Turing Algorithm for Prime Numbers and Factorization

Shaimaa said soltan[1]

[1] Computer Engineer, Toronto, Canada

Correspondence: Shaimaa Soltan, 3050 Constitution Blvd, Mississauga, ON., L4Y 3X1, Canada. Tel: 1-647-801-6063
E-mail: shaimaasultan@hotmail.com

Suggested Reviewers (Optional)

Please suggest 3-5 reviewers for this article. We may select reviewers from the list below in case we have no appropriate reviewers for this topic.

Name:	E-mail:
Affiliation:	

Name:	E-mail:
Affiliation:	

Name:	E-mail:
Affiliation:	

Name:	E-mail:
Affiliation:	

Name:	E-mail:
Affiliation:	

Turing Algorithm for Prime Numbers and Factorization

Abstract

This paper introduces a new algorithm to represent a whole set of numbers in one binary number representation and we use algorithm to fetch a list of Prime numbers with an execution time related to only the number of digits in the number. Some other applications for this algorithm in number theory will be factoring a number or checking if a number is prime or not.

Keywords: Prime numbers; Factorization.

1. Introduction

1.1 Binary Turing representation.

In classic computers the limitaion of data types ranges makes it hard to represent decimal numebrs in binary representations and doing binary operations and goes back to decimal numbers after this operations evene for small numbers.

In this paper we are going to introduce new algorithm to represent numbers into binary turing layout for each number and its multipliers.

We are realying on one note that each number its multipliers are exisits on the same step a way

For Example: - for number 7 and its multipliers

{7 , 14 , 21, 28, 35, 42, 49, 56, 63 , 70 , 77, …}

The diffirence all the time between each two consecounce numbers = 7 as those are multipliers of 7

Same will be for any other number

{11, 22, 33, 44, 55, 66, 77, 88, 99, ….}

So we are going to represent each number and its multipliers in order untill a specific end length or cutoff number(lentgth)

And at each step = N we are going to set this place = 1 and the rest of the digits in the number length will be 0

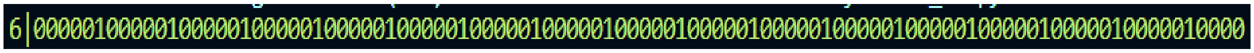
For example : - if we want to represent deimal system 7 number in this turing binary algorithm for length =100

```
7|0000001000000100000010000001000000100000010000001000000100000010000001000000100000010000001000000100
```

First 1 in this binary turing number = 7 ; second 1 in this binary turning number = 14 ; third 1 in this turing number = 21 ; fourth 1 in this binary turing number = 28; ….

```
3|0010010010010010010010010010010010010010010010010010010010010010010010010010010010010010010010010010
```

So in egenral rule for our binary turing algorithm by default our base in zero[0] and we only have one at position N and each N step away from it up untill the end of the of the legth



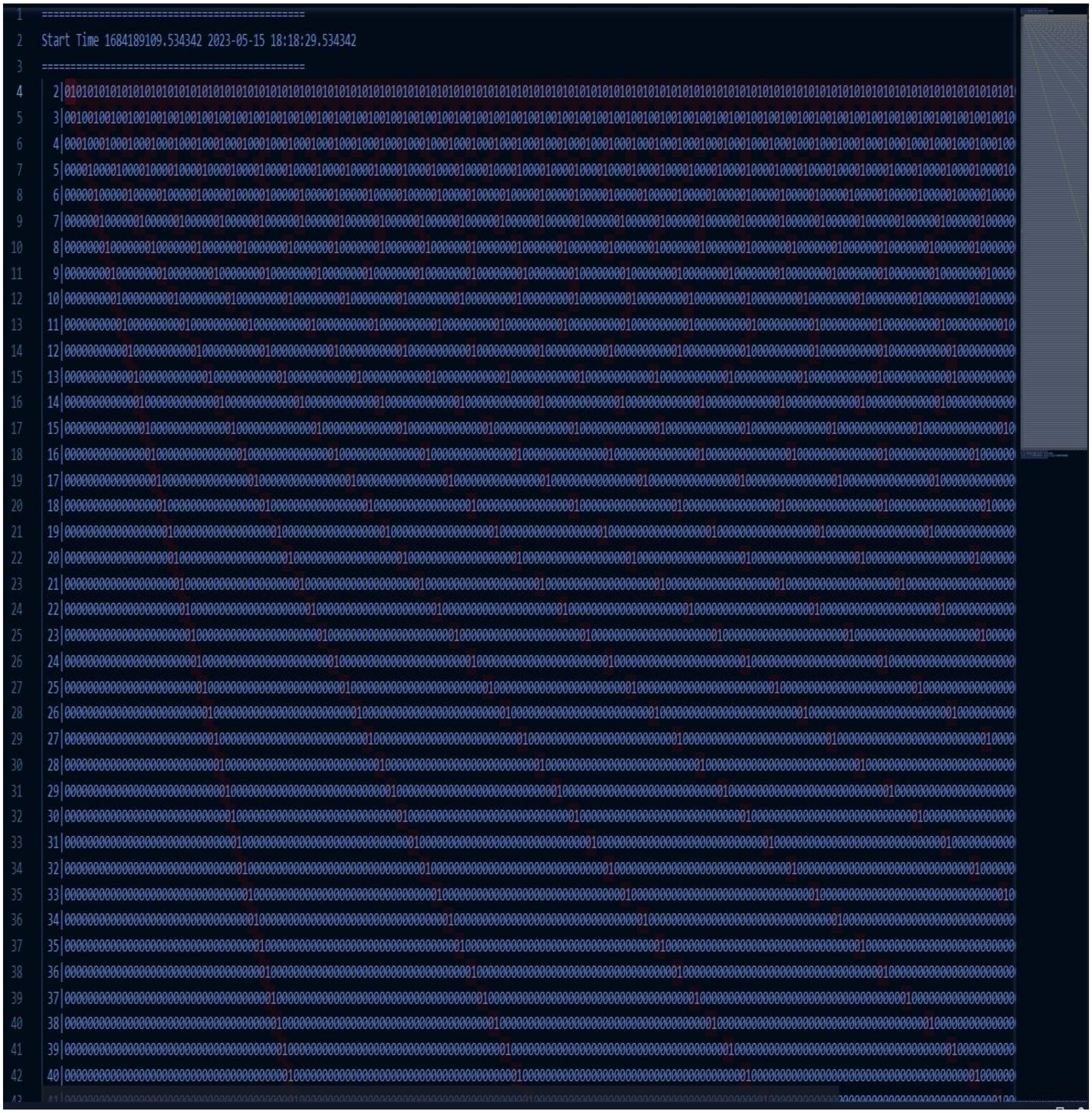


Figure 1. Turing Binary representation for All numbers and its multipliers

As it is shown from ones positions it is a tree with an origin at 1. And all the rest of positions are Zeros.

1 positions like a skewed diagonal lines with one origin at 1

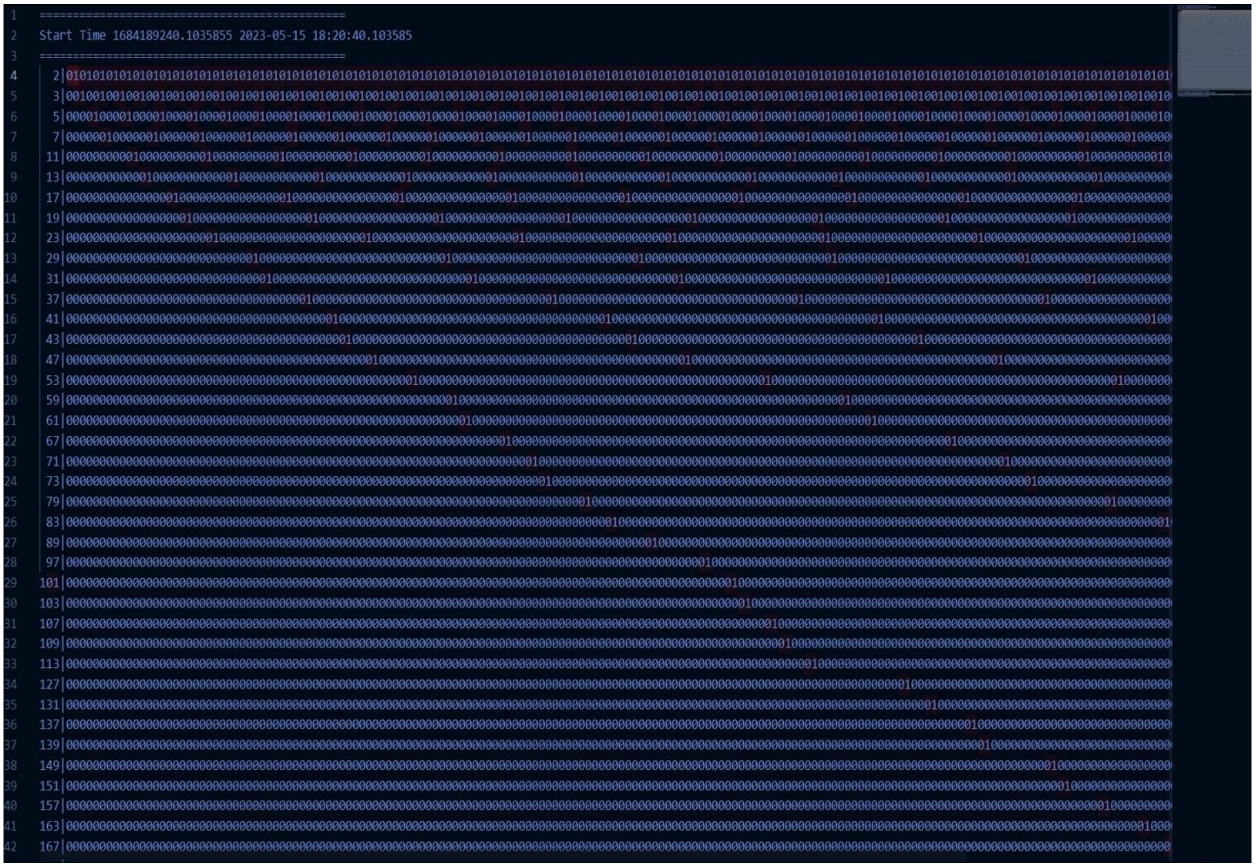
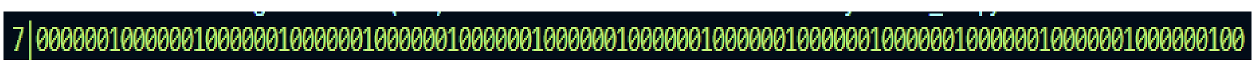


Figure 2. Turing Binary representation for Prime numbers and its multipliers

1.2 Binary Turing operations.

This representation gives us an advantage on getting the value from the index of the position of [1] in the representation. [we do not calculate value in binary system we calculate the value from the index still in base 10 system]

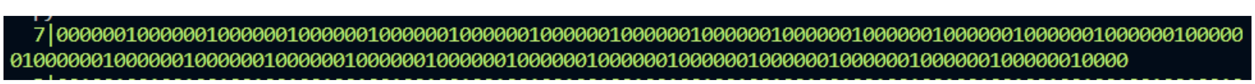


So, this representation is a set of numbers not only one number.

This representation is for SET = {7, 14, 21 ,28, 35, 42, 49, 56, 63, 70, 77, 84, 91,98} as the length we used limited to 100.

If the limit is 200

Our set will be larger until its max number <= 200.



And this is the representation for this Set.

[7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91, 98, 105, 112, 119, 126, 133, 140, 147, 154, 161, 168, 175, 182, 189, 196]

Each time we increase the length condition we represent more range of numbers as we include more multipliers for the numbers.

For example, if our length condition =1000

Our Turing binary length will be = 1000

7|0000001000000100000010000001000000100000010000001000000100000010000001000000100000010000001000000100000
0100000010000001000000100000010000001000000100000010000001000000100000010000001000000100000010000
0010000001000000100000010000001000000100000010000001000000100000010000001000000100000010000001000
0001000000100000010000001000000100000010000001000000100000010000001000000100000010000001000000100
0000100000010000001000000100000010000001000000100000010000001000000100000010000001000000100000010
0000010000001000000100000010000001000000100000010000001000000100000010000001000000100000010000001
0000001000000100000010000001000000100000010000001000000100000010000001000000100000010000001000000
1000000100000010000001000000100000010000001000000100000010000001000000100000010000001000000100000
0100000010000001000000100000010000001000000100000010000001000000100000010000001000000100000010000
0010000001000000100000010000001000000100000010000001000000100000010000001000000100000010000001000
0001000000100000010000001000000100000010000001000000100000010000001000000100000010000001000000100
0000100000010000001000000100000010000001000000100000010000001000000100000010000001000000100000010
00000

Representation for this set of numbers

[7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91, 98, 105, 112, 119, 126, 133, 140, 147, 154, 161, 168, 175, 182, 189, 196, 203, 210, 217, 224, 231, 238, 245, 252, 259, 266, 273, 280, 287, 294, 301, 308, 315, 322, 329, 336, 343, 350, 357, 364, 371, 378, 385, 392, 399, 406, 413, 420, 427, 434, 441, 448, 455, 462, 469, 476, 483, 490, 497, 504, 511, 518, 525, 532, 539, 546, 553, 560, 567, 574, 581, 588, 595, 602, 609, 616, 623, 630, 637, 644, 651, 658, 665, 672, 679, 686, 693, 700, 707, 714, 721, 728, 735, 742, 749, 756, 763, 770, 777, 784, 791, 798, 805, 812, 819, 826, 833, 840, 847, 854, 861, 868, 875, 882, 889, 896, 903, 910, 917, 924, 931, 938, 945, 952, 959, 966, 973, 980, 987, 994]

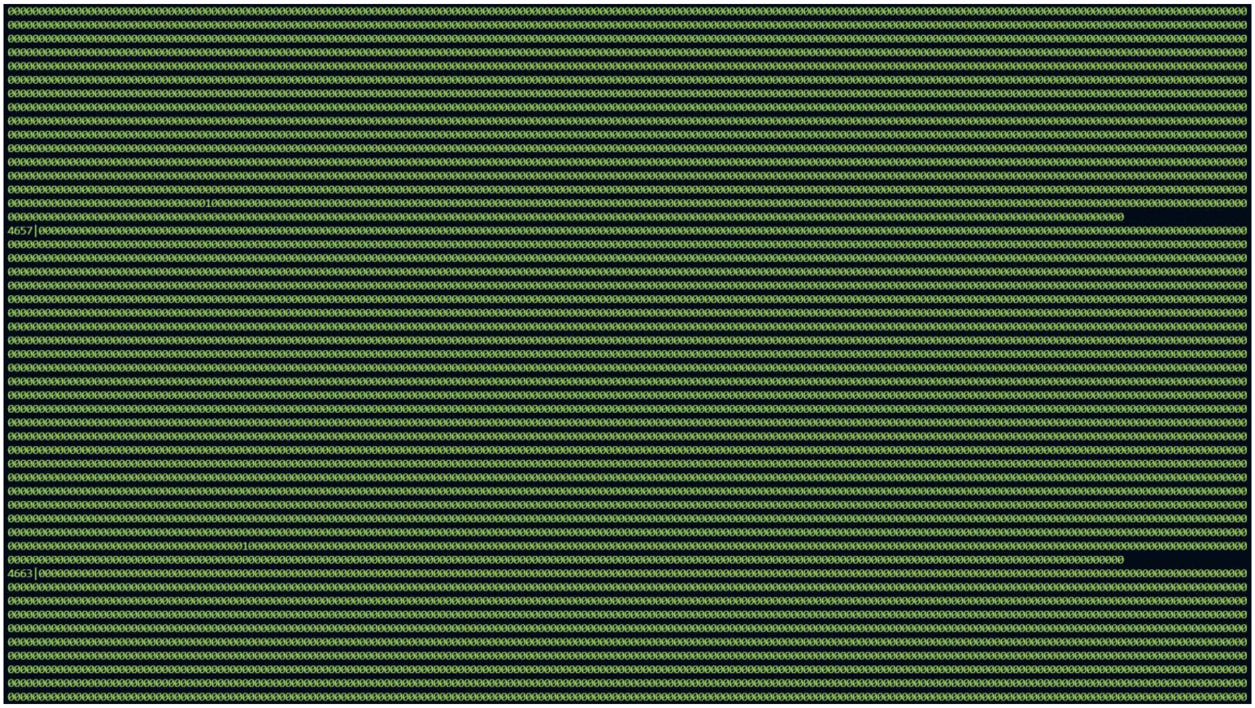


Figure 2. Turing Binary Representations for large numbers have less frequency occurrence for 1 in its representations as the first occurrence will be at position equal this huge number.

Figure 3. Turing Binary Representations for small numbers have higher frequency occurrence for 1 in each given length.

To do operations on this Turing representation we are going to use similar length binary Turing representation for natural number [2] and [1]

For length = 100

1|1111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111

2|0101010101010101010101010101010101010101010101010101010101010101010101010101010101
010101010101010101

In this example we are going to natural number 7

7|0000001000000100000010000001000000100000010000001000000100000010000001000000100000
010000001000000100

We are going to add ones and zeros in its positions as they are.

For example: -

7|0000001000000100000010000001000000100000010000001000000100000010000001000000100000

+

2|0101010101010101010101010101010101010101010101010101010101010101010101010101010101
= 0101011101010201010111010102010101110101020101011101010201010111010102010101110101

And as this is only adding two numbers the max number, we get in the sum will 2 and we can get values {0,1,2} in the final add result. So, we can filter the result on these three values o or 1 or 2.

Filter result on 0:

1- ODD + 1 = []
   7|000000100000010000001000000100000010000001000000100000010000001000000100000
   0100000010000001000000100
   +
   1|111111111111111111111111111111111111111111111111111111111111111111111111111
   1111111111111111111111111
   =
   11111121111112111111211111121111112111111211111121111112111111211111121111112
   11111121111112111111211

   So, filter on 0 will give us nothing.
2- EVEN +1 = []
   6|000001000001000001000001000001000001000001000001000001000001000001000001000
   0010000010000010000010000
   +
   1|111111111111111111111111111111111111111111111111111111111111111111111111111
   1111111111111111111111111
   =

111112111112111112111112111112111112111112111112111112111112111112111112111111
21111121111121111121111

So, filter on 0 will give us nothing.

3- ODD + 2 = [ALL natural ODD numbers list except this ODD numbers and its multipliers]
   For example: - if ODD = 7 the add operation results Turing binary number

7|0000001000000100000010000001000000100000010000001000000100000010000001000000100000

+

2|0101010101010101010101010101010101010101010101010101010101010101010101010101010101
= 0101011101010201010111010102010101110101020101011101010201010111010102010101110101

   And filter for only Zero they indexes will be set.
   [1, 3, 5, 9, 11, 13, 15, 17, 19, 23, 25, 27, 29, 31, 33, 37, 39, 41, 43, 45, 47, 51, 53, 55, 57, 59, 61, 65, 67, 69, 71, 73, 75, 79, 81, 83, 85, 87, 89, 93, 95, 97, 99]

4- EVEN +2 = [ODD List]

6|0000010000010000010000010000010000010000010000010000010000010000010000010000010000
010000010000010000

+

2|0101010101010101010101010101010101010101010101010101010101010101010101010101010101
010101010101010101

=

01010201010201010201010201010201010201010201010201010201010201010201010201010201010
20101020101020101

Indexes of Zeros = set of ODD numbers

[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99]

Now let use see if we filtered on if the value of the add = 1 what index values we are going to get

Filter result on 1:

1- ODD + 1 = [ALL Numbers Except this ODD Multipliers]
   11|00000000001000000000010000000000100000000001000000000010000000000100000000
   00100000000001000000000010
   +
   1|111111111111111111111111111111111111111111111111111111111111111111111111111
   1111111111111111111111111
   =
   11111111112111111111121111111111211111111112111111111121111111111211111111112
   11111111112111111111121
   Indexes of Ones = All natural numbers except multipliers of the number we used in the addition.

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 100]

2- EVEN +1 = [ALL Numbers Except this EVEN Multipliers]
8|000000010000000100000001000000010000000100000001000000010000000100000001000
0000100000001000000010000
+
1|111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111
=
11111112111111121111111211111112111111121111111211111112111111121111111211111
11211111112111111121111
[1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 25, 26, 27, 28, 29, 30, 31, 33, 34, 35, 36, 37, 38, 39, 41, 42, 43, 44, 45, 46, 47, 49, 50, 51, 52, 53, 54, 55, 57, 58, 59, 60, 61, 62, 63, 65, 66, 67, 68, 69, 70, 71, 73, 74, 75, 76, 77, 78, 79, 81, 82, 83, 84, 85, 86, 87, 89, 90, 91, 92, 93, 94, 95, 97, 98, 99, 100]

3- ODD + 2 = [ALL EVEN Numbers + ALL this ODD Multipliers except it’s even multipliers]
As 2 has value 1 every two bits first occurrence for this odd number will be in (odd/2-1/2 +1)
And its multipliers will be every step = ODD-1.
11|00000000001000000000010000000000100000000001000000000010000000000100000000
00100000000001000000000010
+
2|010101010101010101010101010101010101010101010101010101010101010101010101010
1010101010101010101010101
=
01010101011101010101020101010101110101010102010101010111010101010201010101011
1010101010201010101011
[2, 4, 6, 8, 10, 11, 12, 14, 16, 18, 20, 24, 26, 28, 30, 32, 33, 34, 36, 38, 40, 42, 46, 48, 50, 52, 54, 55, 56, 58, 60, 62, 64, 68, 70, 72, 74, 76, 77, 78, 80, 82, 84, 86, 90, 92, 94, 96, 98, 99, 100]
List of even numbers including this odd number and its odd multipliers only.

4- EVEN +2 = [ALL EVEN Numbers Except this Even number Multipliers]
4|000100010001000100010001000100010001000100010001000100010001000100010001000
1000100010001000100010001
+
2|010101010101010101010101010101010101010101010101010101010101010101010101010
1010101010101010101010101
=
01020102010201020102010201020102010201020102010201020102010201020102010201020
10201020102010201020102
[2, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62, 66, 70, 74, 78, 82, 86, 90, 94, 98]

Filter adds result on 2:

1- ODD + 1 = [Multipliers of this ODD Number]
3|001001001001001001001001001001001001001001001001001001001001001001001001001
0010010010010010010010010
+
1|111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111
=
11211211211211211211211211211211211211211211211211211211211211211211211211211
21121121121121121121121
=
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99]

2- EVEN +1 = [Multipliers of this EVEN Number]
10|00000000010000000001000000000100000000010000000001000000000100000000010000
00000100000000010000000001
+
1|111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111
=
11111111121111111112111111111211111111121111111112111111111211111111121111111
11211111111121111111112
=
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

3- ODD + 2 = [EVEN Multipliers of this ODD]
13|00000000000010000000000001000000000000100000000000010000000000001000000000
00010000000000001000000000
+
2|010101010101010101010101010101010101010101010101010101010101010101010101010
1010101010101010101010101
=
01010101010111010101010102010101010101110101010101020101010101011101010101010
20101010101011101010101
=
[26, 52, 78]

4- EVEN +2 = [Multipliers of this EVEN]

8|0000000100000001000000010000000100000001000000010000000100000001000000010000000100
000001000000010000

+

2|0101010101010101010101010101010101010101010101010101010101010101010101010101010101
010101010101010101

=

01010102010101020101010201010102010101020101010201010102010101020101010201010102010
10102010101020101

=

[8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96]

1.3 Binary Turing Prime Numbers.

We are going to use these operations in getting multipliers of numbers list so we can exclude it in order of getting a final list of prime numbers without doing any complex looping.

Also, we are not going to go through all numbers we only need to go through numbers that satisfies.

(C-1) / 6 and (C - 5) /6

i.e., only go through list.

[7, 13, 19, 25, 31, 37, 43, ….] and list [11, 17, 23, 29, 35, 41, 47, 53, ….]

This is a beta code for the algorithm to get the list of prime numbers using the operations of adding Turing binary representation of natural number 1 to each number in those both lists.

```
def OneBinaryI(N , L , padd):
    s=[]
    for v in range(0,L):
        s.append(0)
    for j in range(0,L,N):
        if j+N-1 > L-1 : pass
        else:
            s[j+N-1] = 1
    ''' row header'''
    s.insert(0,str(N).rjust(padd)+'|')
    return s
```

```python
def PrimeList(N):
    m = []
    P = [2 , 3, 5 ]
    f = open('Prime_List.lg','+w')
    padd = len(str(N))
    for j in range(0,N,6):
        a = 7 + j

        if a%5 == 0 : pass
        elif a not in m:
            Vn11 = OneBinaryI(a,N, padd)
            Vn14 = OneBinaryI(1,N, padd)
            VAn1 = [a+b for a,b in zip(Vn11[1:], Vn14[1:])]
            l1 = [i+1 for i , val in enumerate(VAn1) if val == 2 ]
            m= m + l1
            P.append(a)

        b = 11 + j

        if b > N : break
        if b%5 == 0 : pass
        elif b not in m :
            Vn21 = OneBinaryI(b,N, padd)
            Vn24 = OneBinaryI(1,N, padd)
            VAn2 = [a+b for a,b in zip(Vn21[1:], Vn24[1:])]
            l2 = [i+1 for i , val in enumerate(VAn2) if val == 2 ]
            m= m + l2
            P.append(b)
    f.write(str(P))
```

```python
L = 100
lg , hnd = InitiateLogger("RunStats" , "RunStats.lg")
lg.info('===============================================')
start_T = time.time()
lg.info('Start Time {} {}'.format(start_T,datetime.now()))
lg.info('===============================================')
PrimeList(L)
lg.info('===============================================')
End_T = time.time()
lg.info('End Time {} {} '.format(End_T,datetime.now() ))
lg.info(" Duration in sec {} ; Duration in minutes {}".format( End_T - start_T , (End_T - start_T)/60.0 ))
lg.info('===============================================')
```

<table>
<tr><td colspan="3">Table 1. Turing Binary Execution Times</td></tr>
<tr><td>First N Prime Numbers List</td><td>Execution Time using Turing Binary</td><td></td></tr>
<tr><td>100</td><td>0</td><td>==
Start Time 1684218271.7750618 2023-05-16 02:24:31.775061
==
==
End Time 1684218271.7750618 2023-05-16 02:24:31.775061
Duration in sec 0.0 ; Duration in minutes 0.0
==</td></tr>
<tr><td colspan="3">[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]</td></tr>
<tr><td>1000</td><td>0.03 sec</td><td>==
Start Time 1684218317.0176797 2023-05-16 02:25:17.017679
==
==
End Time 1684218317.0550637 2023-05-16 02:25:17.055063
Duration in sec 0.037384033203125 ; Duration in minutes 0.0006230672200520834
==</td></tr>
<tr><td colspan="3">[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]</td></tr>
<tr><td>10000</td><td>2.9 sec</td><td>==
Start Time 1684218510.2120397 2023-05-16 02:28:30.212039
==
==
End Time 1684218513.1682866 2023-05-16 02:28:33.168286
Duration in sec 2.956246852874756 ; Duration in minutes 0.04927078088124593
==</td></tr>
<tr><td colspan="3">[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,…, 9511, 9521, 9533, 9539, 9547, 9551, 9587, 9601, 9613, 9619, 9623, 9629, 9631, 9643, 9649, 9661, 9677, 9679, 9689, 9697, 9719, 9721, 9733, 9739, 9743, 9749, 9767, 9769, 9781, 9787, 9791, 9803, 9811, 9817, 9829, 9833, 9839, 9851, 9857, 9859, 9871, 9883, 9887, 9901, 9907, 9923, 9929, 9931, 9941, 9949, 9967, 9973]</td></tr>
</table>

100000	4.035 minutes	== Start Time 1684218808.7414348 2023-05-16 02:33:28.741434 == == End Time 1684219050.8653667 2023-05-16 02:37:30.865366 Duration in sec 242.12393188476562 ; Duration in minutes 4.035398864746094 ==
500000	121.26 minutes	== Start Time 1684221315.3232112 2023-05-16 03:15:15.323211 == == End Time 1684228591.028291 2023-05-16 05:16:31.028291 Duration in sec 7275.70507979393 ; Duration in minutes 121.261 ==

There will be future enhancement for this algorithm execution time on classic computers by using accumulative operations and batching to avoid hardware limitations as well.

Conclusion

Classic computers have some limitations in datatypes ranges for numbers that have big number of digits. In this paper we introduced a new algorithm that can be used to in representing set of numbers in a form of Binary Turing number controlled by the number of digits as a parameter. An enhancement for this algorithm will be to use the same algorithm but using batches instead of running the algorithm every time starting from 1 up until the max length parameter. As we showed this algorithm currently shows a promising execution time in handling large number of numbers. Also increasing the number of operations between the Turing Binary representation for each number Sets which will give better execution time and many other applications that can benefits from this algorithm. Also doing operations accumulatively, operate on more than two number at the same time (add more than two binary representations at a time accumulatively). Witch will give us an easy way for factorization without doing any division operations on classical computers.

References

[1] Brumfiel, G. Student snags maths prize. *Nature* (2007). https://doi.org/10.1038/news.2007.190

[2] Universality in Turing Machines and Other Systems: A New Kind of Science | Online by Stephen Wolfram [Page 707] (wolframscience.com); https://www.wolframscience.com/nks/p707--universality-in-turing-machines-and-other-systems/m [Page 707] (wolframscience.com)

Copyrights