ACM ICPC Training

Bruce Merry

August 2013

Outline

Problem description

The RSA encryption algorithm requires two large prime numbers to compute an encryption / decryption key pair. The bigger the prime numbers, the harder it becomes to compromise your encrypted data. Thus, the search for large prime numbers begins...

You have developed a novel algorithm for generating potential prime numbers. However, since no known deterministic algorithm exists that generates prime numbers, you know that you will have a few non-primes in your output, so you have to filter your algorithm's output to extract the largest prime number. Your task is simple: given a sequence of numbers, find the largest prime number amongst them.

Problem description

The RSA encryption algorithm requires two large prime numbers to compute an encryption / decryption key pair. The bigger the prime numbers, the harder it becomes to compromise your encrypted data. Thus, the search for large prime numbers begins...

You have developed a novel algorithm for generating potential prime numbers. However, since no known deterministic algorithm exists that generates prime numbers, you know that you will have a few non-primes in your output, so you have to filter your algorithm's output to extract the largest prime number.

Your task is simple: given a sequence of numbers, find the largest prime number amongst them.

Problem description

The RSA encryption algorithm requires two large prime numbers to compute an encryption / decryption key pair. The bigger the prime numbers, the harder it becomes to compromise your encrypted data. Thus, the search for large prime numbers begins...

You have developed a novel algorithm for generating potential prime numbers. However, since no known deterministic algorithm exists that generates prime numbers, you know that you will have a few non-primes in your output, so you have to filter your algorithm's output to extract the largest prime number.

Your task is simple: given a sequence of numbers, find the largest prime number amongst them.

Input

Your input will consist of a number of records. Each record is a sequence of integers in the range 2 to $2^{32} - 1$, terminated by the value 0.
The last record in the input set will be terminated by the number -1.

```
5
7
21
9
11
101
13
0
65535
131071
-1
```

Figure: Sample input

Input

Your input will consist of a number of records. Each record is a sequence of integers in the range 2 to $2^{32} - 1$, terminated by the value 0.
The last record in the input set will be terminated by the number -1.

```
5
7
21
9
11
101
13
0
65535
131071
-1
```

Figure: Sample input

Output

For each input record, your output should be the largest prime number in the sequence.

```
101
131071
```

Figure: Sample output

Output

For each input record, your output should be the largest prime number in the sequence.

```
101
131071
```

Figure: Sample output

Time Limit

5 seconds

Outline

Solution Outline

```
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    long biggest = 0;
    long N;
    do {
        N = in.nextLong();
        if (N <= 0) {
            System.out.println(biggest);
            biggest = 0;
        }
        else if (N > biggest && isPrime(N))
            biggest = N;
    } while (N != -1);
}
```

Drilling Down

```
public static boolean isPrime(long x) {
    if (x <= 1) return false;
    for (long i = 2; i * i <= x; i++)
        if (x % i == 0)
            return false;
    return true;
}
```

Testing

Start by testing the sample input

- Type the sample input into `sample.in`
- Type the sample output into `sample.ref`
- Run:

```
java PrimeConcern < sample.in > sample.out
```

- Compare: `diff sample.out sample.ref`

Testing

Start by testing the sample input

- Type the sample input into `sample.in`
- Type the sample output into `sample.ref`
- Run:

```
java PrimeConcern < sample.in > sample.out
```

- Compare: `diff sample.out sample.ref`

Testing

Start by testing the sample input

- Type the sample input into `sample.in`
- Type the sample output into `sample.ref`
- Run:

```
java PrimeConcern < sample.in > sample.out
```

- Compare: `diff sample.out sample.ref`

Testing

Start by testing the sample input

- Type the sample input into `sample.in`
- Type the sample output into `sample.ref`
- Run:
  ```
  java PrimeConcern < sample.in > sample.out
  ```
- Compare: `diff sample.out sample.ref`

Testing

Also test the extreme cases:

```
1
4294967291
2
-1
```

Outline

The Problem

The problem-setters in Pretoria are evil.

```
23
4
100
-1
```

The Problem

The problem-setters in Pretoria are evil.

```
1  23␣
2  4
3  100␣␣
4  -1
5
```

The Problem

```
BufferedReader in = new BufferedReader(new
      InputStreamReader(System.in));
long biggest = 0;
long N;
do {
   N = Long.parseLong(in.readLine());
   if (N <= 0) {
      System.out.println(biggest);
      biggest = 0;
   }
   else if (N > biggest && isPrime(N))
      biggest = N;
} while (N != -1);
```

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "23 "
```

The Problem

```
BufferedReader in = new BufferedReader(new
      InputStreamReader(System.in));
long biggest = 0;
long N;
do {
   N = Long.parseLong(in.readLine());
   if (N <= 0) {
      System.out.println(biggest);
      biggest = 0;
   }
   else if (N > biggest && isPrime(N))
      biggest = N;
} while (N != -1);
```

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "23 "
```

Solutions (Java)

- Use `java.util.Scanner` (slow but easy)
- Use `String.trim` on the result of `readLine()`

```
N = Long.parseLong(
      in.readLine().trim());
```

Solutions (C++)

- Both `cin >>` and `scanf` skip whitespace
- Add `std::ios::sync_with_stdio(false);` for better `iostream` performance if not using C stdio

Outline

Conventions

There are three common ways to end the input

- A count of test cases at the top of the file
- A special value at the end of the file
- No marker, the file just ends

Count of Test Cases

```
Scanner in = new Scanner(System.in);
int T = in.nextInt();
for (int casenum = 0; casenum < T; casenum++) {
    // process a test case
}
```

Special Marker

Java

If each test case is a single integer, test cases terminated by 0:

```
Scanner in = new Scanner(System.in);
int N = in.nextInt();
while (N != 0) {
    // process N
    N = in.nextInt();
}
```

Special Marker

C++

```
int N;
while (cin >> N && N != 0) {
    // process N
}
```

End of File

Java

Do not do this:

```
BufferedReader in = new BufferedReader(
      new InputStreamReader(System.in));
String line = in.readLine();
while (line != null) {
   N = Integer.parseInt(line.trim());
   // process N
   line = in.readLine();
}
```

```
Exception in thread "main" java.lang.NumberFormatException: For input string: ""
```

End of File

Java

Do not do this:

```
BufferedReader in = new BufferedReader(
      new InputStreamReader(System.in));
String line = in.readLine();
while (line != null) {
   N = Integer.parseInt(line.trim());
   // process N
   line = in.readLine();
}
```

```
Exception in thread "main" java.lang.NumberFormatException: For input string: ""
```

End of File

Java

Do not do this either:

```
try {
    BufferedReader in = new BufferedReader(
          new InputStreamReader(System.in));
    String line = in.readLine();
    while (line != null) {
        N = Integer.parseInt(line.trim());
        // process N
        line = in.readLine();
    }
}
catch (Exception e) {}
```

It will swallow other exceptions, making testing difficult.

End of File

Java

Do not do this either:

```
try {
    BufferedReader in = new BufferedReader(
          new InputStreamReader(System.in));
    String line = in.readLine();
    while (line != null) {
        N = Integer.parseInt(line.trim());
        // process N
        line = in.readLine();
    }
}
catch (Exception e) {}
```

It will swallow other exceptions, making testing difficult.

End of File

Java

This should work:

```
BufferedReader in = new BufferedReader(
      new InputStreamReader(System.in));
String line = in.readLine().trim();
while (line != null && !line.equals("")) {
   N = Integer.parseInt(line);
   // process N
   line = in.readLine().trim();
}
```

End of File

C++

This does not work:

```
while (!cin.eof()) {
    int N;
    cin >> N;
    // process N
}
```

End of File

C++

This is simpler and robust:

```
int N;
while (cin >> N) {
    // process N
}
```

Outline

Avoiding Failure

Sample cases are not exhaustive

- Test with a large case
- Test with corner cases (e.g. $N = 1$)
- Test all code paths

Don't be afraid to use PC time for debugging

Avoiding Failure

Sample cases are not exhaustive

- Test with a large case
- Test with corner cases (e.g. $N = 1$)
- Test all code paths

Don't be afraid to use PC time for debugging

Avoiding Failure

Sample cases are not exhaustive

- Test with a large case
- Test with corner cases (e.g. $N = 1$)
- Test all code paths

Don't be afraid to use PC time for debugging

Avoiding Failure

Sample cases are not exhaustive

- Test with a large case
- Test with corner cases (e.g. $N = 1$)
- Test all code paths

Don't be afraid to use PC time for debugging

Avoiding Failure

Sample cases are not exhaustive

- Test with a large case
- Test with corner cases (e.g. $N = 1$)
- Test all code paths

Don't be afraid to use PC time for debugging

Handling Failure

Sometimes your solutions will be rejected

- Write more test cases
- Print out and examine the source
- Re-read the problem statement
- Explain your solution to someone else
- Test your bug fixes

Handling Failure

Sometimes your solutions will be rejected

- Write more test cases
- Print out and examine the source
- Re-read the problem statement
- Explain your solution to someone else
- Test your bug fixes

Handling Failure

Sometimes your solutions will be rejected

- Write more test cases
- Print out and examine the source
- Re-read the problem statement
- Explain your solution to someone else
- Test your bug fixes

Handling Failure

Sometimes your solutions will be rejected

- Write more test cases
- Print out and examine the source
- Re-read the problem statement
- Explain your solution to someone else
- Test your bug fixes

Handling Failure

Sometimes your solutions will be rejected

- Write more test cases
- Print out and examine the source
- Re-read the problem statement
- Explain your solution to someone else
- Test your bug fixes

Coding Strategy

Make it as simple as possible, but no simpler

- Can usually avoid `new` in C/C++
- `Scanner` in Java is slow, but often fast enough
- Pay attention to the constraints

Coding Strategy

Make it as simple as possible, but no simpler

- Can usually avoid `new` in C/C++
- `Scanner` in Java is slow, but often fast enough
- Pay attention to the constraints

Coding Strategy

Make it as simple as possible, but no simpler

- Can usually avoid `new` in C/C++
- `Scanner` in Java is slow, but often fast enough
- Pay attention to the constraints

Outline

Scoring

Each problem is worth one point. Ties are broken by time penality.

Time penality is the sum for each solved problem of

- Time from start of the contest until correct submission
- 20 minutes per incorrect submission

Scoring

Each problem is worth one point. Ties are broken by time penality.
Time penality is the sum for each solved problem of

- Time from start of the contest until correct submission
- 20 minutes per incorrect submission

Scoring

Strategy should thus be:

- Solve problems easiest to hardest.
- Don't submit without doing some testing.
- Don't jump between problems without solving any of them.
- There is no harm in last-minute attempts.

Scoring

Strategy should thus be:

- Solve problems easiest to hardest.
- Don't submit without doing some testing.
- Don't jump between problems without solving any of them.
- There is no harm in last-minute attempts.

Scoring

Strategy should thus be:

- Solve problems easiest to hardest.
- Don't submit without doing some testing.
- Don't jump between problems without solving any of them.
- There is no harm in last-minute attempts.

Scoring

Strategy should thus be:

- Solve problems easiest to hardest.
- Don't submit without doing some testing.
- Don't jump between problems without solving any of them.
- There is no harm in last-minute attempts.

Team Strategy

Appoint a team manager

- Keeps track of who is doing what
- Makes sure problems are not forgotten
- Keeps track of time

Team Strategy

Appoint a team manager

- Keeps track of who is doing what
- Makes sure problems are not forgotten
- Keeps track of time

Team Strategy

Appoint a team manager

- Keeps track of who is doing what
- Makes sure problems are not forgotten
- Keeps track of time

Pair Programming

One person codes, co-pilot watches

- Reads the problem
- Understands the solution
- Points out bugs
- Looks things up for pilot

Pair Programming

One person codes, co-pilot watches

- Reads the problem
- Understands the solution
- Points out bugs
- Looks things up for pilot

Pair Programming

One person codes, co-pilot watches

- Reads the problem
- Understands the solution
- Points out bugs
- Looks things up for pilot

Pair Programming

One person codes, co-pilot watches

- Reads the problem
- Understands the solution
- Points out bugs
- Looks things up for pilot