Loop 1

```
for (i=0; i<N; i++) {
   a[(N+1)*i+k1] = b[i]*c[i];
   d[i] = a[(N+1)*i+k2*N]+e;
}
```

1. Substitute N=5590, k1=k2=559 to the above loop:

```
for (int i = 0; i < 5590; i++) {
   a[(5591)*i+559] = b[i]*c[i];  // statement 1: a1 = 5591, a0 = 559
   d[i] = a[(5591)*i+3124810]+e; // statement 2: b1 = 5591, b0 = 3124810
}
```

2. We have let `s1 = 5591 * i1 + 559` and `s2 = 5591 * i2 + 3124810`
3. By GCD Test, data dependence occur if `s1 = s2`, which implies

```
  5591 * i1 + 559 = 5591 * i2 + 3124810
⇒ 5591 * i1 - 5591 * i2 = 3124810 - 559
⇒ 5591 * (i1 - i2) = 3124251              (1)
```

Observe that equation (1) has integral solution iff `gcd(3124251, 5591)` exists. And, in fact, only 1 is gcd of them. Hence, by GCD test, i1 and i2 are not dependent.

4. Banerjee's Test

`s1 - s2 ⇔ 5591 * (i1 - i2) - 3124251`

Consider the four points:

```
(i1 = 0, i2 = 0)       ⇒ s1 - s2 = 5591 * (0 - 0) - 3124251       = -3124251
(i1 = 0, i2 = 5589)    ⇒ s1 - s2 = 5591 * (0 - 5589) - 3124251    = -34372350
(i1 = 5589, i2 = 0)    ⇒ s1 - s2 = 5591 * (5589 - 0) - 3124251    =  28123848
(i1 = 5589, i2 = 5589) ⇒ s1 - s2 = 5591 * (5589 - 5589) - 3124251 = -3124251
```

Thus, `L = -34372350, U = 28123848`.

Note that `b0 - a0 = 3124810 - 559 = 3124251`. So `L < b0 - a0 < U`. This doesn't break the dependence between statement 1 and statement 2.

Now let's add constraint `i1 ≤ i2`, which allows us to exclude the point `(i1 = 5589, i2 = 0)` from our test. Hence, we have

```
(i1 = 0, i2 = 0)       ⇒ s1 - s2 = 5591 * (0 - 0) - 3124251       = -3124251
(i1 = 0, i2 = 5589)    ⇒ s1 - s2 = 5591 * (0 - 5589) - 3124251    = -34372350
(i1 = 5589, i2 = 5589) ⇒ s1 - s2 = 5591 * (5589 - 5589) - 3124251 = -3124251
```

This implies `L = min(-3124251, -34372350, -3124251) = -34372350` and `U = max(-3124251, -34372350, -3124251) = -3124251`. But `b0 - a0 = 3124810 - 559 = 3124251 > U`. This breaks data dependence between statement1 and statement2.

Loop 2

1. Substitution:

Given the loop as below:

```
for (i=0; i<N; i++) {
   a[k3*N+i] = b[i]*c[i];
   d[i] = a[(k3+1)*N+i]+e;
}
```

Now substitute N=5590,k3=124 to obtain the new loop where statement 1 writes to and statement 2 reads from a

```
for (i=0; i<5590; i++) {
   a[693160+i] = b[i]*c[i];  // statement 1: a1 = 1, a0 = 693160
   d[i] = a[698750+i]+e;     // statement 2: b1 = 1, b0 = 698750
}
```

Let s1 = 693160+i and s2 = 698750+i

2. Calculation:

The data dependence occurs iff

```
  s1              = s2 for any 0 ≤ i1, i2 ≤ 5590
⇔ 693160+i1       = 698750+i2
⇔ i1 - i2 - 5590 = 0
```

3. By GCD Test, the data dependence occurs iff equation `i1 - i2 - 550 = 0 ⇔ i1 - i2 = 550` has integral solutions. Indeed, gcd(a1 = 1, b1 = 1) = 1 divides b0 - a0 = 550. Hence, GCD test doesn't break data dependence between statement 1 and statement 2.

4. By Banerjee's Test,

Consider the four points:

```
(i1 = 0, i2 = 0)       ⇒ s1 - s2 = 0 - 0 - 5590       = -5590
(i1 = 0, i2 = 5589)    ⇒ s1 - s2 = 0 - 5589 - 5590    = -11179
(i1 = 5589, i2 = 0)    ⇒ s1 - s2 = 5589 - 0 - 5590    = -1
(i1 = 5589, i2 = 5589) ⇒ s1 - s2 = 5589 - 5589 - 5590 = -5590
```

Now `L = min(-5590, -11179, -1, -5590) = -11179`, `U = max(-5590, -11179, -1, -5590) = -1`. And `b0 - a0 = 698750 - 693160 = 5590` which is not in the interval [-11179, -1]. This breaks the dependence between statement 1 and statement 2.

Loop 3

```
for (i=0; i<N; i++) {
   a[N*i+k4] = b[i]*c[i];
```

```
    d[i] = a[(N+1)*i+k5]+e;
}
```

1. Substitute N=5590, k4=94, k5=144 to the above loop:

```
for (int i = 0; i < 5590; i++) {
    a[5590*i+94] = b[i]*c[i];     // statement 1: a1 = 5590, a0 = 94
    d[i] = a[5591*i+144]+e;       // statement 2: b1 = 5591, b0 = 144
}
```

2. Let `s1 = 5590*i+94` and `s2 = 5591*i+144`

3. Data dependence occurs iff s1 = s2, which means

```
    5590 * i1 + 94         = 5591 * i2 + 144 for any 0 ≤ i1, k2 ≤ 5590
⇔   5590 * i1 - 5591 * i2 = 50                (1)
```

4. GCD Test, to check if equation (1) has integral solution, we verify if `gcd(5590, 5591)` divides `50`. Since `gcd(5590, 5591) = 1` and 1 divides 50, the GCD test doesn't break data dependence between statement 1 and statement 2.

5. Banerjee's test

`s1 - s2 = 5590 * i1 + 94 - 5591 * i2 - 144 = 5590 * i1 - 5591 * i2 - 50`

Consider the four points:

```
(i1 = 0, i2 = 0)        ⇒ s1 - s2 = 5590 * 0 - 5591 * 0 - 50       = -50
(i1 = 0, i2 = 5589)     ⇒ s1 - s2 = 5590 * 0 - 5591 * 5589 - 50    = -31248149
(i1 = 5589, i2 = 0)     ⇒ s1 - s2 = 5590 * 5589 - 5591 * 0 - 50    = 31242460
(i1 = 5589, i2 = 5589) ⇒ s1 - s2 = 5590 * 5589 - 5591 * 5589 - 50 = -5639
```

Thus, `L = min(-50, -31248149, 31242460, -5639) = -31248149` and `U = max(-50, -31248149, 31242460, -5639) = 31242460`.

Note that `b0 - a0 = 144 - 94 = 50`. So `L ≤ b0 - a0 ≤ U` which doesn't break the data dependence.

Now let's add constraint `i1 ≤ i2`, which allows us to exclude the point `(i1 = 5589, i2 = 0)` from our test. Hence, we have

```
(i1 = 0, i2 = 0)        ⇒ s1 - s2 = 5590 * 0 - 5591 * 0 - 50       = -50
(i1 = 0, i2 = 5589)     ⇒ s1 - s2 = 5590 * 0 - 5591 * 5589 - 50    = -31248149
(i1 = 5589, i2 = 5589) ⇒ s1 - s2 = 5590 * 5589 - 5591 * 5589 - 50 = -5639
```

Thus, `L = min(-50, -31248149, -5639) = -50` and `U = max(-50, -31248149, -5639) = -31248149`. This means `b0 - a0 = 144 - 94 = 50 > U` and breaks the data dependence between statement 1 and statement 2.

However, by using a python script to verify the overlapping between sets of indices of statement 1 and 2, we find that the two indices overlap. This means our banerjee's test, especially the complete test which restrict i1 <= i2, is overly conservative.

```python
# Generate sets of indices accessed by each statement
statement1_indices = set([5590 * i + 94 for i in range(5590)])  # Write indices
statement2_indices = set([5591 * i + 144 for i in range(5590)])  # Read indices

# Find intersection
intersection = statement1_indices.intersection(statement2_indices)

print("Statement 1 (Write) accesses indices:", min(statement1_indices), "to",
max(statement1_indices))
print("Statement 2 (Read) accesses indices:", min(statement2_indices), "to",
max(statement2_indices))
print("Intersection of indices:", intersection)
print("Number of intersecting indices:", len(intersection))

if len(intersection) == 0:
    print("No data dependency exists between Statement 1 and Statement 2")
else:
    print("Data dependency exists! Conflicting indices:", sorted(intersection))
```

Loop 4

```c
for (i=0; i<N; i++) {
   a[(N+1)*i] = b[i]*c[i];
   d[i] = a[(k6+1)*N+i]+e;
}
```

1. Substitute `N=5590, k6 = 559` to the above loop:

```c
for (int i = 0; i < 5590; i++) {
   a[5591*i] = b[i]*c[i];           // statement 1: a1 = 5591, a0 = 0
   d[i] = a[3130400+i]+e;           // statement 2: b1 = 1, b0 = 3130400
}
```

2. Let `s1 = 5591 * i1` and `s2 = 3130400 + i2`

3. Data dependence occurs iff s1 = s2, which means

```
    5591 * i1       = 3130400 + i2 for any 0 ≤ i1, k2 ≤ 5590
⇔  5591 * i1 - i2 = 3130400      (1)
```

For equation (1) to have integral solutions, `gcd(5591, 1) = 1` must divide `3130400`. Since 1 divides any integer, the GCD test doesn't break data dependence between statement 1 and statement 2.

4. Banerjee's Test

`s1 - s2 = 5591 * i1 - 3130400 - i2`

Consider the four corner points:

```
(i1 = 0, i2 = 0)       ⇒ s1 - s2 = 5591 * 0 - 3130400 - 0       = -3130400
(i1 = 0, i2 = 5589)    ⇒ s1 - s2 = 5591 * 0 - 3130400 - 5589    = -3135989
(i1 = 5589, i2 = 0)    ⇒ s1 - s2 = 5591 * 5589 - 3130400 - 0    = 28117699
(i1 = 5589, i2 = 5589) ⇒ s1 - s2 = 5591 * 5589 - 3130400 - 5589 = 28112110
```

Thus, `L = min(-3130400, -3135989, 28117699, 28112110) = -3135989` and `U = max(-3130400, -3135989, 28117699, 28112110) = 28117699`

Note that `b0 - a0 = 3130400 - 0 = 3130400`. Since `L ≤ b0 - a0 ≤ U` (-3135989 ≤ 3130400 ≤ 28122999), this doesn't break the data dependence.

Now let's add constraint `i1 ≤ i2`, which allows us to exclude the point `(i1 = 5589, i2 = 0)` from our test:

```
(i1 = 0, i2 = 0)       ⇒ s1 - s2 = 5591 * 0 - 3130400 - 0       = -3130400
(i1 = 0, i2 = 5589)    ⇒ s1 - s2 = 5591 * 0 - 3130400 - 5589    = -3135989
(i1 = 5589, i2 = 5589) ⇒ s1 - s2 = 5591 * 5589 - 3130400 - 5589 = 28112110
```

Thus, `L = min(-3130400, -3135989, 28112110) = -3135989` and `U = max(-3130400, -3135989, 28112110) = 28112110`

Since `b0 - a0 = 3130400` and `L ≤ b0 - a0 ≤ U` (-3135989 ≤ 3130400 ≤ 28112110), the Banerjee test with the `i1 ≤ i2` constraint still doesn't break the data dependence.