Understanding the problem statement

Consider an array such that 1<=arr[i]<=n i.e all the elements are in the range of [1,n]. All the elements in the array occur once except for two repeating elements. suppose if n=7 then elements will be like [1,2,3,4,5,1,3]. In the example 1 and 3 are repeated, but any elements could repeat but the range will be from 1 to n-2. Actual range 1<=arr[i]<=n-2. Input : [1,2,3,4,5,1,3] Output : [1,3]

Approach 1

Find all possible pairs out of n elements (nc2) using two for loops and check if they are repeating or not.

Time Complexity : O(n^2)

Space Complexity : O(1)

Approach 2

1. Maintain a list of size n.
2. Traverse the given array and increment value at that index in the list. If 1 is repeated increment count at list[1] i.e [0,1,0,0,0]
3. Traverse the list and print the indexes where count = 2. [1,2,1,1,2]

Time Complexity: O(n)

Space Complexity: O(n)

Approach 3

1. Find the sum of the given array S.
2. Find the sum of first (n-2) natural numbers using n(n+1)/2 formula S1.
3. Compute the difference S-S1 which gives sum of repeating elements. (X + Y)
4. Find the product of the given array. P
5. Find the product of first (n-2) natural numbers using n!. P1
6. Compute the division P/P1 which gives product of repeating elements.(X * Y)
7. Compute the difference of repeating elements using (X-Y) = sqrt(((X+Y)^2)- 4XY)
8. We have X+Y and X-Y we can compute X and Y.
9. Although the time complexity is O(n) we dont use this approach because this involves too multiplcations and multiplication is computationally expensive.

Time Complexity : O(n)

Space Complexity : O(1)

Approach 4

1. Find the xor of all elements in the given array. [1,2,3,4,1,2]
2. Xor the obtained result with the first (n-2) natural numbers to obtain the xor of repeating elements. (X^Y). Xor with same element cancels out each other. (1^2^3^4^1^2)^(1^2^3^4) = 1^2
3. Now find the right most set bit in X^Y using formula a & ~(a-1).
4. Divide the given array based on the right most set bit. Now the two repeating elements goes into two different lists. [1,1,3],[2,2,4]
5. Also divide first n-2 natural numbers based on right most set bit. [1,3],[2,4]
6. Now xor the divided natural numbers list and divided array list to get repeating elements [1,1,3] ^ [1,3] = [1] [2,2,4] ^ [2,4] = [2]
7. [1,2] are repeating elements

Time Complexity : O(n)

Space Complexity : O(1)

Implementation

In [3]:
```
def check_set(n,k):
    if (1<<k)&(n):
        return 1
    return 0

def findRepeating(arr,size):
    arr_xor = 0
    for i in range(size):
        arr_xor = arr_xor ^ arr[i]
    for i in range(1,size-1):
        arr_xor = arr_xor ^ i
    set_bit = arr_xor & (~(arr_xor -1))
    first_num = 0
    second_num = 0
    for i in range(size):
        if check_set(arr[i],set_bit):
            first_num = first_num ^ arr[i]
        else:
            second_num = second_num ^ arr[i]
    for i in range(1,size-1):
        if check_set(i,set_bit):
            first_num = first_num ^ i
        else:
            second_num = second_num ^ i
    return [first_num,second_num]
```

In [4]:
```
arr = [1,2,3,4,1,2]
size = len(arr)
print(findRepeating(arr,size))
# ans
# [2, 1]
```

```
[2, 1]
```

In [6]:
```
arr = [1,2,2,3,1,4]
size = len(arr)
print(findRepeating(arr,size))
```

```
# ans
# [2, 1]
```

```
[2, 1]
```

In [7]:
```
arr = [1,3,2,3,5,4,5]
size = len(arr)
print(findRepeating(arr,size))
# ans
# [5, 3]
```

```
[5, 3]
```

Approach 4

1. This is similar to finding duplicates program in O(n) time and O(1) extra space.
2. Iterate through the array and make integer at index arr[i] to negative if it is positive. If it is already negative it means we already have arr[i] at some other place in array.
3. So arr[i] is considered as repeated.

In [ ]:
```

```