# Lecture 3: Finding integer solutions to systems of linear equations

Algorithmic Number Theory (Fall 2014)
Rutgers University
Swastik Kopparty
Scribe: Abhishek Bhrushundi

## 1 Overview

The goal of this lecture is to develop machinery in order to efficiently find integer solutions to a given system of linear equations with integer coefficients. We will begin by looking at how one solves the same problem over finite fields and over the rationals, and what kind of problem one runs into while doing so naively. In particular, when using Gaussian elimination on a given matrix with rational entries, it's important to make sure that the entries don't blow up too much during the intermediate stages, if one is aiming for an efficient algorithm (polynomial time, to be more specific).
After the detour, we shall get back to the original problem, introducing the Hermite Normal Form (HNF) for a matrix with integer entries, and also proving its existence by giving a constructive proof. It turns out that the algorithm implicit in the constructive proof is not efficient since the intermediate entries can blow up, and thus we will propose an efficient algorithm to find the HNF of a given integer matrix, which shall be the stepping to stone to the algorithm for finding integer solutions to a system of linear equation. This algorithm (for finding integer solutions) will be described in full detail in the next lecture, along with its analysis.

## 2 Solving systems of linear equations over finite fields

### 2.1 The setup

Since we haven't yet dealt with the construction of fields whose size is a power of a prime, we shall restrict our attention to fields $\mathbb{F}_p$, where $p$ is a prime. Suppose we are given $n$ linear equations over $\mathbb{F}_p$ in $n$ variables, and want to find solutions to the this system.

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1\\
&a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2\\
&\ldots\\
&\ldots\\
&a_{n1}x_1 + a_{m2}x_2 + \ldots + a_{nn}x_n = b_n
\end{aligned}$$

Suppose that the coefficient matrix for the above system is represented by the $n \times n$ matrix $A$. Recall that the standard technique to solve such a system is to use *Gaussian elimination* to bring $A$ into a *row reduced form* $A'$ (or column reduced form, but we shall stick to the row form), such that the solution space of $A'$ is exactly equal to the solution space of $A$. Once we have this $A'$ it's straight-forward to find the solution(s) of the system.

**Remark**: When employing Gaussian elimination, we will not worry about making the leading coefficient of every row into a '1'. Thus, we shall only use elementary row operations of two kinds: row operations for swapping rows, and row operations for adding a multiple of one row to another. This will ensure that the determinant of the matrix remains invariant as far as the magnitude goes.

## 2.2 Computational complexity

Every entry of the matrix $A$ is $O(\log p)$ bits, so the size of the input to this problem is roughly $n \times n \times \log p$. Recall that Gaussian elimination involves a sequence of elementary row operations on the matrix. It is not hard to see that the total number of such row operations is roughly $O(n^2)$. Moreover, each row operation requires $O(n)$ field operations, thus making the whole process require $O(n^3)$ field operations.
Another important fact is that, since we are working over $\mathbb{F}_p$, the result of all field operations on field elements will result in another field element, which means that at any intermediate step, every entry of the matrix is at most $\log p$ bits in size.
Let's talk a bit about the complexity of the field operations now. Firstly, addition, subtraction, and multiplication in $\mathbb{F}_p$ basically boils down to doing standard arithmetic over the integers and going mod $p$. In the previous lectures we saw how this can be done in time polynomial in $\log p$. But what about division in $\mathbb{F}_p$? Note that performing $a/b$ in $\mathbb{F}_p$ boils down to finding $b^{-1}$ and computing $ab^{-1}$. Can we find inverses efficiently?

**Claim 1.** *For any $a \in \mathbb{F}_p$, $a^{-1}$ can be computed in time polynomial in $\log p$.*

*Proof.* First, note that over the integers, $gcd(a, p) = 1$. This means we can always find integers $s$ and $t$ such that $as + pt = 1$. If we go mod $p$ on both sides, we get $as \equiv 1 \mod p$. Thus, we are done if we can find $s$ and $t$ (we just need $s$ and $t$ mod $p$!) efficiently.
Suppose we feed $a$ and $p$ into Euclid's algorithm (refer to Lecture 1), and get the following equations:

$$\begin{aligned}
p &= aq_0 + r_0 \\
a &= r_0q_1 + r_1 \\
r_0 &= r_1q_2 + r_2 \\
&\dots \\
r_{k-3} &= r_{k-2}q_{k-1} + r_{k-1} \\
r_{k-2} &= r_{k-1}q_k + r_k
\end{aligned}$$

such that $r_{k-1} = 1, r_k = 0$. Thus, the euclidean algorithm takes $k + 1$ steps, and $gcd(p, a) = gcd(r_{k-2}, r_{k-1}) = r_{k-1} = 1$.
We know $r_0 = p - aq_0$, $r_1 = a - r_0q_1$ and $r_2 = r_0 - r_1q_2$. Thus, using these three equations, we can write $r_1$ and $r_2$ as an integer linear combination of $a$ and $p$.
Now it's easy to see that by repeating this forward-substitution process roughly $k$ times, we can express $r_{k-1}$ as an integer combination of $a$ and $p$, where the integers coefficients of $a$ and $p$ will be in terms of sums and products over $\{q_j\}_{j=0}^{k}$. The only concern is that the intermediate or even the final coefficients (i.e. $s$ and $t$) might be too large (Can this happen?). Since we are only interested in the value of $s$ mod $p$, we can do all the multiplications and additions involved during the intermediate steps mod $p$, and only keep the value of the coefficients mod $p$.
Thus, the complete algorithm needs roughly $k$ steps to find $gcd(a, p)$ (which also gives us $\{q_j\}_{j=0}^{k}$)

and another $k$ steps to do the forward substitutions, where each step requires a constant number arithmetic operations (here, by a constant number we mean a value independent of $k$), making the total number of arithmetic operations $O(k)$. Now, every arithmetic operation requires $O(\log p)$ steps, and, it follows from our discussion in Lecture 1 that $k \leq O(\log p)$, making the overall complexity $O(\log^2 p)$. □

We conclude this section by noting that, in the light of the above discussion, solving a system of linear equations over $\mathbb{F}_p$ needs at most $O(n^3 \log^2 p)$ operations, which is polynomial in the input size.

# 3 Solving systems of linear equations over the rationals

Suppose we have the same setup as in Section 2, the only difference being that this time the $a_{ij}$ are in $\mathbb{Q}$ and are presented to us as $(p_{ij}, q_{ij})$ such that $a_{ij} = p_{ij}/q_{ij}$ and $\forall i, j$, $p_{ij}, q_{ij}$ are $m$ bit integers. We want to find solution vectors $(x_1, \ldots, x_n) \in \mathbb{Q}^n$ for this system. Our algorithm in this case will the same as before (i.e. Gaussian elimination), with the additional condition that the result of every arithmetic operation on two rationals will be brought to the reduced form (The reason for doing so will become clear later).

## 3.1 Computational complexity

Note that the size of the input to this problem is $O(n^2 m)$, and following the argument from Section 2, the total number of arithmetic operations (reducing a fraction to its lowest terms will be considered an arithmetic operation) is $O(n^3)$.
There is, however, a problem that comes up, which did not arise in the case of finite fields: are the intermediate and final entries of the matrix small (i.e. the number of bits is polynomial in $n, m$)? Assuming that they were small after every step, it is easy to see that the complexity of the algorithm would then be $O(poly(n, m))$, since every arithmetic operation can be performed in $poly(n, m)$ steps (Here, by $poly(n, m)$ we mean some polynomial in $n, m$). We now turn to proving that the intermediate entries are indeed small.

## 3.2 Bounding intermediate values

Recall that the Gaussian elimination will have $n-1$ rounds, where, broadly speaking, the $k^{th}$ round involves, possibly, a row swap, followed by the subtraction of multiples of the $k^{th}$ row from the rows below. Suppose, after $k$ round of Gaussian elimination, we have reduced $A$ to the matrix $A_k$:

$$A_k = \begin{pmatrix} B_k & C_k \\ 0 & D_k \end{pmatrix}$$

where $B_k$ is an $k \times k$ upper triangular matrix, the 0 denotes a 0-matrix of size $(n-k) \times k$, $C_k$ is a matrix of size $k \times (n-k)$, and $D_k$ is then a matrix of size $(n-k) \times (n-k)$.

**Claim 2.** *$\forall k \in \{0, \ldots n-1\}$, every entry of $A_k$ can be represented using at most $poly(n, m)$-bits.*

*Proof.* We can assume without loss of generality that none of the rounds involve swapping rows [1]. Let us write $A$ in a similar manner as $A_k$:

$$A = \begin{pmatrix} B & C \\ E & D \end{pmatrix}$$

Let us first look at entries of $D_k$. Consider any entry $(D_k)_{i,j}$. Consider the $(k+1) \times (k+1)$ matrix $J_{ijk}$:

$$J_{ijk} = \begin{pmatrix} B_k & (C_k)_j \\ 0 & (D_k)_{i,j} \end{pmatrix}$$

where $(C_k)_j$ is the $j^{th}$ column of $C_k$, 0 denotes a 0-matrix of dimension $1 \times k$. Let $J_{ij}$ be the matrix:

$$J_{ij} = \begin{pmatrix} B & C_j \\ 0 & D_{i,j} \end{pmatrix}$$

Since $J_{ijk}$ can been obtained from $J_{ij}$ by just doing row operations that involve subtracting a multiple of one row from another, we have

$$det(J_{ijk}) = det(J_{ij})$$

By the same argument, we have

$$det(B) = det(B_k)$$

Moreover, $det(J_{ijk}) = det(B_k)(D_k)_{ij}$. Combining these equations, we get:

$$(D_k)_{ij} = \frac{det(J_{ij})}{det(B)}$$

We can assume that the matrix $A$ has all integer entries to begin with because, if not, we can always clear the denominators while blowing up the number of bits required for each entry of the matrix only polynomially (to be precise, this blows up the number of bits from $m$ to $m^3$, in the worst case).

Now, the determinant of any sub-matrix of $A$ can be represented using a polynomial (in $n, m$) number of bits[2], which then implies that $(D_k)_{ij}$ can be expressed as a rational, such that both its numerator and denominator can be expressed using a polynomial (in $n, m$) number of bits.

Let us now turn our attention to an entry $(C_k)_{ij}$, of the matrix $C_k$. Note that the round in which this entry gets affected for the last time is the $(i-1)^{th}$ round. Consider the matrix $A_{i-1}$ as above, representing the entries after the $(i-1)^{th}$ round. Clearly, $(C_k)_{ij}$ is an entry of $(D_{i-1})$ in $A_{i-1}$, and since we already proved that, for all $l \in \{1, \ldots, n-1\}$, every entry of $D_l$ can be represented using polynomially many bits, the same holds for $(C_k)_{ij}$. A similar argument works for the entries of the matrix $B_k$, which proves the complete claim. □

Up till now, we have only shown that the intermediate values we obtain during Gaussian elimination *have* a "short" representation. The fact that we reduce all intermediate fractions to their lowest terms *ensures* that we actually end up using this "short" representation after every round.

[1] We can assume that all the required row swaps are done before hand. Let $A_k$ represent the matrix in which row swaps are done before hand, and $A_k^0$ represent the matrix in which they are done during the rounds, then each of $B_k, C_k$ and $D_k$ are equivalent to their counterparts in $A_k^0$ upto a permutation of the rows, which really does not affect the claim we are proving.

[2] The determinant is given by $\sum_{\sigma \in S_n} sign(\sigma) \prod_{i=1}^{n} a_{i\sigma(i)}$, so there are $n!$ terms, each at most $2^{poly(m)}$, and they can sum up to at most $n! 2^{poly(m)}$, which is representable using $O(n \log n + poly(m))$ bits.

# 4 Solving systems of linear equations over the integers

Suppose we are given $m$ linear equations over $\mathbb{Z}$ in $n$ variables, and want to find integer solutions to this system. We are assured that each $a_{ij}$ is at most $N$ bits, making the total input size $mnN$.

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\
&a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\
&\ldots \\
&\ldots \\
&a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m
\end{aligned}$$

Notice that we are no longer working over a field and this basically rules out using Gaussian elimination, although we shall use something that's similar.

## 4.1 Enter Hermite Normal Form (HNF)

Let $A$ be the $m \times n$ integer matrix representing the above system. Finding solutions to this system can be phrased as finding integer linear combinations of the columns of $A$ that are equal to $b$, where $b$ is the column vector $(b_1, \ldots, b_m)$. In other words, is $b$ in the integer span of the columns of $A$?[3] To mimic Gaussian elimination, it is important that we understand what kind of row or column operations preserve the integer span of the columns of $A$. The following are such operations:

1. Subtracting an integer multiple of one column from another i.e. $y_j \to y_j - c \times y_i$, where $c \in \mathbb{Z}$.
2. Multiplying the entries of a column by $-1$ i.e. $y_j \to -y_j$.
3. Swapping two columns, i.e. $y_i \leftrightarrow y_j$.

Our goal will be to take an arbitrary integer matrix $A$, and perform column operations of the above time to bring it into a nice form.

**Definition 3** (Hermite Normal Form)**.** *An $m \times n$ matrix is said to be in Hermite Normal Form (HNF) if it has the following form:*

$$\begin{pmatrix} B & 0 \end{pmatrix}$$

*such that:*

- *$B$ is an $m \times m$ lower triangular matrix.*
- *$0$ denotes an $m \times (n-m)$ 0-matrix.*
- *$\forall i$, $\forall j < i$, $0 \leq B_{ij} < B_{ii}$.*

**Proposition 4.** *Let $A$ be an $m \times n$ matrix with integer entries and column rank $m$. Then $A$ can be brought into Hermite Normal Form through a sequence of column operations of the above type. Furthermore, the resulting matrix in Hermite Normal Form is unique.*

[3] Additionally, we can assume that the columns of $A$ have rank $m$ over $\mathbb{Q}$, otherwise we can "reduce" the size of the problem and proceed.

*Proof.* We shall give a constructive proof to show the existence of the HNF. *The procedure we give now is not efficient! This procedure is only to show the existence of an HNF for every $A$. We address efficiency in the next section.*

Consider the following algorithms:
**Algorithm**: **FindPreHNF**
**Input**: An $m' \times n'$ integer matrix $A$ with column rank $m'$
**Output**: HNF of $A$ without the guarantee that off-diagonal entries are smaller than the diagonal ones.

1. Since we have a full rank matrix, the first row is non-zero. By multiplying columns with $-1$, we can ensure that all non-zero entries of the first row are positive.
2. By using column operations of the first kind (See 4.1), we compute the GCD of all the non-zero entries of the first row, such that we are left with only one non-zero entry at the end of the process, which is equal to the GCD itself.
3. We then use a column swap to ensure that this non-zero entry is at position $(1, 1)$ (the other entries in the first row are all zero). At this stage the matrix has the following form:
$$\begin{pmatrix} B_{11} & 0 & \dots & 0 \\ B_{21} & * & \dots & * \\ \vdots & \vdots & \vdots & \vdots \\ B_{m1} & * & \dots & * \end{pmatrix}$$
4. If $m = 1$, the above matrix is just a row $\begin{pmatrix} B_{11} & 0 & \dots & 0 \end{pmatrix}$, and we return it.
5. Otherwise, let us denote the lower-left $(m-1) \times (n-1)$ matrix in the above matrix by $A'$. Notice that $A'$ is an integer matrix with column rank $m-1$. We can now recurse on $A'$: $C$ = **FindPreHNF**$(A')$.
6. We then return the following matrix:
$$\begin{pmatrix} B_{11} & 0 & \dots & 0 \\ B_{21} & C_{11} & \dots & C_{1,n-1} \\ B_{21} & C_{21} & \dots & C_{2,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ B_{m1} & C_{m-1,1} & \dots & C_{m-1,n-1} \end{pmatrix}$$
where $C_{i,j}$ denotes the $(i, j)$ entry of the matrix $C$ obtained in the last step.

Running the above algorithm on $A$ will produce a matrix of the form
$$\begin{pmatrix} B & 0 \end{pmatrix}$$
where $B$ is an $m \times m$ lower-triangular matrix, though, along a row in $B$, the off-diagonal entries can be larger than the diagonal ones.
To ensure that this does not happen, we use the following algorithm:
**Algorithm**: **ReduceOffDiagonal**

**Input**: An $m' \times m'$ integer lower-triangular matrix $B$ with full rank
**Output**: An $m' \times m'$ integer lower-triangular matrix $B'$ with the condition that $\forall i$, $\forall j < i$, $0 \leq B'_{ij} < B'_{ii}$, such that to go from $B$ to $B'$ we just need elementary column operations (See 4.1).

**for** $j = 1$ to $m' - 1$ **do**
  **for** $i = j + 1$ to $m'$ **do**

1. If $B_{i,j} < 0$, then we add a large enough of multiple of column $B_i$ to column $B_j$ so that $0 \leq B_{i,j} < B_{i,i}$. This is achieved by $B_j \to B_j + \lceil \frac{|B_{i,j}|}{B_{i,i}} \rceil B_i$.
2. Otherwise if $B_{i,j} > B_{i,i}$, we subtract a large enough multiple of column $B_i$ from column $B_j$ so that $0 \leq B_{i,j} < B_{i,i}$. This is achieved by $B_j \to B_j - \lfloor \frac{B_{i,j}}{B_{i,i}} \rfloor B_i$.

  **end for**
**end for**

It can easily be verified that the above algorithm is correct, the proof idea being that after the $j^{th}$ iteration of the outer loop, all the off-diagonal entries present in columns 1 to $j$ are less than their respective diagonal entries, and this inequality is not perturbed by the subsequent loop iterations.

Combining **FindPreHNF** along with **ReduceOffDiagonal** proves the existence of the HNF. To prove the uniqueness, say the full-rank matrix $A$ has two HNFs, $H$ and $H'$. Then look at the smallest $i$ such that there are entries $h_{ij}$ and $h'_{ij}$ such that $h_{ij} \neq h'_{ij}$. Without loss of generality, $h_{ij} > h'_{ij}$. Let us look at the difference $h_j - h'_j$, where $h_j$ and $h'_j$ are the $j^{th}$ columns of $H$ and $H'$ respectively.
Since both $H$ and $H'$ have the same integer linear span, $h_j - h'_j$ is in the integer span of $H$ and can be written as a linear combination of the columns of $H$. But given our choice of $i$ and $j$, $h_j - h'_j$ has its first non-zero entry at $i$ (where the entries with a lower index are all zero), which means that $h_j - h'_j$ can be written as a linear combination of only columns $i$ to $m'$. Among these, since the $i^{th}$ column is the only one with a non-zero entry at row $i$, we have that $h_{ij} - h'_{ij} = c \times h_{ii}$, where $c \in \mathbb{Z}$.
Now, $h'_{ij} < h_{ij} < h_{ii}$, which means that $h_{ij} - h'_{ij} < h_{ii}$. Thus, $c$ must be zero, which is a contradiction. Thus, there cannot exist an $i$ and $j$ such that $h_{ij} \neq h'_{ij}$. □

**Corollary 5.** *Let $A$ and $A'$ be two full-rank integer matrices with dimensions $m \times n$ and $m \times n'$ $(n > m, n' > m)$ respectively, such that the integer span of $A$ is equal to that of $A'$, then:*

- *If $n > n'$, the HNF of $A$ is just the HNF of $A'$ appended with $n - n'$ zero columns.*
- *If $n < n'$, the HNF of $A'$ is just the HNF of $A$ appended with $n' - n$ zero columns.*
- *else if $n = n'$, the HNF of $A$ is the same as the HNF of $A'$.*

*Proof.* The proof is exactly the same as the proof for the uniqueness of HNF given above: we apply the algorithm (i.e. **FindPreHNF** + **ReduceOffDiagonal**) to both $A$ and $A'$ to get $H$ and $H'$, and since the integer spans of $H$ and $H'$ are the same, the uniqueness argument from the previous proof tells us that $H$ and $H'$ are identical in their respective lower-triangular parts. The only difference is that one of them might have more zero columns towards the end than the other if $n - n' \neq 0$. □

### 4.2 A polynomial time algorithm for finding the HNF

The first question one might ask is, since we gave a constructive proof for the existence of the HNF, why not use the algorithm implicit in the proof? Unfortunately, unlike the previous section, there is no way of showing that the intermediate values encountered during the algorithm cannot be too large (not too large here means polynomial in $n, m, N$). We will now suggest a way to get around this problem.
Let us choose some $m$ independent columns from the matrix $A$ and put them together in an $m \times m$ matrix $Z$. Let $k = det(Z)$.

**Claim 6.** *If $I_{m\times m}$ denotes the identity matrix, then the columns of the matrix $kI_{m\times m}$ are in the integer span of the columns of $Z$ (and thus the columns of $A$).*

*Proof.* $kI_{m\times m}$ can be written as $(Z \times Z^{-1}) \times k$, which is the same as $Z \times (Z^{-1} \times det(Z))$. If we can show that $Z^{-1} \times det(Z)$ has all integer entries, we would be done.
Using Cramer's rule, $Z^{-1} \times det(Z)$ can be written as $((1/det(Z)) \times \hat{Z} \times det(Z) = \hat{Z}$, where $\hat{Z}$ is the matrix of cofactors of $Z$. Since every entry of the cofactor matrix is an integer, the claim follows. □

Consider the $m \times 2m$ matrix $\hat{A} = \begin{pmatrix} Z & kI_{m\times m} \end{pmatrix}$. Using Corollary 5, we observe that the HNF for $A$ can be obtained from the HNF of $\hat{A}$, since both have the same integer span, and thus it's enough to find the HNF for $\hat{A}$.

Note that computing $\hat{A}$ from $A$ can be done in polynomial time based on the results of Section 3 (In particular, it involves using Gaussian elimination over $\mathbb{Q}$ to find $m$ independent column vectors and to compute the determinant). The next step is to find the HNF for $\hat{A}$, and the algorithm for this is exactly the same as before (i.e. **FindPreHNF** + **ReduceOffDiagonal**) with a slight modification in the **FindPreHNF** algorithm. The modification is as follows:

*In step 2, whenever we use column operations of the first kind i.e. adding/subtracting a multiple of one column from another, and as a result of the computation, some entry of the matrix, say the $(i, j)$ entry, exceeds $k$, we pick the $i^{th}$ column from the $kI_{m\times m}$ sub-matrix and subtract a large enough multiple of this column from the $j^{th}$ column of the matrix so that the $(i, j)$ entry becomes less than $k$ (This amounts to subtracting a large enough integer multiple of $k$ from the $(i, j)$ entry without disturbing the other entries of the $j^{th}$ column of the matrix).*

The next step of our algorithm is to make sure that the off-diagonal entries are less than the diagonal ones and this is done by simply running the **ReduceOffDiagonal** algorithm on the matrix.

### 4.3 Analysis of the modified algorithm

Since we are not interested in computing the exact complexity of the algorithm, we only need a rough analysis in order to show that the running time is polynomial in $n, m$ and $N$.
Firstly, note that if we can show that all intermediate values during the algorithm have at most polynomially many bits, then in order to prove that the algorithm runs in polynomial time, it suffices to show that:

1. the total number of arithmetic operations is polynomial, and

2. the total number of non-arithmetic operations (swapping etc.) is also polynomial.

Since both 1. and 2. can easily be verified, we will focus our attention on proving that the entries don't blow up too much during the algorithm.

It can easily be verified that, since $k = det(Z)$ is at most $poly(m, N)$ bits long, all the intermediate values encountered during this modified **FindPreHNF** algorithm are polynomial in $n, m$ and $N$, since, after every arithmetic operation, we keep making sure that they do not become larger than $k$.
What about the **ReduceOffDiagonal** algorithm? Note that after the $j^{th}$ iteration of the outer loop, the entries in columns $j$ are never altered. In fact the only time the entries of the $j^{th}$ column are altered is in the $j^{th}$ iteration of the outer loop. Thus it suffices to prove the following claim:

**Proposition 7.** *Let $1 \leq j < m$ (Assume that the input parameter $m'$ of the algorithm is equal to $m$), then, after the $j^{th}$ iteration of the outer loop, every entry of the $j^{th}$ column is at most $mN$ bits long.*

The proposition follows from the following claim:

**Claim 8.** *Consider the $j^{th}$ iteration of the outer loop. Let $j + 1 \leq i \leq m$, then all the entries in column $j$ are at most $(i - j + 1)N$ bits long after the $(i - j)^{th}$ iteration of the inner loop.*

*Proof.* Let us look at the base case when $i = j + 1$ i.e. the $1^{st}$ iteration. Recall that, as discussed before, the $j^{th}$ column is unaltered up till this point and thus every entry is at most $N$ bits. Let us assume without loss of generality that $B_{ij} > B_{ii}$. This will mean that we do the operation $B_j \to B_j - \lfloor \frac{B_{i,j}}{B_{i,i}} \rfloor B_i$. By how much can this operation blow up the size of the entries of column $j$? In the worst case, an entry of $B_j$ could be zero, $\lfloor \frac{B_{i,j}}{B_{i,i}} \rfloor$ could be as large as $N$ bits (for example, if $B_{i,i}$ is 1), and the other entries (barring the $j^{th}$ entry) of $B_i$ could be $N$ bits, making an entry of column $j$ as large as $2N$. Notice that this proves the base case.
Let us assume that the assertion holds up till $i - 1$ i.e. after the $(i - 1 - j)^{th}$ iteration. So all the entries in column $j$ are at most $(i - j)N$ bits. We will show that the assertion holds even after the $(i - j)^{th}$ iteration, thus proving the claim. Again, in the worst case, without loss of generality, we perform the operation $B_j \to B_j - \lfloor \frac{B_{i,j}}{B_{i,i}} \rfloor B_i$. Now, this time, $\lfloor \frac{B_{i,j}}{B_{i,i}} \rfloor$ can be as large as $(i - j)N$ bits (by a similar argument as the base case), and the entries of $B_i$ (except $B_{i,i}$) could be as large as $N$ bits. In this situation, the entries of $B_j$ can become at most $(i - j)N + N = (i - j + 1)N$ bits. □