Chapter 6  Practical Implementation of the Attacks on RSA

6.1  The RSA Algorithm (Concept)

The RSA algorithm is named after the scientist, who invented it in 1977. The basic technique was first discovered in 1973 by Clifford Cocks of CESG (part of the British GCHQ) but this was a secret until 1997. The patent taken out by RSA Labs has expired. The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

Public-Key Applications can classify uses into 3 categories:

- encryption/decryption (provide secrecy)
- digital signatures (provide authentication)
- key exchange (of session keys)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Encryption/Decryption</th>
<th>Digital Signature</th>
<th>Key Exchange</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Elliptic Curve</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Diffie-Hellman</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DSS</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 6.1 Applications of Public Key Cryptosystem

a.  Key Generation Algorithm

Generate two large random primes, \( p \) and \( q \), of approximately equal size such that their product \( n = pq \) is of the required bit length, e.g. 1024 bits.

1. Compute \( n = pq \) and \( (\varphi) \) phi = \((p-1)(q-1)\).
2. Choose an integer \( e \), \( 1 < e < \varphi \), such that \( \gcd(e, \varphi) = 1 \).
3. Compute the secret exponent \( d \), \( 1 < d < \varphi \), such that \( ed \equiv 1 \pmod{\varphi} \).
4. The public key is \((n, e)\) and the private key is \((n, d)\). Keep all the values \( d \), \( p \), \( q \) and \( \varphi \) secret.
   - \( n \) is known as the modulus.
   - \( e \) is known as the public exponent or encryption exponent or just the exponent.
   - \( d \) is known as the secret exponent or decryption exponent.

b.  Encryption

Sender A does the following:-
1. Obtains the recipient B's public key (n, e).
2. Represents the plaintext message as a positive integer \( m \).
3. Computes the ciphertext \( c = m^e \mod n \).
4. Sends the ciphertext \( c \) to B.

c. **Decryption**

Recipient B does the following:
1. Uses his private key (n, d) to compute \( m = c^d \mod n \).
2. Extracts the plaintext from the message representative \( m \).

d. **Digital Signing**

Sender A does the following:
1. Creates a message digest of the information to be sent.
2. Represents this digest as an integer \( m \) between 0 and \( n-1 \).
3. Uses her private key (n, d) to compute the signature \( s = m^d \mod n \).
4. Sends this signature \( s \) to the recipient, B.

e. **Signature Verification**

Recipient B does the following:
1. Uses sender A's public key (n, e) to compute integer \( v = s^e \mod n \).
2. Extracts the message digest from this integer.
3. Independently computes the message digest of the information that has been signed.
4. If both message digests are identical, the signature is valid.

f. **Key Length**

When we talk about the key length of an RSA key, we are referring to the length of the modulus, \( n \), in bits. The minimum recommended key length for a secure RSA transmission is currently 1024 bits. A key length of 512 bits is now no longer considered secure, although cracking it is still not a trivial task for the likes of you and me. The longer your information is needed to be kept secure, the longer the key you should use. Keep up to date with the latest recommendations in the security journals.

There is small one area of confusion in defining the key length. One convention is that the key length is the position of the most significant bit in \( n \) that has
value '1', where the least significant bit is at position 1. Equivalently, key length = ceiling \((\log_2(n+1))\). The other convention, sometimes used, is that the key length is the number of bytes needed to store \(n\) multiplied by eight, i.e. ceiling \((\log_{256}(n+1))\).

The most significant byte 0x0A in binary is 00001010B. The most significant bit is at position 508, so its key length is 508 bits. On the other hand, this value needs 64 bytes to store it, so the key length could also be referred to by some as 64 \(\times\) 8 = 512 bits. We prefer the former method. You can get into difficulties with the X9.31 method for signatures if you use the latter convention.

<table>
<thead>
<tr>
<th>Number of Decimal Digits</th>
<th>Approximate Number of Bits</th>
<th>Date Achieved</th>
<th>MIPS-years</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>332</td>
<td>April 1991</td>
<td>7</td>
<td>quadratic sieve</td>
</tr>
<tr>
<td>110</td>
<td>365</td>
<td>April 1992</td>
<td>75</td>
<td>quadratic sieve</td>
</tr>
<tr>
<td>120</td>
<td>308</td>
<td>June 1993</td>
<td>830</td>
<td>quadratic sieve</td>
</tr>
<tr>
<td>129</td>
<td>428</td>
<td>April 1994</td>
<td>5000</td>
<td>quadratic sieve</td>
</tr>
<tr>
<td>130</td>
<td>431</td>
<td>April 1996</td>
<td>1000</td>
<td>generalized number field sieve</td>
</tr>
<tr>
<td>140</td>
<td>465</td>
<td>February 1999</td>
<td>2000</td>
<td>generalized number field sieve</td>
</tr>
<tr>
<td>155</td>
<td>512</td>
<td>August 1999</td>
<td>8000</td>
<td>generalized number field sieve</td>
</tr>
</tbody>
</table>

Table 6.2 Progress in Factorization

6.2 Converting Modular equations of RSA into CNF form to apply on SAT SOLVER

- \(n = pq\), where \(p\) and \(q\) are distinct primes.
- \(\phi = (p-1)(q-1)\)
- \(e < n\) such that \(gcd(e, \phi) = 1\)
- \(d = e^{-1}\) mod \(\phi\).
- \(c = m^e\) mod \(n\), \(1 < m < n\).
- \(m = c^d\) mod \(n\).

6.3 Conjunctive Normal Form

In Boolean logic, a formula is in conjunctive normal form (CNF) if it is a conjunction of clauses, where a clause is a disjunction of literals, where a literal and its complement cannot appear in the same clause. As a normal form, it is useful in
automated theorem proving. It is similar to the product of sums form used in circuit theory.

All conjunctions of literals and all disjunctions of literals are in CNF, as they can be seen as conjunctions of one-literal clauses and conjunctions of a single clause, respectively. As in the disjunctive normal form (DNF), the only propositional connectives a formula in CNF can contain are and, or, and not. The not operator can only be used as part of a literal, which means that it can only precede a propositional variable.

Examples and counterexamples

All of the following formulas are in CNF:

\[-A \land (B \lor C)\]
\[(A \lor B) \land (\neg B \lor C \lor \neg D) \land (D \lor \neg E)\]
\[A \land B.\]

The last formula is in CNF because it can be seen as the conjunction of the two single-literal clauses A and B. Incidentally; this formula is also in disjunctive normal form. The following formulae are not in CNF:

\[\neg (B \lor C)\]
\[(A \land B) \lor C\]
\[A \land (B \lor (D \land E)).\]

The above three formulae are respectively equivalent to the following three formulas that are in CNF:

\[-B \land \neg C\]
\[(A \lor C) \land (B \lor C)\]
\[A \land (B \lor D) \land (B \lor E).\]
6.3.1 Conversion into CNF

Every propositional formula can be converted into an equivalent formula that is in CNF. This transformation is based on rules about logical equivalences: the double negative law, De Morgan's laws, and the distributive law. Since all logical formulae can be converted into an equivalent formula in conjunctive normal form, proofs are often based on the assumption that all formulae are CNF. However, in some cases this conversion to CNF can lead to an exponential explosion of the formula. For example, translating the following non-CNF formula into CNF produces a formula with $2^n$ clauses:

\[(X_1 \land Y_1) \lor (X_2 \land Y_2) \lor \ldots \lor (X_n \land Y_n)\].

In particular, the generated formula is:

\[(X_1 \lor \ldots \lor X_{n-1} \lor X_n) \land (X_1 \lor \ldots \lor X_{n-1} \lor Y_n) \land \ldots \land (Y_1 \lor \ldots \lor Y_{n-1} \lor Y_n)\].

In plain English, this formula contains $2^n$ clauses; each clause contains either $X_i$ or $Y_i$ for each $i$. There exist transformations into CNF that avoid an exponential increase in size by preserving satisfiability rather than equivalence\(^1\)[2]. These transformations are guaranteed to only linearly increase the size of the formula, but introduce new variables. For example, the above formula can be transformed into CNF by adding variables $Z_1, \ldots, Z_n$ as follows:

\[(Z_1 \lor \ldots \lor Z_n) \land (\neg Z_1 \lor X_1) \land (\neg Z_1 \lor Y_1) \land \ldots \land (\neg Z_n \lor X_n) \land (\neg Z_n \lor Y_n)\].

An interpretation satisfies this formula only if at least one of the new variables is true. If this variable is $Z_i$, then both $X_i$ and $Y_i$ are true as well. This means that every model that satisfies this formula also satisfies the original one. On the other hand, only some of the models of the original formula satisfy this one: since the $Z_i$ are not mentioned in the original formula, their values are irrelevant to satisfaction of it, which is not the case in the last formula. This means that the original formula and the result of the translation are equi satisfiable but not equivalent.

An alternative translation includes also the clauses $Z_i \lor \neg X_i \lor \neg Y_i$. With these clauses, the formula implies $Z_i \equiv X_i \land Y_i$; this formula is often regarded to "define" $Z_i$ to be a name for $X_i \land Y_i$. 

72
6.4 Introduction to SAT

Propositional satisfiability problem (SAT) is the problem of deciding if there is a truth assignment under which a given propositional formula (in conjunctive normal form) evaluates to true. It is a canonical NP-complete problem (S. A. Cook, USA, 1971) and it holds a central position in the field of computational complexity. SAT problem is also important in many practical applications such as electronic design automation, software and hardware verification, artificial intelligence, and operations research. Thanks to recent advances in propositional solving.

SAT solvers are becoming a tool suitable for attacking more and more practical problems. Some of the solvers are complete, while others are stochastic. For a given SAT instance, complete SAT solvers can either find a solution (i.e., a satisfying variable assignment) or show that no solution exists. Stochastic solvers, on the other hand, cannot prove that an instance is unsatisfiable although they may be able to find a solution for certain kinds of large satisfiable instances quickly. The majority of the state-of-the-art complete SAT solvers are based on the branch and backtracking algorithm called Davis-Putnam-Logemann-Loveland, or DPLL (Davis, M., Putnam, H, 1960; Davis, M.; Logemann, G.; Loveland, D., 1962). Starting with the work on the GRASP and SATO systems (J P. Marques-Silva and K A. Sakallah, 1999; H. Zhang, SATO: 1997), and continuing with Chaff, Berk Min and MiniSAT (M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. 2001; E. Goldberg and Y. Novikov. Berkmin: 2002, N. Een and N. Sorensson. 2004). The spectacular improvements in the performance of DPLL-based SAT solvers achieved in the last years are due to the following

(i) Several conceptual enhancements of the original DPLL procedure, aimed at reducing the amount of explored search space, such as backjumping, conflict-driven lemma learning, and restart.
(ii) Better implementation techniques, such as the two-watch literals scheme for unit propagation. These advances make it possible to decide the satisfiability of industrial SAT problems with tens of thousands of variables and millions of clauses.

While SAT solvers have become complex, describing their underlying algorithms and data structures has become a nontrivial task. Unfortunately, there is
still a large gap between these two approaches. Higher level presentations, although clean and accompanied with correctness proofs, omit many details that are vital to efficient solver implementation. Lower level presentations usually give SAT solver algorithms in a form of pseudo-code. The open source SAT solvers themselves are, in a sense, the most detailed presentations or specifications of SAT solving techniques. The success of MiniSAT (N. Een and N. Sörensson.2004), and the number of its re-implementations, indicates that detailed descriptions of SAT solvers are needed and welcome in the community. However, in order to achieve the highest possible level of efficiency, these descriptions are far from the abstract, algorithmic level. Often, one procedure in the code contains several higher level concepts or one higher level algorithm is spread across several code procedures.

6.5 SAT Solver Tool

Mostly SAT solvers are based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm and require the input formula to be in Conjunctive Normal Form (CNF). However, typical formulas that arise in practice are non-clausal, that is, not in CNF. Converting a general formula to CNF introduces overhead in the form of new variables and may destroy the structure of the initial formula, which can be useful to check satisfiability efficiently (Davis, M., Putnam, H. 1960).

Satisfiability (often written in all capitals or abbreviated SAT) is the problem of determining if the variables of a given Boolean formula can be assigned in such a way as to make the formula evaluate to TRUE. Equally important is to determine whether no such assignments exist, which would imply that the function expressed by the formula is identically FALSE for all possible variable assignments. In this latter case, we would say that the function is unsatisfiable; otherwise it is satisfiable. For example, the formula a AND b is satisfiable because one can find the values a = TRUE and b = TRUE, which make a AND b TRUE. To emphasize the binary nature of this problem, it is frequently referred to as Boolean or propositional satisfiability.

Boolean satisfiability (SAT) solvers are used heavily in verification tools as decision procedures for propositional logic. In complexity theory, the satisfiability problem (SAT) is a decision problem, whose instance is a Boolean expression written
using only AND, OR, NOT, variables, and parentheses. A formula of propositional logic is said to be satisfiable if logical values can be assigned to its variables in a way that makes the formula true.

**What is a SAT Solver?**

Solves a problem in CNF

CNF is an “and of or-s” \((x_1 \lor \neg x_3) \land (\neg x_2 \lor x_3) \land (x_1 \lor x_2)\)

Using DPLL (φ) Algorithm

- If formula φ is trivial, return SAT/UNSAT
- ret = DPLL(φ with \(v \leftarrow true\))
  - if ret == SAT, return SAT
  - ret = DPLL(φ with \(v \leftarrow false\))
  - if ret == SAT, return SAT
- return UNSAT

6.6 **CNF Snapshot generated from SAT Solver (First Output)**

\[
\begin{align*}
-21 & -11 1 & -21 & 11 & -1 & -21 & 11 1 & 21 & -11 & -1 \\
-21 & -11 7 & -21 & 11 & -7 & -21 & 11 7 & 21 & -11 & -7 \\
\end{align*}
\]

\[
\begin{align*}
-22 & -11 1 & -22 & 11 & -1 & -22 & 11 1 & 22 & -11 & -1 \\
-22 & -11 7 & -22 & 11 & -7 & -22 & 11 7 & 22 & -11 & -7 \\
\end{align*}
\]
| -23 11 1 | -23 11 -1 | -23 11 1 | 23 -11 -1 |
| -23 11 2 | -23 11 -2 | -23 11 2 | 23 -11 -2 |
| -23 11 3 | -23 11 -3 | -23 11 3 | 23 -11 -3 |
| -23 11 4 | -23 11 -4 | -23 11 4 | 23 -11 -4 |
| -23 11 5 | -23 11 -5 | -23 11 5 | 23 -11 -5 |
| -23 11 6 | -23 11 -6 | -23 11 6 | 23 -11 -6 |
| -23 11 7 | -23 11 -7 | -23 11 7 | 23 -11 -7 |
| -23 11 8 | -23 11 -8 | -23 11 8 | 23 -11 -8 |
| -23 11 9 | -23 11 -9 | -23 11 9 | 23 -11 -9 |
| -23 11 10 | -23 11 -10 | -23 11 10 | 23 -11 -10 |

| -24 11 1 | -24 11 -1 | -24 11 1 | 24 -11 -1 |
| -24 11 2 | -24 11 -2 | -24 11 2 | 24 -11 -2 |
| -24 11 3 | -24 11 -3 | -24 11 3 | 24 -11 -3 |
| -24 11 4 | -24 11 -4 | -24 11 4 | 24 -11 -4 |
| -24 11 5 | -24 11 -5 | -24 11 5 | 24 -11 -5 |
| -24 11 6 | -24 11 -6 | -24 11 6 | 24 -11 -6 |
| -24 11 7 | -24 11 -7 | -24 11 7 | 24 -11 -7 |
| -24 11 8 | -24 11 -8 | -24 11 8 | 24 -11 -8 |
| -24 11 10 | -24 11 -10 | -24 11 10 | 24 -11 -10 |

| -25 11 1 | -25 11 -1 | -25 11 1 | 25 -11 -1 |
| -25 11 3 | -25 11 -3 | -25 11 3 | 25 -11 -3 |
| -25 11 4 | -25 11 -4 | -25 11 4 | 25 -11 -4 |
| -25 11 6 | -25 11 -6 | -25 11 6 | 25 -11 -6 |
| -25 11 7 | -25 11 -7 | -25 11 7 | 25 -11 -7 |
| -25 11 8 | -25 11 -8 | -25 11 8 | 25 -11 -8 |
| -25 11 10 | -25 11 -10 | -25 11 10 | 25 -11 -10 |

| -26 11 1 | -26 11 -1 | -26 11 1 | 26 -11 -1 |
| -26 11 2 | -26 11 -2 | -26 11 2 | 26 -11 -2 |
| -26 11 3 | -26 11 -3 | -26 11 3 | 26 -11 -3 |
| -26 11 4 | -26 11 -4 | -26 11 4 | 26 -11 -4 |
| -26 11 5 | -26 11 -5 | -26 11 5 | 26 -11 -5 |
| -26 11 6 | -26 11 -6 | -26 11 6 | 26 -11 -6 |
| -26 11 7 | -26 11 -7 | -26 11 7 | 26 -11 -7 |
| -26 11 8 | -26 11 -8 | -26 11 8 | 26 -11 -8 |
|---------|---------|---------|---------|---------|---------|
| 26-11 10| 26-11 10| 26-11 10| 26-11 10| 26-11 10| 26-11 10|

<table>
<thead>
<tr>
<th>27-11 1</th>
<th>27-11 1</th>
<th>27-11 1</th>
<th>27-11 1</th>
<th>27-11 1</th>
<th>27-11 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>27-11 7</td>
<td>27-11 7</td>
<td>27-11 7</td>
<td>27-11 7</td>
<td>27-11 7</td>
<td>27-11 7</td>
</tr>
<tr>
<td>27-11 8</td>
<td>27-11 8</td>
<td>27-11 8</td>
<td>27-11 8</td>
<td>27-11 8</td>
<td>27-11 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>28-11 1</th>
<th>28-11 1</th>
<th>28-11 1</th>
<th>28-11 1</th>
<th>28-11 1</th>
<th>28-11 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>28-11 8</td>
<td>28-11 8</td>
<td>28-11 8</td>
<td>28-11 8</td>
<td>28-11 8</td>
<td>28-11 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>29-11 1</th>
<th>29-11 1</th>
<th>29-11 1</th>
<th>29-11 1</th>
<th>29-11 1</th>
<th>29-11 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>29-11 2</td>
<td>29-11 2</td>
<td>29-11 2</td>
<td>29-11 2</td>
<td>29-11 2</td>
<td>29-11 2</td>
</tr>
<tr>
<td>29-11 3</td>
<td>29-11 3</td>
<td>29-11 3</td>
<td>29-11 3</td>
<td>29-11 3</td>
<td>29-11 3</td>
</tr>
<tr>
<td>29-11 4</td>
<td>29-11 4</td>
<td>29-11 4</td>
<td>29-11 4</td>
<td>29-11 4</td>
<td>29-11 4</td>
</tr>
<tr>
<td>29-11 5</td>
<td>29-11 5</td>
<td>29-11 5</td>
<td>29-11 5</td>
<td>29-11 5</td>
<td>29-11 5</td>
</tr>
<tr>
<td>29-11 7</td>
<td>29-11 7</td>
<td>29-11 7</td>
<td>29-11 7</td>
<td>29-11 7</td>
<td>29-11 7</td>
</tr>
<tr>
<td>29-11 8</td>
<td>29-11 8</td>
<td>29-11 8</td>
<td>29-11 8</td>
<td>29-11 8</td>
<td>29-11 8</td>
</tr>
<tr>
<td>29-11 9</td>
<td>29-11 9</td>
<td>29-11 9</td>
<td>29-11 9</td>
<td>29-11 9</td>
<td>29-11 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>30-11 1</th>
<th>30-11 1</th>
<th>30-11 1</th>
<th>30-11 1</th>
<th>30-11 1</th>
<th>30-11 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>30-11 2</td>
<td>30-11 2</td>
<td>30-11 2</td>
<td>30-11 2</td>
<td>30-11 2</td>
<td>30-11 2</td>
</tr>
<tr>
<td>30-11 3</td>
<td>30-11 3</td>
<td>30-11 3</td>
<td>30-11 3</td>
<td>30-11 3</td>
<td>30-11 3</td>
</tr>
<tr>
<td>30-11 4</td>
<td>30-11 4</td>
<td>30-11 4</td>
<td>30-11 4</td>
<td>30-11 4</td>
<td>30-11 4</td>
</tr>
<tr>
<td>30-11 5</td>
<td>30-11 5</td>
<td>30-11 5</td>
<td>30-11 5</td>
<td>30-11 5</td>
<td>30-11 5</td>
</tr>
<tr>
<td>30-11 6</td>
<td>30-11 6</td>
<td>30-11 6</td>
<td>30-11 6</td>
<td>30-11 6</td>
<td>30-11 6</td>
</tr>
</tbody>
</table>
-34 -12 5  -34 12 -5  -34 12 5  34 -12 -5
-34 -12 6  -34 12 -6  -34 12 6  34 -12 -6
-34 -12 7  -34 12 -7  -34 12 7  34 -12 -7
-34 -12 8  -34 12 -8  -34 12 8  34 -12 -8
-34 -12 9  -34 12 -9  -34 12 9  34 -12 -9
-34 -12 10  -34 12 -10  -34 12 10  34 -12 -10

-35 -12 1  -35 12 -1  -35 12 1  35 -12 -1
-35 -12 2  -35 12 -2  -35 12 2  35 -12 -2
-35 -12 3  -35 12 -3  -35 12 3  35 -12 -3
-35 -12 4  -35 12 -4  -35 12 4  35 -12 -4
-35 -12 5  -35 12 -5  -35 12 5  35 -12 -5
-35 -12 6  -35 12 -6  -35 12 6  35 -12 -6
-35 -12 7  -35 12 -7  -35 12 7  35 -12 -7
-35 -12 8  -35 12 -8  -35 12 8  35 -12 -8
-35 -12 9  -35 12 -9  -35 12 9  35 -12 -9
-35 -12 10  -35 12 -10  -35 12 10  35 -12 -10

-36 -12 1  -36 12 -1  -36 12 1  36 -12 -1
-36 -12 2  -36 12 -2  -36 12 2  36 -12 -2
-36 -12 3  -36 12 -3  -36 12 3  36 -12 -3
-36 -12 4  -36 12 -4  -36 12 4  36 -12 -4
-36 -12 5  -36 12 -5  -36 12 5  36 -12 -5
-36 -12 6  -36 12 -6  -36 12 6  36 -12 -6
-36 -12 7  -36 12 -7  -36 12 7  36 -12 -7
-36 -12 8  -36 12 -8  -36 12 8  36 -12 -8
-36 -12 9  -36 12 -9  -36 12 9  36 -12 -9
-36 -12 10  -36 12 -10  -36 12 10  36 -12 -10

-37 -12 1  -37 12 -1  -37 12 1  37 -12 -1
-37 -12 2  -37 12 -2  -37 12 2  37 -12 -2
-37 -12 3  -37 12 -3  -37 12 3  37 -12 -3
-37 -12 4  -37 12 -4  -37 12 4  37 -12 -4
-37 -12 5  -37 12 -5  -37 12 5  37 -12 -5
-37 -12 6  -37 12 -6  -37 12 6  37 -12 -6
-37 -12 7  -37 12 -7  -37 12 7  37 -12 -7
-37 -12 8  -37 12 -8  -37 12 8  37 -12 -8
-37 -12 9  -37 12 -9  -37 12 9  37 -12 -9
-37 -12 10  -37 12 -10  -37 12 10  37 -12 -10

-38 -12 1  -38 12 -1  -38 12 1  38 -12 -1
-38 -12 2  -38 12 -2  -38 12 2  38 -12 -2
-38 -12 3  -38 12 -3  -38 12 3  38 -12 -3
<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-38</td>
<td>-12</td>
<td>4</td>
<td>-38</td>
<td>12</td>
<td>-4</td>
<td>-38</td>
<td>12</td>
<td>4</td>
<td>38</td>
</tr>
<tr>
<td>-38</td>
<td>-12</td>
<td>5</td>
<td>-38</td>
<td>12</td>
<td>-5</td>
<td>-38</td>
<td>12</td>
<td>5</td>
<td>38</td>
</tr>
<tr>
<td>-38</td>
<td>-12</td>
<td>6</td>
<td>-38</td>
<td>12</td>
<td>-6</td>
<td>-38</td>
<td>12</td>
<td>6</td>
<td>38</td>
</tr>
<tr>
<td>-38</td>
<td>-12</td>
<td>7</td>
<td>-38</td>
<td>12</td>
<td>-7</td>
<td>-38</td>
<td>12</td>
<td>7</td>
<td>38</td>
</tr>
<tr>
<td>-38</td>
<td>-12</td>
<td>8</td>
<td>-38</td>
<td>12</td>
<td>-8</td>
<td>-38</td>
<td>12</td>
<td>8</td>
<td>38</td>
</tr>
<tr>
<td>-38</td>
<td>-12</td>
<td>9</td>
<td>-38</td>
<td>12</td>
<td>-9</td>
<td>-38</td>
<td>12</td>
<td>9</td>
<td>38</td>
</tr>
<tr>
<td>-38</td>
<td>-12</td>
<td>10</td>
<td>-38</td>
<td>12</td>
<td>-10</td>
<td>-38</td>
<td>12</td>
<td>10</td>
<td>38</td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-39</td>
<td>-12</td>
<td>1</td>
<td>-39</td>
<td>12</td>
<td>-1</td>
<td>-39</td>
<td>12</td>
<td>1</td>
<td>39</td>
</tr>
<tr>
<td>-39</td>
<td>-12</td>
<td>3</td>
<td>-39</td>
<td>12</td>
<td>-3</td>
<td>-39</td>
<td>12</td>
<td>3</td>
<td>39</td>
</tr>
<tr>
<td>-39</td>
<td>-12</td>
<td>4</td>
<td>-39</td>
<td>12</td>
<td>-4</td>
<td>-39</td>
<td>12</td>
<td>4</td>
<td>39</td>
</tr>
<tr>
<td>-39</td>
<td>-12</td>
<td>6</td>
<td>-39</td>
<td>12</td>
<td>-6</td>
<td>-39</td>
<td>12</td>
<td>6</td>
<td>39</td>
</tr>
<tr>
<td>-39</td>
<td>-12</td>
<td>7</td>
<td>-39</td>
<td>12</td>
<td>-7</td>
<td>-39</td>
<td>12</td>
<td>7</td>
<td>39</td>
</tr>
<tr>
<td>-39</td>
<td>-12</td>
<td>8</td>
<td>-39</td>
<td>12</td>
<td>-8</td>
<td>-39</td>
<td>12</td>
<td>8</td>
<td>39</td>
</tr>
<tr>
<td>-39</td>
<td>-12</td>
<td>10</td>
<td>-39</td>
<td>12</td>
<td>-10</td>
<td>-39</td>
<td>12</td>
<td>10</td>
<td>39</td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-40</td>
<td>-12</td>
<td>1</td>
<td>-40</td>
<td>12</td>
<td>-1</td>
<td>-40</td>
<td>12</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>-40</td>
<td>-12</td>
<td>2</td>
<td>-40</td>
<td>12</td>
<td>-2</td>
<td>-40</td>
<td>12</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>-40</td>
<td>-12</td>
<td>3</td>
<td>-40</td>
<td>12</td>
<td>-3</td>
<td>-40</td>
<td>12</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>-40</td>
<td>-12</td>
<td>4</td>
<td>-40</td>
<td>12</td>
<td>-4</td>
<td>-40</td>
<td>12</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>-40</td>
<td>-12</td>
<td>5</td>
<td>-40</td>
<td>12</td>
<td>-5</td>
<td>-40</td>
<td>12</td>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>-40</td>
<td>-12</td>
<td>6</td>
<td>-40</td>
<td>12</td>
<td>-6</td>
<td>-40</td>
<td>12</td>
<td>6</td>
<td>40</td>
</tr>
<tr>
<td>-40</td>
<td>-12</td>
<td>7</td>
<td>-40</td>
<td>12</td>
<td>-7</td>
<td>-40</td>
<td>12</td>
<td>7</td>
<td>40</td>
</tr>
<tr>
<td>-40</td>
<td>-12</td>
<td>8</td>
<td>-40</td>
<td>12</td>
<td>-8</td>
<td>-40</td>
<td>12</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>-40</td>
<td>-12</td>
<td>9</td>
<td>-40</td>
<td>12</td>
<td>-9</td>
<td>-40</td>
<td>12</td>
<td>9</td>
<td>40</td>
</tr>
<tr>
<td>-40</td>
<td>-12</td>
<td>10</td>
<td>-40</td>
<td>12</td>
<td>-10</td>
<td>-40</td>
<td>12</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-41</td>
<td>-13</td>
<td>1</td>
<td>-41</td>
<td>13</td>
<td>-1</td>
<td>-41</td>
<td>13</td>
<td>1</td>
<td>41</td>
</tr>
<tr>
<td>-41</td>
<td>-13</td>
<td>2</td>
<td>-41</td>
<td>13</td>
<td>-2</td>
<td>-41</td>
<td>13</td>
<td>2</td>
<td>41</td>
</tr>
<tr>
<td>-41</td>
<td>-13</td>
<td>3</td>
<td>-41</td>
<td>13</td>
<td>-3</td>
<td>-41</td>
<td>13</td>
<td>3</td>
<td>41</td>
</tr>
<tr>
<td>-41</td>
<td>-13</td>
<td>4</td>
<td>-41</td>
<td>13</td>
<td>-4</td>
<td>-41</td>
<td>13</td>
<td>4</td>
<td>41</td>
</tr>
<tr>
<td>-41</td>
<td>-13</td>
<td>5</td>
<td>-41</td>
<td>13</td>
<td>-5</td>
<td>-41</td>
<td>13</td>
<td>5</td>
<td>41</td>
</tr>
<tr>
<td>-41</td>
<td>-13</td>
<td>6</td>
<td>-41</td>
<td>13</td>
<td>-6</td>
<td>-41</td>
<td>13</td>
<td>6</td>
<td>41</td>
</tr>
<tr>
<td>-41</td>
<td>-13</td>
<td>7</td>
<td>-41</td>
<td>13</td>
<td>-7</td>
<td>-41</td>
<td>13</td>
<td>7</td>
<td>41</td>
</tr>
<tr>
<td>-41</td>
<td>-13</td>
<td>8</td>
<td>-41</td>
<td>13</td>
<td>-8</td>
<td>-41</td>
<td>13</td>
<td>8</td>
<td>41</td>
</tr>
<tr>
<td>-41</td>
<td>-13</td>
<td>9</td>
<td>-41</td>
<td>13</td>
<td>-9</td>
<td>-41</td>
<td>13</td>
<td>9</td>
<td>41</td>
</tr>
<tr>
<td>-41</td>
<td>-13</td>
<td>10</td>
<td>-41</td>
<td>13</td>
<td>-10</td>
<td>-41</td>
<td>13</td>
<td>10</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-45</td>
<td>-10</td>
<td>-45</td>
<td>-10</td>
<td>-45</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-46</td>
<td>-11</td>
<td>-46</td>
<td>-1</td>
<td>-46</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-46</td>
<td>-12</td>
<td>-46</td>
<td>-2</td>
<td>-46</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-46</td>
<td>-13</td>
<td>-46</td>
<td>-3</td>
<td>-46</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-46</td>
<td>-14</td>
<td>-46</td>
<td>-4</td>
<td>-46</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-46</td>
<td>-15</td>
<td>-46</td>
<td>-5</td>
<td>-46</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-46</td>
<td>-16</td>
<td>-46</td>
<td>-6</td>
<td>-46</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-46</td>
<td>-17</td>
<td>-46</td>
<td>-7</td>
<td>-46</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-46</td>
<td>-18</td>
<td>-46</td>
<td>-8</td>
<td>-46</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-46</td>
<td>-19</td>
<td>-46</td>
<td>-9</td>
<td>-46</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-46</td>
<td>-20</td>
<td>-46</td>
<td>-10</td>
<td>-46</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-47</td>
<td>-11</td>
<td>-47</td>
<td>-1</td>
<td>-47</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-47</td>
<td>-12</td>
<td>-47</td>
<td>-2</td>
<td>-47</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-47</td>
<td>-13</td>
<td>-47</td>
<td>-3</td>
<td>-47</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-47</td>
<td>-14</td>
<td>-47</td>
<td>-4</td>
<td>-47</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-47</td>
<td>-15</td>
<td>-47</td>
<td>-5</td>
<td>-47</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-47</td>
<td>-16</td>
<td>-47</td>
<td>-6</td>
<td>-47</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-47</td>
<td>-17</td>
<td>-47</td>
<td>-7</td>
<td>-47</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-47</td>
<td>-18</td>
<td>-47</td>
<td>-8</td>
<td>-47</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-47</td>
<td>-19</td>
<td>-47</td>
<td>-9</td>
<td>-47</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-47</td>
<td>-20</td>
<td>-47</td>
<td>-10</td>
<td>-47</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-48</td>
<td>-11</td>
<td>-48</td>
<td>-1</td>
<td>-48</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-48</td>
<td>-12</td>
<td>-48</td>
<td>-2</td>
<td>-48</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-48</td>
<td>-13</td>
<td>-48</td>
<td>-3</td>
<td>-48</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-48</td>
<td>-14</td>
<td>-48</td>
<td>-4</td>
<td>-48</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-48</td>
<td>-15</td>
<td>-48</td>
<td>-5</td>
<td>-48</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-48</td>
<td>-16</td>
<td>-48</td>
<td>-6</td>
<td>-48</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-48</td>
<td>-17</td>
<td>-48</td>
<td>-7</td>
<td>-48</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-48</td>
<td>-18</td>
<td>-48</td>
<td>-8</td>
<td>-48</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-48</td>
<td>-19</td>
<td>-48</td>
<td>-9</td>
<td>-48</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-48</td>
<td>-20</td>
<td>-48</td>
<td>-10</td>
<td>-48</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-49</td>
<td>-11</td>
<td>-49</td>
<td>-1</td>
<td>-49</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-49</td>
<td>-12</td>
<td>-49</td>
<td>-2</td>
<td>-49</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-49</td>
<td>-13</td>
<td>-49</td>
<td>-3</td>
<td>-49</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-49</td>
<td>-14</td>
<td>-49</td>
<td>-4</td>
<td>-49</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-49</td>
<td>-15</td>
<td>-49</td>
<td>-5</td>
<td>-49</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-49</td>
<td>-16</td>
<td>-49</td>
<td>-6</td>
<td>-49</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-49</td>
<td>-17</td>
<td>-49</td>
<td>-7</td>
<td>-49</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-49</td>
<td>-18</td>
<td>-49</td>
<td>-8</td>
<td>-49</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-49</td>
<td>-19</td>
<td>-49</td>
<td>-9</td>
<td>-49</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-49</td>
<td>-20</td>
<td>-49</td>
<td>-10</td>
<td>-49</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

82
-49 -13 8  -49 13 -8  -49 13 8  49 -13 -8
-49 -13 10 - 49 13 -10 - 49 13 10  49 -13 -10

-50 -13 1  -50 13 -1  -50 13 1  50 -13 -1
-50 -13 2  -50 13 -2  -50 13 2  50 -13 -2
-50 -13 3  -50 13 -3  -50 13 3  50 -13 -3
-50 -13 4  -50 13 -4  -50 13 4  50 -13 -4
-50 -13 5  -50 13 -5  -50 13 5  50 -13 -5
-50 -13 6  -50 13 -6  -50 13 6  50 -13 -6
-50 -13 7  -50 13 -7  -50 13 7  50 -13 -7
-50 -13 8  -50 13 -8  -50 13 8  50 -13 -8
-50 -13 9  -50 13 -9  -50 13 9  50 -13 -9
-50 -13 10 - 50 13 -10 - 50 13 10  50 -13 -10

-51 -14 1  -51 14 -1  -51 14 1  51 -14 -1
-51 -14 2  -51 14 -2  -51 14 2  51 -14 -2
-51 -14 3  -51 14 -3  -51 14 3  51 -14 -3
-51 -14 4  -51 14 -4  -51 14 4  51 -14 -4
-51 -14 5  -51 14 -5  -51 14 5  51 -14 -5
-51 -14 6  -51 14 -6  -51 14 6  51 -14 -6
-51 -14 7  -51 14 -7  -51 14 7  51 -14 -7
-51 -14 8  -51 14 -8  -51 14 8  51 -14 -8
-51 -14 9  -51 14 -9  -51 14 9  51 -14 -9
-51 -14 10 - 51 14 -10 - 51 14 10  51 -14 -10

-52 -14 1  -52 14 -1  -52 14 1  52 -14 -1
-52 -14 2  -52 14 -2  -52 14 2  52 -14 -2
-52 -14 3  -52 14 -3  -52 14 3  52 -14 -3
-52 -14 4  -52 14 -4  -52 14 4  52 -14 -4
-52 -14 5  -52 14 -5  -52 14 5  52 -14 -5
-52 -14 6  -52 14 -6  -52 14 6  52 -14 -6
-52 -14 7  -52 14 -7  -52 14 7  52 -14 -7
-52 -14 8  -52 14 -8  -52 14 8  52 -14 -8
-52 -14 9  -52 14 -9  -52 14 9  52 -14 -9
-52 -14 10 - 52 14 -10 - 52 14 10  52 -14 -10

-53 -14 1  -53 14 -1  -53 14 1  53 -14 -1
-53 -14 3  -53 14 -3  -53 14 3  53 -14 -3
-53 -14 4  -53 14 -4  -53 14 4  53 -14 -4
-53 -14 5  -53 14 -5  -53 14 5  53 -14 -5
-53 -14 6  -53 14 -6  -53 14 6  53 -14 -6 
-53 -14 7  -53 14 -7  -53 14 7  53 -14 -7 
-53 -14 8  -53 14 -8  -53 14 8  53 -14 -8 
-53 -14 10  -53 14 -10  -53 14 10  53 -14 -10 

-54 -14 1  -54 14 -1  -54 14 1  54 -14 -1 
-54 -14 2  -54 14 -2  -54 14 2  54 -14 -2 
-54 -14 3  -54 14 -3  -54 14 3  54 -14 -3 
-54 -14 4  -54 14 -4  -54 14 4  54 -14 -4 
-54 -14 5  -54 14 -5  -54 14 5  54 -14 -5 
-54 -14 6  -54 14 -6  -54 14 6  54 -14 -6 
-54 -14 7  -54 14 -7  -54 14 7  54 -14 -7 
-54 -14 8  -54 14 -8  -54 14 8  54 -14 -8 
-54 -14 9  -54 14 -9  -54 14 9  54 -14 -9 
-54 -14 10  -54 14 -10  -54 14 10  54 -14 -10 

-55 -14 1  -55 14 -1  -55 14 1  55 -14 -1 
-55 -14 2  -55 14 -2  -55 14 2  55 -14 -2 
-55 -14 3  -55 14 -3  -55 14 3  55 -14 -3 
-55 -14 4  -55 14 -4  -55 14 4  55 -14 -4 
-55 -14 5  -55 14 -5  -55 14 5  55 -14 -5 
-55 -14 6  -55 14 -6  -55 14 6  55 -14 
-55 -14 7  -55 14 -7  -55 14 7  55 -14 -7 
-55 -14 8  -55 14 -8  -55 14 8  55 -14 -8 
-55 -14 9  -55 14 -9  -55 14 9  55 -14 
-55 -14 10  -55 14 -10  -55 14 10  55 -14 -10 

-56 -14 1  -56 14 -1  -56 14 1  56 -14 -1 
-56 -14 2  -56 14 -2  -56 14 2  56 -14 -2 
-56 -14 3  -56 14 -3  -56 14 3  56 -14 -3 
-56 -14 4  -56 14 -4  -56 14 4  56 -14 -4 
-56 -14 5  -56 14 -5  -56 14 5  56 -14 -5 
-56 -14 6  -56 14 -6  -56 14 6  56 -14 -6 
-56 -14 7  -56 14 -7  -56 14 7  56 -14 -7 
-56 -14 8  -56 14 -8  -56 14 8  56 -14 -8 
-56 -14 9  -56 14 -9  -56 14 9  56 -14 -9 
-56 -14 10  -56 14 -10  -56 14 10  56 -14 -10 

-57 -14 1  -57 14 -1  -57 14 1  57 -14 -1 
-57 -14 2  -57 14 -2  -57 14 2  57 -14 -2 
-57 -14 3  -57 14 -3  -57 14 3  57 -14 -3 

84
\begin{align*}
-57 \cdot 14 & -57 \cdot 14 -4 & -57 \cdot 14 & 57 \cdot 14 -4 \\
-57 \cdot 14 & -57 \cdot 14 -5 & -57 \cdot 14 & 57 \cdot 14 -5 \\
-57 \cdot 14 & -57 \cdot 14 -6 & -57 \cdot 14 & 57 \cdot 14 -6 \\
-57 \cdot 14 & -57 \cdot 14 -7 & -57 \cdot 14 & 57 \cdot 14 -7 \\
-57 \cdot 14 & -57 \cdot 14 -8 & -57 \cdot 14 & 57 \cdot 14 -8 \\
-57 \cdot 14 & -57 \cdot 14 -9 & -57 \cdot 14 & 57 \cdot 14 -9 \\
-57 \cdot 14 & -57 \cdot 14 -10 & -57 \cdot 14 & 57 \cdot 14 -10 \\
-58 \cdot 14 & -58 \cdot 14 -1 & -58 \cdot 14 & 58 \cdot 14 -1 \\
-58 \cdot 14 & -58 \cdot 14 -2 & -58 \cdot 14 & 58 \cdot 14 -2 \\
-58 \cdot 14 & -58 \cdot 14 -3 & -58 \cdot 14 & 58 \cdot 14 -3 \\
-58 \cdot 14 & -58 \cdot 14 -4 & -58 \cdot 14 & 58 \cdot 14 -4 \\
-58 \cdot 14 & -58 \cdot 14 -5 & -58 \cdot 14 & 58 \cdot 14 -5 \\
-58 \cdot 14 & -58 \cdot 14 -6 & -58 \cdot 14 & 58 \cdot 14 -6 \\
-58 \cdot 14 & -58 \cdot 14 -7 & -58 \cdot 14 & 58 \cdot 14 -7 \\
-58 \cdot 14 & -58 \cdot 14 -8 & -58 \cdot 14 & 58 \cdot 14 -8 \\
-58 \cdot 14 & -58 \cdot 14 -9 & -58 \cdot 14 & 58 \cdot 14 -9 \\
-58 \cdot 14 & -58 \cdot 14 -10 & -58 \cdot 14 & 58 \cdot 14 -10 \\
-59 \cdot 14 & -59 \cdot 14 -1 & -59 \cdot 14 & 59 \cdot 14 -1 \\
-59 \cdot 14 & -59 \cdot 14 -2 & -59 \cdot 14 & 59 \cdot 14 -2 \\
-59 \cdot 14 & -59 \cdot 14 -3 & -59 \cdot 14 & 59 \cdot 14 -3 \\
-59 \cdot 14 & -59 \cdot 14 -4 & -59 \cdot 14 & 59 \cdot 14 -4 \\
-59 \cdot 14 & -59 \cdot 14 -5 & -59 \cdot 14 & 59 \cdot 14 -5 \\
-59 \cdot 14 & -59 \cdot 14 -6 & -59 \cdot 14 & 59 \cdot 14 -6 \\
-59 \cdot 14 & -59 \cdot 14 -7 & -59 \cdot 14 & 59 \cdot 14 -7 \\
-59 \cdot 14 & -59 \cdot 14 -8 & -59 \cdot 14 & 59 \cdot 14 -8 \\
-59 \cdot 14 & -59 \cdot 14 -9 & -59 \cdot 14 & 59 \cdot 14 -9 \\
-59 \cdot 14 & -59 \cdot 14 -10 & -59 \cdot 14 & 59 \cdot 4 -10 \\
-60 \cdot 14 & -60 \cdot 14 -1 & -60 \cdot 14 & 60 \cdot 14 -1 \\
-60 \cdot 14 & -60 \cdot 14 -2 & -60 \cdot 14 & 60 \cdot 14 -2 \\
-60 \cdot 14 & -60 \cdot 14 -3 & -60 \cdot 14 & 60 \cdot 14 -3 \\
-60 \cdot 14 & -60 \cdot 14 -4 & -60 \cdot 14 & 60 \cdot 14 -4 \\
-60 \cdot 14 & -60 \cdot 14 -5 & -60 \cdot 14 & 60 \cdot 14 -5 \\
-60 \cdot 14 & -60 \cdot 14 -6 & -60 \cdot 14 & 60 \cdot 14 -6 \\
-60 \cdot 14 & -60 \cdot 14 -7 & -60 \cdot 14 & 60 \cdot 14 -7 \\
-60 \cdot 14 & -60 \cdot 14 -8 & -60 \cdot 14 & 60 \cdot 14 -8 \\
-60 \cdot 14 & -60 \cdot 14 -9 & -60 \cdot 14 & 60 \cdot 14 -9 \\
-60 \cdot 14 & -60 \cdot 14 -10 & -60 \cdot 14 & 60 \cdot 14 -10
\end{align*}
| -61 | 15 | 1 | -61 | 15 | -61 | 15 | 1 | 61 | 15 | 1 |
| -61 | 15 | 2 | -61 | 15 | 2 | -61 | 15 | 2 | 61 | 15 | 2 |
| -61 | 15 | 3 | -61 | 15 | 3 | -61 | 15 | 3 | 61 | 15 | 3 |
| -61 | 15 | 4 | -61 | 15 | 4 | -61 | 15 | 4 | 61 | 15 | 4 |
| -61 | 15 | 5 | -61 | 15 | 5 | -61 | 15 | 5 | 61 | 15 | 5 |
| -61 | 15 | 6 | -61 | 15 | 6 | -61 | 15 | 6 | 61 | 15 | 6 |
| -61 | 15 | 7 | -61 | 15 | 7 | -61 | 15 | 7 | 61 | 15 | 7 |
| -61 | 15 | 8 | -61 | 15 | 8 | -61 | 15 | 8 | 61 | 15 | 8 |
| -61 | 15 | 9 | -61 | 15 | 9 | -61 | 15 | 9 | 61 | 15 | 9 |
| -61 | 15 | 10 | -61 | 15 | 10 | -61 | 15 | 10 | 61 | 15 | 10 |

| -62 | 15 | 1 | -62 | 15 | 1 | -62 | 15 | 1 | 62 | 15 | 1 |
| -62 | 15 | 2 | -62 | 15 | 2 | -62 | 15 | 2 | 62 | 15 | 2 |
| -62 | 15 | 4 | -62 | 15 | 4 | -62 | 15 | 4 | 62 | 15 | 4 |
| -62 | 15 | 5 | -62 | 15 | 5 | -62 | 15 | 5 | 62 | 15 | 5 |
| -62 | 15 | 6 | -62 | 15 | 6 | -62 | 15 | 6 | 62 | 15 | 6 |
| -62 | 15 | 7 | -62 | 15 | 7 | -62 | 15 | 7 | 62 | 15 | 7 |
| -62 | 15 | 8 | -62 | 15 | 8 | -62 | 15 | 8 | 62 | 15 | 8 |
| -62 | 15 | 10 | -62 | 15 | 10 | -62 | 15 | 10 | 62 | 15 | 10 |

| -63 | 15 | 1 | -63 | 15 | 1 | -63 | 15 | 1 | 63 | 15 | 1 |
| -63 | 15 | 2 | -63 | 15 | 2 | -63 | 15 | 2 | 63 | 15 | 2 |
| -63 | 15 | 3 | -63 | 15 | 3 | -63 | 15 | 3 | 63 | 15 | 3 |
| -63 | 15 | 4 | -63 | 15 | 4 | -63 | 15 | 4 | 63 | 15 | 4 |
| -63 | 15 | 5 | -63 | 15 | 5 | -63 | 15 | 5 | 63 | 15 | 5 |
| -63 | 15 | 6 | -63 | 15 | 6 | -63 | 15 | 6 | 63 | 15 | 6 |
| -63 | 15 | 7 | -63 | 15 | 7 | -63 | 15 | 7 | 63 | 15 | 7 |
| -63 | 15 | 8 | -63 | 15 | 8 | -63 | 15 | 8 | 63 | 15 | 8 |
| -63 | 15 | 9 | -63 | 15 | 9 | -63 | 15 | 9 | 63 | 15 | 9 |
| -63 | 15 | 10 | -63 | 15 | 10 | -63 | 15 | 10 | 63 | 15 | 10 |

| -64 | 15 | 1 | -64 | 15 | 1 | -64 | 15 | 1 | 64 | 15 | 1 |
| -64 | 15 | 2 | -64 | 15 | 2 | -64 | 15 | 2 | 64 | 15 | 2 |
| -64 | 15 | 3 | -64 | 15 | 3 | -64 | 15 | 3 | 64 | 15 | 3 |
| -64 | 15 | 4 | -64 | 15 | 4 | -64 | 15 | 4 | 64 | 15 | 4 |
| -64 | 15 | 5 | -64 | 15 | 5 | -64 | 15 | 5 | 64 | 15 | 5 |
| -64 | 15 | 6 | -64 | 15 | 6 | -64 | 15 | 6 | 64 | 15 | 6 |
| -64 | 15 | 7 | -64 | 15 | 7 | -64 | 15 | 7 | 64 | 15 | 7 |
| -64 | 15 | 8 | -64 | 15 | 8 | -64 | 15 | 8 | 64 | 15 | 8 |
| -64 | 15 | 9 | -64 | 15 | 9 | -64 | 15 | 9 | 64 | 15 | 9 |

86
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-64</td>
<td>-15</td>
<td>10</td>
<td>-64</td>
<td>15</td>
<td>-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-65</td>
<td>-15</td>
<td>1</td>
<td>-65</td>
<td>15</td>
<td>-1</td>
</tr>
<tr>
<td>-65</td>
<td>-15</td>
<td>2</td>
<td>-65</td>
<td>15</td>
<td>-2</td>
</tr>
<tr>
<td>-65</td>
<td>-15</td>
<td>3</td>
<td>-65</td>
<td>15</td>
<td>-3</td>
</tr>
<tr>
<td>-65</td>
<td>-15</td>
<td>4</td>
<td>-65</td>
<td>15</td>
<td>-4</td>
</tr>
<tr>
<td>-65</td>
<td>-15</td>
<td>5</td>
<td>-65</td>
<td>15</td>
<td>-5</td>
</tr>
<tr>
<td>-65</td>
<td>-15</td>
<td>6</td>
<td>-65</td>
<td>15</td>
<td>-6</td>
</tr>
<tr>
<td>-65</td>
<td>-15</td>
<td>7</td>
<td>-65</td>
<td>15</td>
<td>-7</td>
</tr>
<tr>
<td>-65</td>
<td>-15</td>
<td>8</td>
<td>-65</td>
<td>15</td>
<td>-8</td>
</tr>
<tr>
<td>-65</td>
<td>-15</td>
<td>9</td>
<td>-65</td>
<td>15</td>
<td>-9</td>
</tr>
<tr>
<td>-65</td>
<td>-15</td>
<td>10</td>
<td>-65</td>
<td>15</td>
<td>-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-66</td>
<td>-15</td>
<td>1</td>
<td>-66</td>
<td>15</td>
<td>-1</td>
</tr>
<tr>
<td>-66</td>
<td>-15</td>
<td>2</td>
<td>-66</td>
<td>15</td>
<td>-2</td>
</tr>
<tr>
<td>-66</td>
<td>-15</td>
<td>3</td>
<td>-66</td>
<td>15</td>
<td>-3</td>
</tr>
<tr>
<td>-66</td>
<td>-15</td>
<td>4</td>
<td>-66</td>
<td>15</td>
<td>-4</td>
</tr>
<tr>
<td>-66</td>
<td>-15</td>
<td>5</td>
<td>-66</td>
<td>15</td>
<td>-5</td>
</tr>
<tr>
<td>-66</td>
<td>-15</td>
<td>6</td>
<td>-66</td>
<td>15</td>
<td>-6</td>
</tr>
<tr>
<td>-66</td>
<td>-15</td>
<td>7</td>
<td>-66</td>
<td>15</td>
<td>-7</td>
</tr>
<tr>
<td>-66</td>
<td>-15</td>
<td>8</td>
<td>-66</td>
<td>15</td>
<td>-8</td>
</tr>
<tr>
<td>-66</td>
<td>-15</td>
<td>9</td>
<td>-66</td>
<td>15</td>
<td>-9</td>
</tr>
<tr>
<td>-66</td>
<td>-15</td>
<td>10</td>
<td>-66</td>
<td>15</td>
<td>-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-67</td>
<td>-15</td>
<td>1</td>
<td>-67</td>
<td>15</td>
<td>-1</td>
</tr>
<tr>
<td>-67</td>
<td>-15</td>
<td>2</td>
<td>-67</td>
<td>15</td>
<td>-2</td>
</tr>
<tr>
<td>-67</td>
<td>-15</td>
<td>3</td>
<td>-67</td>
<td>15</td>
<td>-3</td>
</tr>
<tr>
<td>-67</td>
<td>-15</td>
<td>4</td>
<td>-67</td>
<td>15</td>
<td>-4</td>
</tr>
<tr>
<td>-67</td>
<td>-15</td>
<td>5</td>
<td>-67</td>
<td>15</td>
<td>-5</td>
</tr>
<tr>
<td>-67</td>
<td>-15</td>
<td>6</td>
<td>-67</td>
<td>15</td>
<td>-6</td>
</tr>
<tr>
<td>-67</td>
<td>-15</td>
<td>7</td>
<td>-67</td>
<td>15</td>
<td>-7</td>
</tr>
<tr>
<td>-67</td>
<td>-15</td>
<td>8</td>
<td>-67</td>
<td>15</td>
<td>-8</td>
</tr>
<tr>
<td>-67</td>
<td>-15</td>
<td>9</td>
<td>-67</td>
<td>15</td>
<td>-9</td>
</tr>
<tr>
<td>-67</td>
<td>-15</td>
<td>10</td>
<td>-67</td>
<td>15</td>
<td>-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-68</td>
<td>-15</td>
<td>1</td>
<td>-68</td>
<td>15</td>
<td>-1</td>
</tr>
<tr>
<td>-68</td>
<td>-15</td>
<td>2</td>
<td>-68</td>
<td>15</td>
<td>-2</td>
</tr>
<tr>
<td>-68</td>
<td>-15</td>
<td>3</td>
<td>-68</td>
<td>15</td>
<td>-3</td>
</tr>
<tr>
<td>-68</td>
<td>-15</td>
<td>4</td>
<td>-68</td>
<td>15</td>
<td>-4</td>
</tr>
<tr>
<td>-68</td>
<td>-15</td>
<td>5</td>
<td>-68</td>
<td>15</td>
<td>-5</td>
</tr>
<tr>
<td>-68</td>
<td>-15</td>
<td>6</td>
<td>-68</td>
<td>15</td>
<td>-6</td>
</tr>
<tr>
<td>-68</td>
<td>-15</td>
<td>7</td>
<td>-68</td>
<td>15</td>
<td>-7</td>
</tr>
</tbody>
</table>

87
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-68</td>
<td>-15</td>
<td>8</td>
<td>-68</td>
<td>15</td>
<td>-8</td>
</tr>
<tr>
<td>-68</td>
<td>-15</td>
<td>9</td>
<td>-68</td>
<td>15</td>
<td>-9</td>
</tr>
<tr>
<td>-68</td>
<td>-15</td>
<td>10</td>
<td>-68</td>
<td>15</td>
<td>-10</td>
</tr>
<tr>
<td>-69</td>
<td>-15</td>
<td>1</td>
<td>-69</td>
<td>15</td>
<td>-1</td>
</tr>
<tr>
<td>-69</td>
<td>-15</td>
<td>2</td>
<td>-69</td>
<td>15</td>
<td>-2</td>
</tr>
<tr>
<td>-69</td>
<td>-15</td>
<td>3</td>
<td>-69</td>
<td>15</td>
<td>-3</td>
</tr>
<tr>
<td>-69</td>
<td>-15</td>
<td>4</td>
<td>-69</td>
<td>15</td>
<td>-4</td>
</tr>
<tr>
<td>-69</td>
<td>-15</td>
<td>5</td>
<td>-69</td>
<td>15</td>
<td>-5</td>
</tr>
<tr>
<td>-69</td>
<td>-15</td>
<td>6</td>
<td>-69</td>
<td>15</td>
<td>-6</td>
</tr>
<tr>
<td>-69</td>
<td>-15</td>
<td>7</td>
<td>-69</td>
<td>15</td>
<td>-7</td>
</tr>
<tr>
<td>-69</td>
<td>-15</td>
<td>8</td>
<td>-69</td>
<td>15</td>
<td>-8</td>
</tr>
<tr>
<td>-69</td>
<td>-15</td>
<td>9</td>
<td>-69</td>
<td>15</td>
<td>-9</td>
</tr>
<tr>
<td>-69</td>
<td>-15</td>
<td>10</td>
<td>-69</td>
<td>15</td>
<td>-10</td>
</tr>
<tr>
<td>-70</td>
<td>-15</td>
<td>1</td>
<td>-70</td>
<td>15</td>
<td>-1</td>
</tr>
<tr>
<td>-70</td>
<td>-15</td>
<td>2</td>
<td>-70</td>
<td>15</td>
<td>-2</td>
</tr>
<tr>
<td>-70</td>
<td>-15</td>
<td>3</td>
<td>-70</td>
<td>15</td>
<td>-3</td>
</tr>
<tr>
<td>-70</td>
<td>-15</td>
<td>4</td>
<td>-70</td>
<td>15</td>
<td>-4</td>
</tr>
<tr>
<td>-70</td>
<td>-15</td>
<td>5</td>
<td>-70</td>
<td>15</td>
<td>-5</td>
</tr>
<tr>
<td>-70</td>
<td>-15</td>
<td>6</td>
<td>-70</td>
<td>15</td>
<td>-6</td>
</tr>
<tr>
<td>-70</td>
<td>-15</td>
<td>7</td>
<td>-70</td>
<td>15</td>
<td>-7</td>
</tr>
<tr>
<td>-70</td>
<td>-15</td>
<td>8</td>
<td>-70</td>
<td>15</td>
<td>-8</td>
</tr>
<tr>
<td>-70</td>
<td>-15</td>
<td>9</td>
<td>-70</td>
<td>15</td>
<td>-9</td>
</tr>
<tr>
<td>-70</td>
<td>-15</td>
<td>10</td>
<td>-70</td>
<td>15</td>
<td>-10</td>
</tr>
<tr>
<td>-71</td>
<td>-15</td>
<td>1</td>
<td>-71</td>
<td>16</td>
<td>-1</td>
</tr>
<tr>
<td>-71</td>
<td>-15</td>
<td>2</td>
<td>-71</td>
<td>16</td>
<td>-2</td>
</tr>
<tr>
<td>-71</td>
<td>-15</td>
<td>3</td>
<td>-71</td>
<td>16</td>
<td>-3</td>
</tr>
<tr>
<td>-71</td>
<td>-15</td>
<td>4</td>
<td>-71</td>
<td>16</td>
<td>-4</td>
</tr>
<tr>
<td>-71</td>
<td>-15</td>
<td>5</td>
<td>-71</td>
<td>16</td>
<td>-5</td>
</tr>
<tr>
<td>-71</td>
<td>-15</td>
<td>6</td>
<td>-71</td>
<td>16</td>
<td>-6</td>
</tr>
<tr>
<td>-71</td>
<td>-15</td>
<td>7</td>
<td>-71</td>
<td>16</td>
<td>-7</td>
</tr>
<tr>
<td>-71</td>
<td>-15</td>
<td>8</td>
<td>-71</td>
<td>16</td>
<td>-8</td>
</tr>
<tr>
<td>-71</td>
<td>-15</td>
<td>9</td>
<td>-71</td>
<td>16</td>
<td>-9</td>
</tr>
<tr>
<td>-71</td>
<td>-15</td>
<td>10</td>
<td>-71</td>
<td>16</td>
<td>-10</td>
</tr>
<tr>
<td>-72</td>
<td>-15</td>
<td>1</td>
<td>-72</td>
<td>16</td>
<td>-1</td>
</tr>
<tr>
<td>-72</td>
<td>-15</td>
<td>2</td>
<td>-72</td>
<td>16</td>
<td>-2</td>
</tr>
<tr>
<td>-72</td>
<td>-15</td>
<td>3</td>
<td>-72</td>
<td>16</td>
<td>-3</td>
</tr>
<tr>
<td>-72</td>
<td>-15</td>
<td>4</td>
<td>-72</td>
<td>16</td>
<td>-4</td>
</tr>
<tr>
<td>-72</td>
<td>-15</td>
<td>5</td>
<td>-72</td>
<td>16</td>
<td>-5</td>
</tr>
</tbody>
</table>
-72 -166 -7216 -6 -7216 6 72 -16 -6
-72 -167 -7216 -7 -7216 7 72 -16 -7
-72 -168 -7216 -8 -7216 8 72 -16 -8
-72 -169 -7216 -9 -7216 9 72 -16 -9
-72 -16 10 -7216 -10 -7216 10 72 -16 -10

-73 -161 -7316 -1 -7316 1 73 -16 -1
-73 -162 -7316 -2 -7316 2 73 -16 -2
-73 -163 -7316 -3 -7316 3 73 -16 -3
-73 -164 -7316 -4 -7316 4 73 -16 -4
-73 -165 -7316 -5 -7316 5 73 -16 -5
-73 -166 -7316 -6 -7316 6 73 -16 -6
-73 -167 -7316 -7 -7316 7 73 -16 -7
-73 -168 -7316 -8 -7316 8 73 -16 -8
-73 -169 -7316 -9 -7316 9 73 -16 -9
-73 -16 10 -7316 -10 -7316 10 73 -16 -10

-74 -161 -7416 -1 -7416 1 74 -16 -1
-74 -162 -7416 -2 -7416 2 74 -16 -2
-74 -163 -7416 -3 -7416 3 74 -16 -3
-74 -164 -7416 -4 -7416 4 74 -16 -4
-74 -165 -7416 -5 -7416 5 74 -16 -5
-74 -166 -7416 -6 -7416 6 74 -16 -6
-74 -167 -7416 -7 -7416 7 74 -16 -7
-74 -168 -7416 -8 -7416 8 74 -16 -8
-74 -169 -7416 -9 -7416 9 74 -16 -9
-74 -16 10 -7416 -10 -7416 10 74 -16 -10

-75 -161 -7516 -1 -7516 1 75 -16 -1
-75 -162 -7516 -2 -7516 2 75 -16 -2
-75 -163 -7516 -3 -7516 3 75 -16 -3
-75 -164 -7516 -4 -7516 4 75 -16 -4
-75 -165 -7516 -5 -7516 5 75 -16 -5
-75 -166 -7516 -6 -7516 6 75 -16 -6
-75 -167 -7516 -7 -7516 7 75 -16 -7
-75 -168 -7516 -8 -7516 8 75 -16 -8
-75 -169 -7516 -9 -7516 9 75 -16 -9
-75 -16 10 -7516 -10 -7516 10 75 -16 -10

-76 -161 -7616 -1 -7616 1 76 -16 -1
-76 -162 -7616 -2 -7616 2 76 -16 -2
-76 -163 -7616 -3 -7616 3 76 -16 -3
| -76 | -16 | -4 | -76 | -16 | -4 | 76 | -16 | -4 |
| -76 | -16 | -5 | -76 | -16 | -5 | 76 | -16 | -5 |
| -76 | -16 | -6 | -76 | -16 | -6 | 76 | -16 | -6 |
| -76 | -16 | -7 | -76 | -16 | -7 | 76 | -16 | -7 |
| -76 | -16 | -8 | -76 | -16 | -8 | 76 | -16 | -8 |
| -76 | -16 | -9 | -76 | -16 | -9 | 76 | -16 | -9 |
| -76 | -16 | -10 | -76 | -16 | -10 | 76 | -16 | -10 |

| -77 | -16 | -1 | -77 | -16 | -1 | 77 | -16 | -1 |
| -77 | -16 | -2 | -77 | -16 | -2 | 77 | -16 | -2 |
| -77 | -16 | -3 | -77 | -16 | -3 | 77 | -16 | -3 |
| -77 | -16 | -4 | -77 | -16 | -4 | 77 | -16 | -4 |
| -77 | -16 | -5 | -77 | -16 | -5 | 77 | -16 | -5 |
| -77 | -16 | -6 | -77 | -16 | -6 | 77 | -16 | -6 |
| -77 | -16 | -7 | -77 | -16 | -7 | 77 | -16 | -7 |
| -77 | -16 | -8 | -77 | -16 | -8 | 77 | -16 | -8 |
| -77 | -16 | -9 | -77 | -16 | -9 | 77 | -16 | -9 |
| -77 | -16 | -10 | -77 | -16 | -10 | 77 | -16 | -10 |

| -78 | -16 | -1 | -78 | -16 | -1 | 78 | -16 | -1 |
| -78 | -16 | -2 | -78 | -16 | -2 | 78 | -16 | -2 |
| -78 | -16 | -3 | -78 | -16 | -3 | 78 | -16 | -3 |
| -78 | -16 | -4 | -78 | -16 | -4 | 78 | -16 | -4 |
| -78 | -16 | -5 | -78 | -16 | -5 | 78 | -16 | -5 |
| -78 | -16 | -6 | -78 | -16 | -6 | 78 | -16 | -6 |
| -78 | -16 | -7 | -78 | -16 | -7 | 78 | -16 | -7 |
| -78 | -16 | -8 | -78 | -16 | -8 | 78 | -16 | -8 |
| -78 | -16 | -9 | -78 | -16 | -9 | 78 | -16 | -9 |
| -78 | -16 | -10 | -78 | -16 | -10 | 78 | -16 | -10 |

| -79 | -16 | -1 | -79 | -16 | -1 | 79 | -16 | -1 |
| -79 | -16 | -2 | -79 | -16 | -2 | 79 | -16 | -2 |
| -79 | -16 | -3 | -79 | -16 | -3 | 79 | -16 | -3 |
| -79 | -16 | -4 | -79 | -16 | -4 | 79 | -16 | -4 |
| -79 | -16 | -5 | -79 | -16 | -5 | 79 | -16 | -5 |
| -79 | -16 | -6 | -79 | -16 | -6 | 79 | -16 | -6 |
| -79 | -16 | -7 | -79 | -16 | -7 | 79 | -16 | -7 |
| -79 | -16 | -8 | -79 | -16 | -8 | 79 | -16 | -8 |
| -79 | -16 | -9 | -79 | -16 | -9 | 79 | -16 | -9 |
| -79 | -16 | -10 | -79 | -16 | -10 | 79 | -16 | -10 |

90
# Code for SAT solver

#include "Solver.h"
#include "Sort.h"
#include <cmath>

// Debug:

// For derivation output (verbosity level 2)
#define L_IND    "%-#d"
#define L_ind    decisionLevel()*3+3,decisionLevel()
#define L_LIT    "%sx%d"
#define L_lit(p) sign(p)?"~":"," var(p)

// Just like 'assert()' but expression will be evaluated in the release version as well.
inline void check(bool expr) { assert(expr); }

// Minor methods:
// Creates a new SAT variable in the solver. If “decision_var” is cleared, variable will not be used as a decision variable
(NOTE! This has effects on the meaning of a SATISFIABLE result).

Var Solver::newVar(bool decision_var)
{
    int index;
    index = nVars();
    watches .push();      // (list for positive literal)
    watches .push();      // (list for negative literal)
    undos .push();
    reason .push(NULL);
    assigns .push(toInt(l_Undef));
    level .push(-1);
    activity.push(0);
    order .newVar();
    return index;
}

// Returns FALSE if immediate conflict.
bool Solver::assume(Lit p)
{
    assert(propQ.size() == 0);
    if (verbosity >= 2) printf(L_IND"assume("L_LIT")\n", L_ind, L_lit(p));
    trail_lim.push(trail.size());
    return enqueue(p); }

// Revert one variable binding on the trail.
inline void Solver::undoOne(void)
{
    if (verbosity >= 2){ Lit p = trail.last(); printf(L_IND"unbind("L_LIT")\n", L_ind, L_lit(p));
    Lit p = trail.last();
    Var x = var(p);
    assigns[x] = toInt(l_Undef);
    reason [x] = NULL;
    level [x] = -1;
    order.undo(x);
    trail.pop();
    while (undos[x].size() > 0)
undos[x].last()->undo(*this, p),
undos[x].pop();
}

// Reverts to the state before last 'assume()'.
void Solver::cancel(void)
{
    assert(propQ.size() == 0);
    if (verbosity >= 2) {
        if (trail.size() != trail_lim.last())
            { Lit p = trail[trail_lim.last()];
                printf(L_IND"cancel("L_LIT")\n", L_ind,
                L_lit(p));
            }
        for (int c = trail.size() - trail_lim.last(); c != 0; c--)
            undoOne();
        trail_lim.pop();
    }
}

// Revert to the state at given level.//
void Solver::cancelUntil(int level)
{
    while (decisionLevel() > level) cancel();
}

// Record a clause and drive backtracking. 'clause[0]' must contain the asserting
literal.//
void Solver::record(const vec<Lit>& clause)
{
    assert(clause.size() != 0);
    Clause* c;
    check(Clause_new(*this, clause, true, c));
    check(enqueue(clause[0], c));
    if (c != NULL) learnts.push(c);
}

//=======================================================
// Major methods:

analyze : (confl : Constr*) (out_learnt : vec<Lit>&) (out_btlevel : int&) -> [void]

Description:
Analyze conflict and produce a reason clause.
Pre-conditions:
* 'out_learnt' is assumed to be cleared.
* Current decision level must be greater than root level.

Post-conditions:
* 'out_learnt[0]' is the asserting literal at level 'out_btlevel'.

Effect:
Will undo part of the trail, upto but not beyond the assumption of the current
decision level.

```cpp
void Solver::analyze(Constr* confl, vec<Lit>& out_learnt, int& out_btlevel)
{
    vec<char>& seen = analyze_seen;
    int pathC = 0;
    Lit p = lit_Undef;
    vec<Lit> p_reason;
    seen.growTo(nVars(), 0);

    // Generate conflict clause:
    out_learnt.push();  // (leave room for the asserting literal)
    out_btlevel = 0;
    do
    {
        assert(confl != NULL);  // (otherwise should be UIP)
        p_reason.clear();
        confl->calcReason(*this, p, p_reason);

        for (int j = 0; j < p_reason.size(); j++)
        {
            Lit q = p_reason[j];
            if (!seen[var(q)])
            {
                seen[var(q)] = 1;
                if (level[var(q)] == decisionLevel())
                    pathC++;
            }
            else
            {
                if (level[var(q)] > 0)  // (exclude variables from decision level 0)
```
out_learnt.push(~q),
    out_btlevel = max(out_btlevel, level[var(q)]);
}
}

// Select next clause to look at:
do{
    p = trail.last();
    confl = reason[var(p)];
    undoOne();
}while (!seen[var(p)]);
    pathC--;
    seen[var(p)] = 0;
}while (pathC > 0);
    out_learnt[0] = ~p;

for (int j = 0; j < out_learnt.size(); j++) seen[var(out_learnt[j])] = 0;
    // (seen[] is now cleared)

if (verbosity >= 2) {
    printf(L_IND"Learnt {", L_ind);
    for (int i = 0; i < out_learnt.size(); i++) printf(" L_LIT,
L_lit(out_learnt[i]));
    printf(" } at level %d\n", out_btlevel); }

enqueue : (p : Lit) (from : Constr*) -> [bool]

Description:
    Puts a new fact on the propagation queue as well as immediately updating the
    variable’s value.
    Should a conflict arise, FALSE is returned.

Input:
    p - The fact to enqueue from - [Optional] Fact propagated from this
        (currently) unit clause. Stored in ‘reason[]’.
    Default value is NULL (no reason).
bool Solver::enqueue(Lit p, Constr* from)
{
    if (value(p) != l_Undef)
    {
        if (value(p) == l_False)
        {
            // Conflicting enqueued assignment
            return false;
        }
        else
        {
            // Existing consistent assignment -- don't enqueue
            return true;
        }
    }
    else
    {
        // New fact -- store it.
        if (verbosity >= 2) printf(L_IND"bind("L_LIT")\n", L_ind, L_lit(p));
        assigns[var(p)] = toInt(lbool(!sign(p)));
        level [var(p)] = decisionLevel();
        reason [var(p)] = (decisionLevel() != 0) ? from : NULL;
        // Paverkar det nagot?
        trail.push(p);
        propQ.insert(p);
        return true;
    }
}

propagate : [void] -> [Constr*]

Description:
Propagates all enqueued facts. If a conflict arises, the conflicting clause is returned, otherwise NULL.

Post-conditions:
Constr* Solver::propagate(void)
{
    while (propQ.size() > 0)
    {
        stats.propagations++;
        Lit p = propQ.dequeue();                // 'p' is enqueued fact to propagate
        vec<Constr*>& tmp = propagate_tmp;      // watch list for 'p' will be emptied, 'tmp' will store its state before that
        watches[index(p)].copyTo(tmp); watches[index(p)].clear();

        for (int i = 0; i < tmp.size(); i++)
        {
            stats.inspects++;
            if (!tmp[i]->propagate(*this, p))
            {
                // Constraint is conflicting -- copy remaining watches to 'watches[p]' and return constraint:
                for (int j = i+1; j < tmp.size(); j++) watches[index(p)].push(tmp[j]);
                // Clear the propagation queue:
                propQ.clear();
                return tmp[i];
            }
        }
    }
}

return NULL;

/*
reduceDB : () -> [void]
Description:
    Remove half of the learnt clauses, minus the clauses locked by the current assignment. Locked clauses are clauses that are reason to a some assignment.
*/

struct reduceDB_lt { bool operator () (Clause* x, Clause* y) { return x->activity() < y->activity(); } ;
void Solver::reduceDB(void)
int i, j;
double extra_lim = cla_inc / learnts.size();
   // Remove any clause below this activity

sort(learnts, reduceDB_lt());
for (i = j = 0; i < learnts.size() / 2; i++)
{
    if (!learnts[i] -> locked(*this))
        learnts[i] -> remove(*this);
    else
        learnts[j++] = learnts[i];
}
for (; i < learnts.size(); i++)
{
    if (!learnts[i] -> locked(*this) && learnts[i] -> activity() < extra_lim)
        learnts[i] -> remove(*this);
    else
        learnts[j++] = learnts[i];
}
learnts.shrink(i - j);

/*
* ________________________________________________________________________

simplifyDB : [void] -> [bool]

Description:
   Simplify all constraints according to the current top-level assignment (redundant
   constraints may be removed altogether).
__________________________________________________________________________ */

bool Solver::simplifyDB(void)
{
    assert(decisionLevel() == 0);

    if (propagate() != NULL)
        return false;

    if (nAssigns() == last_simplify)
        return true;

    return false;
}
last_simplify = nAssigns();

for (int type = 0; type < 2; type++)
{
    vec<Constr*>& cs = type ? (vec<Constr*>&)learnts : constrs;
    int     j = 0;
    for (int i = 0; i < cs.size(); i++)
    {
        if (cs[i]->simplify(*this))
            cs[i]->remove(*this);
        else
            cs[j++] = cs[i];
    }
    cs.shrink(cs.size()-j);
}
return true;

/*________________________________________________________

search : (nof_conflicts : int) (nof_learnts : int) (params : const SearchParams&) - > [lbool]

Description:
Search for a model the specified number of conflicts, keeping the number of learnt clauses below the provided limit. NOTE! Use negative value for 'nof_conflicts' or 'nof_learnts' to indicate infinity.

Output:
'l_True' if a partial assignment that is consistent with respect to the clauseset is found. If all variables are decision variables, this means that the clause set is satisfiable. 'l_False'
if the clause set is unsatisfiable. 'l_Undef' if the bound on number of conflicts is reached.
________________________________________________________*/

lbool Solver::search(int nof_conflicts, int nof_learnts, const SearchParams& params)
{
    assert(root_level == decisionLevel());
    stats.starts++;
    int conflictC = 0;
    var_decay = 1 / params.var_decay;

cla_decay = 1 / params.clause_decay;
model.clear();

for (; ;)
{
    Constr* confl = propagate();
if (confl != NULL)
{
    // CONFLICT
    if (verbosity >= 2) printf(L_IND"**CONFLICT**\n", L_ind);
    stats.conflicts++; conflictC++;
    vec<Lit> learnt_clause;
    int backtrack_level;

    if (decisionLevel() == root_level)
        return l_False;
    analyze(confl, learnt_clause, backtrack_level);
    cancelUntil(max(backtrack_level, root_level));
    record(learnt_clause);
    varDecayActivity(); claDecayActivity();
}
else
{
    // NO CONFLICT
    if (nof_conflicts >= 0 && conflictC >= nof_conflicts)
    {
        // Reached bound on number of conflicts:
        progress_estimate = progressEstimate();
        propQ.clear();
        cancelUntil(root_level);
        return l_Undef; }

    if (decisionLevel() == 0)
        // Simplify the set of problem clauses:
        check(simplifyDB());

    if (nof_learnts >= 0 && learnts.size()-nAssigns() >= nof_learnts)
        // Reduce the set of learnt clauses:
        reduceDB();

    // New variable decision:
    stats.decisions++;
Var next = order.select(params.random_var_freq);

if (next == var_Undef)
{
    // Model found:
    model.growTo(nVars());
    for (int i = 0; i < nVars(); i++) model[i] = (value(i) == l_True);
    cancelUntil(root_level);
    return l_True;

    check(assume(~Lit(next)));
}
}

// Return search-space coverage. Not extremely reliable.
double Solver::progressEstimate(void)
{
    double progress = 0;
    double F = 1.0 / nVars();
    for (int i = 0; i < nVars(); i++)
        if (value(i) != l_Undef)
            progress += pow(F, level[i]);
    return progress / nVars();
}

// Divide all variable activities by 1e100.
void Solver::varRescaleActivity(void)
{
    for (int i = 0; i < nVars(); i++)
        activity[i] *= 1e-100;
    var_inc *= 1e-100;
}

// Divide all constraint activities by 1e100.
void Solver::claRescaleActivity(void)
{
    for (int i = 0; i < learnts.size(); i++)
        learnts[i]->activity() *= 1e-20;
    cla_inc *= 1e-20;
}
solve : (assumps : const vec<Lit>&) -> [bool]

Description:
Top-level solve. If using assumptions (non-empty 'assumps' vector), you must call 'simplifyDB()' first to see that no top-level conflict is present (which would put the solver in an undefined state).

bool Solver::solve(const vec<Lit>& assumps)
{
    SearchParams params(0.95, 0.999, 0.02);
    double nof_conflicts = 100;
    double nof_learnts   = nConstrs()/3;
    lbool status        = l_Undef;

    for (int i = 0; i < assumps.size(); i++)
        if (!assume(assumps[i]) || propagate() != NULL)
        {
            propQ.clear();
            cancelUntil(0);
            return false;
        }
    root_level = decisionLevel();

    while (status == l_Undef)
    {
        if (verbosity >= 1) printf("Solving -- conflicts=%d    learnts=%d
progress=%.4f  \n", (int)nof_conflicts, (int)nof_learnts,
progress_estimate*100);
        status = search((int)nof_conflicts, (int)nof_learnts, params);
        nof_conflicts *= 1.5;
        nof_learnts   *= 1.1;
    }

    cancelUntil(0);
    return status == l_True;
}
6.8 Example of Attack on RSA

Here we will group the characters into blocks of three and compute a message representative integer for each block.

ATTACK*AT*SEVEN= ATT ACK * AT *SEVEN

In the same way that a decimal number can be represented as the sum of powers of ten,

e.g. \( 135 = 1 \times 10^2 + 3 \times 10^1 + 5 \)

We could represent our blocks of three characters in base 26 using A=0, B=1, C=2, ..., Z=25

\[
\begin{align*}
ATT &= 0 \times 26^2 + 19 \times 26^1 + 19 = 513 \\
ACK &= 0 \times 26^2 + 2 \times 26^1 + 10 = 62 \\
XAT &= 23 \times 26^2 + 0 \times 26^1 + 19 = 15567 \\
XSE &= 23 \times 26^2 + 18 \times 26^1 + 4 = 16020 \\
VEN &= 21 \times 26^2 + 4 \times 26^1 + 13 = 14313
\end{align*}
\]

For this example, to keep things simple, we'll not worry about numbers and punctuation characters, or what happens with groups AAA or AAB.

In this system of encoding, the maximum value of a group (ZZZ) would be \( 26^3 - 1 = 17575 \), so we require a modulus \( n \) greater than this value.

1. We "generate" primes \( p=137 \) and \( q=131 \)
2. \( n = p.q = 137.131 = 17947 \)
   \( \phi = (p-1)(q-1) = 136.130 = 17680 \)
3. Select \( e = 3 \)
   check \( \gcd(e, p-1) = \gcd(3, 136) = 1 \), and
   check \( \gcd(e, q-1) = \gcd(3, 130) = 1 \)
4. Compute \( d = e^{-1} \text{ mod } \phi = 3^{-1} \text{ mod } 17680 = 11787 \)
5. Hence Public Key, \( (n, e) = (17947, 3) \)
   Private Key \( (n, d) = (17947, 11787) \)

To encrypt the first integer that represents "ATT", we have
\( c = m^e \text{ mod } n = 513^3 \text{ mod } 17947 = 8363 \).

We can verify that our private key is valid by decrypting
\( m' = c^d \text{ mod } n = 8363^{11787} \text{ mod } 17947 = 513 \).

Overall, our plaintext is represented by the set of integer \( m \)
We compute corresponding cipher text integers $c = m^e \mod n$.

The security of RSA algorithm depends on the ability of the hacker to factorize numbers. New, faster and better methods for factoring numbers are constantly being devised. The Trent best for long numbers is the Number Field Sieve. Prime Numbers of a length that was unimaginable a mere decade ago are now factored easily. Obviously the longer a number is, the harder is to factor, and so the better the security of RSA.

6.9 A General discussion on General Number Field Sieve (GNFS) & Quadratic Sieve factoring algorithms

The foundation of the most popular public-key cryptography algorithm in use today, RSA, rests on the difficulty of factoring large integers. When keys are generated, efficient algorithms are used to generate two very large prime numbers and multiply them together. The person except the one who generated the keys knows these two numbers, but everyone else only knows the product. By factoring, we can extract the two prime factors from the product and break the encryption.

At the time that RSA was invented in 1977, factoring integers with as few as 80 decimal digits was intractable; all known algorithms were either too slow or required the number to have a special form. This made even small, 256-bit keys relatively secure. The first major breakthrough was quadratic sieve, a relatively simple factoring algorithm invented by Carl Pomerance in 1981, which can factor numbers up to 100 digits and more. It's still the best known method for numbers under 110 digits or so; for larger numbers, the general number field sieve (GNFS) is now used. However, the general number field sieve is extremely complicated, and requires extensive explanation and background for even the most basic implementation. However, GNFS is based on the same fundamental ideas as quadratic sieve, so if factoring the largest numbers then GNFS is the best choice (Pomerance, Carl December 1996).

In addition, due to the large amount of computation required, the GNFS was designed to support parallel computing, which further speeds up the actual
computation by the number of processors involved. As a result, various achievements have been recorded in efforts of integer factorization, such as the 193-digit challenge number RSA-640 in 2005. This sequence of breakthroughs has urgently raised a question about the security level of the RSA algorithm, a public key encryption method being used in almost every major information system nowadays. In particular, it provides an ability to break the main security principle of the RSA algorithm, the best known public key encryption method. In addition, several aspects of applied mathematics also benefit from integers that come along with their primes factors. As an example, if prime factors of an arbitrarily composite integer $n$ are available, then it is simple to compute the completely multiplicative function on $n$, i.e., an arithmetic homomorphism $f(x)$ that is defined on $\mathbb{Z}$ (Weisstein E.W. November 2005).

Quadratic Sieve is a general-purpose factorization algorithm, meaning that its running time depends solely on the size of the integer to be factored, and not on special structure or properties. The algorithm attempts to set up a congruence of squares modulo $n$ (the integer to be factorized), which often leads to a factorization of $n$. The algorithm works in two phases: the data collection phase, where it collects information that may lead to a congruence of squares; and the data processing phase, where it puts all the data it has collected into a matrix and solves it to obtain a congruence of squares. The naive approach to finding a congruence of squares is to pick a random number, square it, and hope the least non-negative remainder modulo $n$ is a perfect square (in the integers). For example, $80^2 \mod 5959$ are 441, which is $21^2$. This approach finds a congruence of squares only rarely for large $n$, but when it does find one, more often than not, the congruence is nontrivial and the factorization is complete.
6.10 For Real life example we have to generate big prime numbers and need an algorithm to factorization

We have used the best known algorithm for factoring large numbers is the General Number Field Sieve (GNFS). It is as follows

```c
#include <stdio.h>
#include <limits.h>
#include <sys/types.h>
#ifdef __ppc__
#include "ppc32/siever-config.h"
#else
#include "asm/lasieve-asm.h"
#include "lasieve.h"
#endif

static double *aux;
static size_t aux_alloc = 0;

/* Action of GL(2,Z) on homogeneous polys. */

/*********************************************************

void tpol(double *rop, double *op, i32_t deg, i32_t x0, i32_t x1,
         i32_t y0, i32_t y1)

/***********************************************************/

{ i32_t d;

if (deg == U32_MAX)
    complain("Degree too large\n");
if (deg == 0)
    { rop[0] = op[0];
        return;
    }
if (aux_alloc < deg + 1)
    { if (aux_alloc > 0)
        free(aux);
        aux_alloc = deg + 1;
        aux = xmalloc(aux_alloc * sizeof(*aux));
    }
```
rop[0] = op[0] * y1;
rop[1] = op[0] * y0;
aux[0] = x1;
aux[1] = x0;
for (d = 1;; d++)
{
    i32_t i;

    for (i = 0; i <= d; i++)
        rop[i] += op[d] * aux[i];
    if (d == deg)
        return;

    /* Multiply rop by (y0*T+y1), where T is the free variable. */
    rop[d + 1] = y0 * rop[d];
    for (i = d; i > 0; i--)
        rop[i] = y1 * rop[i] + y0 * rop[i - 1];
    rop[0] *= y1;

    /* Multiply aux by (x0*T+x1). */
    aux[d + 1] = x0 * aux[d];
    for (i = d; i > 0; i--)
        aux[i] = x1 * aux[i] + x0 * aux[i - 1];
    aux[0] *= x1;
}

double rpol_eval(double *p, i32_t d, double x, double y)
{ double v, z;
i32_t i;

    for (i = 0, v = p[0], z = x; i < d;)
    {
        v = y * v + p[++i] * z;
        z *= x;
    }
    return v;
}
/***************************************************************************/

double rpol_lb(double *pol, i32_t poldeg, double a, double b)
/***************************************************************************/
{
    i32_t i;
    double m1, m2, s1, s2;

    if (poldeg == U32_MAX)
        complain("Degree too large\n");
    if (aux_alloc < poldeg + 1)
    {
        if (aux_alloc > 0)
            free(aux);
        aux_alloc = poldeg + 1;
        aux = xmalloc(aux_alloc * sizeof(*aux));
    }

    if (a < 0)
    {
        double rv1, rv2;
        i32_t s;

        for (i = 0, s = 1; i <= poldeg; i++, s = -s)
            aux[i] = pol[i] * s;
        if (b < 0)
            return rpol_lb(aux, poldeg, -b, -a);
        rv1 = rpol_lb(aux, poldeg, 0, -a);
        if ((rv2 = rpol_lb(pol, poldeg, 0, b)) < rv1)
            return rv2;
        return rv1;
    }

    if (pol[poldeg] < 0)
    {
        s1 = -pol[poldeg];
        s2 = -pol[poldeg];
        m1 = 0;
        m2 = 0;
    }
    else
    {
        m1 = pol[poldeg];

    }
m2 = pol[poldeg];
s1 = 0;
s2 = 0;
}

for (i = 1; i <= poldeg; i++)
{
    if (pol[poldeg - i] < 0)
    {
        s1 = s1 * a - pol[poldeg - i];
        s2 = s2 * b - pol[poldeg - i];
        m1 *= a;
        m2 *= b;
    }
    else
    {
        s1 *= a;
        s2 *= b;
        m1 = m1 * a + pol[poldeg - i];
        m2 = m2 * b + pol[poldeg - i];
    }
}

if (s2 < m1)
    return m1 - s2;
if (s1 > m2)
    return s1 - m2;
return -1;

#if 0
/* Use this to debug tpol and rpol_eval. */
void Usage()
{
    exit(1);
}

main()
{
    i32_t d = 3, i;
    double p[4], q[4], a, b;
    i32_t x0, x1, y0, y1;
```c
#include

#define 1

p[0] = 43;
p[1] = 15;
p[2] = 18;
p[3] = -100;
x0 = 5;
x1 = -3;
y0 = 3;
y1 = -2;
#else
p[0] = 0;
p[1] = 0;
p[2] = 0;
p[3] = 1;
x0 = 1;
x1 = 1;
y0 = 1;
y1 = 0;
#endif
tpol(q, p, d, x0, x1, y0, y1);
scanf("%lf %lf", &a, &b);
printf("%f == %f ?
", rpol_eval(p, d, x0 * a + x1 * b, y0 * a + y1 * b),
rpol_eval(q, d, a, b));
for (i = 0; i <= d; i++)
printf(i == 0 ? "%f" : " %f", q[i]);
printf("\n");
exit(0);
#endif

6.11 Code for Prime Number Generation

import java.util.*;

// finds all prime numbers up to max
public static List<Integer> generatePrimes(int max)
{
    List<Integer> primes = new ArrayList<Integer>();
    // start from 2
    OUTERLOOP:
        for (int i = 2; i <= max; i++)
        {
            // try to divide it by all known primes
```
for (Integer p : primes)
    if (i % p == 0)
        continue OUTERLOOP; // i is not prime

    // i is prime
    primes.add(i);
}
return primes;

Sieve of Erastosthenes, from here

import java.util.*;

public class Sieve
{
    private BitSet sieve;

    public Sieve(int size)
    {
        sieve = new BitSet((size+1)/2);
    }

    public boolean is_composite(int k)
    {
        assert k >= 3 && (k % 2) == 1;
        return sieve.get((k-3)/2);
    }

    public void set_composite(int k)
    {
        assert k >= 3 && (k % 2) == 1;
        sieve.set((k-3)/2);
    }

    public static List<Integer> sieve_of_eratosthenes(int max)
    {
        Sieve sieve = new Sieve(max + 1); // +1 to include max itself
        for (int i = 3; i*i <= max; i += 2)
        {
            if (sieve.is_composite(i))
                continue;

            // We increment by 2*i to skip even multiples of i
            for (int multiple_i = i*i; multiple_i <= max; multiple_i += 2*i)
            {
            }
        }
    }
sieve.set_composite(multiple_i);

}  
List<Integer> primes = new ArrayList<Integer>();  
primes.add(2);  
for (int i = 3; i <= max; i += 2)  
if (!sieve.is_composite(i))  
primes.add(i);  
return primes;

6.12 Second Output (hexadecimal format)

n=3337   n=33    n=17947  
e=79      e=3      e=3

n=  
E08973398DD8F5F5E88776397F4EB005BB5383DE0FB7ABDC7DC775290D0
52E6D
12DFA68626D4D26FAA5829FC97ECFA82510F3080BEB1509E4644F12CBB
D832CF
C6686F07D9B060ACBEEE34096A13F5F7050593DF5EBA3556D961FF197FC
981E6
F86CEA874070EFAC6D2C749F2DFA553AB99977702A648528C4EF357385774
575F

e=010001
Bob's 1024-bit RSA encryption key in hex format:

n=  
A9E167983F39D55FF2A093415E6798985C8355D9A915BFB1D01DA197026
170F
BDA522D035856D7A986614415CCFB7B7083B09C991B81969376DF9651E7B
D9A9
3324A37F3BBBAF460186363432CB07035952FC858B3104B8CC18081448E64
F1C
FB5D60C4E05C1F53D37F53D86901F105F87A70D1BE83C65F38CF1C2CAA6
AA7EB

e=010001
6.13 Conclusion & Future Work

I conjecture that the most efficient way to break RSA algorithm is still unknown because in most of the cases researchers & scientist illustrated the improper use of RSA cryptosystem or the wrong choice of the keys. During the research we have gone through many other algorithms, analysing each & every aspects theoretical as well as practical approaches of the cryptosystem as a cryptanalyst. The point matters are that the cryptanalyst need to have detail knowledge of cryptosystem to break through any system. Security attacks are classified as either passive attacks & active attacks so being cryptanalyst we need to analyse each and every point.

A security mechanism is any process that should be able to detect, prevent & recover from any type of attack. Now a day’s information security is major issue as in a progressive environment every piece of data is automated, so as to be used by every person around the world. During the research work we have seen that in Hill Cipher if the inverse of the matrix is available then the system security has been compromised. So in modified Hill Cipher is easy to implement & difficult to crack.

Public Key Cryptosystems are better than symmetric cryptosystems as they provide better environment in the area of confidentiality, key distribution & authentication. The mathematical operations in public-key cryptography required considerable computational resources relative to computer performance. As a result, public-key cryptography was a slow sell through that 80-90’s. I’ve had the privilege to observe the public-key cryptography (RSA Algorithm) in my research work, we have needed to understand how best to use the algorithm to establish of the value $m$ to the actual message or key, some of which are much better for security than others. We have needed to understand the impact of various proposed improvements to integer factorization methods, especially in terms of recommended key sizes. This requires assessment of the effectiveness of those methods.

The mathematics like – prime numbers, integer factorization, and modular exponentiation – has had a great impact on computer security, particularly in the field of internet. The theory is working well in practice through algorithms like Diffie-Hellman key agreement, the RSA public-key cryptosystem and elliptic curve cryptography.
In cryptography, “it’s not broken” is no reason to avoid trying to fix it. Mathematicians still don’t know whether or not there are faster methods for integer factorization than the ones currently available. Research is needed to try to find faster methods, as well to try to prove that there aren’t any. A related research problem is to confirm whether integer factorization is hard or some new methods can have a better chance.

Interestingly, much faster methods for integer factorization already exist in theory, but they run on computers that haven’t yet been built. In particular, if one could build a full-scale quantum computer, it will be possible to break a large number into its factors essentially as easily as it is to put the number together by multiplication.