# Daily Coding Problem #141

## Problem

This problem was asked by Microsoft.

Implement 3 stacks using a single list:

```
class Stack:
    def __init__(self):
        self.list = []

    def pop(self, stack_number):
        pass

    def push(self, item, stack_number):
        pass
```

## Solution

In order to implement 3 stacks using one list, we separate the list into 3 separate parts, one for each stack.

- The first stack starts from `list[0]` and grows up.
- The second stack starts from `list[len(list) / 2]` and grows up.
- The third stack starts from `list[len(list) - 1]` and grows down.

In order to know where to put the next `item` on `push`, we store three pointers `s0`, `s1`, and `s2` for the above three stacks.

When one of the stacks is about to overwrite another stack, we `resize` the stack, similar to how

lists themselves are resized.

```
class Stacks:
    def __init__(self):
        self.size = 10
        self.list = [None] * self.size
        self.s0 = 0 # Grows up
        self.s1 = len(self.list) / 2 # Grows up
        self.s2 = len(self.list) - 1 # Grows down

    def pop(self, stack_number):
        if stack_number == 0:
            self.s0 -= 1
            return self.list[self.s0]
        elif stack_number == 1:
            self.s1 -= 1
            return self.list[self.s1]
        else:
            self.s2 += 1
            return self.list[self.s2]

    def push(self, item, stack_number):
        if stack_number == 0:

            self.list[self.s0] = item
            self.s0 += 1
        elif stack_number == 1:
            self.list[self.s1] = item
            self.s1 += 1
        else:
            self.list[self.s2] = item
            self.s2 -= 1

        if self.is_resize_needed():
            self.resize(self.size * 2)

    def is_resize_needed(self):
        return self.s0 == len(self.list) / 2 or self.s1 > self.s2

    def resize(self, size):
        prev_list = self.list
        prev_s0 = self.s0
        prev_s1 = self.s1
        prev_s2 = self.s2
```

```
        self.list = [None] * size
        self.s0 = 0 # Grows up
        self.s1 = len(self.list) / 2 # Grows up
        self.s2 = len(self.list) - 1 # Grows down
        self.size = size

        for i in range(prev_s0):
            self.push(prev_list[i], 0)

        for i in range(len(prev_list) / 2, prev_s1):
            self.push(prev_list[i], 1)

        for i in reversed(range(prev_s2 + 1, len(prev_list))):
            self.push(prev_list[i], 2)
```