Discrete Hyperbolic Polylogarithmic Sieve For Integer Factorization (Draft)

Ka.Shrinivaasan (ka.shrinivaasan@gmail.com)

June 16, 2013

Abstract

Best known algorithm for factorization is the General number field sieve algorithm which takes time exponential in number bits in the input or simply called exponential algorithm. In this article, a polylogarithmic time (in size of input) sieve for factorization is discussed that uses elementary principles.

1 Discretization of a hyperbolic arc

1. A plot of a hyperbolic function is ($xy = N$) drawn on a 2-dimensional plane. This plot of hyperbola is continuous. This hyperbola can be discretized (or tesselated) with set of rectangles that cover the hyperbolic arc.

2. Discretization step divides the x-axis of the hyperbola into $log^c N$ intervals where each interval is of length $N/log^c N$. Infact it suffices to calculate upto $\sqrt{N}$ length along the x-axis since atleast 1 factor must lie within $\sqrt{N}$.Correspondingly the y-axis is also split into intervals whose width is $y * \sqrt{N}/(xlog^c N + \sqrt{N})$ where (x, y) are coordinates of bottom left corners of each rectangle.

3. The constant c is chosen to be quite large so as to shrink the size of each rectangle as much as possible.Thus each rectangle is of length $\sqrt{N}/log^c N$ and width $y * \sqrt{N}/(xlog^c N + \sqrt{N})$ where (x, y) are coordinates of topleft corners of each rectangle. The factors (say p,q) of N are sure to be present within one of the rectangles if N is composite. Algorithm then is to sieve out rectangles to get to the factors. Looping through $log^c N$ rectangles takes time $O(log^c N * TimePerRectangle)$.

4. Within each rectangle,the values of coordinates are hypothetically considered to be products of x and y coordinates. This is just supposed to exist and no lookup table or memoization is needed (though it can be made into a dynamic programming recurrence algorithm if knowledge of factors of numbers lower than N is assumed). To find the coordinates where N is located within the rectangle, binary search is applied on the coordinate products with some specialized modifications as described in next section below.

5. The line equation corresponding to leading diagonal within each rectangle obtained previously, intersects with the hyperbola,thereby creating a bow-shaped area (hemmed between hyperbolic arc and diagonal) of coordinate products within each rectangle.This bow shaped area is only a small percentage of the area of each rectangle.

6. The row slices of coordinate products within this bow shaped area are in sorted order from left to right. Also the coordinate products in diagonal of the rectangle from bottom-left to top-right of each rectangle are in sorted ascending order.

2 Binary search on the above bow-shaped area within each rectangle

1. Since coordinate products along bottom-left to top-right diagonal are in ascending order and border coordinates for each rectangle are already known, doing a binary search along the diagonal helps in reaching to the closest point within the rectangle to the factor coordinates (p, q) such that $pq = N$. Closest point (x, y) is where $xy - pq$ is least.

2. Once the closest point (x, y) is reached on the diagonal, from that row x, remaining row slices within bow-shaped area can be sequentially scanned and each row slice within the bow-shaped area can be binary searched for factor coordinates. This is repeated for each rectangle. Time per rectangle is $O(log^2 N)$ since each side of the rectangle is a function of $N/log^c N$ which is polylog in input size.As mentioned in a later section below for suitably chosen c, $N/log^c N$ is upperbounded by $log(N)$and thus area of each rectangle is $O(log^2 N)$.

3.
```
Functions BinarySearchSubroutine(xleft, yleft, xright, yright)
{
if(yleft == yright) //horizontal scan
{
if (xleft+xright)/2 * yleft == N
{
return (xleft+xright)/2, yleft as factors of N
}
else if (xleft + xright)/2 * yleft > N
BinarySearchSubroutine(xleft, yleft, (xleft+xright)/2, yright)
else
BinarySearchSubroutine((xleft+xright)/2, yleft, xright, yright)
}
else //diagonal scan
{
same as above except that midpoint is computed as
([xleft+xright]/2 , [yleft+yright]/2)
and proceed as above outer if clause
}
}
```

3 Discrete Hyperbolic Polylogarithmic Sieve for Integer Factorization - Algorithm Steps

1. Before the factorization loop starts for a number N to be factorized, the constant c has to be chosen in such a way that $N/log^c N$ is less than $logN$.

2. It can be derived that if c is greater than $logN/loglogN - 1$ then it makes sure that $N/log^c N$ is less than $logN$.

3. Thus a choice of value of c is made adhering to above inequality to make the sides of rectangles sublogarithmic in N.

4. Split interval $[0 - N]$ into $log^c N$ intervals of width $N/log^c N$ each.

5. For each of $log^c N$ rectangle of width $N/log^c N$ and height $y * \sqrt{N}/(xlog^c N + \sqrt{N})$ where (x, y) is the coordinate of bottom-left corner of each rectangle, compute the following loop:

   (a) By Binary Search, using BinarySearchSubroutine described above, along the bottomleft-topright diagonal, reach to the closest point (x, y) to the factor coordinates (p, q) such that $pq = N$ and $xy - pq$ is least within the rectangle.

   (b) From point (x, y) and within the bow shaped area confined within the hyperbolic area and the diagonal of the rectangle, do horizontal binary search for coordinate of $N(p, q)$ such that $pq = N$.

   (c) Output $p, q(pq = N)$ if found in previous step and break; else continue loop

Thus total time taken is $O(log^c N * logN * logN) = O(log^{c+2} N)$ for choice of c described below.

4 $log(N)$ Upperbounds $N/log^c N$ for suitable choice of c

Key to this algorithm is that side of each rectangle $N/log^c N$ is upperbounded by $logN$ for suitable choice of c. The constant c has to be chosen in such a way that $N/log^c N$ is less than $logN$. It can be derived that if c is greater than $logN/log(log(N)) - 1$ then it makes sure that $N/log^c N$ is less than $logN$. This is just a lowerbound on what values c can have and c is not a function of N as it might seem. For example if N is 10^{200} then c has to be greater than $200log(10)/log(200log(10)) - 1$ which is approximately 85. Choosing c as 100 satisifies this lowerbound for c to be greater than 85. Since sides of rectangle are upperbounded by $log(N)$, areas of all rectangles are upperbounded by $log^2 N$. Since choice of c is predecided before the factorization loop starts, c is invariant in the loop and it depends only on number to be factorized.Hence even if the complete rectangle is scanned instead of bow-shaped area, time per rectangle is still $O(log^2 N)$. For arbitrarily large values of c, the upperbound widens between $logN$ and $N/log^c N$. Though by fixing c as $log(N)/log(log(N)$ would be sufficient for $logN$ to upperbound $N/log^{log(N)/log(log(N)} N$ for all N, leaving it as a lowerbound minimum value for c gives some flexibility in deciding the number of rectangles that tesselate the hyperbola. If we have $log^c N$ number of processors which would imply an NC circuit, above factorization loop can infact be done using a logdepth, bounded width $NC1$ circuit with each rectangle assigned to one processor if binary search per rectangle is also in logdepth and constant c decides the extent of parallelism.

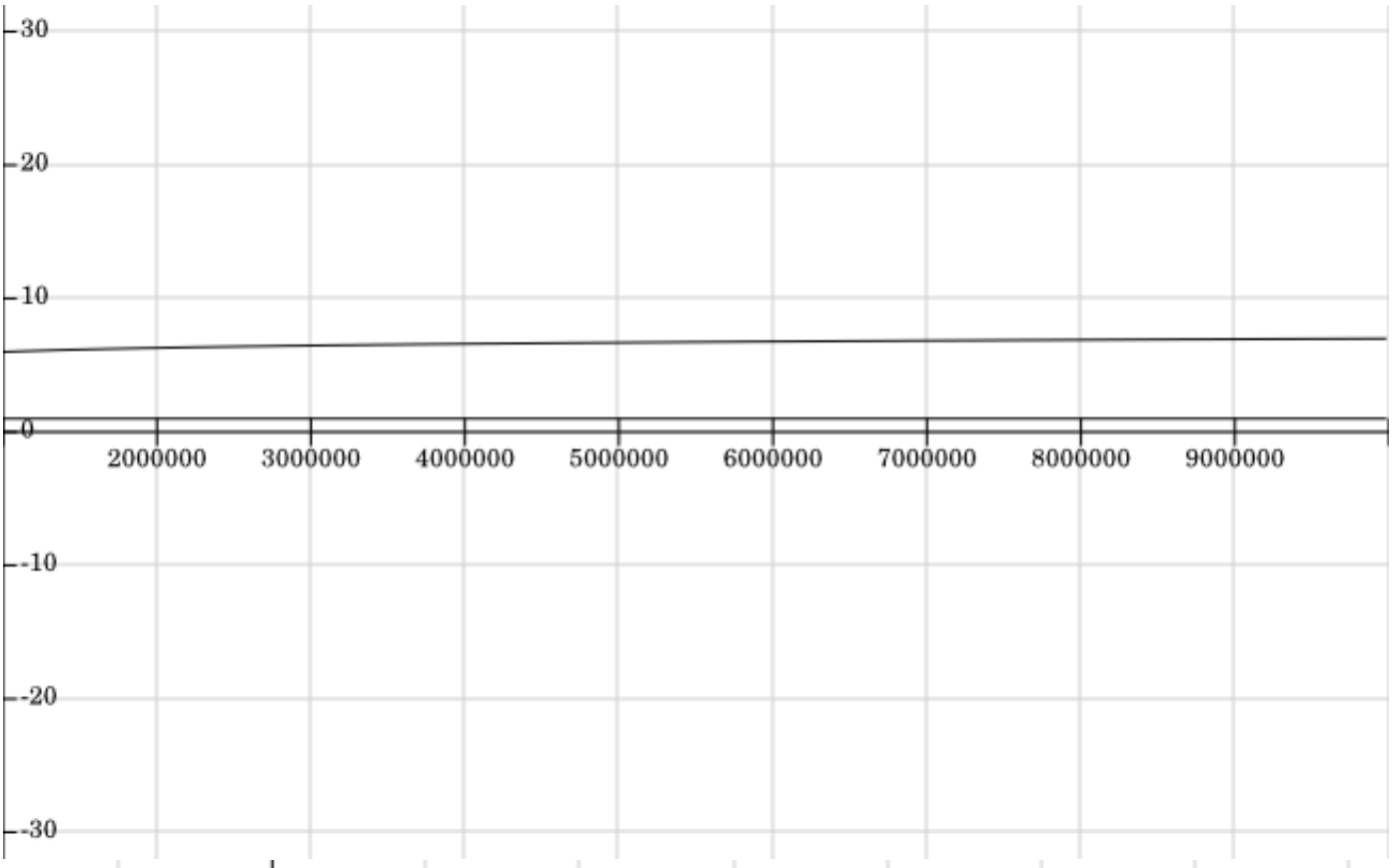
5 Implications

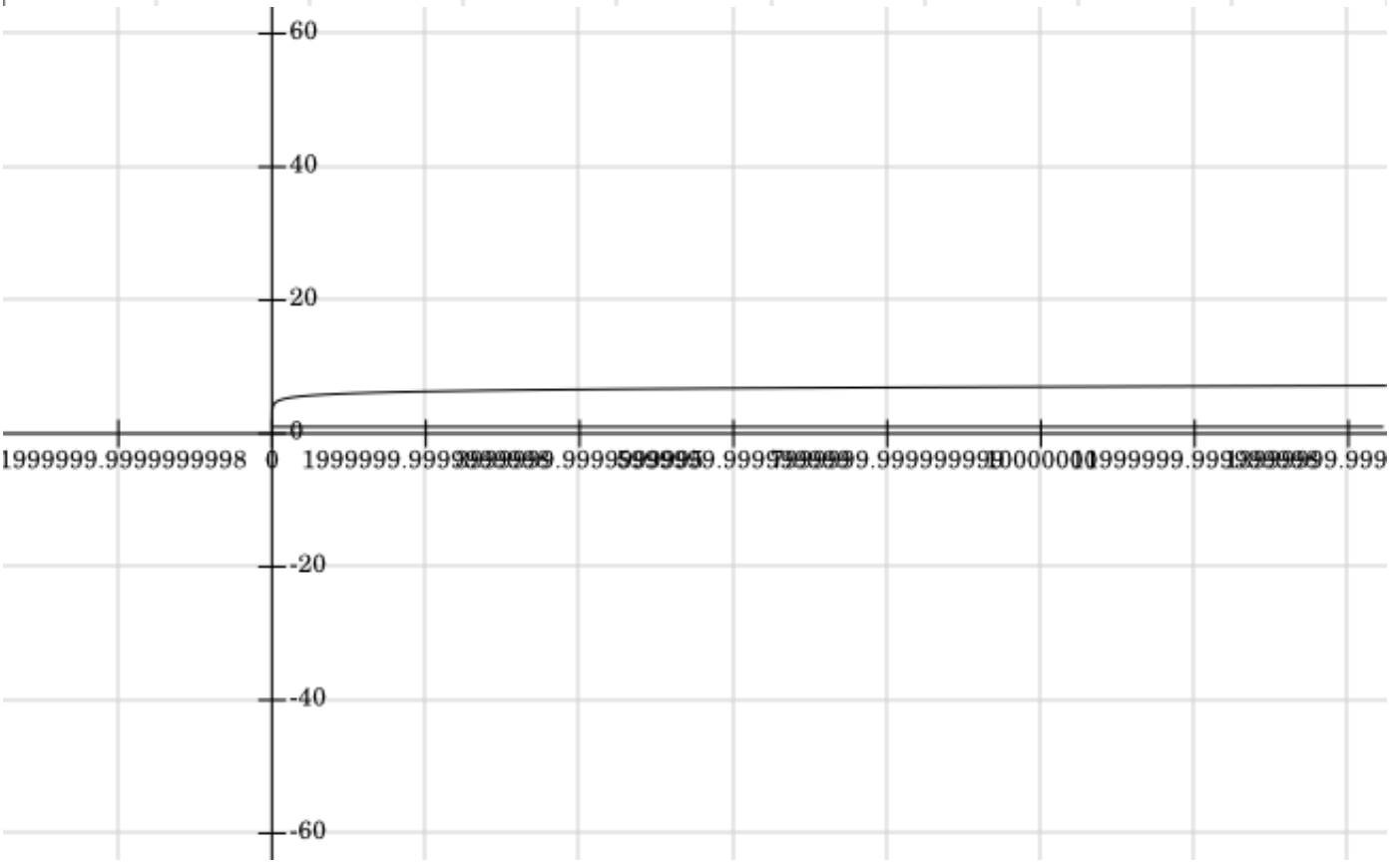
As a consequence of this polylog algorithm for factorization, RSA cryptosystem's large prime keypair can be found in polynomial time by factorization for which till now there are only BQP quantum polynomial algorithms available. Hence factorization in P implies probably that $BQP = P$ (if factorization is BQP-complete) since there is only a quantum polynomial algorithm for factorization and implies therefore that $BPP = P$ since BQP contains BPP (implying pseudorandomness and determinism are equivalent in computational power). This algorithm outputs all factors of an integer within polylogarithmic time in input bits and as a special case this is also a polynomial time primality test similar to PRIMES in P (AKS).

6 Acknowledgement

I dedicate this article to God.

7 Appendix 1 - function plots for $logN$ upperbounding $N/log^{log(N)/log(log(N)}N$





8 Appendix 2 - Indefinite Elliptic Integral for Hyperbolic Arc Length which is attained when number of rectangles tend to infinity

$$\int \sqrt{1+\frac{N^2}{x^4}}\,dx = -x\sqrt{\frac{N^2}{x^4}+1} + \frac{2Nx^2\sqrt{\frac{N^2}{x^4}+1}\sqrt{1-\frac{i\,x^2}{N}}\sqrt{1+\frac{i\,x^2}{N}}\left(E\left(i\sinh^{-1}\left(\sqrt{\frac{i}{N}}\,x\right)\middle|-1\right)-F\left(i\sinh^{-1}\left(\sqrt{\frac{i}{N}}\,x\right)\middle|-1\right)\right)}{\sqrt{\frac{i}{N}}\,(N^2+x^4)}$$

$$+\text{constant}$$

Computed by Wolfram|Alpha

9 Bibliography

References

[1] Elementary algebraic principles

[2] Binary Search

[3] Various Function plotter software

[4] Python function plot (text) code written by self

[5] Scanned rough notes containing illustrations - https://docs.google.com/file/d/0B8TCub8qrCY8czZJZFJMT1V

[6] Scanned rough notes containing illustrations(annexures): 1. https://docs.google.com/file/d/0B8TCub8qrCY8Ykt4cUFDZmpDX1U/edit

2. https://docs.google.com/file/d/0B8TCub8qrCY8U1pEeUxrUENfcFU/edit

3. https://docs.google.com/file/d/0B8TCub8qrCY8SkpwcjBDYkMtSXM/edit