 are:

012 021 102 120 201 210

What is the millionth lexicographic permutation of the digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9?

In [41]:

sum(map(x **->** 10**^**(x[1]**-**1) **\*** x[2], enumerate(nthperm([0**:**9], 1000000))))

Out[41]:

645193872

**1000-digit Fibonacci number   
Problem 25**

The Fibonacci sequence is defined by the recurrence relation:

Fn = Fn−1 + Fn−2, where F1 = 1 and F2 = 1.

Hence the first 12 terms will be:   
F1 = 1   
F2 = 1   
F3 = 2   
F4 = 3   
F5 = 5   
F6 = 8   
F7 = 13   
F8 = 21   
F9 = 34   
F10 = 55   
F11 = 89   
F12 = 144

The 12th term, F12, is the first term to contain three digits.

What is the first term in the Fibonacci sequence to contain 1000 digits?

In [13]:

**function** Fib\_num\_digits(n)

fib\_seq **=** ["1","1"]

**while** length(fib\_seq[**end**]) **<** n

push!(fib\_seq, string(BigInt(fib\_seq[**end-**1]) **+** BigInt(fib\_seq[**end**])))

*# print(fib\_seq[end], " ")*

**end**

**return** length(fib\_seq)

**end**

Out[13]:

Fib\_num\_digits (generic function with 1 method)

In [17]:

begin\_time **=** time()

answer **=** Fib\_num\_digits(1000)

print("runtime: ", time() **-** begin\_time)

**return** answer

runtime: 0.20959210395812988

Out[17]:

4782

**Reciprocal cycles   
Problem 26**

A unit fraction contains 1 in the numerator. The decimal representation of the unit fractions with denominators 2 to 10 are given:

1/2 = 0.5

1/3 = 0.(3)

1/4 = 0.25

1/5 = 0.2

1/6 = 0.1(6)

1/7 = 0.(142857)

1/8 = 0.125

1/9 = 0.(1)

1/10 = 0.1

Where 0.1(6) means 0.166666..., and has a 1-digit recurring cycle. It can be seen that 1/7 has a 6-digit recurring cycle.

Find the value of d < 1000 for which 1/d contains the longest recurring cycle in its decimal fraction part.

In [15]:

**function** ReciprocalCycles(n)

seq\_len **=** 0

answer **=** 0

​

**for** i **=** 2**:**n

remainders **=** zeros(Int64,i)

value **=** 1

len **=** 0

**while** value **<** i

value **\*=** 10

**end**

**while** value **!=** 0

**if** value **in** remainders

**break**

**end**

push!(remainders, value)

**while** (value **<** i)

value **\*=** 10

len **+=** 1

**end**

value **%=** i

**end**

**if** value **!=** 0 **&&** len **>** seq\_len

seq\_len **=** len

answer **=** i

**end**

**end**

**return** answer, seq\_len

**end**

Out[15]:

ReciprocalCycles (generic function with 1 method)

In [16]:

begin\_time **=** time()

answer **=** ReciprocalCycles(1000)

print("runtime: ", time() **-** begin\_time)

**return** answer

runtime: 0.3948509693145752

Out[16]:

(983,982)

**Quadratic primes   
Problem 27**

Euler discovered the remarkable quadratic formula:

n² + n + 41

It turns out that the formula will produce 40 primes for the consecutive values n = 0 to 39. However, when n = 40, 402 + 40 + 41 = 40(40 + 1) + 41 is divisible by 41, and certainly when n = 41, 41² + 41 + 41 is clearly divisible by 41.

The incredible formula n² − 79n + 1601 was discovered, which produces 80 primes for the consecutive values n = 0 to 79. The product of the coefficients, −79 and 1601, is −126479.

Considering quadratics of the form:

n² + an + b, where |a| < 1000 and |b| < 1000

where |n| is the modulus/absolute value of n e.g. |11| = 11 and |−4| = 4 Find the product of the coefficients, a and b, for the quadratic expression that produces the maximum number of primes for consecutive values of n, starting with n = 0.

In [83]:

**function** QuadraticPrimes()

max\_len **=** 0

best\_a **=** 0

best\_b **=** 0

**for** a **=** **-**999**:**999

**for** b **=** **-**999**:**999

**if** isprime(b)

n **=** 0

**while** isprime(abs((n**^**2) **+** (a**\***n) **+** b))

n **+=** 1

**end**

**if** n **>** max\_len

max\_len **=** n

best\_a **=** a

best\_b **=** b

**end**

**end**

**end**

**end**

**return** best\_a **\*** best\_b

**end**

Out[83]:

QuadraticPrimes (generic function with 1 method)

In [84]:

begin\_time **=** time()

answer **=** QuadraticPrimes()

print("runtime: ", time() **-** begin\_time)

**return** answer

runtime: 6.396510124206543

Out[84]:

-59231

**Number spiral diagonals   
Problem 28**

Starting with the number 1 and moving to the right in a clockwise direction a 5 by 5 spiral is formed as follows:

21 22 23 24 25

20 7 8 9 10

19 6 1 2 11

18 5 4 3 12

17 16 15 14 13

It can be verified that the sum of the numbers on the diagonals is 101.

What is the sum of the numbers on the diagonals in a 1001 by 1001 spiral formed in the same way?

In [19]:

**function** NumSpiralDiag(n)

i **=** 1

result **=** 1

**while** i **<** n

i **+=** 2

result **=** result **+** (i**^**2 **\*** 4) **-** ((i**-**1) **\*** 6)

**end**

**return** result

**end**

Out[19]:

NumSpiralDiag (generic function with 2 methods)

In [20]:

begin\_time **=** time()

answer **=** NumSpiralDiag(1001)

print("runtime: ", time() **-** begin\_time)

**return** answer

runtime: 0.005847930908203125

Out[20]:

669171001

**Consecutive prime sum   
Problem 50**

The prime 41, can be written as the sum of six consecutive primes:

41 = 2 + 3 + 5 + 7 + 11 + 13

This is the longest sum of consecutive primes that adds to a prime below one-hundred.

The longest sum of consecutive primes below one-thousand that adds to a prime, contains 21 terms, and is equal to 953.

Which prime, below one-million, can be written as the sum of the most consecutive primes?

In [4]:

**function** GeneratePrimes(n)

primes **=** Int[]

**for** i **=** 1**:**n

**if** isprime(i)

push!(primes, i)

**end**

**end**

**return** primes

**end**

Out[4]:

GeneratePrimes (generic function with 1 method)

In [13]:

**function** ConsecutivePrimes(n)

primes\_list **=** GeneratePrimes(n**/**250)

l **=** length(primes\_list)

count **=** 0

prime **=** 0

count\_result **=** 0

prime\_result **=** 0

**for** i **=** 1**:**l

**for** j **=** 1**:**l

x **=** l**-**j

count **=** x**-**i

**if** count **>** count\_result

num **=** sum({primes\_list[y] **for** y**=**i**:**x})

**if** num **<** n

**if** isprime(num)

count\_result **=** count

prime\_result **=** num

print("Result: $prime\_result, $count\_result\n")

**end**

**end**

**end**

**end**

**end**

**return** prime\_result, count\_result

**end**

Out[13]:

ConsecutivePrimes (generic function with 1 method)

In [14]:

begin\_time **=** time()

answer **=** ConsecutivePrimes(1000000)

print("runtime: ", time() **-** begin\_time)

**return** answer

Result: 958577, 535

Result: 978037, 538

Result: 997651, 542

runtime: 0.0333561897277832

Out[14]:

(997651,542)

**Distinct powers   
Problem 29**

Consider all integer combinations of ab for 2 ≤ a ≤ 5 and 2 ≤ b ≤ 5:

22=4, 23=8, 24=16, 25=32   
32=9, 33=27, 34=81, 35=243   
42=16, 43=64, 44=256, 45=1024   
52=25, 53=125, 54=625, 55=3125

If they are then placed in numerical order, with any repeats removed, we get the following sequence of 15 distinct terms:

4, 8, 9, 16, 25, 27, 32, 64, 81, 125, 243, 256, 625, 1024, 3125

How many distinct terms are in the sequence generated by ab for 2 ≤ a ≤ 100 and 2 ≤ b ≤ 100?

In [1]:

**function** DistinctPowers(a, b)

matrix **=** zeros(BigFloat, a, b)

matrix **=** [ BigFloat(row)**^**col **for** row**=**2**:**a, col**=**2**:**b ]

**return** length(unique(matrix))

**end**

Out[1]:

DistinctPowers (generic function with 1 method)

In [2]:

begin\_time **=** time()

answer **=** DistinctPowers(100, 100)

print("runtime: ", time() **-** begin\_time)

**return** answer

runtime: 0.6840958595275879

Out[2]:

9183

In [59]:

​

In [60]:

​

In [ ]:

​