def calculator(x,y,z):  
 result = 0 # 결과값을 0 으로 초기화  
 # 연산자가 +,-,/,\* 일 때를 구분하여 연산을 진행후 반환한다  
 if z == '+': result =x+y  
 elif z == '-': result =x-y  
 elif z == '\*': result =x\*y  
 else: result =x/y  
 return result  
while True: # 파이썬에는 do-while 이 없기 때문에 while True와 break 를 통해 0 0 0 입력시 탈출 할 수 있도록 해준다  
 print('수식 (0 0 0 입력 시 종료)?',end='')  
 x,z,y = input().split() # 파이썬의 기본 입력 형태는 문자열이다. 문자열로 x,y,z를 받아온다  
 if x=='0' and y=='0' and z=='0': break # 만약 x,y,z 모두 0이라면 종료 되도록 break  
 print(calculator(float(x),float(y),z)) # 문자열x,y 를 실수형 float으로 변환후 함수로 넘겨준다  
  
def calculator2(s):  
 return eval(s) # eval 함수는 연산자와 수로 이루어진 문자열을 자동 계산하여 반환한다  
while True:  
 print('수식 (0 0 0 입력 시 종료)?',end='')  
 s = input(); # 문자열 s 를 입력받는다  
 if s=='0 0 0': break # 문자열 s 가 0 0 0 이라면 종료되도록 하고 아니라면 아래 수행  
 print(calculator2(s)) # 계산된 값을 반환한다

리뷰: 우선 연산자와 2개의 피연산자를 '인자로' 즉 매개변수로 받아와 사칙 연산을 수행해야 하는것이 요구사항이기에 두번째 eval 함수를 이용한 방법은 요구사항을 충족하지 못한다 생각됩니다. 파이썬 내에서는 공백을 기준으로 여러 변수를 입력받을 때 기본 default input을 받을 때 문자 형으로 받게 됩니다. 그렇기에 수식의 계산을 위해 함수로 매개변수를 줄때 float(실수형)으로 형변환을 거친 뒤 계산에 들어갔습니다. (+,-,\*,/) 네번에 거쳐 경우의 수를 나누는 것 외에 다른 방법이 없었을까 아쉬웠습니다.

oneCnt = 0  
zeroCnt = 0  
def quadTree(x,y,n):  
 target = arr[x][y] # 타겟은 지정된 배열 영역 i,j 안의 첫번째 요소이다  
 global oneCnt # 재귀호출을 하기 때문에 Cnt를 0으로 설정하면 다음 호출시 0 이되어버리기 때문에  
 # global 을 통하여 함수 밖에서 초기값 0을 설정하여 전역 변수로 사용한다  
 global zeroCnt  
 # 만약 타겟이 지정된 영역에서 모두 같다면 영역을 4분할 할 필요가 없다  
 flag = True # 만약 지정한 영역 안에서 모두 같은 값일 경우  
 ######## 영역 순회 ( 지정된 영역을 돌면서 같은 값만 있는지 판단 ) ##############  
 for i in range(n): # n 은 매 반복마다 영역 S 의 길이가 된다  
 for j in range(n):  
 if target != arr[i+x][j+y]: # 타겟과 영역안의 수가 다르다면  
 flag = False # 지정 영역 안에서 다른 값이 존재한다  
 break # 더이상 탐색할 필요가 없기 때문에 종료  
 if not flag: break # 만약 flag 가 False 라면 종료 되도록  
 # (위 break를 통해 안쪽 for 문을 탈출 하였지만 바깥 for문도 더이상 탐색할 필요가 없기 때문에 종료한다)  
  
 if flag: # 만약 영역안에 존재하는 수가 모두 같을 때  
 if target == 1: oneCnt+=1 # 해당 영역은 모두 1로 채워져있기 때문에 하나의 큰 1로  
 else: zeroCnt+=1 # 하나의 큰 0으로 채워져있기 때문에 큰 0으로  
 else: # 지정 영역 안에서 다른 값이 존재한다면 4분할 한다  
 n=n//2 # n을 반으로 나눠준다  
 quadTree(x,y,n) # 1사분면  
 quadTree(x,y+n,n) # 2사분면  
 quadTree(x+n,y,n) # 3사분면  
 quadTree(x+n,y+n,n) # 4사분면  
 # 최종적으로 n=1 이 될때 2로 나눈 몫은 0 이므로 위의 for 문을 수행하지 않고 flag는 True인  
 # 상태로 내려와 타겟값의 값에 따라 카운트가 증가된다. ex) quadTree(0,0,0) 을 수행하였을때  
 # if flag 문에서 target 에 따라 카운트가 증가된다 이후 모든 재귀적 호출이 끝난뒤 마지막에  
 # 0의 개수와 1의 개수를 리스트로 묶어 반환하도록 했다  
 answer = [zeroCnt,oneCnt]  
 return answer  
  
n = int(input()) # 배열의 크기  
arr = [list(map(int, input().split())) for \_ in range(n)] # n\*n의 배열을 공백을 기준으로 입력 받는다  
# print(\*arr)  
print(quadTree(0,0,n)) # 초기값은 0,0 부터 n,n 까지 진행된다 (이때 n은 배열의 크기이다)

리뷰 : DFS 나 BFS 를 통하여 푸는 문제인줄 알고 다소 불필요한 수행과정을 겪었지만 지정한 영역을 분할과 정복을 통하여 나눈뒤 각 영역에 대하여 같은 수를 가진 경우를 하나의 "덩이"라고 보고 이 값들을 모두 더하였습니다. 우선 어떤 방식으로 탐색을 해야 할지 몰라 몇가지 분할과 정복에 대한 개념적 설명들을 이해한 뒤 그것을 바탕으로 이 문제에서 제시한 4분할을 통하여 1~4사분면 까지 나눈뒤 모두 같은 값을 이루고 있을 때 하나의 "덩이"라고  
보고 문제에 접근하게 되었습니다. 마지막에 탈출 조건에 대해 처음에는 따로 설정을 해야하는 것인가 고민을 하였지만n이 1일때 2로 나눈 몫이 0 이기 때문에 자연스럽게 재귀호출을 멈출 수 있다 판단하였고 마지막에 정답을 반환하도록 구현해 보았습니다. 모든 경우를 탐색하다 보니 브루트포스 같이 효율성이 떨어지는 것처럼 보여서 어떻게 최적화를 할지 조금 더 고민하고 개선 해야 할 것 같습니다.