Kelvin Mei
COSI121B
PS4 Write-up

1. To create a stream with integers not divisible by 2, 3, or 5, I just used Stream-filter on integers with the predicate of (not (or (divisible? x 2) (divisible? x 3) (divisible? x 5))).

2. I defined not7 and not 3, then interleaved them and called that weave. To see the results, I called (print-stream weave). The first few elements are:

"1" "1" "2" "2" "3" "4" "4" "5" "5" "7" "6" "8" "8" "10" "9" "11" "10" "13" "11" "14" "12" "16" "13" "17"

3. The alt stream is generated using the integer stream and itself. If alt is 2 elements, it will interleave with 2 elements from the integer stream. Then alt will have 4 and interleave with the next 4 from the integer stream. The zeros are in the 0,2,6,16,30th place in the list and beyond, generated by adding double of the previous amount e.g. +2, +4, +8 and so on. Then the ones are in the positions generated by 0+3, 3+6, 9+12. Twos are in 0+4, 4+8, 12+16 and so on.

4. The first occurrence of an integer k in the i column is $-1 + 2^k$ e.g. (4, 4) will appear in the $-1 + 2^4$ position, which is 15th, assuming the count starts at 1. The second element begins with the number it starts with, so the first occurrence of 4 has to be (4,4). The next occurrence is at the first position + $2^k/2$. Then every element after that is at the second position + 2^k. Each element after the first will be have jth element incremented by 1.

For (1,100):
First occurrence of 1 is Position 1
Second is Position 2 with (1,2)
100th is Position $98(2^k)+2$, where k = 1, which is position 198.
Thus, 197 pairs precede this pair.

For (99,100):
First occurrence of (99,99) is position $-1 + 2^{99}$.
The second occurrence is $-1 + 2^{99} + 2^{98}$.
Thus there are $-2 + 2^{99} + 2^{98}$ pairs that precede this.
For (100,100):
First occurrence of (100,100) is position $-1 + 2^{100}$.
Thus there are $-2 + 2^{100}$ pairs that precede this
5. For merge-weighted, I just took the weights and ordered the pair based on weights.

6.

a → "(1 . 1)" "(1 . 2)" "(1 . 3)" "(2 . 2)" "(1 . 4)" "(2 . 3)" "(1 . 5)" "(2 . 4)" "(3 . 3)" "(1 . 6)" "(2 . 5)" "(3 . 4)" "(1 . 7)" "(2 . 6)" "(3 . 5)" "(4 . 4)" "(1 . 8)" "(2 . 7)" "(3 . 6)" "(4 . 5)"

(print-stream (weighted-pairs integers integers (lambda(x y) (+ x y))))

b → "(1 . 1)" "(1 . 2)" "(2 . 2)" "(1 . 3)" "(2 . 3)" "(3 . 3)" "(1 . 4)" "(2 . 4)" "(3 . 4)" "(1 . 5)" "(4 . 4)" "(2 . 5)" "(3 . 5)" "(4 . 5)" "(1 . 6)" "(2 . 6)" "(3 . 6)" "(5 . 5)" "(4 . 6)" "(5 . 6)" "(1 . 7)" "(2 . 7)" "(3 . 7)" "(4 . 7)" "(6 . 6)" "(5 . 7)" "(1 . 8)" "(2 . 8)" "(3 . 8)" "(6 . 7)" "(4 . 8)"

```
(print-stream (weighted-pairs integers integers (lambda(x y) (+ (* x x x) (* y y y)))))
```

c → "(1 . 1)" "(1 . 7)" "(1 . 11)" "(1 . 13)" "(1 . 17)" "(1 . 19)" "(1 . 23)" "(1 . 29)" "(1 . 31)" "(7 . 7)" "(1 . 37)" "(1 . 41)" "(1 . 43)" "(1 . 47)" "(1 . 49)" "(1 . 53)" "(7 . 11)" "(1 . 59)" "(1 . 61)" "(7 . 13)"

```
(define newstream (weighted-pairs integers integers (lambda(x y) (+ (* 2 x) (* 3 y) (* 5 x y)))))
(define newno235 (stream-filter(lambda (x) (not (or (divisible? (car x) 2) (divisible? (car x) 3)
(divisible? (car x) 5) (divisible? (cdr x) 2) (divisible? (cdr x) 3) (divisible? (cdr x) 5)))) newstream))
(print-stream newno235)
```

7. To combine the same weight, I got the pair-weight of the first element in the stream. Then I returned the (cons-stream of a list generated using the makelist method and recursively called combine-same-weights on a shortened stream using the advstream method.

Makelist takes in a number, a stream, and a procedure. It goes through the stream and applies the procedure to the numbers in the stream. If they match the number then it concatenates the pair into a single list and returns that as the output.

Advstream does the same thing; it takes in a number, stream, and a procedure. However, it returns out a new stream with elements that does not equal the number when the procedure is applied. This is used for the recursive application of combine-same-weights.

8. The first 5 romanujan numbers are 1729, 4104, 13832, 20683, 32832.

9.

a) 5 smallest numbers are:

"(325 (1 . 18) (6 . 17) (10 . 15))"

"(425 (5 . 20) (8 . 19) (13 . 16))"

"(650 (5 . 25) (11 . 23) (17 . 19))"

"(725 (7 . 26) (10 . 25) (14 . 23))"

"(845 (2 . 29) (13 . 26) (19 . 22))"

This is generated via:

```
(define (square x) (* x x))
(define nums (same-weight-pairs integers
                integers
                (lambda (i j) (+ (square i) (square j)))))
(define 3sumsquares (stream-filter (lambda (x) (= (length x) 4)) nums))
(print-stream 3sumsquares)
```

b) 5 smallest numbers are:

"(4901 (1 . 70) (13 . 52))"

"(6209 (5 . 78) (17 . 36))"

"(6427 (3 . 80) (7 . 78))"

"(6849 (5 . 82) (17 . 44))"

"(7731 (11 . 80) (15 . 66))"

This is generated via:

```
(define nums (same-weight-pairs integers
                integers
                (lambda (i j) (+ (cube i) (square j)))))
(define twoway (stream-filter (lambda (x) (= (length x) 3)) nums))
(define icubejsquare (stream-filter (lambda (x) (and (and (= (modulo (caadr x) 2) 1) (= (modulo (cdadr x) 2)
0))(and (= (modulo (caaddr x) 2) 1) (= (modulo (cdaddr x) 2) 0)))) twoway))
(print-stream icubejsquare)
```