Thesis Defense: A Transparent Square Root Algorithm to Beat Brute Force for Sufficiently Large Primes of the Form $p = 4n + 1$

Michael R. Spink,
Dr. Manuel Lopez, PhD, as advisor

Rochester Institute of Technology

School of Mathematical Sciences

October 29, 2018

Overview

Table of Symbols I

p	Prime number, usually of form $p = 4n + 1$
$\mathbb{Z}_p^\times$	The group of units mod p, i.e $\{1, 2, \cdots, p-2, p-1\}$
C	Potential Quadratic Residue
x	Solution to the Square Root Problem, if it exists
λ	Shifted Solution to the Square Root Problem, if it exists
QR_p, NR_p	Sets of QR's and NR's
a, b	Integers in the decomposition of $p = 4n + 1 = a^2 + b^2$
V	Vector space in which a symplectic manifold is set
Ω	Two form for manifold
$\tilde{\Omega}$	Dual mapping for Ω
π	Symplectomorphism

Table of Symbols II

R	Table of Determinants for mod p.
"Quadrant"	If $p = 4n + 1$, integer intervals $[1, n][n + 1, 2n] \cdots$
ρ	Ratio between Quadratic Residues
ξ	Difference between Quadratic Residues
Q, s	Coefficients in decomposition of $p - 1 = Q * 2^s$
y_i	Perfect integer squares to use in algorithm
χ	Algorithmic current value
Ψ	Algorithmic integer multiple
κ	Algorithmic sign
$[x]$	x rounded to the nearest integer

The Square Root Problem: A definition

Definition- Square Root Problem

Consider $\mathbb{Z}_p^\times$ for given odd prime integer p. Integer solutions to the modular equation

$$x^2 \equiv C \pmod{p} \tag{1}$$

where $C \in \mathbb{Z}_p^\times$ is also given solves the so called **Square Root Problem**, if they exist.

Since we can write this as $\sqrt{C} \equiv x \pmod{p}$, we say that x is the Square Root of C modulo p.

Must a Solution always exist?

Definition- Quadratic Residue Modulo p (QR_p)

The variable C in the square root problem is known as a Quadratic Residue modulo p whenever a solution exists.

Definition- Quadratic Non-residue Modulo p (NR_p)

Values in $\mathbb{Z}_p^\times$ that are not quadratic residues mod p are known as Quadratic Non-residues modulo p.

Theorem- Cardinality of Sets of Reciprocity

Let p be prime. Then

$$|QR_p| = |NR_p| = \frac{p-1}{2} \tag{2}$$

Easy Square Roots

Theorem- Euler's Criterion

Let p be a prime, and $m \in \mathbb{Z}_p^\times$. $m \in QR_p$ iff

$$m^{\frac{p-1}{2}} \equiv 1 \pmod{p} \tag{3}$$

Theorem- Square root for Quadratic residues of $p = 4n + 3$ primes

Let $p = 4n + 3$ be prime. The square root, x, of quadratic residue C is:

$$x \equiv \pm C^{\frac{p+1}{4}} \pmod{p} \tag{4}$$

So what do we do with $p = 4n + 1$?

No deterministic algorithm exists. These rely on the Discrete Log Problem or Quadratic extensions. We could also use Brute force.

Algorithm-Tonelli Shanks

Decompose $p = Q * 2^s + 1$, computing C^Q, finding the least i such that $t^{(2^i)} \equiv 1 \pmod{p}$, and a lot of book-keeping

Algorithm- Cipolla

Find a t such that $t^2 - a \in NR_p$, Exponentiation

Algorithm-Pocklington

Decompose p type, find α, β such that $\alpha^2 + C\beta^2 \in NR_p$, recursion

Others do exist, but they aren't simple to execute. EX: Elliptic curves

So now what?

We use properties of $4n + 1$ primes:

Theorem- Product of Residues

Let p be prime and let $x, y \in QR_p$.Then $xy \in QR_p$,

Corollary- Quadratic Character of -1

Let p be an odd prime. Then $-1 \in QR_p$ iff $p = 4n + 1$

Theorem- Pythagorean Decomposition of $p = 4n + 1$ primes

Let $p = 4n + 1$ be prime. Then we can write p uniquely as
$p = a^2 + b^2, a, b \in \mathbb{Z}$.

Our model: an overview

What we need for a symplectic Manifold:

- A Skew Symmetric Bilinear Two-Form
- A Manifold in Symplectic Space V

We then combine them into a Symplectic Manifold.
This gives us access to a unique visual element, as well as Darboux's Theorem.

The Skew Symmetic Biliear Two form

Begin with an even dimensional Vector Space: $V = \mathbb{R}^2$.
Add in a skew symmetric bilinear two form in V – this will be

$$\Omega(\vec{u}, \vec{v}) = det \begin{bmatrix} -\vec{u}- \\ -\vec{v}- \end{bmatrix} \tag{5}$$

Note the skew symmetric and bilinear nature.
However, we want to relate this to our problem.

Let's do this shift

We want to solve

$$x^2 \equiv C \pmod{p} \tag{6}$$

Let's move away from conventional approaches and shift the problem

$$(\lambda - 1)^2 \equiv \lambda^2 - 2\lambda + 1 \equiv C \pmod{p} \tag{7}$$

Companion Matrix:

$$\begin{bmatrix} 2 & C-1 \\ 1 & 0 \end{bmatrix} \tag{8}$$

and now we can find our version of Ω by taking the determinant.

Putting Ω into the space

Definition- Dual Map

Let $\Gamma : V \times V \to \mathbb{R}$ be a skew symmetric bilinear map. Then $\tilde{\Gamma}$, called a dual map for Γ, is the mapping from V into its dual, denoted V^*. Further, this map is

$$\tilde{\Gamma}(\vec{u}) = \Gamma(\vec{u}, \_) \in Hom(V, \mathbb{R}) \tag{9}$$

Definition- Symplectic Vector Space

Let $\Gamma : V \times V \to \mathbb{R}$ be a skew symmetric bilinear map. Γ is symplectic, if $\tilde{\Gamma} : V \to V^*$ is bijective. We call (Γ, V) a symplectic vector space, and Γ a symplectic linear structure on V.

The manifold

Definition- Manifold

A Manifold M is a topological space that is locally Euclidean at every point in the space.

Definition- Symplectomorphism

Let (V, Γ) and (V', Γ') be symplectic vector spaces in vector spaces V, V'. These two spaces are symplectomorphic if there exists a linear mapping $\pi : V \to V'$, the symplectomorphism, such that

$$\Gamma'(\pi(x)) = \Gamma(x) \forall x \in V \tag{10}$$

We also want the mainfold to remain smooth like V: Symplectomorphism. Let $V' = V$. Consider π:

$$\pi(a, b) = \pi(c, d) \quad \text{if} \quad p|(a - c) \quad \text{and} \quad p|(b - c) \tag{11}$$

What does it look like?

$$\begin{tikzcd}
V \times V \arrow[rr, "\Omega"] \arrow[dr, "\Pi"'] & & \mathbb{R} \\
& V' \times V' \arrow[ur, "\Omega'"'] &
\end{tikzcd}$$

Ω' on the equivalency classes in $V' \times V'$

Putting it all together

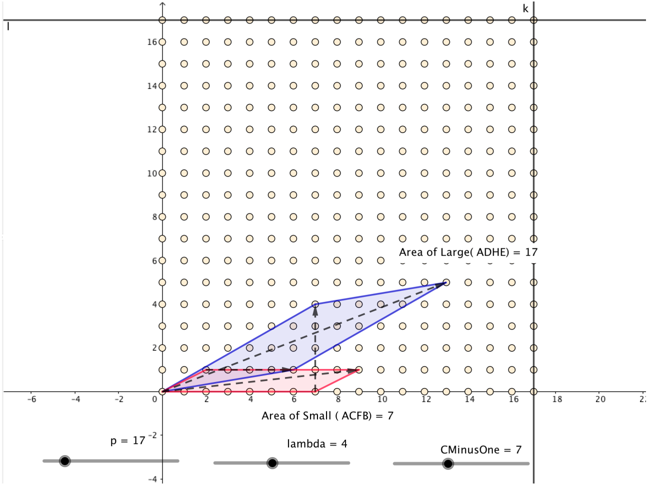
Definition- Symplectic Manifold

A Manifold M in symplectic vector space (Γ, V) is a Symplectic Manifold if at every point p on the manifold, there corresponds a tangental plane, denoted T_pM, and on this tangental plane the map Γ_p is a symplectic linear structure in V. We denote the manifold (M, Γ)

Lay an integer lattice on the torus
So what does Ω mean in the tangental planes?

Gathering the Information Linearly

What if we just work around the torus and apply Ω as we change λ's



Darboux's Thorem

Theorem- Darboux's Theorem

Let (M^{2n}, Ω) be a symplectic manifold. For every point in the manifold, there is an open neighborhood diffeomorphic to an open neighborhood of the origin in $\mathbb{R}^{2n}$ and every pair of points on the manifold have diffeomorphic open neighborhoods.

Darboux's Theorem implies that for every point on the manifold (M^{2n}, Ω), there exists a coordinate chart ϕ such that

$$\phi^* \Omega_0 = \Omega. \tag{12}$$

So we want to gather global information, but the question is how?

Looking at the bigger picture

		Values of λ					
		0	**1**	**2**	**3**	**4**	**5**
Values of C	**1**	**0**	3	8	15	24	35
	2	-1	2	7	14	23	34
	3	-2	1	6	**13**	22	33
	4	-3	**0**	5	12	21	32
	5	-4	-1	4	11	20	31
	6	-5	-2	3	10	19	30
	7	-6	-3	2	9	18	29
	8	-7	-4	1	8	17	28
	9	-8	-5	**0**	7	16	27
	10	-9	-6	-1	6	15	**26**
	11	-10	-7	-2	5	14	25
	12	-11	-8	-3	4	**13**	24

Looking at the bigger picture

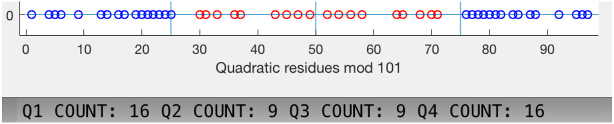
		Values of λ							
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Values of C	**1**	**0**	3	8	15	24	35	48	63
	2	-1	2	7	14	23	**34**	47	62
	3	-2	1	6	13	22	33	46	61
	4	-3	**0**	5	12	21	32	45	60
	5	-4	-1	4	11	20	31	44	59
	6	-5	-2	3	10	19	30	43	58
	7	-6	-3	2	9	18	29	42	57
	8	-7	-4	1	8	**17**	28	41	56
	9	-8	-5	**0**	7	16	27	40	55
	10	-9	-6	-1	6	15	26	39	54
	11	-10	-7	-2	5	14	25	38	53
	12	-11	-8	-3	4	13	24	37	52
	13	-12	-9	-4	3	12	23	36	**51**
	14	-13	-10	-5	2	11	22	35	50
	15	-14	-11	-6	1	10	21	**34**	49
	16	-15	-12	-7	**0**	9	20	33	48

The Conjecture

Conjecture- The Front-Loading Conjecture

Let $p = 4n + 1$ be prime. We partition $\mathbb{Z}_p^\times$ into four distinct regions, from $[1, n]$, $[n + 1, 2n]$, $[2n + 1, 3n]$,$[3n + 1, 4n]$, respectively called quadrants I,II,III, IV. Furthermore, we call the union of quadrants I, IV the “RICH” regions, and the union of quadrants II and III the “POOR” regions. Then there are more quadratic residues in the RICH regions than the POOR ones.

$p = 17, QR_p = \{1, 2, 4\ \ , 8,\ \ 9,\ \ 13, 15, 16\}$
$p = 29, QR_p = \{1, 4, 5, 6, 7\ \ , 9, 13,\ \ 16, 20,\ \ 22, 23, 24, 25, 28\}$



Ok, but how do we quantify this?

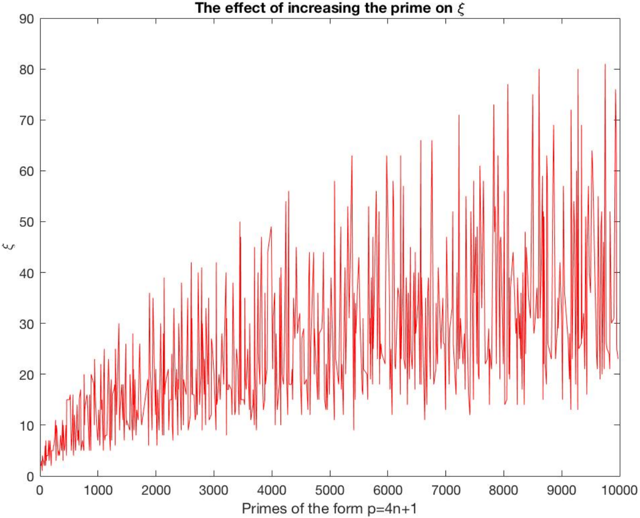
$$\rho := \frac{\text{Number of QR's in Quadrant II}}{\text{Number of QR's in Quadrant I}} \tag{13}$$

$$\xi := \text{Number of QR's in Quadrant I} - \text{Number of QR's in Quadrant II} \tag{14}$$

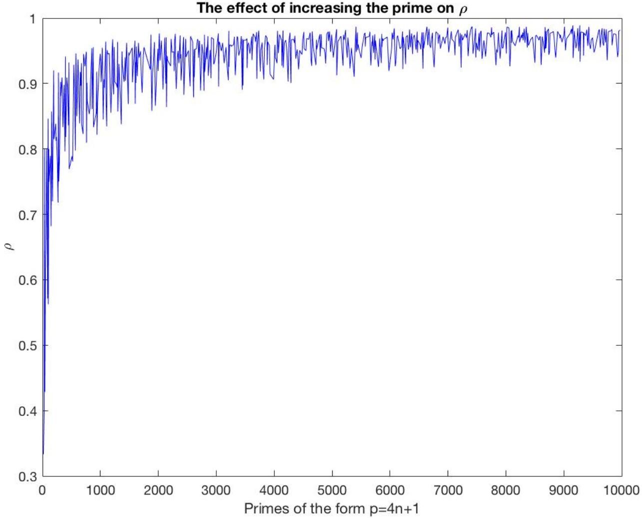
Examples of these constants.

		Values of interest			
		QR I	QR II	ρ	ξ
Primes	**5**	1	0	N/A	1
	13	2	1	.5000	1
	17	3	1	.3333	2
	29	5	2	.4000	3
	37	5	4	.8000	1
	41	7	3	.4286	4
	53	8	5	.6250	3
	61	9	6	.6667	3
	73	10	8	.8000	2
	89	14	8	.5714	6
	97	13	11	.8462	2
	⋮	⋮	⋮	⋮	⋮
	233	32	26	.8125	6
	617	80	74	.9250	6
	73529	9275	9107	.9819	168

Confirming ξ's trends



Confirming ρ's trends



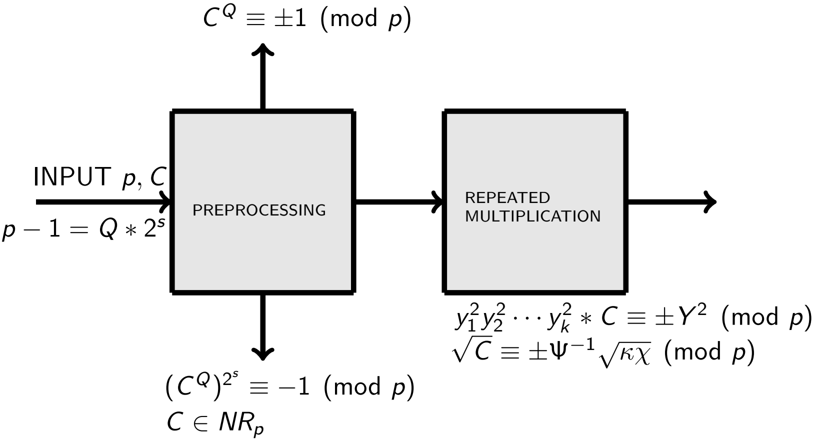
The Algorithm

Goals of the algorithm:

- Perform better than brute force for large enough primes
  i.e. Running at $\mathcal{O}(\frac{n}{c}), c > 1$
- Avoid the trappings of Tonelli-Shanks, Cipolla, Pocklington, etc.
  i.e Rely on basic operations
- Incorporate decidability into the algorithm, finding the square root when applicable.

Front-Loading, while not instrumental, was our inspiration and will speed up the algorithm if true.

Overview of algorithm find $x^2 \equiv C \pmod{p}$



The MATLAB Code

```matlab
function root =Algo(p,C)

%%%%%%%%%%%%%%%%% INITIALIZE VALUES %%%%%%%%%%%%%%%%
n=(p-1)/4; % compute n, p=4n+1
curr=C; %set current value
%%%%%%%%%%%%%%% PREPROCESSING %%%%%%%%%%%%%%%%%%%%%
%write p-1=Q*2^s
Q=p-1;
s=0;
while mod(Q,2)==0
   Q=Q/2;
   s=s+1;
end
tempcurrq=SquareAndMultiply(curr,(Q-1)/2,p); %compute ans if in path 1/2
currq=mod(tempcurrq^2*curr,p);%% curr Curr^Q
twopow=1; % power of two currenrtly at

%% COMPUTE IF LIFE IN THE FAST LANE
if currq==1 % curr^odd=1  // PATH ONE
    [root,~,~]=ExtendedGCD(tempcurrq,p); % compute root
    root=mod(root,p);
    return
elseif currq==p-1 %curr^odd =-1 // PATH TWO
    if mod(p,4)==3
        root=0;
        return
    end
    [root,~,~]=ExtendedGCD(tempcurrq,p);
    root=mod(root*SquareRootOfMinusOne(p),p);
    return
else %life is hard
    if mod(p,8)==5
       root=0;
       return
    end
    rootminusone=currq; %sqrt(-1) is two iterations back
    previous=currq;
    while(twopow<s)
        currq=mod(currq^2,p); %repeated squaring
        if currq==1 %if QR
            break;
        end
        rootminusone=previous; %update values if QR
        previous=currq;
        twopow=twopow+1;
    end
    if currq ==p-1 % IF NR, PRINT RESULTS
       root=0;
       return
    end
end
%% END PREPROCESSING %%%%%%%%%%%%%
```

```matlab
%%%%%%%% REPEATED MULTIPLICATION %%%%%%%%%%%%%%%%%%%%%%%%%%
    %% INITALIZE VALUES FOR MULTIPLICATION
chi=C; %value that we are currently at
kappa=1; %compute number of sign switches
psi=1; %compute multiple
%create previous visited list
prev = containers.Map('KeyType','int32','ValueType','int32');
prev(chi)=1; %% visit curr
hashnum=2; %to try to speed up lookup
while true  % MAIN ALGO LOOP
   if floor(sqrt(chi))==sqrt(chi) %check if integer square
       break; %WE'RE DONE HERE
   end
   temp=p-chi;
   if floor(sqrt(temp))==sqrt(temp) %check if negative is integer square
      chi=p-chi; %FLIP
      kappa=kappa*-1; %UPDATE
      break; %WE'RE DONE HERE
   end
   if chi>2*n % FLIP IF IN Q3,Q4
       chi=p-chi;
       kappa=kappa*-1;
   end
   %% DETERMINE VALUE TO MULTIPLY BY
   pmult=1; %what target are we aiming for?
   while(true)
       sq=(round(sqrt((pmult*p)/chi)))^2; %compute possible square
       temp=mod(sq*chi,p); %compute temp
         %check if we've been here before
       if temp>3*n|| temp<=n
           if isKey(prev,temp)==0 && isKey(prev,p-temp)==0
              break
           end
       end
       % other wise update to next target and increment
       pmult=pmult+1;
   end
   chi=temp;% determine next value
   if(chi>2*n) %flip if need be
      chi=p-chi;
      kappa=kappa*-1;
   end
   psi=mod(psi*sqrt(sq),p);  %update
   prev(chi)=hashnum; %update previously visited values
   hashnum=hashnum+1;
end
if kappa==1
    rootminusone=1;
end
[inv,~,~]=ExtendedGCD(psi,p); %compute multiplicative inv
inv=mod(inv,p); %invert
root=mod(inv*kappa*sqrt(chi)*rootminusone,p);  %compute root
return
```

Goal of Preprocessing

Work out the easy cases

- Find the square root of these if applicable.
- Determine if NR, if so return no solution (see analysis)
- If the solution is not easy, find the $\sqrt{-1}$ quickly and proceed to repeated multiplication

What easy cases?

input $p = 4n + 1, C$. decompose $p - 1 = Q * 2^s$, s as large as possible.
Compute $C^Q \pmod{p}$.
If $C^Q \equiv \pm 1$, then $C \in QR_p$,

$$\sqrt{C} \equiv \pm (C^{\frac{Q-1}{2}})^{-1} \sqrt{\pm 1} \pmod{p} \tag{15}$$

Side note: Zagier Example

Exercise: Find $\sqrt{-1} \pmod{73}$.
Decompose $p = a^2 + b^2$.

Algorithm- Method of Zagier

Let $p = 4n + 1$ be prime. Initialize

$$(x, y, z) = (1, 1, \frac{p-1}{4}) \tag{16}$$

Iterate on

$$f(x, y, z) = \begin{cases} (x + 2z, y - z - x, z) & z + x < y \\ (2y - x, z + x - y, y) & z + x > y \end{cases} \tag{17}$$

until $y = z$.
Then $p = x^2 + (2y)^2$

Side note: Zagier Example

Exercise: Find $\sqrt{-1} \pmod{73}$.
$73 = 3^2 + 8^2$

Theorem- Finding $\sqrt{-1} \pmod{p}$

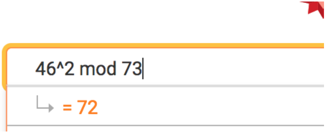
Let $p = a^2 + b^2$. Then

$$\sqrt{-1} = a^{-1}b, ab^{-1} \tag{18}$$

Side note: Zagier Example

Exercise: Find $\sqrt{-1} \pmod{73}$.
$73 = 3^2 + 8^2$
So

$$\sqrt{-1} = 3 * 8^{-1} \equiv 3 * 64 = 46 \pmod{73} \tag{19}$$



Harder cases

Let us assume it falls into neither of these cases.
Then we repeatedly square to get $(C^Q)^{2^k}, 0 < k < s$.
If we get $(C^Q)^{2^k} \equiv 1 \pmod{p}$, then $C \in QR_p$,

$$\sqrt{-1} \equiv (C^Q)^{2^{k-2}} \pmod{p} \tag{20}$$

Otherwise, $C \in NR_p$.

How much is caught by Preprocessing?

Case Number	Qualifier	Number of Instances
1	$C^Q \equiv 1 \pmod{p}$	Q
2	$C^Q \equiv -1 \pmod{p}$	Q
3	$C^{\frac{p-1}{2}} \equiv -1 \pmod{p}$	$\frac{p-1}{2}$
4	none of the above	$\frac{p-1}{2} - 2Q$

How much is caught by Preprocessing?

Case Number	Qualifier	Number of Instances
1	$C^Q \equiv 1 \pmod{p}$	Q
2	$C^Q \equiv -1 \pmod{p}$	Q
3	$C^{\frac{p-1}{2}} \equiv -1 \pmod{p}$	$\frac{p-1}{2}$
4	none of the above	$\frac{p-1}{2} - 2Q$

Theorem- Maximal value of Q

Let p be prime and $p - 1 = Q * 2^s$. If $p = 8n + 5$, then

$$2Q = \frac{p-1}{2} \tag{21}$$

What if we put $p = 4n + 3$ into preprocessing?

Goal of Repeated multiplication

Work out those cases that got through

Preconditions:

- $p = 4n + 1$
- $C \in QR_p$

We are jumping around the torus in a non trivial way, and then tracing back when we get to a perfect integer square.
Inspired by Front-Loading, and will benefit from it.

Process

Algorithm- Repeated Mulitplication

1. INPUT: $p = 4n + 1, C \in QR_p$
2. Initialize $\chi = C, \Psi = 1, \kappa = 1,$
3. While χ is not a perfect integer square and $-\chi \pmod{p}$ is not a perfect integer square:
    1. if χ is not in Quadrant I or II update $\chi \equiv -\chi \pmod{p}, \kappa = -\kappa$
    2. Select perfect integer square y_i^2 such that $\pm\chi y_i^2 \pmod{p}$ has yet to have been assigned to χ and update $\chi \equiv \chi * y_i^2 \pmod{p}, \Psi \equiv \Psi * y_i \pmod{p}$
4. if $-\chi \pmod{p}$ was the perfect integer square, update $\chi \equiv -\chi \pmod{p}, \kappa = -\kappa$
5. Return

$$\sqrt{C} \equiv \pm\Psi^{-1}\sqrt{\kappa\chi} \pmod{p} \tag{22}$$

Proof of accuracy

$$(y_1^2 y_2^2 \cdots y_{k-1}^2 y_k^2) C \equiv \sqrt{\chi}^2 \pmod{p} \tag{23}$$

$$(y_1 y_2 \cdots y_{k-1} y_k) \sqrt{C} \equiv \pm \sqrt{\chi} \sqrt{\pm 1} \pmod{p} \tag{24}$$

$$\sqrt{C} \equiv \pm (y_1 y_2 \cdots y_{k-1} y_k)^{-1} \sqrt{\chi} \sqrt{\pm 1} \tag{25}$$

$$\sqrt{C} \equiv \pm \Psi^{-1} \sqrt{\kappa \chi} \tag{26}$$

Proof of Termination

Theorem- Cardinality of Sets of Reciprocity

Let p be prime. Then

$$|QR_p| = \frac{p-1}{2} < \infty \tag{27}$$

Lemma- QR closure by integer squares

Let $p = 4n+1$, $C_1, C_2 \in QR_p$. Then there exists an integer y_i such that

$$y_i^2 C_1 \equiv C_2 \pmod{p} \tag{28}$$

But how do we choose y_i^2?

Algorithm- Selecting y_i^2

Select the least integer k such that

$$y_i^2 = \left[\sqrt{\frac{kp}{\chi}}\right]^2, \quad k \in \mathbb{N} \tag{29}$$

corresponds to a $\chi y_i^2 \pmod{p}$ has yet to be visited in the proceedings of the algorithm and this χy_i^2 is in Front-Loading Quadrants I or IV, where $[x]$ in this case rounds x to the nearest integer.

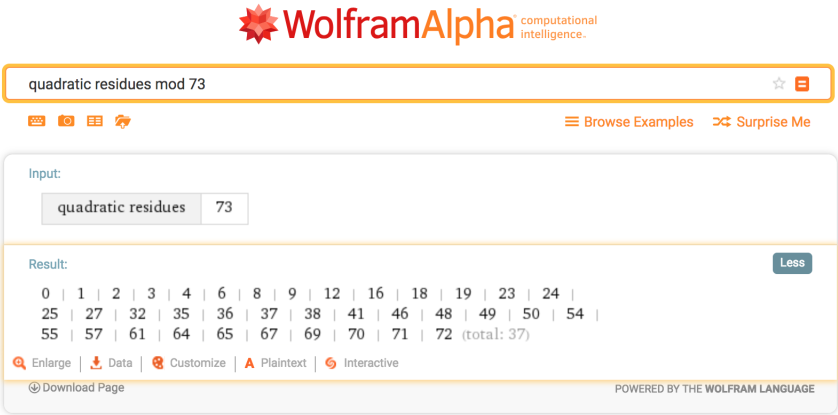
This is the place of Front-Loading.

Let's do an example!

Assume C got through preprocessing.

$p = 73 = 4(18) + 1$

$\sqrt{-1} \equiv 46 \pmod{73}$



Let's do an example (or two if we have time)!

$p = 73 = 4(18) + 1$
$\sqrt{-1} \equiv 46 \pmod{73}$

Algorithm- Repeated Mulitplication with y_i^2 choice

1. Initialize $\chi = C, \Psi = 1, \kappa = 1,$
2. While χ is not a perfect integer square and $-\chi \pmod{p}$ is not a perfect integer square:
    1. if χ is not in Quadrant I or II update $\chi \equiv -\chi \pmod{p}, \kappa = -\kappa$
    2. Select $y_i^2 = \left[\sqrt{\frac{kp}{\chi}}\right]^2$.
    3. update $\chi \equiv \chi * y_i^2 \pmod{p}, \Psi \equiv \Psi * y_i \pmod{p}$
3. if $-\chi \pmod{p}$ was the perfect integer square, update $\chi \equiv -\chi \pmod{p}, \kappa = -\kappa$
4. Return$\sqrt{C} \equiv \pm\Psi^{-1}\sqrt{\kappa\chi} \pmod{p}$

See thesis for more examples with other primes

What is there to analyze?

- Runtime
- Iteration Count/Operation count (kind of)
- Potential improvements to the algorithm's design

Runtime statistics

p	AlgoRuntime	Brute-force Runtime
17	.010s	.001s
97	.016s	.002s
617	.047s	.018s
977	.081s	.043s
1361	.115s	.080s
2377	.260s	.246s
3041	.378s	.414s
3313	.484s	.482s
4721	.594s	1.023s

Conjecture- Runtime of Algorithm

Let $w \approx 3500$. Our algorithm beats brute force in terms of runtime for a prime p if $p = 4n + 1 > w$. This is an empirical result.

What is slowing the algorithm down?

Algo (Calls: 10000, Time: 5.433 s)

Generated 03-Sep-2018 15:02:38 using performance time.
function in file /Users/michaelspink/Documents/MATLAB/Algo.m
Copy to new window for comparing multiple runs

Refresh

- [x] Show parent functions
- [x] Show busy lines
- [x] Show child functions
- [x] Show Code Analyzer results
- [x] Show file coverage
- [x] Show function listing

Parents (calling functions)

Function Name	Function Type	Calls
TestSuitev2	function	10000

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
119	`prev(chi)=hashnum; %update pre...`	96625	1.898 s	34.9%	
36	`tempcurrq=SquareAndMultiply(cu...`	10000	1.257 s	23.1%	
106	`if isKey(prev,temp)==0 &&a...`	96625	1.170 s	21.5%	
51	`root=mod(root*SquareRootOfMinu...`	1231	0.569 s	10.5%	
82	`prev = containers.Map('KeyType...`	2436	0.113 s	2.1%	
All other lines			0.426 s	7.8%	
Totals			5.433 s	100%	

String to Binary
Lower Level Implementation?

Side note: A discussion of MATLAB

Why bother using MATLAB?

- Ease of use
- Graphing abilities
- GNU Octave
- Detailed runtime stats- We use the time and run option

A different view on performance- Iteration count

Definition- Iteration for Brute Force

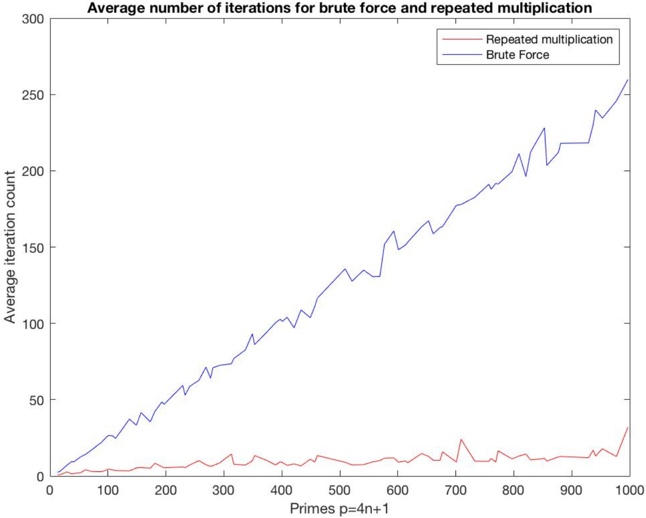
We define one iteration for brute force as every integer that we square and compare to C.

Definition- Iteration for our algorithm

We define one iteration for our algorithm as every instance we compute an invalid y_i^2, and every time we update our bookkeeping for Ψ.

Does this translate to Operation count? Kind of.

Iteration count



Improving design

- Can we remove Preprocessing?
  - Work decidability into Repeated Multiplication
  - Collatz interpretation
  - Overflow
- Optimal choice of y_i^2

Theorem- Consecutive set of QR_p

Let $p = 4n + 1$ be prime. Then $QR_p = \{1^2, 2^2, 3^2 \cdots (2n-1)^2, (2n)^2\}$

Future Work

For the Future:

- Prove the Front-Loading Conjecture.
- Improve the algorithm by implementing in a lower level language?
- Improve the algorithm by optimization?
- Implement an NR condition into the repeated multiplication?
- Explore other shifts of the problem?

Conclusion

We have shown a new approach to the well studied Square Root Problem. We have found the Front-Loading Conjecture to be fascinating and to have merit into understanding the distribution of quadratic residues. And from this, we found a new algorithm to find the square root, and have shaken it for all it is worth. We find the algorithm to be

- Easy to work by hand
- Cross Disciplinary approach
- More tractable than Brute Force
- More transparent than the other algorithms for the problem

And thus we think that our algorithm is worth discussing and plausible to implement when refined.

Acknowledgements to people at home

Names redacted to protect identities of people.

In summary, thanks to my parents, siblings, and friends for constant support and making me the person I am today.

Acknowledgements to people at RIT

- My Committee
  - Dr. Lopez – My advisor; guiding me here, PiRIT help, and for putting up with my erratic schedule. I couldn't have done this without you.
  - Dr. Agarwal – The smartest man I've had the pleasure of working with.
  - Dr. Marengo – Helping me through Exam P and the Putnam
- Everyone else at RIT that has supported me. Names redacted for the same reason as above.

And everyone else that I forgot due to trying to keep this as concise as possible. This slide alone was initially more than four slides long.

References I

Agarwal, A. [Lecture notes] (Fall 2017). MATH 771: Mathematics of Cryptography Lectures. Lectures presented at Rochester Institute of Technology, Rochester NY.

Bernstein, D. (2001). Faster Square Roots in Annoying Finite Fields; Draft. Retrieved from https://www.researchgate.net/publication/2381439_Faster_Square_Roots_in_Annoying_Finite_Fields

Cannas da Silva, A. (2006). Lectures on Symplectic Geometry. Retrieved from https://people.math.ethz.ch/~acannas/Papers/lsg.pdf

Guy, Q. (2012, October 10). Fast String to Double Conversion. Retrieved August 16, 2018, from Mathworks: File Exchange website: https://www.mathworks.com/matlabcentral/fileexchange/28893-fast-string-to-double-conversion?s_tid=mwa_osa_a

Jones, G. A., & Jones, J. M. (2005). Springer Undergraduate Mathematics Series: Elementary Number Theory (8th ed.). Springer.

LeVeque, W. J. (1977). Fundamentals of Number Theory (2015 ed.). New York, NY: Dover Publications.

Nichols, John (2016). A New Algorithm For Computing the Square Root of a Matrix. Retrieved from https://scholarworks.rit.edu/theses/9265/

Peralta, R. (1992). On the Distribution of Quadratic Residues and Nonresidues Modulo a Prime Number. Mathematics of Computation, 58(197), 433-440. https://doi.org/10.1090/S0025-5718-1992-1106978-9

Pocklington, H.C. (1917). The Direct Solution of the Quadratic and Cubic Binomial Congruences with Prime Moduli. Proceedings of the Cambridge Philosophical Society, XIX, 57-59. Retrieved from https://archive.org/stream/proceedingsofcam1920191721camb/proceedingsofcam1920191721camb_djvu.txt

Printer, C. C. (1990). A Book of Abstract Algebra (2nd ed.). New York, NY: Dover.

References II

Quadratic Nonresidue. (n.d.). Retrieved August 26, 2018, from Wolfram Mathworld website: http://mathworld.wolfram.com/QuadraticNonresidue.html

Quadratic Residue. (n.d.). Retrieved August 16, 2018, from Wolfram Mathworld website: http://mathworld.wolfram.com/QuadraticResidue.html

Schlenk, F. (2018), Symplectic Embedding Problems, Old and New, Bulletin of the AMS, 55(2), p.139-182 https://doi.org/10.1090/bull/1587

Schoof, R. (1985). Elliptic Curves Over Finite Fields and the Computation of Square Roots mod p. Mathematics of Computation, 44(170), 483-494. https://doi.org/10.2307/2007968

Shirali, S. A. (2003). 6: On Fermat's Two Squares Theorem. In S. A. Shirali (Author)& C. S. Yogananda (Ed.), Number Theory (pp. 30-33). Retrieved from https://books.google.com/books?id=BkBSfFjT1BgC&pg=PA30&lpg=PA30&dq#v=onepage&q&f=false

Tornaría, G. (2002). Square Roots Modulo p. Latin American Symposium on Theoretical Informatics, 430-434. https://doi.org/10.1007/3-540-45995-2_38

Turner, S. M. (1994). Square Roots mod *p*. The American Mathematical Monthly, 101(5), 443-449. https://doi.org/10.2307/2974905

Thanks again everyone for coming.