1. Decrement of Dynamic Array
   Data
   - L = current length
   - array B of some length |B| >= L where |B|/4 <= L <= |B|

   Define $\phi = |2L - |B||$

   Decrement (before decrement |B| = 4L. After decrement |B| = 2L)

   Actual time = time for copying the array + time for other constant operations

   Actual time = O(L+1), so for some constant c, we have the following

   $actual\ time \le 2c * (L + 1)$

   $\phi_{old} = |2L - 4L| = 2L$, $\phi_{new} = |2L - 2L| = 0$

   $\Delta\phi = -2L$

   Amortized time = O(1)

   $A = actual\ time + c * \Delta\phi \le 2c * (L + 1) + c * (-2L) = 2c$

2. Superstack
   Data
   - L = length of the stack
   - k = number of elements to superpop

   Define $\phi = |L|$ where L = length of the stack

   for create, pop, push operations

   O(1) actual time

   $\Delta\phi = 0$

   O(1) amortized time

   for superpop

   O(k) actual time for popping k elements from stack to A

   $\Delta\phi = \phi_{new} - \phi_{old} = L - (L - k) = k$

   amortized time = $c * k + c * k = 2c * k$ = O(1)

3. New Superstack
   Yes, for superpush

   O(k) actual time for pushing k elements from A to the stack

   $\Delta\phi = \phi_{new} - \phi_{old} = (L + k) - L = k$

   amortized time = $c * k + c * k = 2c * k$ = O(1)

4. Binary Counter
   $\phi = |L|$ where L = number of rightmost 1-bits.

   for increase

   O(N) actual time where N = number of total bits

   $\Delta\phi = \phi_{new} - \phi_{old}$

   If $\phi_{old} = 0$, then $\phi_{new} = 1$. $\Delta\phi = 1$

   If $\phi_{old} = L > 0$, then $\phi_{new} = 0$. $\Delta\phi = -L$

   amortized time = c*N+c*(-L) = c*(N-L) = O(N)

5. Dynamic Arrays
   To remove A[i] in constant time, exchange A[i] with the last element in the array. Then A[i] becomes the last element in the array and we can remove it with constant time. Pseudocode as the following:

```
tmp = A.get(end_idx)
A.set(end_idx, A[i])
A.set(i, tmp)
A.decrement()
```

6. Binary Counter with Decrement
   a. From right to left, we look at the current bit. If the current bit is 0, we change it to 1 and go to its left bit. If the current bit is 1, we change it to 0 and stop.
   b. Assume 2^n possible bit patterns are equally likely for n-bit counter
      expected time for decrement

$$s = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + ... + n \cdot \frac{1}{2^n}$$

$$S = 2S - S = 1 + \frac{1}{2} + \frac{1}{4} + ... + \frac{1}{2^{n-1}} - \frac{n}{2^n} = 2 - \frac{1}{2^{n-1}} = o(1)$$

7. Overlapping Rectangles
   We will need a list of rectangles ordered by its x_L in ascending order so that we can sweep from left to right. We also need a list to keep track of what rectangles we are sweeping currently. For each rectangle R in the sorted array. We compare R with every rectangle SR in the sweeping array. If SR's x_R is not on the right of the R's x_L, meaning SR is not being sweeped anymore. We remove it from the sweeping list. If R and SR overlap, meaning their x and y both overlaps, we return True. After we finish comparing every R in the sorted rectangle list, we know we didn't encounter any overlapping rectangles and return False.

```
def overlap(rs):        # rs = array of rectangles ordered by its x_L ascending
    srs = None          # rectangles that are currently sweeping (initially None)
    for r in rs:
        for sr in srs:
            if sr.x_R <= r.x_L:
                srs.remove(sr)
            elif sr.y_T > r.y_B or sr.y_B < r.y_T:
                return True
        srs.add(r)
    return False
```