Incremental Sieve

Detailed & Simple

Mausam Kar

February 4, 2026

Contents

1 What Is Incremental Sieve?

The Big Idea

Generate primes ONE-BY-ONE, not all at once!
Like a prime number machine:

- Press button → Get next prime
- Press again → Get next prime
- Can generate infinitely!

1.1 Difference from Normal Sieve

Normal Sieve:

- Need to know N in advance
- Create array for ALL numbers up to N
- Get ALL primes at once

Incremental Sieve:

- Don't need to know how many primes
- Generate one prime at a time
- Can continue forever!

2 Algorithm - Detailed Explanation

2.1 The Core Trick

Memory Map Concept

Keep a "memory" map of future composite numbers:
`composite number → prime that divides it`
When checking number n:

- NOT in map? → n is PRIME!
- IN map? → n is composite, skip it

2.2 How It Works - Step by Step

Let's generate first few primes:

Number 2:

1. Check: Is 2 in memory map? NO
2. → 2 is PRIME!
3. Add to memory: `map[4] = 2`
4. Why 4? Because $2 \times 2 = 4$ is first composite
5. Move to number 3

Memory now: {4: 2}

Number 3:

1. Check: Is 3 in memory? NO
2. → 3 is PRIME!
3. Add: `map[9] = 3` (because $3 \times 3 = 9$)
4. Move to 4

Memory now: {4: 2, 9: 3}

Number 4:

1. Check: Is 4 in memory? YES! [2]

2. → 4 is COMPOSITE (divisible by 2)

3. Remove 4 from memory

4. Move 2 to next multiple: $4 + 2 = 6$

5. Add: `map[6] = 2`

6. Move to 5

Memory now: {6: 2, 9: 3}

Number 5:

1. Check: Is 5 in memory? NO

2. → 5 is PRIME!

3. Add: `map[25] = 5`

Memory now: {6: 2, 9: 3, 25: 5}

2.3 Why Add p×p?

When we find prime p, why add $p \times p$ to memory (not $2 \times p$)?

Same Logic as Normal Sieve!

All smaller multiples are handled by smaller primes.
$p \times p$ is the FIRST multiple we need to track!

Example: p = 5

Multiple	Already handled?
$5 \times 2 = 10$	Will be handled by 2
$5 \times 3 = 15$	Will be handled by 3
$5 \times 4 = 20$	Will be handled by 2
$5 \times 5 = 25$	NOT handled yet!

So we add 25 to memory!

3 Pseudocode with Explanation

```
INCREMENTAL_SIEVE():

1. Create empty map D
2. current = 2

3. While true (infinite):
4.     If current NOT in D:
5.         Print current (it's prime)
6.         D[current * current] = current

7.     Else:
8.         prime = D[current]
9.         nextMultiple = current + prime
10.        D[nextMultiple] = prime
11.        Remove D[current]

12.    current = current + 1
```

3.1 Line-by-Line Explanation

Line 1: Create empty map (dictionary/HashMap)

Line 4: Check if current number is in map

Lines 5-6: If NOT in map:

- It's prime! Print it
- Add $current^2$ to map with this prime

Lines 8-11: If IN map:

- Get the prime that divides it
- Calculate next multiple: current + prime
- Move prime to that next multiple
- Remove current from map (don't need it anymore)

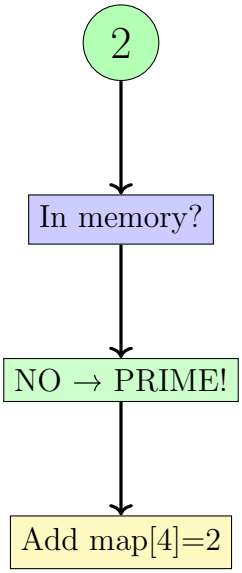
Line 12: Move to next number

4 Detailed Visual Trace

Let's trace numbers 2 to 12 with visual diagrams:

4.1 Number 2

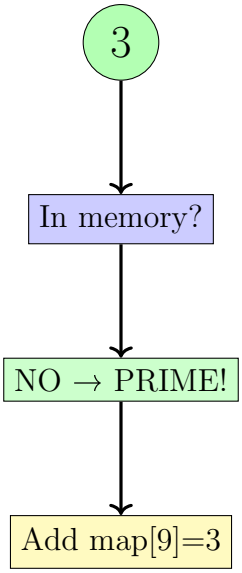
Checking: 2



Memory: {4: 2}

4.2 Number 3

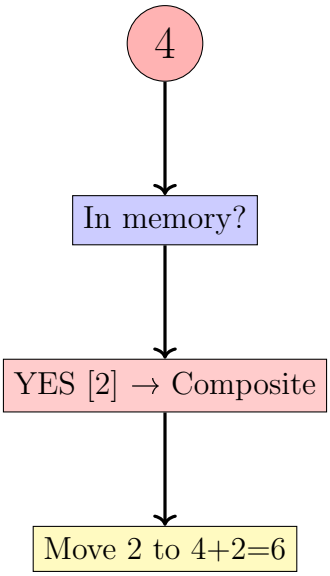
Checking: 3



Memory: {4: 2, 9: 3}

4.3 Number 4

Checking: 4



Memory: {6: 2, 9: 3}

4.4 Complete Trace Table

n	In Map?	Prime?	Action
2	NO	YES	Add map[4]=2
3	NO	YES	Add map[9]=3
4	YES[2]	NO	Move 2→6
5	NO	YES	Add map[25]=5
6	YES[2]	NO	Move 2→8
7	NO	YES	Add map[49]=7
8	YES[2]	NO	Move 2→10
9	YES[3]	NO	Move 3→12
10	YES[2]	NO	Move 2→12
11	NO	YES	Add map[121]=11
12	YES[2,3]	NO	Move 2→14, 3→15

Primes found: 2, 3, 5, 7, 11
Pattern: Every number NOT in map is prime!

5 Java Code - Simple & Working

Listing 1: Incremental Sieve - Complete Working Code

```java
import java.util.*;

public class IncrementalSieve {

    // Map: composite number -> prime that divides it
    Map<Integer, Integer> memory;
    int current;

    // Constructor: Initialize
    public IncrementalSieve() {
        memory = new HashMap<>();
        current = 2;  // Start from 2
    }

    // Get next prime number
    public int nextPrime() {
        while (true) {  // Keep checking until we find prime

            // Check: Is current in memory?
            if (!memory.containsKey(current)) {
                // NOT in memory -> it's PRIME!
                int prime = current;

                // Add prime^2 to memory
                memory.put(prime * prime, prime);

                // Move to next number
                current++;

                // Return the prime we found
                return prime;

            } else {
                // IN memory -> it's COMPOSITE

                // Get prime that divides current
                int prime = memory.get(current);

                // Calculate next multiple
                int nextMultiple = current + prime;

                // Move prime to next multiple
                memory.put(nextMultiple, prime);
```

```
                // Remove current from memory
                memory.remove(current);

                // Move to next number
                current++;
            }
        }
    }

    public static void main(String[] args) {
        IncrementalSieve sieve = new IncrementalSieve();

        // Example 1: Get first 20 primes
        System.out.println("First 20 primes:");
        for (int i = 0; i < 20; i++) {
            System.out.print(sieve.nextPrime() + " ");
        }
        System.out.println();

        // Example 2: Get next 5 primes
        System.out.println("\nNext 5 primes:");
        for (int i = 0; i < 5; i++) {
            System.out.print(sieve.nextPrime() + " ");
        }
        System.out.println();
    }
}
```

Output:

```
First 20 primes:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71

Next 5 primes:
73 79 83 89 97
```

6 Summary

Incremental Sieve - One-by-one generator

How it works:

1. Keep map of future composites
2. For each number:
   - NOT in map? → Prime! Add p^2 to map
   - IN map? → Composite. Move prime to next multiple
3. Can generate infinite primes!

Key points:

- Add $p \times p$ (not $2 \times p$)
- Next multiple = current + prime
- Remove from map after use

Use when:

- Don't know how many primes needed
- Want one-by-one generation
- Iterator/streaming pattern
- Infinite prime generation