## Segregate Odd and Even Numbers

### Approach 1

Two pointer approach that creates a single partition

def swap(numbers, i, j):

tmp = numbers[i]

numbers[i] = numbers[j]

numbers[j] = tmp

def segregate\_evens\_and\_odds(numbers):

n = len(numbers)

if n < 2:

return numbers

even = 0

for i in range(n):

if numbers[i] % 2 == 0:

swap(numbers, even, i)

even += 1

return numbers

### Approach 2

Two pointer approach that creates two partitions

def swap(numbers, i, j):

tmp = numbers[i]

numbers[i] = numbers[j]

numbers[j] = tmp

def segregate\_evens\_and\_odds(numbers):

n = len(numbers)

if n < 2:

return numbers

left = 0

right = n - 1

while left < right:

if numbers[left] % 2 == 0:

left += 1

elif numbers[right] % 2 == 1:

right -= 1

else:

swap(numbers, left, right)

left += 1

right -= 1

return numbers

## Dutch National Flag

Three pointers approach

def swap(balls, i, j):

tmp = balls[i]

balls[i] = balls[j]

balls[j] = tmp

def dutch\_flag\_sort(balls):

n = len(balls)

if n < 2:

return balls

left = 0

right = n - 1

i = 0

while i <= right:

if balls[i] == "R":

swap(balls, left, i)

left += 1

i += 1

elif balls[i] == "B":

swap(balls, i, right)

right -= 1

else:

i += 1

return balls

## 3 Sum

def find\_zero\_sum(A):

n = len(A)

if n < 3:

return []

A.sort()

result = []

i = 0

while i < n - 2 and A[i] <= 0:

j = i + 1

k = n - 1

while j < k:

triplet\_sum = A[i] + A[j] + A[k]

if triplet\_sum > 0:

k -= 1

elif triplet\_sum < 0:

j += 1

else:

result.append("{},{},{}".format(A[i], A[j], A[k]))

j += 1

k -= 1

# Skipped over all duplicate A[j]'s

while j < k and A[j] == A[j-1]:

j += 1

# Skipped over all duplicate A[k]'s

while j < k and A[k] == A[k+1]:

k -= 1

i += 1

# Skipped over all duplicate A[i]'s

while i < n - 2 and A[i] == A[i-1] :

i += 1

return result