```
  Recursive Analysis:

A1+M1*f(M2*n/D1-S1) + M3*f(M4*n/D2-S2)

Assume S1 = S2 = 0, M2 = M4 = 1, and D1 = D2 > 1

Then
In terms of Big-Oh:
  t(1) = 1
    because this is the base step and that takes a
constant amount of time.

  t(n) = 1 + 2*t(n/d)
    where d is just a constant > 1 and the definiti
on 2*f(n/d)
    because the function is called twice in the rec
ursive
    definition.

If you define the function recursively:
  t(n/d) = 1 + 2*t(n/d^2),
  t(n/d^2) = 1 + 2*t(n/d^4),
  t(n/d^4) = 1 + 2*t(n/d^8),

So,
  t(n) = 1 + 2*t(n/d) = 1 + 2*[1 + 2*t(n/d^2)]
       = 1 + 2 + 4*t(n/d^2)
       = 1 + 2 + 4*[1 + 2*t(n/d^4)]
       = 1 + 2 + 4 + 8*t(n/d^4)

    folling the pattern we get
      = 2^k - 1 +2^k *t(n/d^(k-1))

Solve for k,
  n/d^(k-1) = 1
  d^(k-1) = n

  log_d (d^(k-1)) = log_d (n)
  k-1 = log_d (n)
  k = log_d (n) - 1

Plug k back into 2^k - 1 +2^k *t(n/d^(k-1)),
```

```
  2^(log_d (n) + 1) -1 + 2^(log_d (n) + 1) * t(1)


  log_d (n) = log_d (2) * log_2 (n)

  2^(log_d (2) * log_2 (n))
  = (2^(log_2 (n)))^log_d (2)
  = n^(log_d (2))

So the asymptotic time complexity is
  O( n^(log_d (2)) )
```

```
  Vector Amortized Time Complexity Analysis:

The time for copying the data = N*O(1) = O(N)

Let 4^k <= N < 4^(k+1)

The time for a call to push data
  O(1+4+16+32...4^(k+1)) = O(4^(k+2)-1)
  = O(4N) = O(N)

So the amortized time complexity is
  O(1)
```