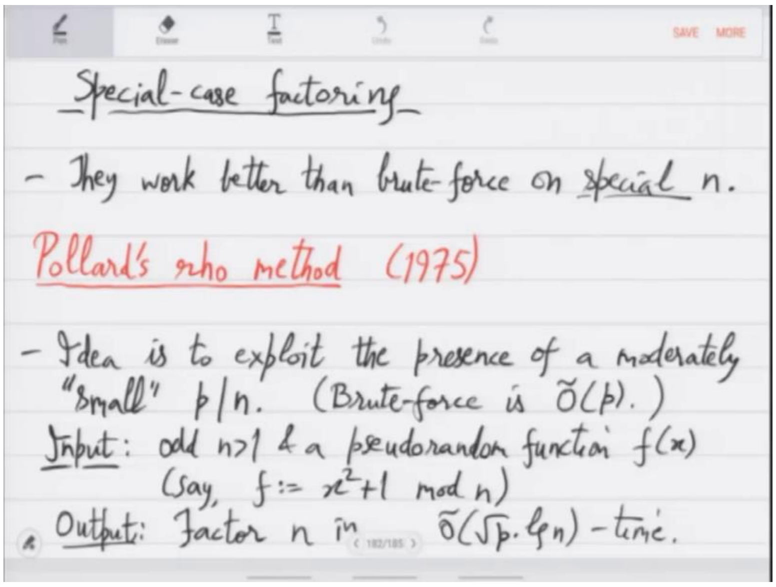
Computational Number Theory and Algebra
Prof. Nitin Saxena
Department of Aerospace Engineering
Indian Institute of Technology - Kanpur

Lecture – 27
Pollard's p-1, Fermat, Morrison-Brillhart, Quadratic and Number Field Sieve Methods

(Refer Slide Time: 00:25)



Special-case factoring

- They work better than brute-force on special n.

Pollard's rho method (1975)

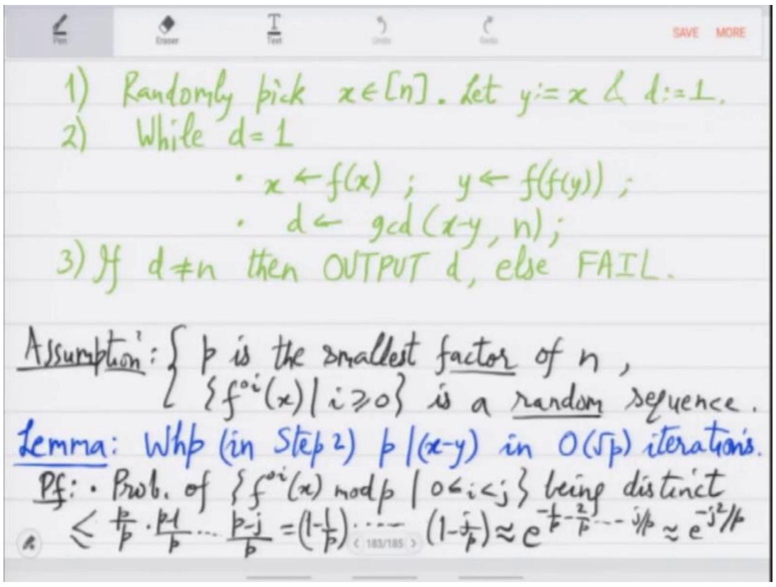
- Idea is to exploit the presence of a moderately "small" $p | n$. (Brute-force is $\tilde{O}(p)$.)

Input: odd $n > 1$ & a pseudorandom function $f(x)$ (Say, $f := x^2+1 \bmod n$)

Output: Factor n in $\tilde{O}(\sqrt{p} \cdot \lg n)$-time.

So in the last class we saw the first integer factoring algorithm which is called Pollard's Rho method what it does is these 3 steps in green.

(Refer Slide Time: 00:27)



```
1) Randomly pick $x \in [n]$. Let $y := x$ & $d := 1$.
2) While $d = 1$
       · $x \leftarrow f(x)$ ; $y \leftarrow f(f(y))$ ;
       · $d \leftarrow \gcd(x-y, n)$ ;
3) If $d \neq n$ then OUTPUT $d$, else FAIL.
```

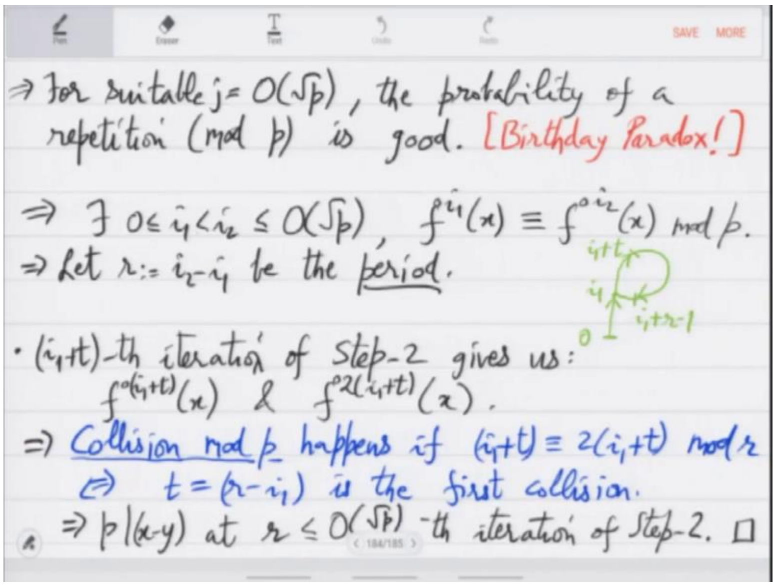
Assumption: $\begin{cases} p \text{ is the smallest factor of } n, \\ \{f^{\circ i}(x) \mid i \geq 0\} \text{ is a random sequence.} \end{cases}$

Lemma: Whp (in Step 2) $p | (x-y)$ in $O(\sqrt{p})$ iterations.

Pf: · Prob. of $\{f^{\circ i}(x) \bmod p \mid 0 \leq i < j\}$ being distinct
$\leq \frac{p}{p} \cdot \frac{p-1}{p} \cdots \frac{p-j}{p} = (1-\frac{1}{p}) \cdots (1-\frac{j}{p}) \approx e^{-\frac{1}{p}-\frac{2}{p}-\cdots-j/p} \approx e^{-j^2/p}$

It basically picks a function f for example you can take f to be x square + 1 and it starts with a random x applies f once iteratively and applies f twice iteratively. So you get a sequence x and a sequence y and then in every iteration it takes the difference and gcd with n. So if this factors n well and good else you declare that the algorithm has failed.

(Refer Slide Time: 01:04)



$\Rightarrow$ For suitable $j = O(\sqrt{p})$, the probability of a repetition (mod p) is good. [Birthday Paradox!]

$\Rightarrow \exists\, 0 \le i_1 < i_2 \le O(\sqrt{p})$, $f^{\circ i_1}(x) \equiv f^{\circ i_2}(x) \bmod p$.

$\Rightarrow$ Let $r := i_2 - i_1$ be the $\underline{\text{period}}$.

i_1+t
i_1
i_1+r-1
0

• (i_1+t)-th iteration of Step-2 gives us:

$$f^{\circ(i_1+t)}(x) \ \& \ f^{\circ 2(i_1+t)}(x).$$

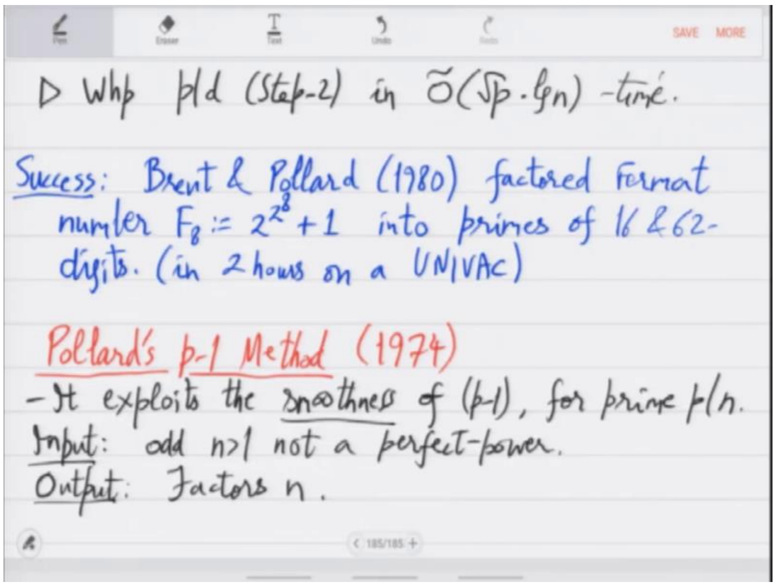
$\Rightarrow$ $\underline{\text{Collision mod } p}$ happens if $(i_1+t) \equiv 2(i_1+t) \bmod r$

$\Leftrightarrow$ $t = (r - i_1)$ is the first collision.

$\Rightarrow p \mid (x-y)$ at $r \le O(\sqrt{p})$-th iteration of Step-2. $\square$

So what yesterday we showed in the previous class we showed by Birthday Paradox that if prime p divides n the let us say the p is the smallest prime dividing n then in the square root ph iteration step 2 will factor.

(Refer Slide Time: 01:25)



$\triangleright$ Whp $p \mid d$ (Step-2) in $\tilde{O}(\sqrt{p} \cdot \lg n)$-time.

$\underline{\text{Success}}$: Brent & Pollard (1980) factored Fermat number $F_8 := 2^{2^8} + 1$ into primes of 16 & 62-digits. (in 2 hours on a UNIVAC)

$\underline{\text{Pollard's } p\text{-1 Method}}$ (1974)

– It exploits the $\underline{\text{smoothness}}$ of $(p-1)$, for prime $p \mid n$.

$\underline{\text{Input}}$: odd $n > 1$ not a perfect-power.

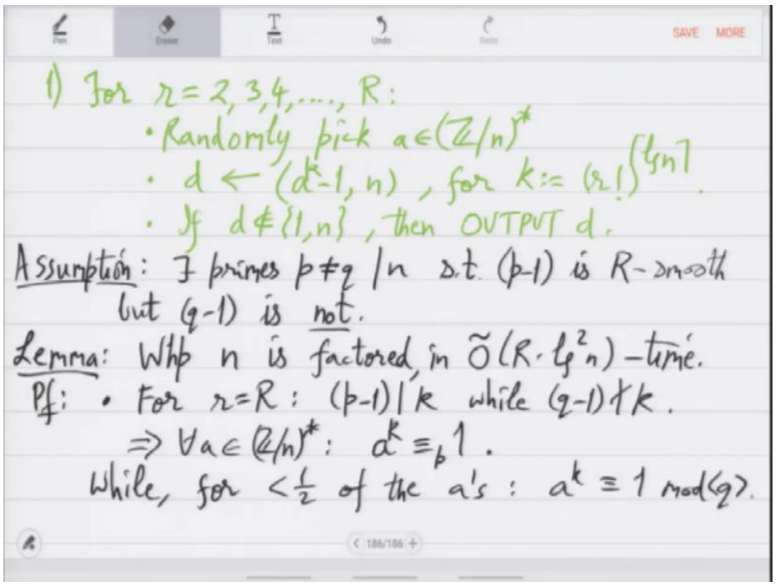
$\underline{\text{Output}}$: Factors n.

So this is a square root p times log n time algorithm and as an example of its success this algorithm Brent and Pollard implemented it in 1980 and they factored a big integer called the Fermat number F 8 which is 2 raised to 2 raised to 8 + 1 and they found 2 prime factors so into 2 primes 16 and 62 digits. So this is around F 8 is a number of around 70, 80 digits and Brent and Pollard were able to factor this into 2 primes 1 is 16 digit other is 62 digit.

16 digit prime is this p that we used in the analysis and since it is relatively smaller so square root of this much time it took which was around 2 hours in those days on an old univac machine. So it was quite fast for such a big integer. Let us move to the next algorithm Pollards p - 1 method. This will be different from the row method around the same time. This will do something different instead of assuming that p is small it will assume that p - 1 has small prime factors so smooth.

So it exploits the smoothness of p - 1 for a prime factor p of n. So in the input again you get an odd n which is not a perfect power. In the output you want a factor. The idea of this method is use the known prime factors of p - 1 to compute a raise to p - 1 roughly mod n.

(Refer Slide Time: 05:02)



1) For $r = 2, 3, 4, \ldots, R$:
- Randomly pick $a \in (\mathbb{Z}/n)^*$
- $d \leftarrow (a^k - 1, n)$, for $k := (r!)^{\lceil \lg n \rceil}$.
- If $d \notin \{1, n\}$, then OUTPUT d.

Assumption: $\exists$ primes $p \neq q \mid n$ s.t. $(p-1)$ is R-smooth but $(q-1)$ is not.

Lemma: Whp n is factored, in $\tilde{O}(R \cdot \lg^2 n)$-time.

Pf: • For $r = R$: $(p-1) \mid k$ while $(q-1) \nmid k$.

$\Rightarrow \forall a \in (\mathbb{Z}/n)^*$: $a^k \equiv_p 1$.

While, for $< \frac{1}{2}$ of the a's : $a^k \equiv 1 \bmod \langle q \rangle$.

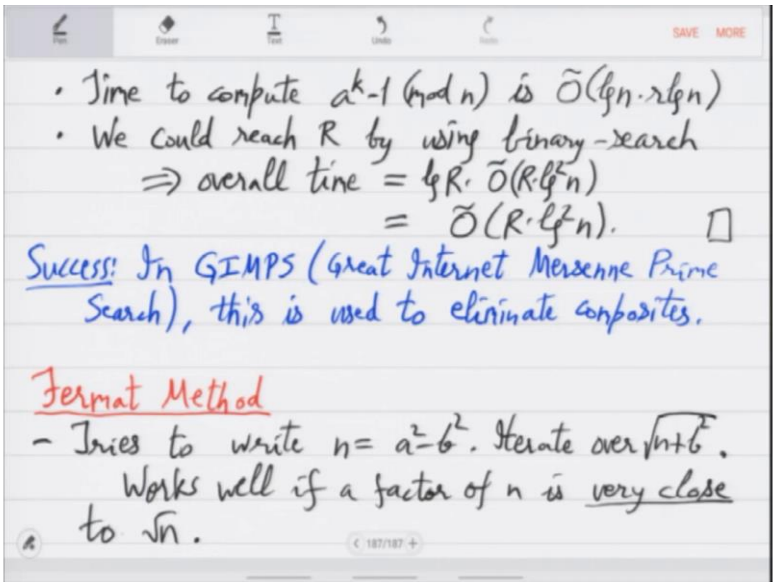
So the algorithm will do the following. So randomly pick and a and compute a raised to k – 1 gcd with n and k what is k k is this basically it is a product of small primes you do not will you will not know p - 1 but you can still use the prime factors of it with maximum possible exponent. So you can use k to be r factorial so product of 1 2 dot dot r and then raise it to a large enough exponent let us say log n.

So if you look at this a raise to k - 1 mod p since p - 1 divides k a raised to p - 1 will be 1. So p actually divides d that is the point of this step. So if d already factors in then you then you succeed. So then you output d as the factor else itself the algorithm has failed. So you can go up to some big bound r do this for all small r from 2 to big R. So let us take the following assumption there exist primes p different from q both dividing n such that p - 1 is r smooth but q - 1 is not.

So this is the difference between p - 1 and q - 1 and using this a raised to k - 1 will factor n. So that is contained in this lemma. So with high probability n is factored in R times log square n times. So as I said the proof is fairly easy you just observe that for this bound when r reaches big R what happens is p - 1 starts dividing k while q - 1 will not divide k which means that for every a in z n star a raised to k is 1 mod p.

While for less than half of these a raised to k is 1 mod q so for all the a's e raised to k is 1 mod p but that is not the case mod q. So there is a difference d will factor n.

(Refer Slide Time: 10:10)



- Time to compute $a^k - 1 \pmod n$ is $\tilde{O}(\lg n \cdot r \lg n)$
- We could reach R by using binary-search

$\Rightarrow$ overall time $= \lg R \cdot \tilde{O}(R \lg^2 n)$

$= \tilde{O}(R \cdot \lg^2 n)$. □

Success: In GIMPS (Great Internet Mersenne Prime Search), this is used to eliminate composites.

Fermat Method

- Tries to write $n = a^2 - b^2$. Iterate over $\sqrt{n+b^2}$. Works well if a factor of n is very close to $\sqrt{n}$.

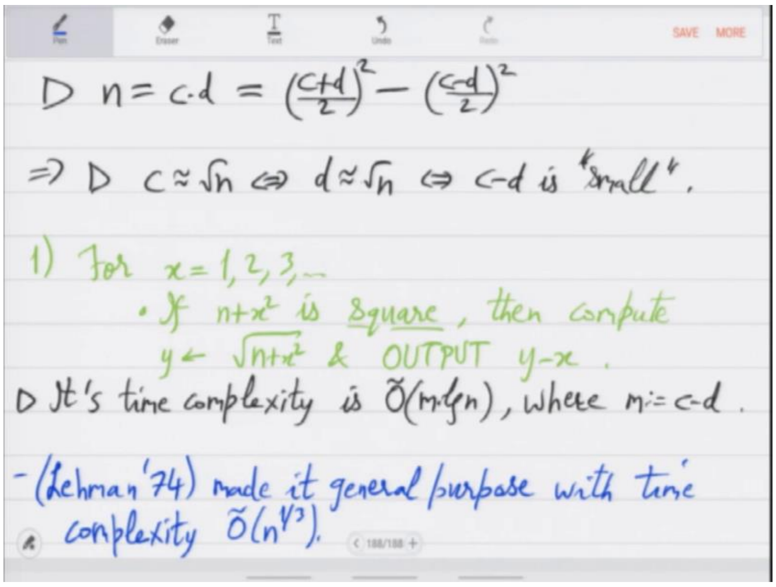
So time to compute a raise to k - 1 mod n is O tilde log n times r log n that is how big log k is yes around r log n. And we could reach R by using binary search. So which implies that overall time is you will be multiplying not by r but log R so which is O tilde R log square n as promised in the lemma statement. So that is the bound what are the successes of this algorithm Pollards p - 1 method.

So this is used in the great internet Mercein prime search so called GIMPS. So in this in this global in this international project GIMPS distributed computing is used to find prime numbers and there the Pollards p - 1 method is used to weed out composites. Let us move to the third method Fermat method. So Fermat method is something even simpler what it does is it promises to factor in when p and q are very close.

The 2 prime factors are actually there if as long as n has 2 factors which are very close this method will factor n. So it basically uses this fact it tries to write n as a difference of squares and so basically iterate over n + b square and take a square root. So it will basically it will vary b and try to and will check whether n + b square is a perfect square and if for some b this works then you have a square - b square representation and we who which factors as a - b times a + b.

So this works well if a factor of n is very close to square root n if 1 factor is very close to square root n. The other is also very close to square root n then the difference actually is relatively small and it is the difference which this method will utilize.

(Refer Slide Time: 15:29)



▷ $n = c\cdot d = \left(\frac{c+d}{2}\right)^2 - \left(\frac{c-d}{2}\right)^2$

$\Rightarrow$ ▷ $c \approx \sqrt{n} \Leftrightarrow d \approx \sqrt{n} \Leftrightarrow c-d$ is 'small'.

1) For $x = 1, 2, 3, \ldots$
- If $n+x^2$ is square, then compute $y \leftarrow \sqrt{n+x^2}$ & OUTPUT $y-x$.

▷ It's time complexity is $\tilde{O}(m \cdot \lg n)$, where $m := c-d$.

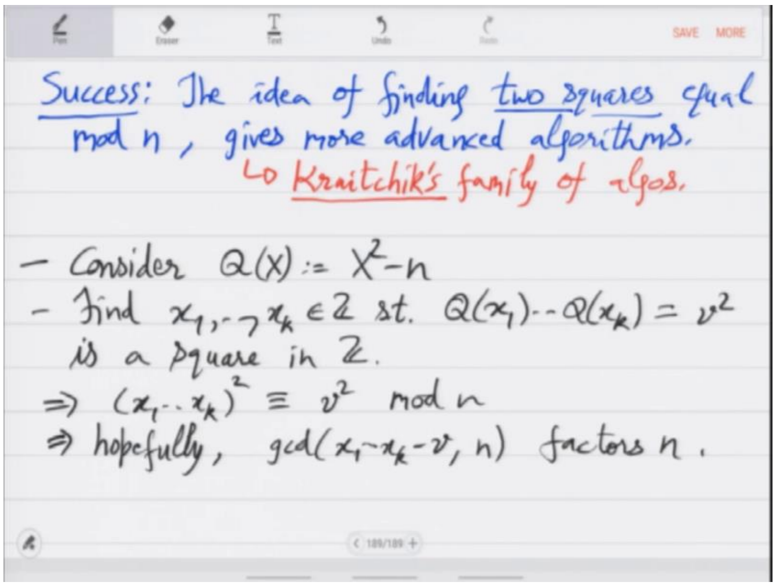
- (Lehman'74) made it general purpose with time complexity $\tilde{O}(n^{1/3})$.

So basically if n is equal to c times d you can write it as c + d by 2 whole square - c - d by 2 whole square and if c - d is small c and d are very close then you can find c - d by 2 by brute force. So if c is extremely close to square root n then d is also very close to square root n which implies that c - d is very small. So find it by brute force that is the idea. So let us see the algorithm just in just a for loop.

So for x 1 2 so on if n + x square is a square then compute its square root and output y - x that is a factor of n right n is equal to y square - x square. So it is a very simple algorithm. So it is time complexity is m times log n where m is the difference between the 2 factors. So if the difference is small then it is a fast algorithm based on this idea or this method of Fermat. Lehman actually gave a n to the 1 third factoring algorithm for general n.

So we will not discuss this but you can read the read the proof online it is not difficult. Sso using forma method you can actually factor any number in n to the 1 third time which is slightly better than brute force which is square root n. So more importantly this Fermat method what it is doing is modulo n it is giving you 2 squares that are equal right. So this is the idea which other people have extended to larger and larger ring constructions and that is what gives the most advanced integer factoring algorithms currently known.

(Refer Slide Time: 19:44)



Success: The idea of finding two squares equal mod n, gives more advanced algorithms.

↳ Kraitchik's family of algos.

- Consider $Q(X) := X^2 - n$
- Find $x_1, \ldots, x_k \in \mathbb{Z}$ st. $Q(x_1) \cdots Q(x_k) = v^2$ is a square in $\mathbb{Z}$.

$\Rightarrow (x_1 \cdots x_k)^2 \equiv v^2 \bmod n$

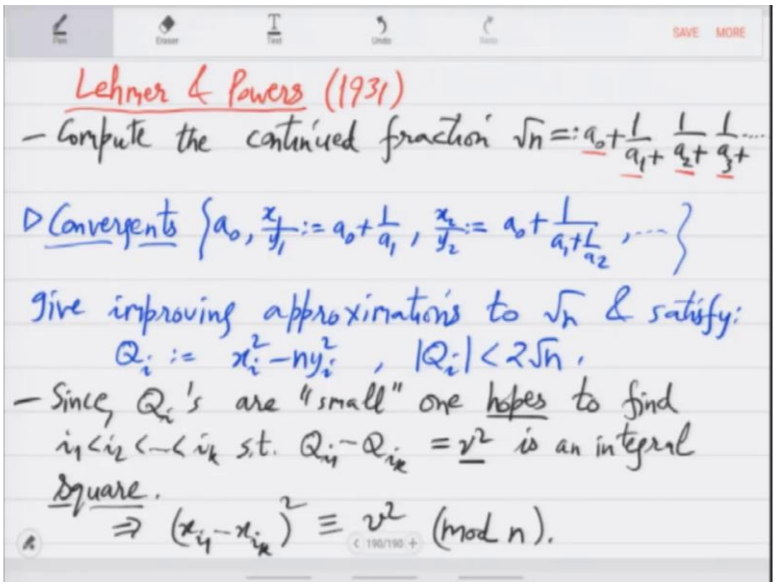
$\Rightarrow$ hopefully, $\gcd(x_1 \cdots x_k - v, n)$ factors n.

So the idea of finding 2 squares that are equal mod n this gives more advanced algorithms. So these algorithms are called Kraitchik's family of algorithms. They are based on finding 2 squares equal mod n. So let us take this idea further to show some famous advanced algorithms so consider this polynomial x square - n mod n it is a square. Find numbers x 1 to x k such that Q x 1 dot dot Q x k is a square.

So now on the lhs Q x 1 dot dot Q x k this is a square mod n. And on the RHS let me not define let me just say that its equal to a integral square. So on the RHS obviously you have a square. So you have 2 squares that are the same mod n they are both squares mod n and they are also same. So that then looks like a square equal to b square and you can hope that a - b will factor n. So this implies that x 1 dot dot x k square is v square mod n.

So hopefully the gcd of x 1 dot dot x k - v with n factors n that is the that is the idea which is common to all these difference of square based algorithms. So let so we will now look at some implementations of this concrete implementations.

(Refer Slide Time: 23:41)



Lehmer & Powers (1931)

– Compute the continued fraction $\sqrt{n} =: a_0 + \frac{1}{a_1+} \frac{1}{a_2+} \frac{1}{a_3+} \cdots$

▷ Convergents $\left\{a_0, \frac{x_1}{y_1} := a_0 + \frac{1}{a_1}, \frac{x_2}{y_2} := a_0 + \frac{1}{a_1 + \frac{1}{a_2}}, \ldots\right\}$

give improving approximations to $\sqrt{n}$ & satisfy:

$Q_i := x_i^2 - ny_i^2$, $|Q_i| < 2\sqrt{n}$.

– Since Q_i's are "small" one hopes to find $i_1 < i_2 < \ldots < i_k$ s.t. $Q_{i_1} \cdots Q_{i_k} = v^2$ is an integral square.

$\Rightarrow (x_{i_1} \cdots x_{i_k})^2 \equiv v^2 \pmod{n}$.

So how do you the whole game is how do you construct these squares right. So the first idea is by lemur and powers it is a fairly old idea they would use what is called continued fractions to construct these x 1 to x k. So compute the continued fraction. So square root n first finds the integer closest to it a 0. Then find the best a1 such that a0 + 1 over a1 is a good approximation for square root n.

Then you go then you add a fraction to a 1 that should give you 1 over a 2 then you add a fraction to a 2 that should give you 1 over a 3 and so on this is called a continued fraction it gives you these integers. Successively giving you a better and better approximation of square root n and what is this representation good for? So if you look at the convergence the ratios that you are getting first is a 0 next is a 0 + 1 over a 1 third is a 0 + 1 over a 1 + 1 over a 2 and so on.

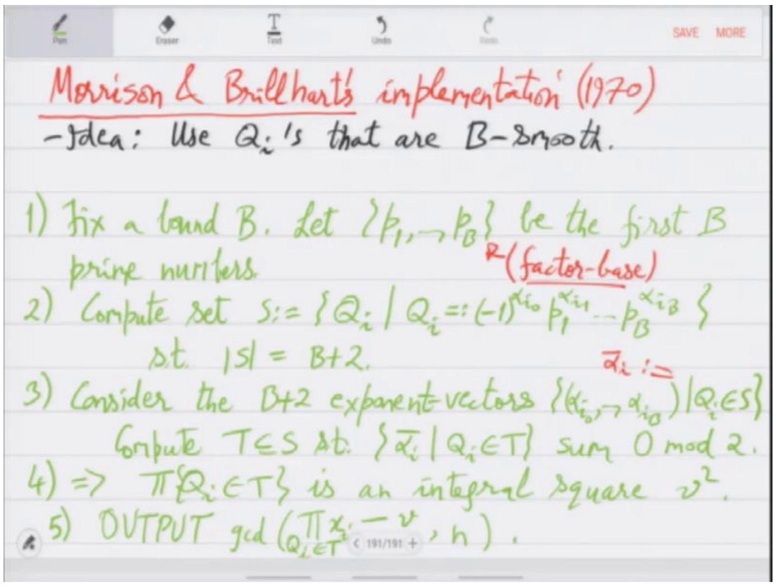
These are called convergence they it is basically a sequence of fractions that is getting closer and closer to square root n. So how close a 0 then this x 1 over y 1 which is a 0 + 1 over a 1 then x 2 over y 2 which is a 0 + 1 over a 1 + 1 over a 2 and so on. This sequence x 0 by y 0 then x 1 by y 1 then x 2 by y 2 they give improving approximations rational approximations to square root n and satisfy the satisfying amazing inequality which is that if you look at x i square - n y i square.

Then this number is at most 2 square root n. So kind of this difference of square but there is an n weight here x. So if you look at x i square - n y i square this number is smaller than 2 root n in value. So this is a property of the convergence from continued fraction right. So this is the deterministic process you write square root n as a continued fraction you find the convergence and now you have these q i's which are smaller than n it is in fact round square root n.

And look at their product so that product mod n is a square. So since these Q i’s are small they are not very small but they are smaller than the trivial thing which is n. So since Q i's are small one hopes to find a subsequence i 1 to i k such that Q i 1 to Q y k is an integral square. And once you have this v then you look mod n. So then this x i 1 dot dot x i k whole square is v square mod n. So you are in this situation of equal squares mod n and then hopefully you will factor n.

So this is the lemur powers approach using continued fraction but still this is not completely giving you all the steps because you do not know how to find this i 1 to i k. So we will see there are several implementations of this let us see one.

(Refer Slide Time: 30:00)



Morrison & Brillhart's implementation (1970)

- Idea: Use Q_i's that are B-smooth.

1) Fix a bound B. Let $\{p_1, \ldots, p_B\}$ be the first B prime numbers (factor-base)
2) Compute set $S := \{Q_i \mid Q_i =: (-1)^{\alpha_{i0}} p_1^{\alpha_{i1}} \cdots p_B^{\alpha_{iB}}\}$ s.t. $|S| = B+2$.
3) Consider the B+2 exponent-vectors $\{\overline{\alpha}_i := (\alpha_{i0}, \ldots, \alpha_{iB}) \mid Q_i \in S\}$. Compute $T \subseteq S$ s.t. $\{\overline{\alpha}_i \mid Q_i \in T\}$ sum 0 mod 2.
4) $\Rightarrow \prod\{Q_i \in T\}$ is an integral square v^2.
5) OUTPUT $\gcd\left(\prod_{Q_i \in T} x_i - v, n\right)$.

So Morrison and Brillhart's implementation it is called. So this implementation will be will also give you an analysis the time complexity etcetera. It will be based on the as promised before it will be based on smooth number density. So this will actually focus on these on finding these Q i 1 Q i 2 Q y k such that they are B smooth for a bound B. It will play with the smoothness parameter.

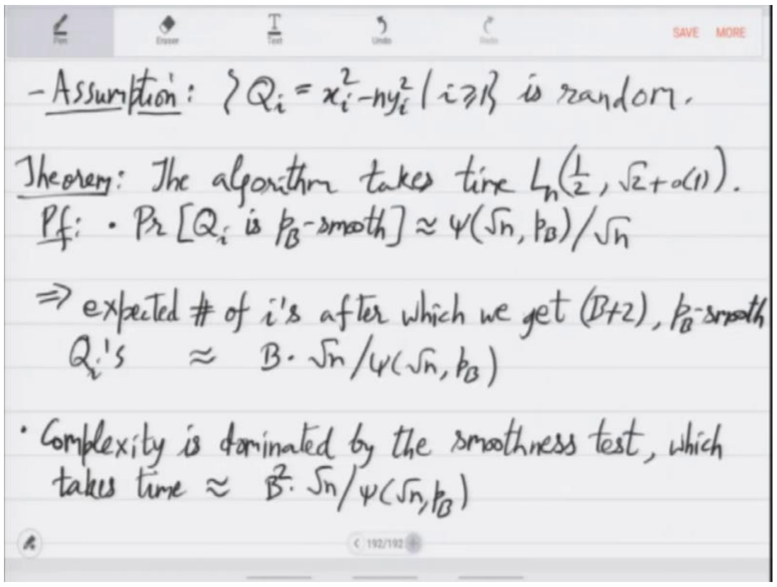
So the idea is use these Q i’s that are B smooth for a suitably chosen parameter B. So let us see the algorithm all the steps. So fix a bound B function of n considers primes up to that bound. So let us let us just consider the first B prime numbers and so this is called the factor base. So you have fixed a factor base and every Q i you will try to check whether it satisfies this factor base.

So basically you look at the Q i's that factor using just p 1 that are p B smooth and find the B + 2 many elements in the set. So you will be trying out Q i's sequentially Q 1 Q 2 Q 3 so on and you will pick those b + 2 many that are p B smooth that is the status. Now look at the exponent vectors. So, alpha 0 dot dot alpha i B for every Q i in s. Let us call this alpha i bar. Now notice that you have B + 2 vectors there are each has B + 1 coordinates.

So they will be linearly dependent mod 2. If you look at them mod 2 they are 01 vectors and since you have more vectors then there are coordinates they will be linearly dependent. So pick such a dependency. So compute the subset T such that alpha i bar in the subset mod 2. So what does that mean? So the above thing implies that now these Q i's if you pick from T C and multiply them that is a square.

So we have this is how it finds these so by picking p B smooth numbers Q i’s a lot of them it finds a subset such that the product is a square. And once it has this it will do the usual thing the gcd. So, product of x i for these Q i's - v gcd within so how do we analyze this? First thing we will assume is that Q i's are random.

(Refer Slide Time: 37:12)



- Assumption: $\{Q_i = x_i^2 - ny_i^2 \mid i \geq 1\}$ is random.

Theorem: The algorithm takes time $L_n(\frac{1}{2}, \sqrt{2}+o(1))$.

Pf: • $\Pr[Q_i \text{ is } p_B\text{-smooth}] \approx \psi(\sqrt{n}, p_B)/\sqrt{n}$

$\Rightarrow$ expected # of i's after which we get $(B+2)$, p_B-smooth Q_i's $\approx B \cdot \sqrt{n}/\psi(\sqrt{n}, p_B)$

• Complexity is dominated by the smoothness test, which takes time $\approx B^2 \cdot \sqrt{n}/\psi(\sqrt{n}, p_B)$

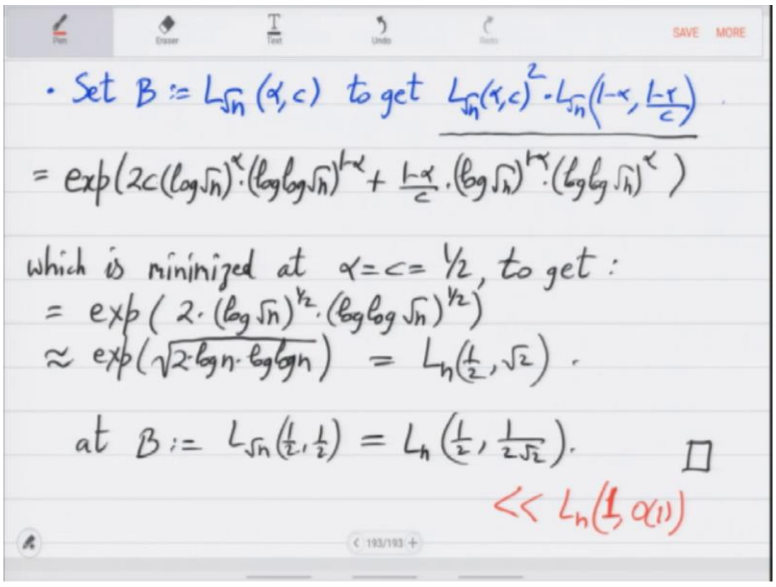
That is a fair assumption because it is a non linear function. This is a random sequence assuming that we will show that the previous algorithm in green it takes time L n half square root 2 slightly more than square root 2 maybe. So this is the first analysis which is significantly better than brute force. Instead of L n 1 this is ln half right that is the important thing. So how is such a thing shown.

So again this will be by smooth number analysis or estimate. So probability that Q i is p B smooth this use the smooth number estimate. So you will get psi note that Q i is around square root n right. So up to square root n how many p B smooth numbers are there? It is this. So which means so how many Q i's do you have to try to get b + 2 p B smooth ones. So expected number of i’s after which we get b + 2 p B smooth Q i’s this expected number is so roughly B + 2 times this 1 over the probability.

So you get square root n divided by psi how much time will the algorithm take? So the time complexity is dominated by the smoothness test which takes so already you know how many i's you will be trying which is B times square root n by psi. For every Q i what are you doing? You are checking whether it is p B smooth. So you are dividing by p 1 to p B which is B divisions right.

So let us multiply by B so you will get b square times square root n divided by psi that is the time complexity. So what is the B you should choose as a function of n right that is the remaining thing.

(Refer Slide Time: 42:09)



- Set $B := L_{\sqrt{n}}(\alpha, c)$ to get $\underline{L_{\sqrt{n}}(\alpha,c)^2 \cdot L_{\sqrt{n}}\left(1-\alpha, \frac{1-\alpha}{c}\right)}$.

$$= \exp\left(2c(\log\sqrt{n})^{\alpha}\cdot(\log\log\sqrt{n})^{1-\alpha} + \frac{1-\alpha}{c}\cdot(\log\sqrt{n})^{1-\alpha}\cdot(\log\log\sqrt{n})^{\alpha}\right)$$

which is minimized at $\alpha = c = 1/2$, to get :

$$= \exp\left(2\cdot(\log\sqrt{n})^{1/2}\cdot(\log\log\sqrt{n})^{1/2}\right)$$

$$\approx \exp\left(\sqrt{2\log n\cdot\log\log n}\right) = L_n\left(\tfrac{1}{2}, \sqrt{2}\right).$$

at $B := L_{\sqrt{n}}\left(\tfrac{1}{2}, \tfrac{1}{2}\right) = L_n\left(\tfrac{1}{2}, \tfrac{1}{2\sqrt{2}}\right)$. $\square$

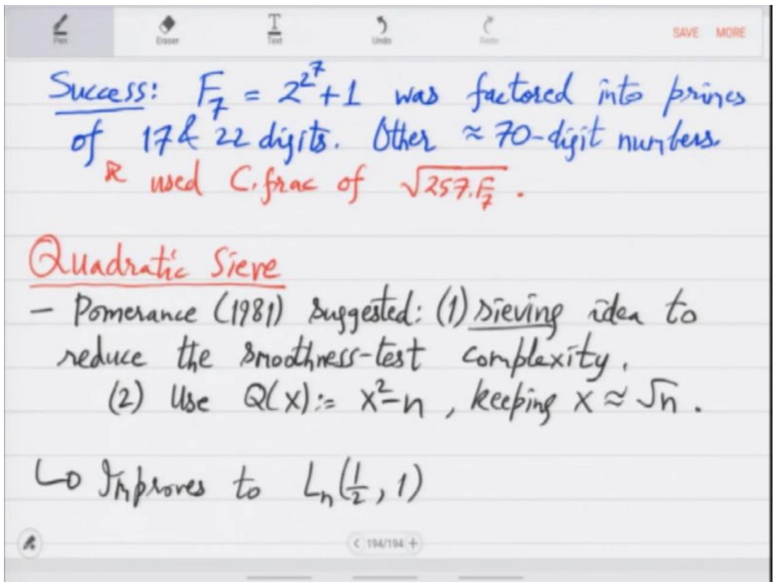
$<< L_n(1, \alpha(1))$

So we will set B to be L square root n alpha, c to get b square times inverse of this probability right. So it is square root n 1 - alpha 1 - alpha by c. So this is the time complexity let us just fix alpha and c constant suitably. So this is what this is by definition a raised to so you are squaring so in the exponent it is doubling + 1 - alpha by c right that is the expression. So first you have alpha then you have 1 - alpha now this will be minimum when you set them to be equal.

So which is minimized at alpha will be fixed to half and then a also make 2 c and 1 - alpha by c equals. So c will also be half. So it will become exp log of square root n half times log log square root n half at the exact and which is so log square root n a half comes out with the square root right. So you get around 2 log n log log n square root which is L n half square root 2 that is the time complexity right L n half, square root 2 this is what you get.

And B is B will be if you work out l square root n half, half. So you will get L n half, 1 over 2 square root 2. So for that large of B you will get similar time complexity. So this is the first integer factoring analysis you see. It is significantly better than Ln 1.

(Refer Slide Time: 47:13)



Success: $F_7 = 2^{2^7}+1$ was factored into primes of 17 & 22 digits. Other ≈ 70-digit numbers.

R used C.frac of $\sqrt{257 \cdot F_7}$.

Quadratic Sieve

– Pomerance (1981) suggested: (1) Sieving idea to reduce the smoothness-test complexity.

(2) Use $Q(x) := x^2 - n$, keeping $x \approx \sqrt{n}$.

↳ Improves to $L_n(\frac{1}{2}, 1)$

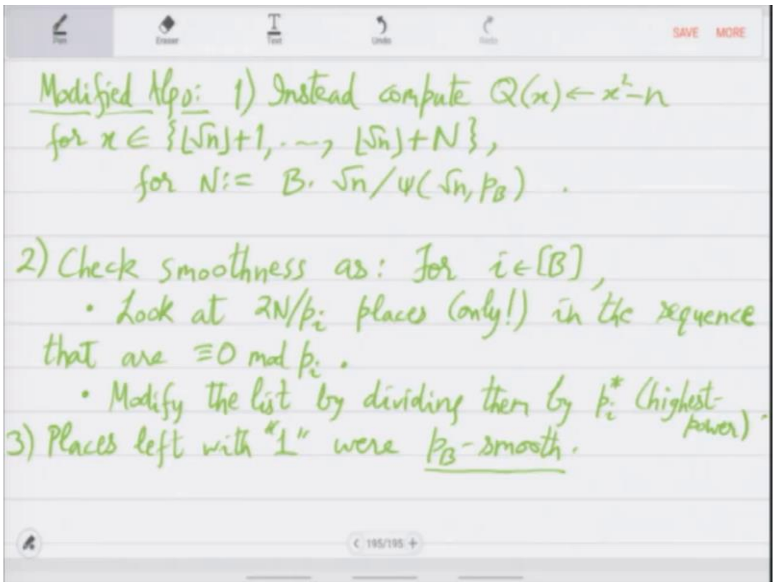
Now what is the success story of this so this factored some large integers for example F 7 this is the first example. So this was factored into primes of 17 and 22 digits and other 70 digit numbers. So this is already quite something this implementation of Morrison and Brillhart could factor numbers up to 70 digits and this F 7 is not that large but the point is that it has prime factor of 17 digits.

So this is slightly more than what you saw before F 8 had a 16 digit prime factor this is slightly larger and it was factored first by this implementation. And this implementation used continued fraction of instead of square root of F 7 it was 257 times F 7 its continued fraction was used. So quadratic sieve is a slight improvement over the previous implementation that you saw suggested by Pomerance.

So Pomerance suggested a sieving idea to reduce the smoothness test complexity. So what he suggested is basically instead of checking each Q i for p B smoothness it will be better if you just divide all the q i's by p B and store the quotient. And also continued fraction method was removed. So there were 2 improvements actually one is sieving idea. Second is use Q x x square - 1 as originally suggested in Krejeriks family keeping x very close to square root n.

So Q x will evaluate to numbers of magnitude around square root n but instead of using continued fraction you just use this sequential growth of x. So this will give an improvement in the constant in the exponent the square root 2 will go away. So we will get L n half 1 instead of square root 2.

(Refer Slide Time: 52:09)



Modified Algo: 1) Instead compute $Q(x) \leftarrow x^2 - n$ for $x \in \{\lfloor\sqrt{n}\rfloor+1, \ldots, \lfloor\sqrt{n}\rfloor+N\}$, for $N := B \cdot \sqrt{n}/\psi(\sqrt{n}, p_B)$.

2) Check smoothness as: For $i \in [B]$,
- Look at $2N/p_i$ places (only!) in the sequence that are $\equiv 0 \bmod p_i$.
- Modify the list by dividing them by p_i^* (highest-power).

3) Places left with "1" were p_B-smooth.

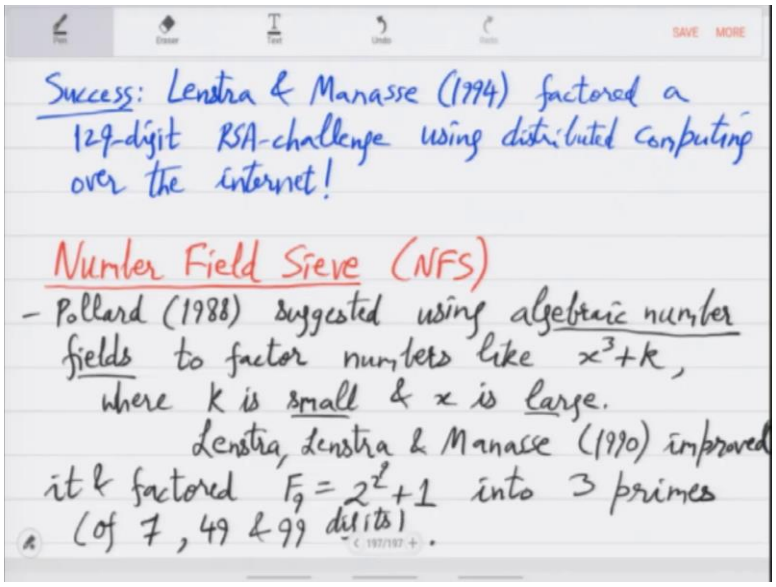
So let us quickly see the modified algorithm this is a famous algorithm. So instead of the continued fraction and using the convergence we have got the Q i right before now you simply compute Q x x square - n for square root n and then square root n + n where this begin is like before its B times square root n divided by psi. So you will get enough p B smooth number as shown in the previous analysis theoretically.

So this is for x in this these are the Q x's you will get at these points and then the smoothness that so that is one modification second was the smoothness test will be slightly improved. So for i 1 to be so the advantage of using this sequential excess is that once you find that p i divides the q x you know the next place that will be divisible by p i. So you will only have to check n over p i begin over p i many places you do not have to check everything that is the advantage we are getting in this sequence.

In the sequence that are divisible by p i. So you can find the first location and then the other locations are just by adding p i and then you divide by p i. So the highest power so in these places you divide by the highest power of p i possible in each place. So now in the sequence nothing is divisible by p i divisibility by p i has been removed. And now you will move to the next this is a for loop.

So, start with p 1 then move to p 2 and so on so in the end the places where you have 1 these are the p B smooth numbers. These are so you have done the smoothness p B smoothness testing this way. So what is the advantage of this? What is the time complexity now?

(Refer Slide Time: 57:44)



Success: Lenstra & Manasse (1994) factored a 129-digit RSA-challenge using distributed computing over the internet!

Number Field Sieve (NFS)

- Pollard (1988) suggested using algebraic number fields to factor numbers like x^3+k, where k is small & x is large.
  Lenstra, Lenstra & Manasse (1990) improved it & factored $F_9 = 2^{2^9}+1$ into 3 primes (of 7, 49 & 99 digits).

So time now is around 2 n by p i in each iteration and you will go 1 to B which is N times 1 sigma 1 over p i. now B h prime is around B size a B in magnitude but since you are only summing over primes this is actually significantly smaller it is log log B which is n is so substitute the value of n by psi which comes out to be l root n alpha c if you substitute B to be that times l root n.

So this is minimized at; so if you again look at the calculation, so what is the difference with the previous calculation? In the previous calculation you had this square right. Now you do not have the square. So alpha you will remain half but c will improve. So you can check that it will become 1 over square root 2. C will be previously it was half now it is different. If you look at 1 - alpha over c this has actually so you get l root n half, 1 over square root 2 square which is around which is L n half 1.

And B is L n half half so we have been able to re remove this square root 2 that you are getting before that is the advantage. So previously you are getting this L n half, square root 2 the square root 2 has been removed that is the improvement by Pomerance. Let us talk about the success of this. So this actually allows for this square root 2 has been removed. So it actually allows u to double n 2 fold increase in the length of n that is significant.

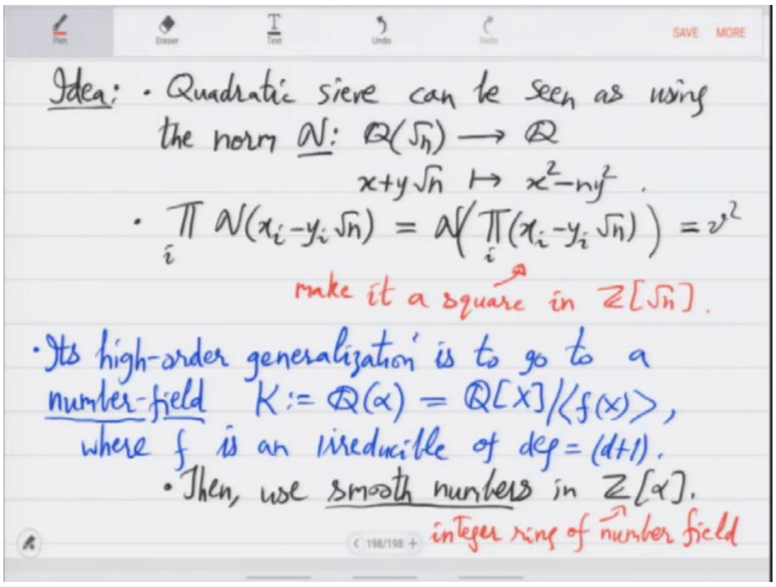
So previously if you were able to factor 70 digit numbers now you can factor 140 digit numbers so that is what happened. So Lenstra and Manus not very long ago actually they factored 129 digit RSA challenge using distributed computing over the internet. What was the success? So you have seen till now factoring algorithm for L n half complexity. In the beginning I had said that the best known complexity is L n one-third.

We do not have time to finish that that is done by the generalized or the general number field sieve. Let me just say few words about the number field sieve and then we will end the course. So this algorithm we saw works with x square - n right. So it is somehow related to square root n. So you can ask why it just restrict to square root n why do not why cannot we go to cube root of n or L n to the 1 fourth or to general number fields right.

So that is what the number field sieve does it goes to number fields. So Pollard suggested using algebraic number fields to factor numbers like x cube + k where k is small and x is large and this suggestion was to use the number field where you are attaching cube root of k. So this idea was picked up and Lenstra, Lenstra and Manus again improved the idea and then factored F 9 2 raise to 2 raise to 9 + 1 like this is this is the biggest Fermat number you have seen till now.

So, F 7 F 8 F 9 this into 3 primes of 7, 49 and 99 digits that is huge, so this is around the 160 digit number and it has 3 prime factors 7 digit which can be found by brute force but then the remaining 49 and 99 digit prime numbers they are really huge there was no way to find this. But this something more complicated than square root n was used to factor it.

(Refer Slide Time: 1:08:18)



Idea: • Quadratic sieve can be seen as using the norm $\mathcal{N}: \mathbb{Q}(\sqrt{n}) \longrightarrow \mathbb{Q}$

$$x+y\sqrt{n} \mapsto x^2-ny^2 .$$

• $\prod_i \mathcal{N}(x_i - y_i\sqrt{n}) = \mathcal{N}\left(\prod_i (x_i - y_i\sqrt{n})\right) = v^2$

make it a square in $\mathbb{Z}[\sqrt{n}]$.

• Its high-order generalization is to go to a number-field $K := \mathbb{Q}(\alpha) = \mathbb{Q}[X]/\langle f(x)\rangle$, where f is an irreducible of $\deg = (d+1)$.

• Then, use smooth numbers in $\mathbb{Z}[\alpha]$.

integer ring of number field

So what is the idea of NFS so quadratic sieve what it is doing it can be seen as using the norm let us call it n which sends Q square root n that s a field right Q rational numbers attached with square root n gives you a field Q root square root n it is a quadratic number field. And the norm will map its elements to Q as follows. So it sends x + y root n to x square - n y square that is the norm.

So this is the norm which was used. Ultimately we actually fix y to be 1. So x square - n was used and note that the norm of a product is product of norm. So if the if the product of x i - if this product of x i - y i root n is a square then the norm is also a square right. So make it a square in z root n or in the; so if this inside thing is a square in the kind of the integer ring of the number field then its norm will be an integral square.

And right so this so there is a chance of getting 2 squares equal mod n in this business. So its higher order generalization is to go to a number field. Let us call it k by attaching an algebraic number alpha which is realized as look at the polynomial ring Q x and mod it out by a polynomial that is irreducible is an irreducible of degree d + 1. And then use smooth numbers in z alpha.

So the smoothness is generalized from integers to extensions of integers called integer rings of the number field. And there again you only look at elements in this integer ring which are smooth you can estimate their number like before. And if you have enough of them then you will get their product to be square and then you take the norm that is the idea. But these details I cannot cover now because of lack of time.