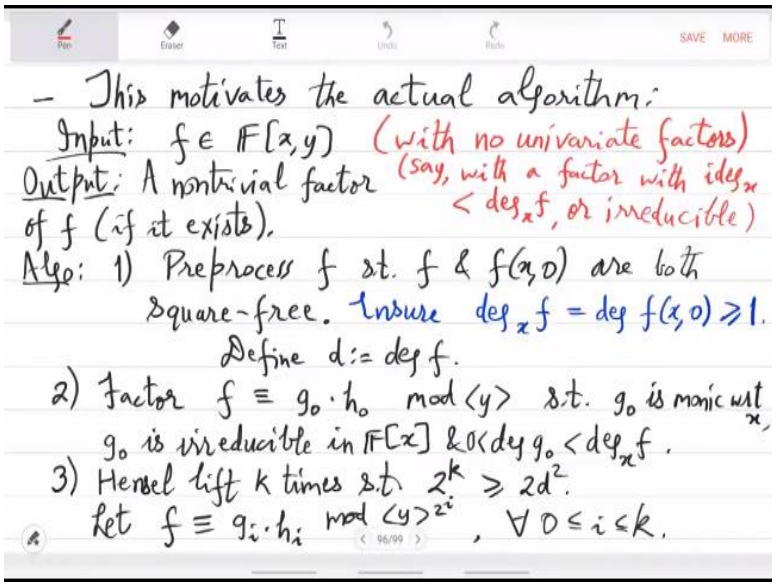
Computational Number Theory and Algebra

Prof. Nitin Saxena
Department of Computer Science and Engineering
Indian Institute of Technology-Kanpur

Lecture – 17
Multivariate Polynomial Factorization

(Refer Slide Time: 00:16)



– This motivates the actual algorithm:

Input: $f \in \mathbb{F}[x,y]$ (with no univariate factors)

Output: A nontrivial factor (say, with a factor with $ideg_x < deg_x f$, or irreducible) of f (if it exists).

Algo: 1) Preprocess f s.t. f & $f(x,0)$ are both square-free. Ensure $deg_x f = deg\, f(x,0) \geq 1$.
Define $d := deg\, f$.

2) Factor $f \equiv g_0 \cdot h_0 \mod \langle y \rangle$ s.t. g_0 is monic wrt x, g_0 is irreducible in $\mathbb{F}[x]$ & $0 < deg\, g_0 < deg_x f$.

3) Hensel lift k times s.t. $2^k \geq 2d^2$.
Let $f \equiv g_i \cdot h_i \mod \langle y \rangle^{2^i}$, $\forall\, 0 \leq i \leq k$.

Okay, so last time we wrote this algorithm, which will reduce bivariate factorization to univariate. So basically in the input you are given a bivariate F over some field where you can factor with univariates. There were some moderate assumptions on the input like there should be no univariate factor and say there is, either F is irreducible or there is a factor with individual degree with respect to x strictly smaller, okay.

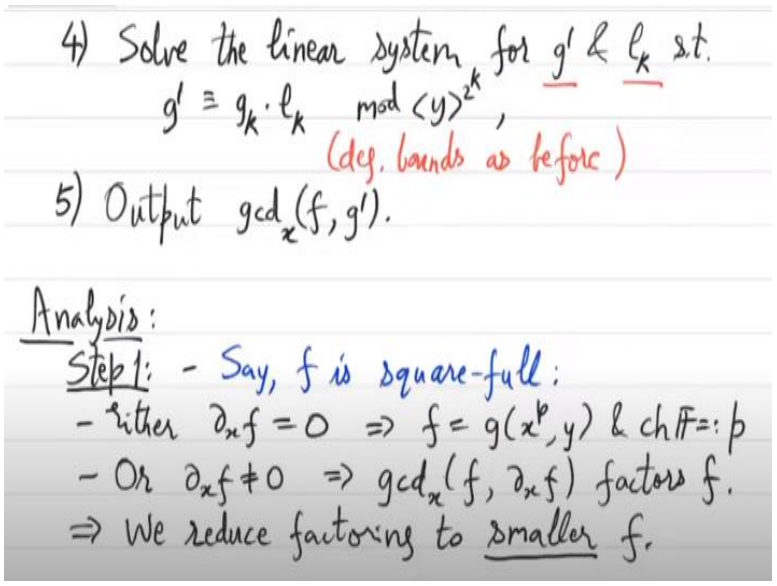
So if there is a non-trivial factor then either its individual degree is smaller in x or in y. So we can assume x. So with these moderate assumptions, this is completely without loss of generality. So what you do is first you preprocess f such that both f and f at y = 0 are square-free, right. If they are not then we had these simple preprocessing steps. If f is square-full then you take GCD with the derivative and factor f.

If f is square-free but f at y = 0 is bad, then you shift, look at some other y equal to alpha. And that will also have the side effect of making the degree of f at y = 0 unchanged. It will be the same as the individual degree of f at x. So let us say degree of f, total degree is d. You factorize first mod y. So mod y there will be a coprime factorization. Why was that? So at y = 0 maybe f maybe irreducible.

If that is the case then well then you can already declare that f is irreducible right. So that is implicit here. So if you cannot find a coprime factorization mod, even mod y then it means that f is irreducible. So then you can just break here. Otherwise mod y there will be factorization, there will be factors and they will be coprime because f is square-free mod y.

So g 0 and h zero are both of them are non-trivial factors mod y, right. So this is a situation where you can Hensel lift. So you will Hensel lift k times to reach y degree 2 raised to k at least 2d square. So you will have g k, h k.

(Refer Slide Time: 03:03)



4) Solve the linear system for g' & ℓ_k s.t.

$g' \equiv g_k \cdot \ell_k \mod \langle y \rangle^{2^k}$,

(deg. bounds as before)

5) Output $\gcd_x(f, g')$.

Analysis:

Step 1: - Say, f is square-full:

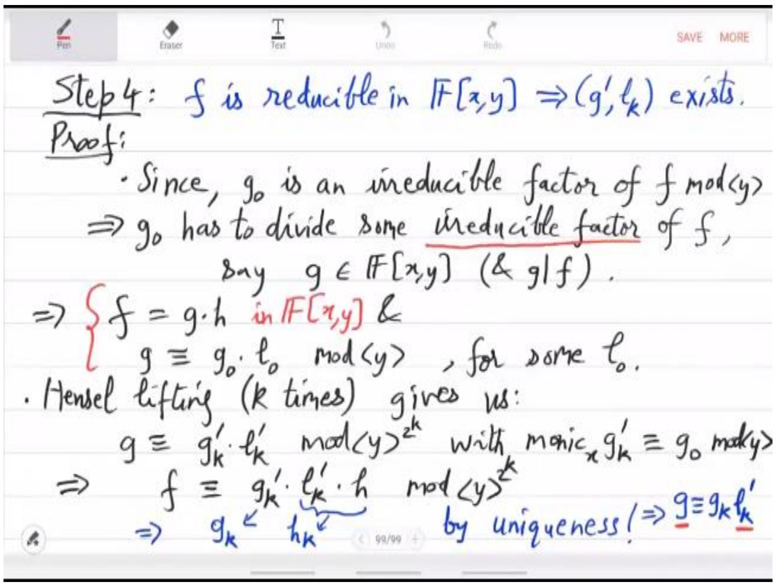
- Either $\partial_x f = 0 \Rightarrow f = g(x^p, y)$ & $\mathrm{ch}\,\mathbb{F} =: p$
- Or $\partial_x f \neq 0 \Rightarrow \gcd_x(f, \partial_x f)$ factors f.

$\Rightarrow$ We reduce factoring to smaller f.

And with g k you try to convert it into a factor g prime with the requisite degree bounds and then take GCD of g prime with f, right. So the claim is that this algorithm works. If f was reducible then step 5 will factor, will give you a factor of f. All the steps we have seen and we have also given the time estimate is clearly polynomial time, deterministic polynomial time. There is no randomness involved. Any questions?

So last time we saw that if f has a factor, let us say g, then this g prime equal to g works in step 4 okay, which means that the linear system is solvable. It is a feasible system. So which means that when you are at step 4, you will find some g prime but that may not be a actual factor of f, right. So that so the last step now we have to analyze. So in Step 5, why are we claiming that this will factorize.

(Refer Slide Time: 04:21)


Step 4: f is reducible in $\mathbb{F}[x,y] \Rightarrow (g', \ell_k)$ exists.

Proof:

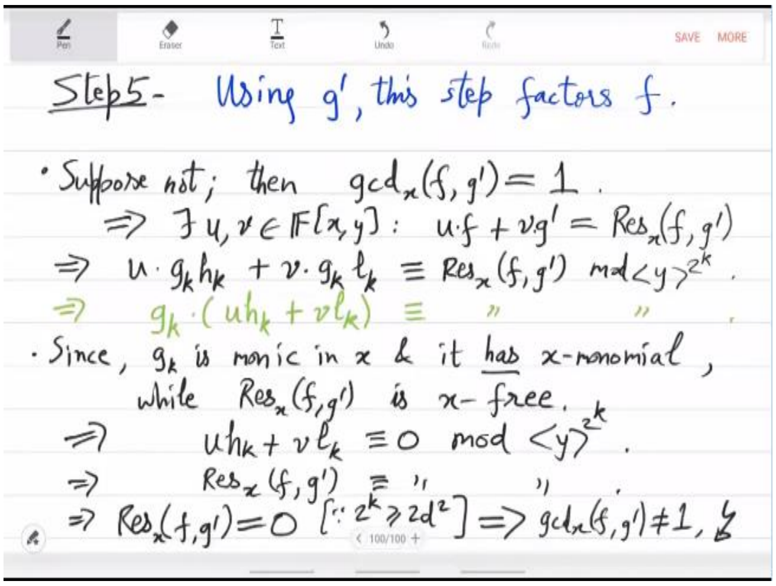
- Since, g_0 is an irreducible factor of $f \bmod \langle y \rangle$
  $\Rightarrow g_0$ has to divide some irreducible factor of f, say $g \in \mathbb{F}[x,y]$ ($\&$ $g | f$).

$\Rightarrow \begin{cases} f = g \cdot h \text{ in } \mathbb{F}[x,y] \ \& \\ g \equiv g_0 \cdot \ell_0 \bmod \langle y \rangle \end{cases}$, for some ℓ_0.

- Hensel lifting (k times) gives us:
  $g \equiv g_k' \cdot \ell_k' \bmod \langle y \rangle^{2^k}$ with $\text{monic}_x\, g_k' \equiv g_0 \bmod \langle y \rangle$
  $\Rightarrow f \equiv g_k' \cdot \underbrace{\ell_k' \cdot h} \bmod \langle y \rangle^{2^k}$
  $\Rightarrow \; g_k \swarrow \quad h_k \swarrow$ by uniqueness! $\Rightarrow g \equiv g_k \ell_k'$

So let us move to that okay. So yeah this proof was done, in particular g works in for the linear system.

(Refer Slide Time: 04:33)


Step 5 - Using g', this step factors f.

- Suppose not; then $\gcd_x(f, g') = 1$.
  $\Rightarrow \exists u, v \in \mathbb{F}[x,y] : u \cdot f + v g' = \text{Res}_x(f, g')$
  $\Rightarrow u \cdot g_k h_k + v \cdot g_k \ell_k \equiv \text{Res}_x(f, g') \bmod \langle y \rangle^{2^k}$.
  $\Rightarrow g_k \cdot (u h_k + v \ell_k) \equiv \quad " \quad "$.
- Since, g_k is monic in x & it has x-monomial, while $\text{Res}_x(f, g')$ is x-free.
  $\Rightarrow u h_k + v \ell_k \equiv 0 \bmod \langle y \rangle^{2^k}$.
  $\Rightarrow \text{Res}_x(f, g') \equiv \quad " \quad "$.
  $\Rightarrow \text{Res}_x(f, g') = 0 \; [\because 2^k \geq 2d^2] \Rightarrow \gcd_x(f, g') \neq 1$, ↯

So let us analyze step 5 now. So here the claim is that, if you have found a g prime you factor f. And if you did not find a g prime then there is, it is a proof that f was irreducible, okay. So suppose not right, proof by contradiction. Then the GCD came out to be in step 5, it came out to be 1, right. There is no other option because g prime has degree of x less than f. So if it does not factor then it means you get 1 as GCD.

So which means that there exist, so now use kind of this generalized Bezout identity. So there will be polynomials u, v such that uf plus vg prime is equal to is equal to what? Yeah, in univariate it would have been 1 but in bivariate it is the resultant. So you are giving importance to the x variable. So in in terms of x variable this uf plus vg prime is 1. But then if you remember how we got resultant there is this denominator which is free of x.

So it has y and that is exactly resultant. Or we can pick it to be resultant. So resultant is free of x, it eliminates x, right. So there are these polynomials u and v that eliminate x. Okay, next what? So just this equality will now give you contradiction. I mean using all your algorithmic steps and this equality you will get a contradiction. So what do you know about f from the algorithm?

f is g k h k, right mod y to the 2 to the k. What do you know about g prime? g prime was g k l k congruent to the resultant. So y to the 2 to the k, this will be the reason for contradiction. So you are getting g k times uh k plus vl k is congruent to the resultant mod this. Why is there a contradiction? So notice that RHS is x free and LHS g k definitely have has x, right.

Because if you look at the starting point g 0 had x and the degree of x does not change when you Hensel lift. So g k has x. But when you multiply with this bracket, you should get x free, which means? Actually not only that g k was monic, g 0 was monic and this was modified Hensel lifting. So g k is actually monic in x. So you can think of coefficient as 1, it is 1.

So which means that uh k plus vl k has to be, it has to be 0. In this modulus it has to be 0. If it is not 0 in this modulus then there is no way to cancel this leading monomial in x. So that is the contradiction. So since g k is monic in x and it has x while the RHS which is resultant is x free. What you deduce is that uh k + vl k is 0 modulo y to the 2 to the k, okay? Which means the resultant is 0.

Why is that a problem? Sorry. No. GCD one was absolutely. No it is zero only mod y to the 2 to the k. No the reason is you have to think about how big is 2 raised to k. 2 raised to k is far bigger than the degree of the resultant, okay. So since 2 raised to k is, so resultant has degree less than 2d square. While 2 raised to k is more than 2d square or at least 2d square.

So this means that resultant is absolutely 0, no modulus. Since 2 raised to k is greater than equal to 2d square. Now this resultant absolutely zero would mean that there is a GCD. It is not 1, which is the contradiction right. So this is a very clever proof, completely invisible in the previous slide. But once you see this you know that the parameters the way we have fixed the parameters in the algorithm g prime is very special.

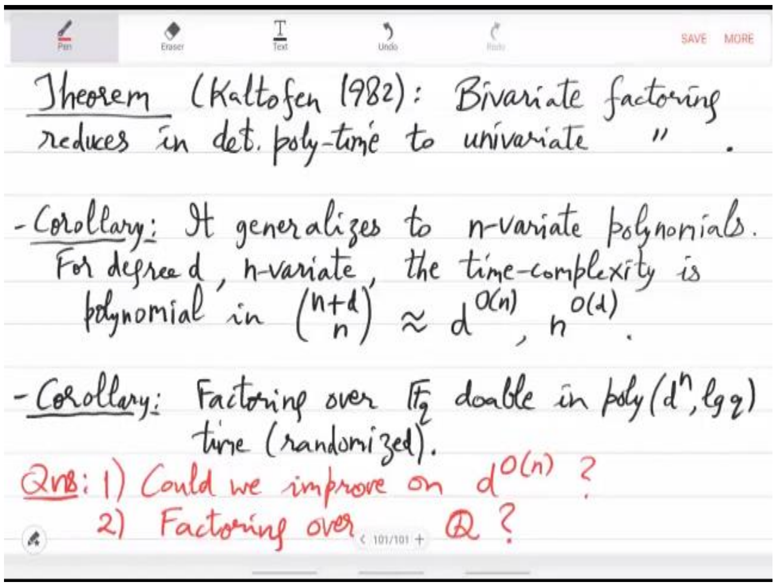
g prime actually contains a factor of f though you pick g prime as some arbitrary solution of a linear system. But whatever you pick it will always contain a factor, okay. Any questions? So that finishes this step 5 analysis and hence the algorithm.

"Professor - student conversation starts" If we, g k is kind of the common factor which has x, f in g prime modulo y. So if I just took Hensel lift to degree greater than max of f or g prime and then say that if f is g k h k and g prime and g k l k. Then directly do I not get a contradiction because g k is a common factor while GCD in x is problem. **"Professor - student conversation ends".**

No GCD, no GCD is 1 absolutely but when you go mod will it remain 1? Mod is a different word. **"Professor - student conversation starts"** If I take greater than 2 to the k greater than degree of both f and g. **"Professor - student conversation ends".** Okay. Okay then. So what you want is what this proof is giving you. This is a highly non-trivial proof.

This is probably the most major application of resultant you will see and very explicit application.

(Refer Slide Time: 13:50)



Theorem (Kaltofen 1982): Bivariate factoring reduces in det. poly-time to univariate ".

- Corollary: It generalizes to n-variate polynomials. For degree d, n-variate, the time-complexity is polynomial in $\binom{n+d}{n} \approx d^{O(n)}, n^{O(d)}$.

- Corollary: Factoring over $\mathbb{F}_q$ doable in $poly(d^n, \lg q)$ time (randomized).

Qns: 1) Could we improve on $d^{O(n)}$?
2) Factoring over $\mathbb{Q}$?

So this was achieved by Kaltofen. So bivariate factoring irrespective of field reduces in deterministic poly-time to univariate factoring, okay. Now what do you do with trivariates? So bivariate is done, what happens to 3-variate? **"Professor - student conversation starts"** It will reduce it to bivariate. Why not reduce it to univariate? So in this in. Consider modulus of two things. Yeah do that. **"Professor - student conversation ends".**

So if you have three variables x 1, x 2, x 3 pick the ideal x 2, x 3 and then go modulo powers of that. So this algorithm is completely general. You can do it for any number of variables up to those moderate assumptions. So you do some preprocessing and after the preprocessing when you go mod ideal there is only one variable that remains, x 1. And x 1 you first factor in that variable and then Hensel lift.

Hensel lift enough number of times then take the GCD. So I will not go into that because, I will actually go into a much bigger thing which will subsume all this discussion. But I will just remark that just the same proof gives you or similar algorithm gives you to n-variate polynomials. So for degree d, n-variate the time complexity is polynomial in, it will be polynomial in what?

In the number of monomials that play a role. So how many monomials are there of degree at most d and n variables, right? That is a combinatorics question. So that number is n plus d choose d. So n plus d choose n is if you think of n as small then it is d to the n. And if d is very small then it is n to the d. So whatever you prefer. So for example, when n is constant then this is fine.

Or when d is constant then also this is fine, this is all polynomial time. But when both of them are changing or growing let us say n equal to d, right. In that case it is very close to 2 raised to m. So that will be exponential. So if you are looking at hundred variate hundred degree polynomials then this algorithm is infeasible, okay. High degree, high variate this will not stop.

This is very expensive. Just because the presentation is actually expensive. Just the number of monomials is so much that you need a huge amount of space and then proportionate amount of time. So that you do not expect to actually improve by this method. You will need a very different representation. So the questions now are okay before that let me finalize this.

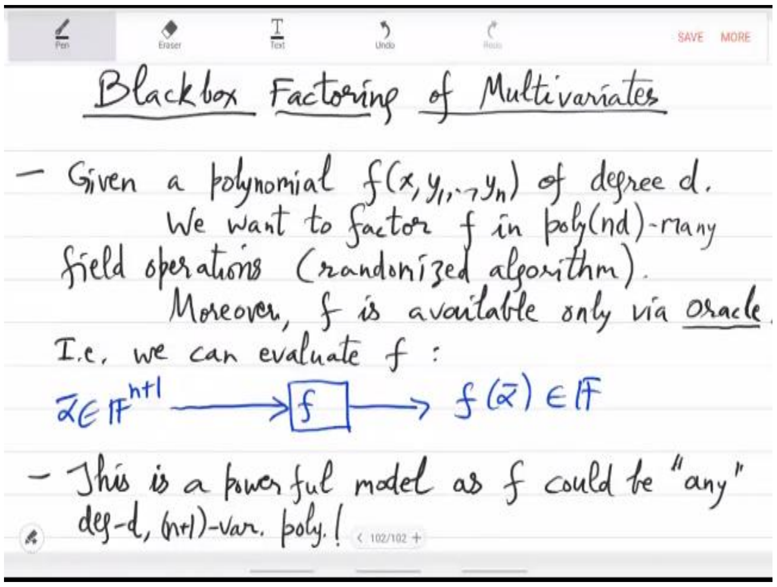
So finite field case is then completely done for small variables. So factoring over q over F q is in polynomial log q and d raised to n. Is in this much time and that is a randomized algorithm, right. So factoring over F q is practical by this algorithm when the number of variables is small. So trivariate four variate and so on. That this algorithm will reduce it to univariate and then at univariate it will use Cantor–Zassenhaus or Berlekamp or whatever.

So in particular Cantor–Zassenhaus will give you log q, right. So d raised to n log q. So this low variate over finite field factorization is done by this. But same cannot be said for other fields, because you do not know even univariate factorization for other fields. So those things we will now focus on. So one is could we improve? And second would be factoring over rationals.

Okay, so can you improve on d raised to n or is this necessary if this is a lower bound? And second question is about other fields. So in computer science, there are only two kinds of fields which are of interest, either finite fields or the field of rationals. We do not worry about, we do not use other fields actually except these two. So we will, next lectures will be devoted to question one.

And subsequent to that, we will move to question two. Okay, and this will cover a lot of topics and techniques. It will gradually get technical, more technical than what you have seen till now. Any questions? Okay, so let us now start question one. So we do not want to get into this number of monomial business. So we will use a different representation, which is called black box representation.

(Refer Slide Time: 22:08)



Blackbox Factoring of Multivariates

- Given a polynomial $f(x, y_1, ..., y_n)$ of degree d. We want to factor f in poly(nd)-many field operations (randomized algorithm). Moreover, f is available only via Oracle. I.e, we can evaluate f:

$$\bar{\alpha} \in \mathbb{F}^{n+1} \longrightarrow \boxed{f} \longrightarrow f(\bar{\alpha}) \in \mathbb{F}$$

- This is a powerful model as f could be "any" deg-d, $(n+1)$-var. poly.!

And then we will factor black boxes. No, just to store a polynomial. How do you store a n-variate degree d polynomial? It is a storage problem first. **"Professor - student conversation starts"** Sir, degree d in terms of a particular variable or **"Professor - student conversation ends".** No total degree. It will not matter. Particular variable average will be d by n, total is d. It is the same thing.

If both are growing then it is, storage is a problem. So black box factoring of multivariates, multivariate polynomials. Yeah, so this is a representation where well everybody here would have heard about the term Oracle. So this is inspired by that. So think of a n-variate polynomial as given by an Oracle, which means that you will give Oracle a point and the Oracle will give you a value of that polynomial at that point.

Okay, without you being able to see anything inside the polynomial. So this is why it is black box. Whatever we were doing till now was white box, because you were able to see the monomials. But here you cannot see the monomials. You just say that a black box has been provided in the input for f, okay. So black box takes in an input and gives an output. That is the kind of representation.

And that is a very mysterious representation because you want to output factors as black boxes, right. So which is very tricky, because you do not even know the number of factors in this case. So suppose the factors are few how do you output a black box for a factor? And it seems to be a very difficult problem in this model. We can just do evaluations. Yeah, black box you can just evaluate.

So essentially, given an Oracle in the input, you want to output a list of Oracle's each corresponding to a factor, right. So it is a much higher level kind of factoring than mere polynomials. **"Professor - student conversation starts"** You want to give all the factors? **"Professor - student conversation ends".** Obviously, yes.

Yeah not just one non-trivial, but this Kaltofen algorithm in the end will give all the black boxes at once. So that part I think is not very hard because the degree is limited. So you cannot have more than d black boxes. There cannot be too many factors. I think number of factors is clearly small. So given a polynomial f, yeah so for the reasons of the presentation of the algorithm we will distinguish the first variable as x and the others are y bar, y 1 to y n okay?

This is because we want to go modulo the ideal generated by y 1 to y n. So that is how we will present the algorithm. So we will keep the notation on the variables like this. So you are given a polynomial n + 1 variate degree d and what you want is a factor, factor f in polynomial in nd field operations and randomization allowed. That we have to allow otherwise even univariates you cannot factor over finite field.

So yeah so you are given an given a black box for f. The black box will tell you will come with the guarantee that there are n + 1 kind of input variables and there is one output of the black box. And it will also promise that the hiding polynomial has degree at most d, okay. So in these parameters you want polynomial. That is the demand. So f is available only via an Oracle.

That is we can evaluate f. So there is a box inside which you cannot see but you can feed a point. Yeah I say f field f but it may also be a small extension of the field f. So Oracle will allow you inputs which are in small extensions of the field f. So from that space you pick any point alpha bar, you feed into the black box and it will output the value. It will output a single field element which is f alpha bar.

So this box is given to you, okay. You can also think of it as a computer program or a subroutine but it is not accessible, inside details are not accessible. Just the, it is a function evaluator. Clearly this representation is complete, in fact much more. It can hide anything in zero space, right? We are not looking inside the box, so in kind of constant space you are able to represent any function.

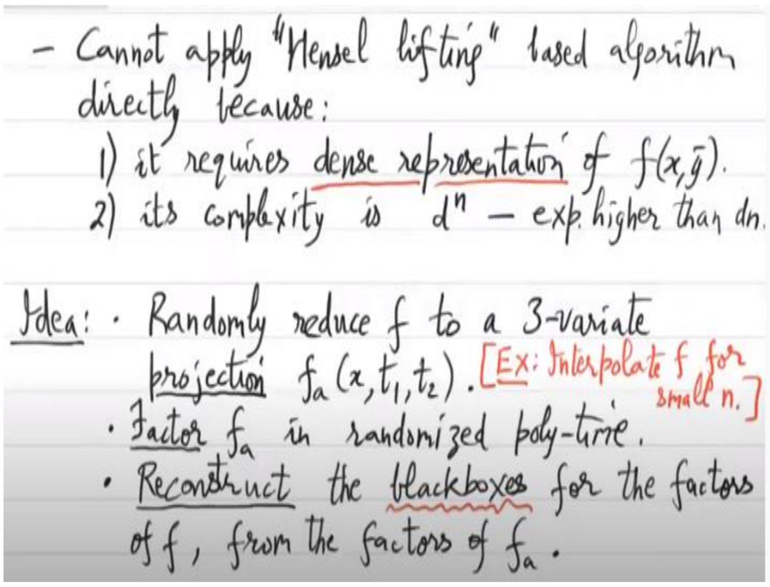
So this is a very powerful model or representation as f could be any degree d, n + 1variate polynomial. **“Professor - student conversation starts”** We are storing this polynomial by means of these evaluations. **“Professor - student conversation ends”.** There is no storage. There is only this box given to you. And the box take space, unit space not dependent on properties of f.

But there is also this promise that it computes degree d polynomial n + 1 variate. And you only want to be polynomial in that nd. So we are not saying that the degree can be 2 raised to m and you have to do polynomial in m. So we are not saying do it in polynomial in log d. It is polynomial in d. So yeah the only factorization algorithm you know at this point is the Hensel lifting based algorithm.

But the problem with that is in every step of the algorithm, you need the full polynomial, monomial by monomial right, which is not possible here. Number of monomials is too much. So if you try to learn, if you have come up with a learning algorithm for the Oracle then it will be a bad algorithm. Necessarily because you have to learn the coefficients of exponentially many monomials, right.

So learning algorithms here will be useless. So you have to actually work directly with these boxes, black boxes.

(Refer Slide Time: 32:27)



- Cannot apply "Hensel lifting" based algorithm directly because:
  1) it requires dense representation of $f(x,\bar{y})$.
  2) its complexity is d^n — exp. higher than dn.

Idea: • Randomly reduce f to a 3-variate projection $f_a(x,t_1,t_2)$. [Ex: Interpolate f for small n.]
• Factor f_a in randomized poly-time.
• Reconstruct the blackboxes for the factors of f, from the factors of f_a.

So we cannot apply Hensel lifting as seen before directly, because it requires dense representation. So dense representation is this monomial by monomial representation. And even if you learn this dense representation by some learning algorithm, it will be too big. So its complexity is d raised to n. So that is exponentially high than dn, right. So you want polynomial in dn.

But this is d raised to n which is exponential as n grows arbitrarily. So we have to do something else. So any ideas what to do? How do you reduce this d raised to n obstruction by reducing n. How do you reduce n? So you project the black box down to few variables, say three variables right. And that trivariate projected polynomial projection you factor in the usual way.

And from that factorization, you try to learn the original black boxes, okay. So it is a projection, drastic projection to n equal to three, and then going back from there to black boxes of the factors. Yeah, but it should not be clear at this point why these things will work, right? Otherwise the whole time that I am going to spend is a waste. So let us do this slowly. There will be a lot of implementation of these ideas.

So randomly reduce f to a 3-variate projection f sub a (x, t 1, t 2). So the x we are saying we are not changing but y 1 to y n we are changing to bivariate. The projection will be quite simple. We are replacing each y i by just a linear function in t 1, t 2, okay. So geometrically you are projecting this higher dimensional thing into let us say a plane, okay. Each of the coordinates you are projecting on a plane or a slightly bigger dimensional flat.

So you project the points there, okay. So let me postpone the details of this, how the prediction is done. Then factor f a in randomized polynomial time. And from these factors, trivariate factors, try to learn the black boxes for the actual factors. **“Professor - student conversation starts”** The projections are also black boxes, right? **“Professor - student conversation ends”.**

No, this is 3-variables. 3-variable black box, you could learn. Because the number of monomials is just d cube. So that the monomials are very few. You can just basically, so how do you do that? I think that we should leave as a small exercise. So in case when, let us say n was just 2, right. So the black box was in variables x, y 1, y 2. From that black box, can you get the polynomial f in dense representation?

Yes, so it is basically a generalization of interpolation. If n was 0, then it is a univariate black box. So univariate black box you will interpolate, that standard interpolation. But if n is 1, 2 and more, then it is a higher dimensional interpolation. But that also you can do. It is just a simple generalization of interpolation. So let me say that here. Interpolate f for small n. It is just interpolation.

So black box and dense representation is the same when n is small. But as n grows, then it exponentially blows. So this f a you can just factorize by Hensel lifting algorithm. And from the factors you try to reconstruct. So reconstruct the black boxes for the factors of f from the factors of f a. Yeah, that seems at this point farfetched, but we will actually implement each of these steps.

“Professor - student conversation starts” Sir, when we say black boxes what is the **“Professor - student conversation ends”.** Oh yes, that is a good question. So basically the white box the box which is given to you this black box for f that black box will be queried let us say many times, values will be obtained. Then those values you will add or multiply and possibly again query the black boxes and so on.

So this you will be outputting a sequence of operations, where you can do addition, multiplication and black box query. So it is kind of a it is you can think of it as a C program which is making free calls to the black box, okay? It does not know the implementation of the black box subroutine, but modulo that it is a valid C program. So you have to output that program, okay. So that program is mostly white box.

You can read the steps except this black box subroutine which was never given to you. I mean with the details were not given to you. So this is a algorithm that will output an algorithm, okay which is why these things are hard. So in the input it was given an algorithm. It itself is an algorithm and the output will be an algorithm. Yeah, so those things will be clear when you see the implementation of this.

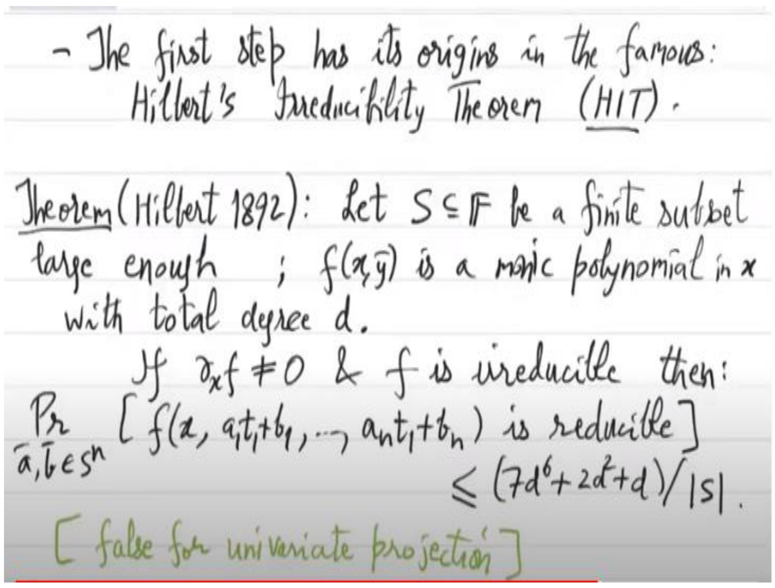
“Professor - student conversation starts” C construction will be dependent on how the other variables are fixed to t 1 and t 2. **“Professor - student conversation ends”.** Yeah. So to give you more, few more details, the this reconstruction is conceivable because so you are given f and suppose you are given a I think that is what I meant by this a probably.

So suppose you are given a point a at which you want to evaluate all the factor black boxes, right. So black boxes, what good are black boxes for to evaluate at a point. So let us say the user has given you this f black box and a point a and the user wants to evaluate factor f 1 at a. So what we will do is we will actually maybe draw a random plane containing a in the space. That is what t 1, t 2 variables are for.

Or maybe it is just a random line probably. So whatever. Some random space will be constructed using two variables containing a. f will be projected down to that, factored and from the factors so it is kind of looking in the vicinity of a. So these factors know everything about vicinity of a. And hence, you can evaluate those factors at a, okay. So in a low dimensional neighborhood of a you are factoring f.

And hence, from the factors you can get the value of the factors of f at a. Is that clear? Does that help your imagination? If not, then be ready for the details. Okay.

(Refer Slide Time: 44:13)


- The first step has its origins in the famous:
Hilbert's Irreducibility Theorem (<u>HIT</u>).

<u>Theorem</u> (Hilbert 1892): Let $S \subseteq \mathbb{F}$ be a finite subset large enough ; $f(x, \bar{y})$ is a monic polynomial in x with total degree d.

If $\partial_x f \neq 0$ & f is irreducible then:

$$\Pr_{\bar{a},\bar{b} \in S^n} \left[f(x, a_1t_1+b_1, \ldots, a_nt_1+b_n) \text{ is reducible} \right] \leq (7d^6+2d^2+d)/|S| .$$

[false for univariate projection]

So the first step is an independently interesting theorem in algebra with a big name. So the first step has its origins in what is called Hilbert's Irreducibility Theorem. So we can call it HIT. So Hilbert proved this theorem which we will probably modify. The modified version is this. The spirit of this is very simple. You are given a univariate n-variate or n + 1 variate polynomial f that is irreducible over a field, okay.

Now what happens if you randomly project that polynomial down to two variables or three variables or so on? Will it remain irreducible, right that is the question? So Hilbert Irreducibility Theorem says that for random projections this is true, okay. So if you for a multivariate polynomial when you randomly project down to fewer variables then this projection will also be irreducible with high probability.

In particular, it tells you that for a general multivariate polynomial f if there were are r number of factors, irreducible factors, when you project f down to fewer variables the number of irreducible factors will remain the same. They will be a bijection, okay. So by projected by projections of polynomials, you can learn the factorization pattern of original f, okay. So this is a major property of polynomials.

This is kind of the starting point of our black box factoring. If this basic thing was not true, then there was little hope, because then it would seem that there is no connection between high variate and low variate predictions, right. But this theorem will tell you that there is actually a strong connection. There is nearly a full connection between n-variate polynomials and their trivariate, bivariate projections.

Yeah, you may think about the case of complex. So suppose you have a high variate polynomial with complex coefficients. So if you project down randomly to univariate what happens? Then it completely splits, right. No matter what was happening in the original polynomial, if you project down all the way to one variable, then you lose the information. So this is why you should not try to come down to univariate.

You should stop at bivariate, okay. Because we want this result for all fields. So for if you go from n-variate to 2-variate then you may hope to bypass this counter example. **"Professor - student conversation starts"** By projection what do you mean? Fixing of variables? **"Professor - student conversation ends".** That also, yeah. But this is slightly more general.

Yes, but this for example, will not work. This is example where it fails because x, y, z. **"Professor - student conversation starts"** Because it is special case right? The z completely vanishes. **"Professor - student conversation ends".** No, but the number of irreducible factors have changed from three to two.

So you do not want that. If you want to preserve the number of irreducible factors of x, y, z, then what you should do is you send x, y, z to random linear functions of x and y, right? If you do that, then three factors will actually just project down to three factors. And there will be a bijection with high probability. There are some bad fixings, but they are very few in density.

Yeah, so that those kind of projections will work. This is what Hilbert says. So let us give the statement with all the gory details. So pick a subset S of size big enough, happens to be 7d 6. f (x, y bar) is a monic polynomial in x. This is again a very moderate assumption. From general polynomial n + 1 variate, you can come to this situation where it is monic in x, right? How do you do that?

Is the same technique that we used before. If the leading monomial of x has some function in y bar, you just randomly shift y bar, y 1 plus alpha 1, y 2 plus alpha 2. And then well even then the coefficient will not become 1, but at least it will become 1 mod y bar, okay. So if you set y 1 to y n to 0, then the leading coefficient of f with respect to x is 1. So we will call it almost monic.

But anyways, let me continue with this because this is really about Hilbert's theorem and not the full algorithm. So f is a monic polynomial in x with total degree d. So if a derivative of f is non-vanishing and f is irreducible then so probability of projections, so probability over a bar, b bar, S to the n, f x comma is irreducible. This probability is less than equal to 7d 6 + 2d square.

So this exact expressions you can forget, it is not very important. But what it is saying is that when you project in this specific way, linear functions in t 1, okay you pick n random linear functions in t 1 and plug that in y 1 to y n. Then the chance that this becomes reducible from the originally irreducible polynomial is very low. I should remove this size condition here, this does not make sense now.

Let S be a finite subset large enough. So if you take S to be larger than let us say d to the 7 or 7 d to the 7 or something right, then this probability, this error probability is sufficiently small. You can make it as small as you want. And the cost of this is that you have to sample the linear functions from a bigger space. a 1, b 1 should be from a bigger subset.

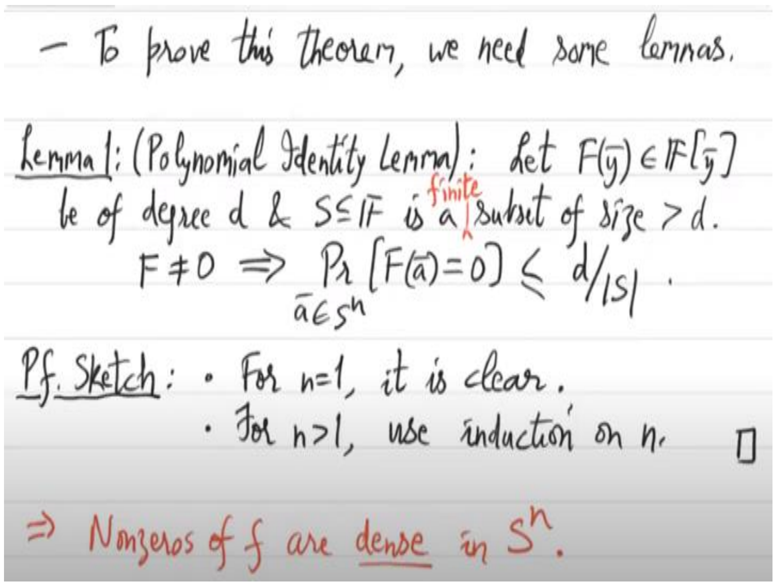
So if you sample from a bigger, big enough large enough subset a 1 b 1 to a n b n then with a very high probability irreducible f goes to irreducible projection and bivariate. As I have as we have discussed, this cannot be improved to univariate. Okay, univariate it is false, but certainly for bivariate it is true for any field. Any questions? Okay. So this is the, this is Hilbert's Irreducibility Theorem.

This is the cornerstone of our future algorithm. **"Professor - student conversation starts"** The univarite case is false for every field? **"Professor - student conversation ends".** Yeah, that I leave for you to answer. Usually it works. Yes. So from this theorem statement, you can now kind of guess vaguely what the algorithm would be. So you anyways need two variable projection minimum two variable.

But since you are given also a point at which you wanted the factors to be evaluated, you need a bit more flexibility to look around in a neighborhood of a. So one more variable will be needed for that. So there will be three variable projection, okay. So from f you will project two, three variate in this specific way. There will be a one to one bijection between factors of f and factors of the projection of f.

And since all the factors are correct up to a neighborhood of f, neighborhood of a you will just evaluate those factors now at a and that will be the output, okay. So this is what we will implement. Yeah, so first we want to prove this theorem which will be some hard work. So let us first prove some baby lemmas for this.

(Refer Slide Time: 57:05)



– To prove this theorem, we need some lemmas.

Lemma 1: (Polynomial Identity Lemma): Let $F(\bar{y}) \in \mathbb{F}[\bar{y}]$ be of degree d & $S \subseteq \mathbb{F}$ is a finite subset of size $> d$.

$$F \neq 0 \Rightarrow \Pr_{\bar{a} \in S^n}[F(\bar{a})=0] \leq d/|S| .$$

Pf. Sketch: • For $n=1$, it is clear.
• For $n>1$, use induction on n. □

$\Rightarrow$ Nonzeros of f are dense in S^n.

So let me first handle this monic business. This condition that we need let me talk a bit more about this. So for that I will need the following lemma and directly state the lemma. So this is called the polynomial identity lemma. So it says that if you have a multivariate polynomial big F of degree d and S is a subset greater than d. So then if f is nonzero, it is nonidentity then what is the probability of it vanishing at a random point.

So what is this bound? If you have a nonzero function and you evaluate it at a random point what is the chance that it vanishes? So what is the chance that you have hit a root? In this space S to the n how many roots are there? If n was 1 what is the answer? Yeah, so degree of a univariate upper bounds the number of roots, right. So roots can be at most d out of S. And this upper bound is true for any n, okay.

This is what the lemma says that no matter what the number of variables is, in a space of S to the n, the number of roots is at most this fraction, the same fraction as n equal to one. So if you take S to be let us say 10d, 10 times d, then the chance of hitting a zero or a root is at most 10%. Okay, if you take 100d, then it is 1% and so on. You can arbitrarily make it small. Again, by sampling in a big enough space.

“Professor - student conversation starts” While the numerator is f g why it is d by S? **“Professor - student conversation ends”.** Why? For n = 1 you mean? This is clear right? Yeah, so I am saying that it is true in general. If you do not believe me, then you have to prove it. So I think that poof is that it is a question in the assignment proving this, in this assignment three.

So when f is a univariate I mean clear as in you still have to show that a degree d polynomial has at most d roots. Do you know how to show that? **“Professor - student conversation starts”** We will use the induction on the degree. Firstly two degree we will assume that one. If it is irreducible then it will have no zeros and if it is reducible it has two, so at most two is satisfied. For higher degree we can factor out. **“Professor - student conversation ends”.**

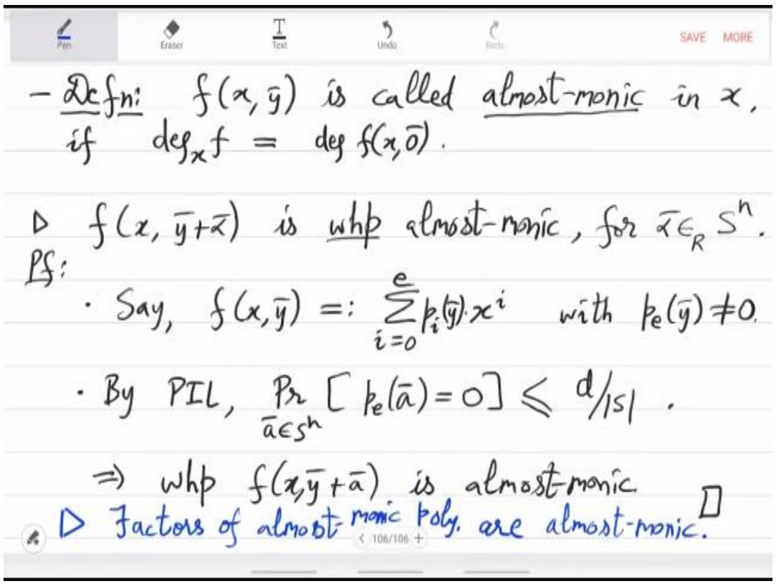
Okay. Yeah. So there are multiple ways you can show that a degree d univariate has at most d roots and for complex or for algebraically close it is exactly d roots. But when you go to bivariates there is a slightly different induction process by which you can prove this. So for n greater than 1, use induction. Use induction on n. So I would discuss. No, that do not have.

Finiteness is gone, which is why we have taken S to be a finite subset. Okay, this I should write. So when f is a infinite field then biariates may have infinite roots. But despite that, when you are looking in a inside a finite domain, sampling from a finite domain, the density remains this. It is not that the density is very high, even though there are infinitely many roots, okay.

So yes, so different things are happening here. So those questions are there in the assignment. I think two questions are dedicated to this. So what you have to remember is that non-zeros are dense in S to the n. That is what this polynomial identity lemma is saying. Okay, so zeros are not dense. But the non-zeros are dense in the space S to the n. This is the, geometrically this is what the lemma says.

But we will actually need these error probabilities. So you actually have to prove this for the analysis of the algorithm. Okay. So once you have this, let us discuss this issue of monic f. That need some preprocessing. So okay let me define that term.

(Refer Slide Time: 1:04:56)



- Defn: $f(x,\bar{y})$ is called almost-monic in x, if $\deg_x f = \deg f(x,\bar{0})$.

▷ $f(x,\bar{y}+\bar{\alpha})$ is whp almost-monic, for $\bar{\alpha} \in_R S^n$.

Pf:

· Say, $f(x,\bar{y}) =: \sum_{i=0}^{e} p_i(\bar{y})\cdot x^i$ with $p_e(\bar{y}) \neq 0$.

· By PIL, $\Pr_{\bar{a}\in S^n}[p_e(\bar{a})=0] \leqslant d/|S|$.

$\Rightarrow$ whp $f(x,\bar{y}+\bar{a})$ is almost-monic. $\square$

▷ Factors of almost-monic poly. are almost-monic.

So we call it almost monic if the individual degree of f at x is remains unchanged around the point 0, okay. So f has let us say x to the i is the leading monomial. It has some coefficient, function of y bar. But that function in y bar has a constant term. This is what it is saying, okay. So these polynomials we call almost monic. Now your input polynomial may not be of this type.

So what you will do then is randomly shift, okay and invoke the previous lemma. So f (x, y bar + alpha bar) is with high probability, almost monic, for alpha bar randomly picked from S to the n. Okay, is this clear? So yeah the proof is just the previous lemma application. So you write f as a polynomial in x, okay. So the coefficients are functions in y, y bar. x to the e is the leading. So x to the e is the leading monomial.

So this coefficient p e is nonzero. So when you will randomly shift this and invoke the previous lemma you will get what you wanted. So by this polynomial identity lemma, probability over choices in S to the n of p e a bar zero is less than equal to d over S. Okay, so we take S to be big enough and then sample from there p y bar shifted by a bar has a constant term and so f is almost monic, okay.

You can also observe one simple thing that factors of almost monic polynomial are, they are also almost monic, right? Factors of monic is monic. So this is a useful property that we will be keeping at all the steps of the algorithm. That we are only working with almost monic polynomials and so whatever factors we are getting they are also almost monic, okay. And this is for, this property is for free.

Once you have PIL it is for free. And this shift is an invertible transformation, right. So whatever you will learn for the shifted f you will also learn for f. So if you can factorize shifted f you can also factorize f, right. So these are, it is if and only if. So we will as well look at such polynomials.