HCF / GCD

In [388]:
```
1
2
3
```

In [500]:
```
1  a = 15
2  b = 12
3
4  n = 2
5
6  def gcd(a,b):
7
8      for i in range(min(a,b),0,-1):
9          if a%i == 0 and b%i == 0:
10             return (i)
11
12 gcd(a,b)
```

Out[500]: 3

In [471]:
```
1  gcd(32,64)
```

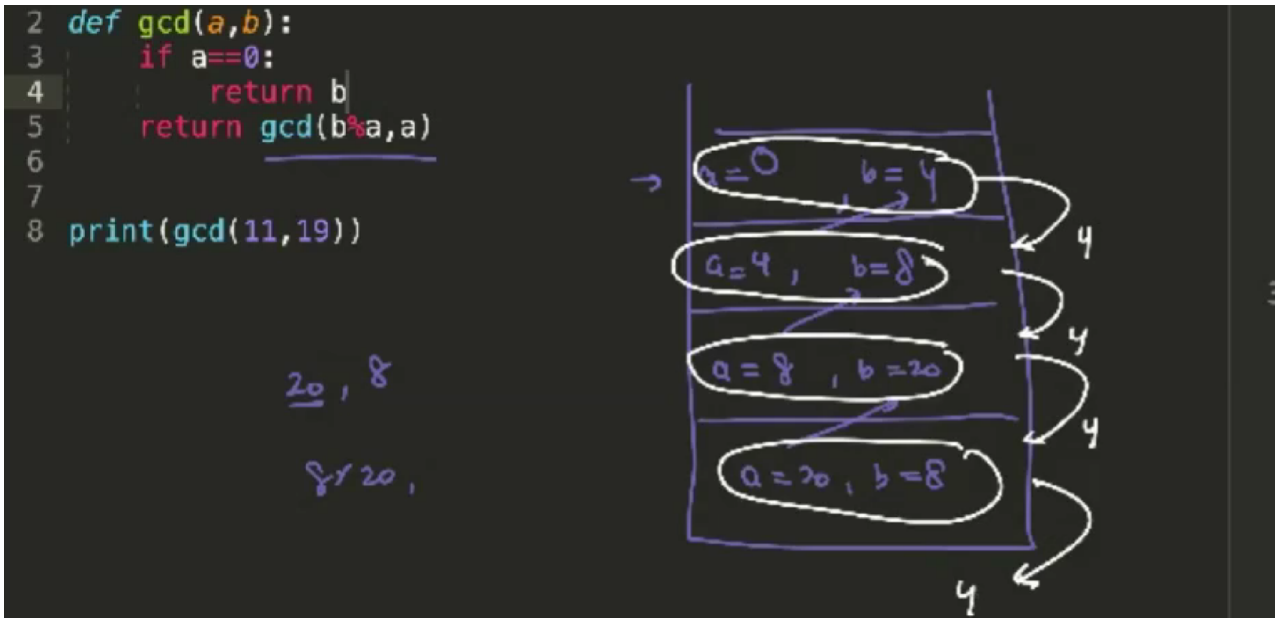
Out[471]: 32

In [479]:
```
1  math.gcd(16,64,36)
```

Out[479]: 4

In [478]:
```
1  a = 16
2  b = 64
3  c = 36
4
5
6  def gcd(a,b,c):
7
8      for i in range(min(a,b,c),0,-1):
9          if a%i == 0 and b%i == 0 and c%i == 0:
10             return (i)
11
12 gcd(a,b,c)
```

Out[478]: 4



In [481]:
```
1  def gcd(a,b):
2      if a == 0:
3          return b
4      return gcd(b%a,a)
5  gcd(50,64)
```

Out[481]: 2

In [489]:
```
A = [16,16,8,16]

def gcd_arr(A):
    def gcd(a,b):
        if a == 0:
            return b
        return gcd(b%a,a)
    ans = gcd(A[0],A[1])
    for i in range(2,len(A)):
        ans = gcd(ans,A[i])
    return ans

gcd_arr(A)
```

Out[489]: 8

In [492]:
```
math.gcd(16,16,8,16)


```

Out[492]: 8

In [ ]:
```

```

In [494]:
```
def lcm(a,b):
    def gcd(a,b):
        if a == 0:
            return b
        return gcd(b%a,a)
    return (a*b)/gcd(a,b)


lcm(15,12)
```

Out[494]: 60.0

In [504]:
```
a = 12
b = 15
m = 1
while b*m % a != 0:
    m = m+1
print(b*m)
```

```
60
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

isprime

In [25]:
```
def isprime(n):
    i = 1
    f = 1

    while i*i <= n:
        if n % i == 0:
            f += 1
        i += 1
    return f == 2

```

In [35]:
```
isprime(10**12+8)
```

Out[35]: False

In [498]:
```
def isprime2(n):
    i = 2

    while i*i <= n:
        if n % i == 0:
            return False
        i+=1
    return True
```

In [56]:
```
isprime2(10**9+7)
```

Out[56]: True

```
In [499]: 1 isprime2(1)
```

```
Out[499]: True
```

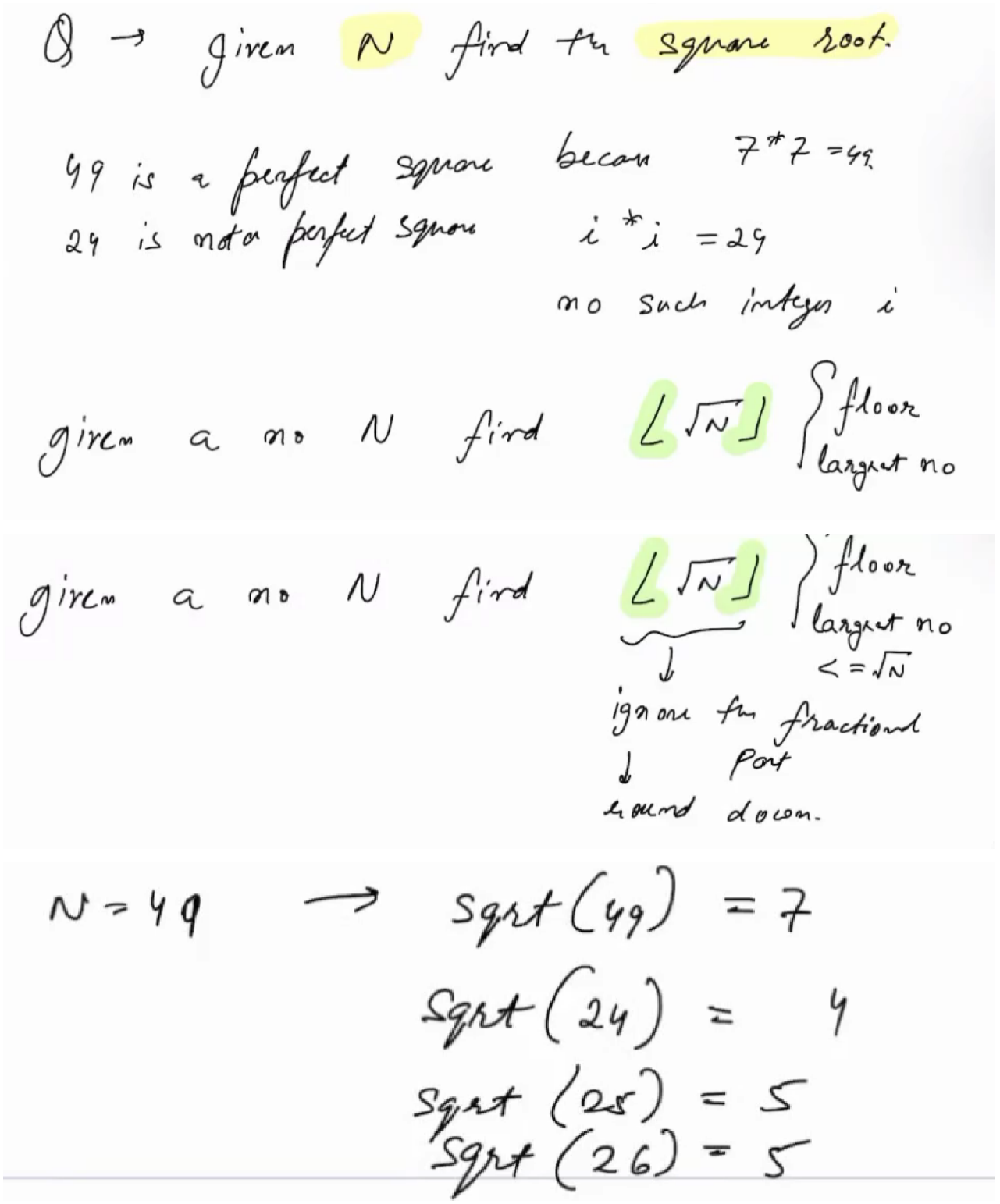
```
In [ ]: 1
```

```
In [57]: 1 from math import *
```

```
In [59]: 1 sqrt(60)
```

```
Out[59]: 7.745966692414834
```

Q!

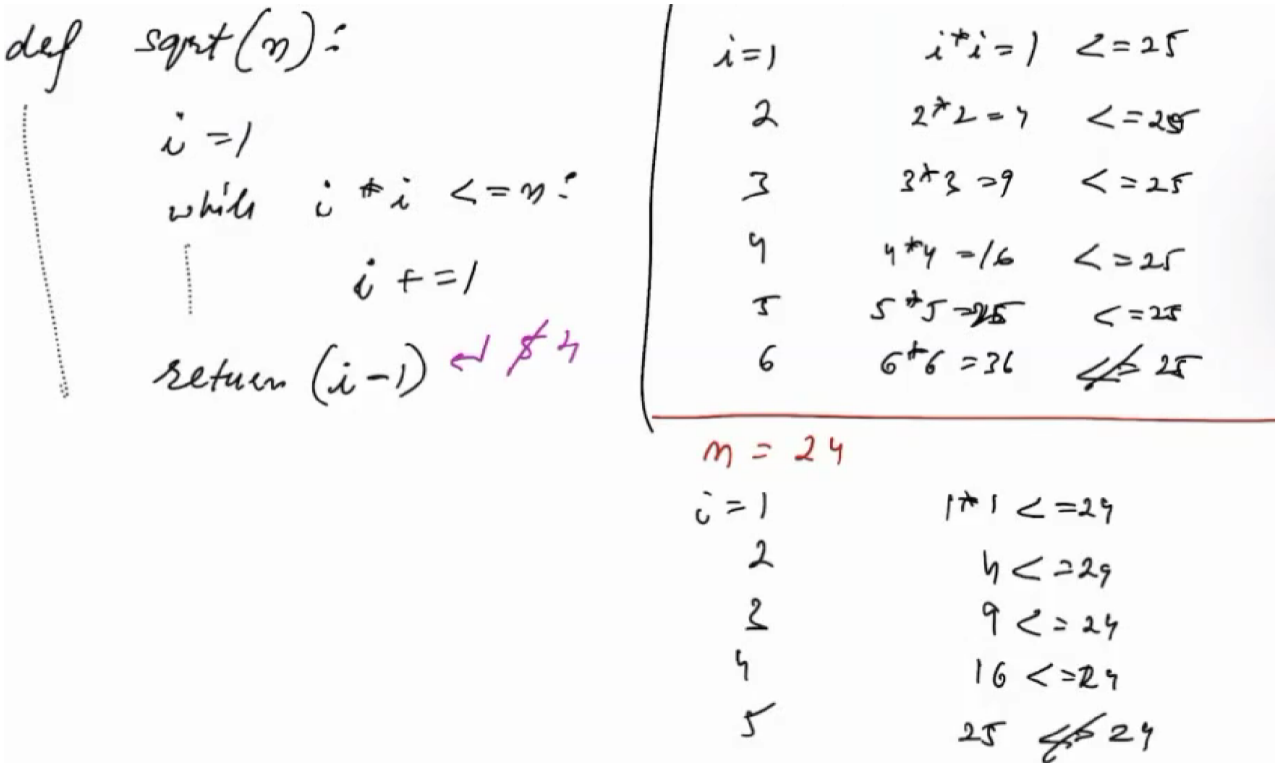


In [513]:
```
def perfect_square(n):
    i = 1
    for i in range(1,n+1):
        if i*i == n:
            return i

perfect_square(24)
```

In [506]:
```
def perfect_square(n):
    i = 1
    while i*i <= n:
        i += 1
    return i-1
```

In [507]:
```
perfect_square(17)
```

Out[507]: 4

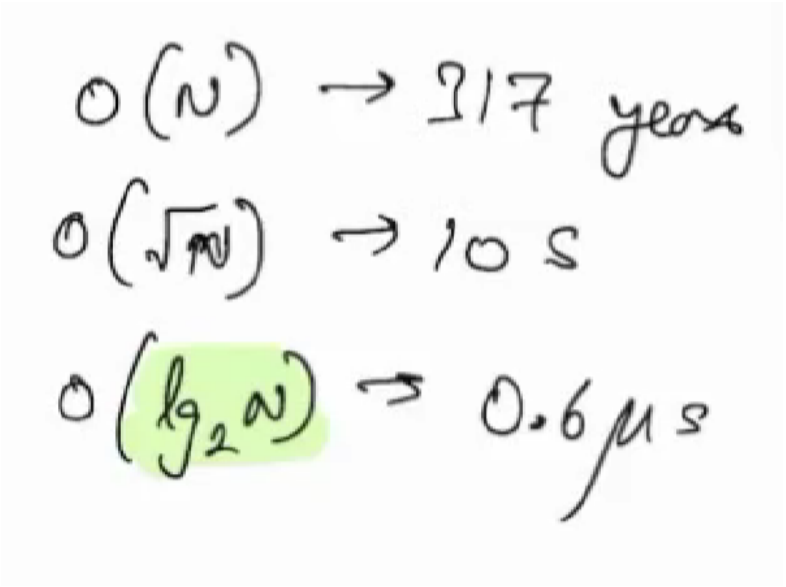


In [ ]:
```

```

In [161]:
```
def perfect_square(n):

    low = 1
    high = n
    mid = (low + high)//2

    while low <= high:
        mid = (low + high)//2
        if mid * mid > n:
            high = mid - 1

        elif mid * mid < n:
            low = mid + 1
        else:
            return mid
    return high


perfect_square(26)

```

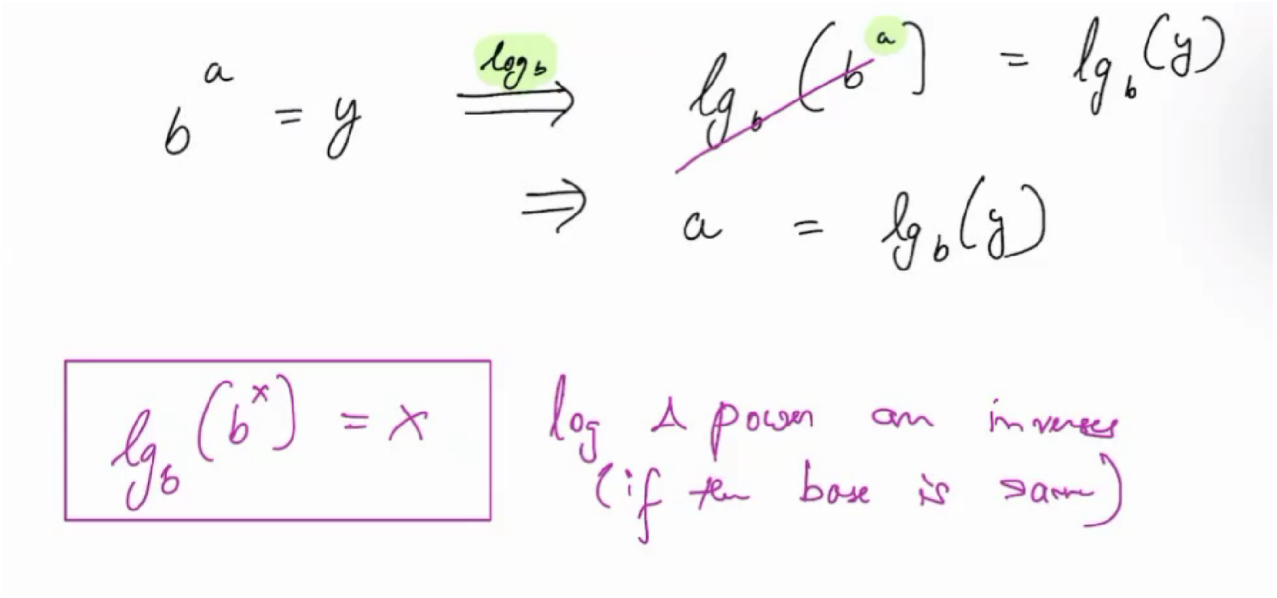
Out[161]: 5

In [ ]:
```

```

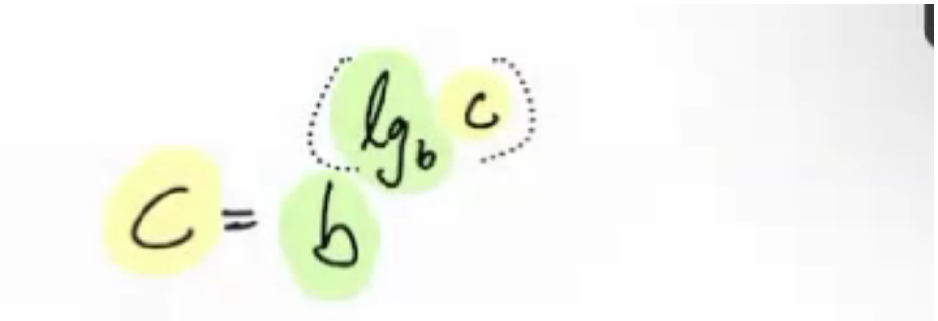

O(N) → 317 years
O(√N) → 10 s
O(log2 N) → 0.6 μs

In [ ]:
```
1
```

In [ ]:
```
1
```


b^a = y
log_b
log_b (b^a) = log_b (y)
⇒ a = log_b (y)
log_b (b^x) = x
log of a power and inverses (if the base is same)


log_b y = x ⇒ b^x = y


C = b^(log_b c)

$$b^x = c^{x/(\log_b c)}$$

$$2^5 = 3^{5/\log_2 3}$$

$$\log_3 1000 = (\log_3 7)\,\log_7 1000$$

$$\log_c a = \log_b a * \log_c b$$

In [ ]:
```
1
```

In [ ]:
```
1
```

$$2^{46} = 10^{46/\log_2 10} = 10^{46/3.322}$$

$$= 14$$

$14+1 = 15$ digits

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

$$\lg(n!) = O(n \lg n) \Rightarrow \text{Stirling's approximation}$$

In [ ]:
```
1
```

In [ ]:
```
1
```

Identities

AP $= a, \quad a+d, \quad a+2d \quad \cdots \quad a+(n-1)d$

$$S_n^{AP} = (\text{first} + \text{last})\frac{n}{2} = \frac{n}{2}\left(2a + (n-1)d\right)$$

GP $= a, \quad ar, \quad ar^2 \quad \cdots \quad ar^{n-1}$

$$S_n^{GP} = a \cdot \frac{r^n - 1}{r - 1}$$

$$S_\infty^{GP} = a \cdot \left(\frac{1 - r^n}{1 - r}\right) \quad r < 1$$

In [ ]:
```
1
```

$$1 + 2 + 3 + \cdots n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + \cdots n^2 = \frac{n(n+1)}{2}\cdot\frac{(2n+1)}{3} = \frac{n(n+1)(2n+1)}{6}$$

$$1^3 + 2^3 + 3^3 + \cdots n^3 = \left(\frac{n(n+1)}{2}\right)^2 = \left(1 + 2 + 3 + \cdots n\right)^2$$

$$1^k + 2^k + \cdots n^k = O\left(n^{k+1}\right)$$

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

append / prepend

insertion at end O(1) in appeding

insertion at start position O(n) prepeding

In [106]:
```
1 sqr = []
2
3 for i in range(1,11):
4     sqr.append(i*i)
5 print(sqr)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In [110]:
```
1 sqr = []
2 index = []
3 idx = 0
4 for i in range(1,11):
5     index.append(idx)
6     sqr.insert(0,i*i)
7     idx += 1
8 print(sqr)
9 print(index)
```

```
[100, 81, 64, 49, 36, 25, 16, 9, 4, 1]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [108]:
```
1 sqr
```

Out[108]:
```
[100, 81, 64, 49, 36, 25, 16, 9, 4, 1]
```

In [109]:
```
1 len(sqr)
```

Out[109]:
```
10
```

In [ ]:
```
1
```

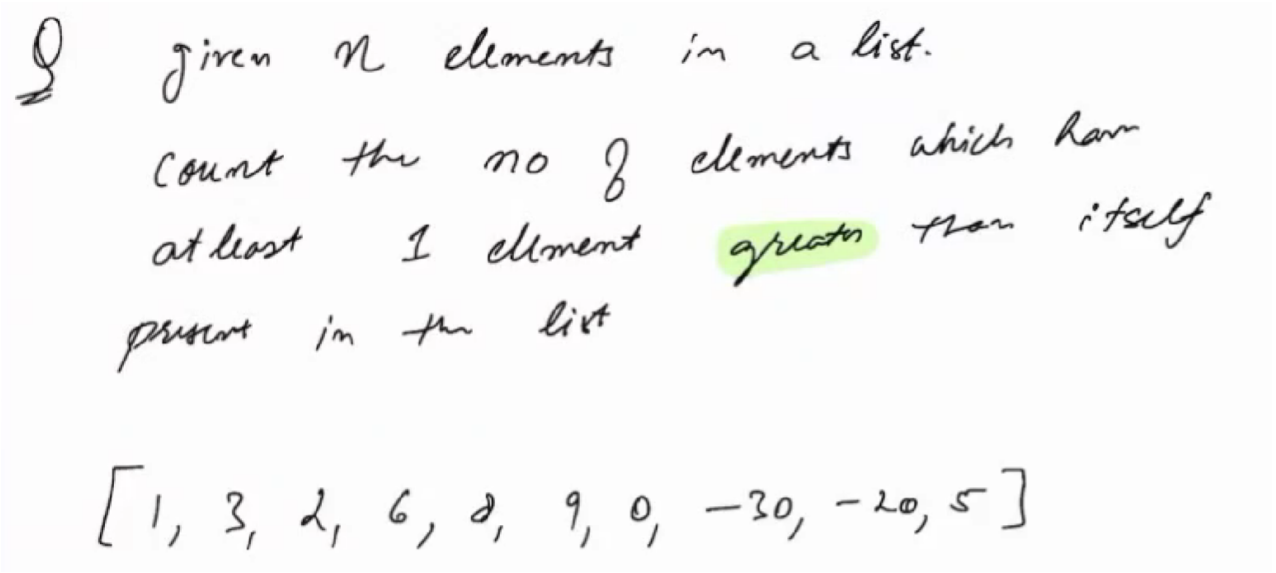
In [111]:
```
1 n = len(sqr)
```

In [114]:
```
1 summation = 0
2
3 for i in range(n):
4     print(summation," ",sqr[i]," ",i)
5     summation += sqr[i]
6
7 print(summation)
```

```
0   100   0
100   81   1
181   64   2
245   49   3
294   36   4
330   25   5
355   16   6
371   9   7
380   4   8
384   1   9
385
```

In [ ]:
```
1
```

Q!



In [117]:
```
A = [2,5,1,4,8,0,8,1,3,8]
n = len(A)

maxn = -float('inf')

for x in A:
    if x > maxn:
        maxn = x


ans = 0
for i in range(n):
    if A[i] < maxn:
        ans += 1
print(ans)
```

7

In [118]:
```
A = [2,5,1,4,8,0,8,1,3,8]
n = len(A)
c = 0
for v in A:
    haslarger = False
    for x in A:
        if x > v:
            haslarger = True
    if haslarger :
        c += 1
print(c)
```

7

In [120]:
```
def greaterthanitself(A):
    n = len(A)

    maxn = -float('inf')

    for x in A:
        if x > maxn:
            maxn = x
    c = 0
    for v in A:
        if v == maxn:
            c += 1
    return n- c
A = [2,5,1,4,8,0,8,1,3,8]
greaterthanitself(A)
```

Out[120]: 7

In [ ]:
```

```

In [ ]:
```

```

reverse the list

In [121]:
```
def revlist(A):
    rev = []
    n = len(A)
    for i in range(n-1,-1,-1):
        rev.append(A[i])
    return rev
```

In [123]:
```
A = [1,10,3,6,8]
revlist(A)
```

Out[123]: [8, 6, 3, 10, 1]

In [ ]:
```

```

In [134]:
```
def revlist(A):
    n = len(A)

    for i in range(n//2):
        A[i],A[n-i-1]  = A[n-i-1],A[i]
    return A

A = [1,10,3,6,8,-99]
print(A)
ans = revlist(A)
print(ans)
```

```
[1, 10, 3, 6, 8, -99]
[-99, 8, 6, 3, 10, 1]
```

In [ ]:
```

```

In [ ]:
```

```

Problem Description

You are given **N** positive integers.

For each given integer **A**, you have to tell whether it is a **perfect number** or not.

Perfect number is a positive integer which is equal to the sum of its proper positive divisors.

In [139]:
```
def divisors(n):
    ans = []
    i = 1
    while i < n:
        if n % i == 0:
            ans.append(i)
        i += 1
    return ans
divisors(6)

```

Out[139]: [1, 2, 3]

In [147]:
```python
def isperfect(n):

    ans = []
    i = 1
    sm = 0
    while i < n:
        if n % i == 0:
            sm += i
        i += 1

    if sm == n:
        return "perfect number"
    return False

isperfect(6)

```

```
6
```

Out[147]: 'perfect number'

In [ ]:
```python

```

In [ ]:
```python

```

In [ ]:
```python

```

Given a number **A**. Return square root of the number if it is perfect square otherwise return -1.

In [515]:
```python
def perfect_square(n):

    low = 1
    high = n
    mid = (low + high)//2

    while low <= high:
        mid = (low + high)//2
        if mid * mid > n:
            high = mid - 1

        elif mid * mid < n:
            low = mid + 1
        else:
            return mid
    return None

def sqrt_num_if_perfect_sq(n):
    if perfect_square(n) != None:
        return perfect_square(n)
    return -1

sqrt_num_if_perfect_sq(25)
```

Out[515]: 5

In [ ]:
```python

```

Problem Description

You are given an integer **N** you need to print all the **Armstrong Numbers** between **1** to **N**.

If sum of cubes of each digit of the number is equal to the number itself, then the number is called an Armstrong number.

For example, 153 = (1 * 1 * 1) + (5 * 5 * 5) + (3 * 3 * 3).

In [174]:
```
def isarmstrong(n):
    num = n
    total = 0
    while n > 0:
        digit = n % 10
        total += digit ** len(str(num))
        n = n//10
    return total == num

print(isarmstrong(407))

for i in range(1,10000):
    if isarmstrong(i):
        print(i)
```

```
True
1
2
3
4
5
6
7
8
9
153
370
371
407
1634
8208
9474
```

In [ ]:

In [ ]:

Problem Description

Given an integer array **A** and an integer **B**, you have to print the same array after rotating it **B** times towards the right.

NOTE: You can use extra memory.

```
[1,2,3,4] => [4,1,2,3] => [3,4,1,2]
```

In [189]:
```
A = [1,2,3,4]
B = 1560405406878
itr = B % len(A)
for i in range(itr):
    A.insert(0,A[-1])
    A.pop()
A
```

Out[189]: [3, 4, 1, 2]

In [187]:
```
1560405406878%4
```

Out[187]: 2

Little Ponny is given an array, **A**, of **N** integers. In a particular operation, he can set any element of the array equal to **-1**.

He wants your help in finding out the minimum number of operations required such that the maximum element of the resulting array is **B**. If it is not possible, then return **-1**.

In [192]:
```
A = [2,4,3,1,5]
B = 3
def ponny_and_max_ele(A,B):
    c = 0
    if B not in A:
        return -1
    for i in A:
        if i > B:
            c += 1
    return(c)

ponny_and_max_ele(A,B)
```

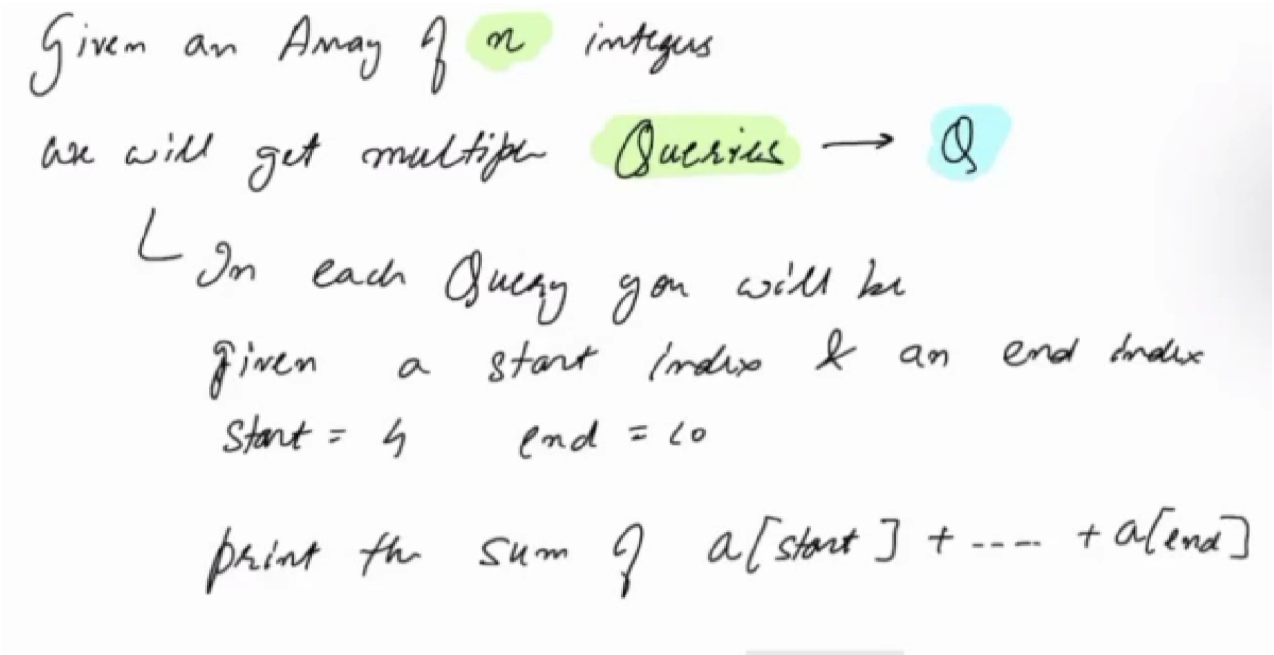
Out[192]: 2

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

>> Prefix Array



In [227]:

```
def getsm(A,start,end):

    n = len(A)

    prefixsm = [A[0]]
    for x in A[1:]:
        prefixsm.append(x+prefixsm[-1])

    if start == 0:
        return prefixsm[end]
    else:
        return prefixsm[end]-prefixsm[start-1]




def query_sm(A,Q):

    while Q > 0:
        Q -= 1
        start = int(input("Start Index : "))
        end = int(input("End Index : "))
        if start < 0 or start > len(A) or end > len(A)-1 or start >= end:
            print("Invalid Input")

        print("Ans: ",getsm(A,start,end))


Q = int(input("Number of queries : "))
A = list(map(int,input("Enter Array : ").split()))
query_sm(A,Q)
```

```
Number of queries : 2
Enter Array : 1 1 3 4 5 8 5 8 2 1 4 5
Start Index : 1
End Index : 5
Ans:  21
Start Index : 2
End Index : 3
Ans:  7
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

equilibrium index in the array:

You are given an array **A** of integers of size **N**.

Your task is to find the equilibrium index of the given array

The equilibrium index of an array is an index such that the sum of elements at lower indexes is equal to the sum of elements at higher indexes.

NOTE:

Array indexing starts from 0.
If there is no equilibrium index then return -1.
If there are more than one equilibrium indexes then return the minimum index.

In [262]:
```
def equilibrium_index(A):

    n = len(A)

    prefix = [A[0]]
    for x in A[1:]:
        prefix.append(x+prefix[-1])

    ans = []
    for i in range(n):

        if i == 0:
            leftsum = 0
        else:
            leftsum = prefix[i-1]

        rightsum = prefix[n-1]-prefix[i]

        if leftsum == rightsum:
            ans.append(i)
    if len(ans) == 0:
        return "NO Equilibrium Index"
    return ans

A = [-7, 1, 5, 2, -4, 3, 0]
equilibrium_index(A)
```

Out[262]: [3, 6]

In [263]:
```
def equilibrium_index_optim(A):
    n = len(A)
    ans = []
    total = 0
    for x in A:
        total += x
    left = 0
    for i in range(n):


        right = total - left - A[i]

        if left == right:
            ans.append(i)
        left += A[i]
    return ans

A = [-7, 1, 5, 2, -4, 3, 0]
equilibrium_index_optim(A)
```

Out[263]: [3, 6]

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Problem Description

Given an array, **arr[]** of size **N**, the task is to find the count of array indices such that removing an element from these indices makes the sum of even-indexed and odd-indexed array elements equal.

"

In [279]:

```python
def odd_even_sum(A):

    n = len(A)
    oddprefix = []
    oddsm = 0
    for i in range(n):
        if i % 2 != 0:
            oddsm += A[i]
        oddprefix.append(oddsm)

    evenprefix = []
    evensm = 0
    for i in range(n):
        if i % 2 == 0:
            evensm += A[i]
        evenprefix.append(evensm)
    print(oddprefix)
    print(evenprefix)
    c = 0
    for i in range(n):
        oddsumm = oddprefix[i-1] + evenprefix[n-1]-evenprefix[i]
        evensumm = evenprefix[i-1] + oddprefix[n-1]-oddprefix[i]

        if oddsumm == evensumm:
            c += 1
    return c


A = [2, 1, 6, 4]
odd_even_sum(A)
```

```
[0, 1, 1, 5]
[2, 2, 8, 8]
```

Out[279]: 1

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

You are given an integer array **A** of size **N**.

You have to pick **B** elements in total. Some (possibly 0) elements from left end of array **A** and some (possibly 0) from the right end of array **A** to get the maximum sum.

Find and return this **maximum possible sum**.

NOTE: Suppose B = 4, and array A contains 10 elements, then

- You can pick the first four elements or can pick the last four elements, or can pick 1 from front and 3 from the back, etc. You need to return the maximum possible sum of elements you can pick.

pick from both sides

In [30]:
```python
def pick_from_both_side(A,B):
    n = len(A)
    maxsum = -float('inf')
    for L in range(B+1):
        R = B - L

        leftsum = 0
        for i in range(L):
            leftsum +=  A[i]

        rightsum = 0
        for i in range(n-R,n):
            rightsum += A[i]

        total = leftsum + rightsum
        maxsum = max(total , maxsum)
    return maxsum



A = [1,2]
B = 1
pick_from_both_side(A,B)
```

Out[30]: 2

In [28]:
```python
def pick_from_both_side(A,B):
    n = len(A)

    prefix = [A[0]]
    for x in A[1:]:
        prefix.append(x+prefix[-1])

    maxsum = -float('inf')
    for L in range(B):
        R = B - L


        if L == 0:
            leftsum = 0
        else:
            leftsum = prefix[L-1]


        if n-R == 0:
            rightsum = prefix[n-1]
        else:
            rightsum = prefix[n-1]-prefix[n-R-1]


        total = leftsum + rightsum
        maxsum = max(total , maxsum)
    return maxsum



A = [1,2]
B = 1
pick_from_both_side(A,B)
```

Out[28]: 2

In [33]:
```python
def pick_from_both_sides(A,B):
    n = len(A)

    maxsum = sum(A[:B])


    rightpicks = n-1
    total = maxsum
    for i in range(B-1,-1,-1):

        total = total + A[rightpicks] - A[i]

        maxsum = max(total, maxsum)
        rightpicks -= 1

    return maxsum




A = [5, -2, 3 , 1, 2]
B = 3
pick_from_both_sides(A,B)
```

Out[33]: 8

In [ ]:
```python

```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

Sum of all elements from `index i to j` =

59 users have participated

A	`pf[j] - pf[i]`	5%
B	`pf[j] - pf[i-1]`	32%
✓ C	`If i!=0: sum = pf[j] - pf[i-1]` `if i==0: sum = pf[j]`	61%
D	None of them	2%

Answered in 6.6 seconds

(Since you've already attempted the quiz before, this attempt wouldn't update your score)

Problem Description

Given an integer array **A** of size **N**. In one second, you can increase the value of one element by 1.

Find the **minimum** time in seconds to make all elements of the array equal.

In [105]:
```
1  A = [2, 4, 1, 3, 2]
2
3  def time_to_equity(A):
4
5      n = len(A)
6      maxnum = max(A)
7      ans = 0
8      for i in range(n):
9          diff = maxnum - A[i]
10         ans += diff
11     return ans
12 time_to_equity(A)
```

Out[105]: 8

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

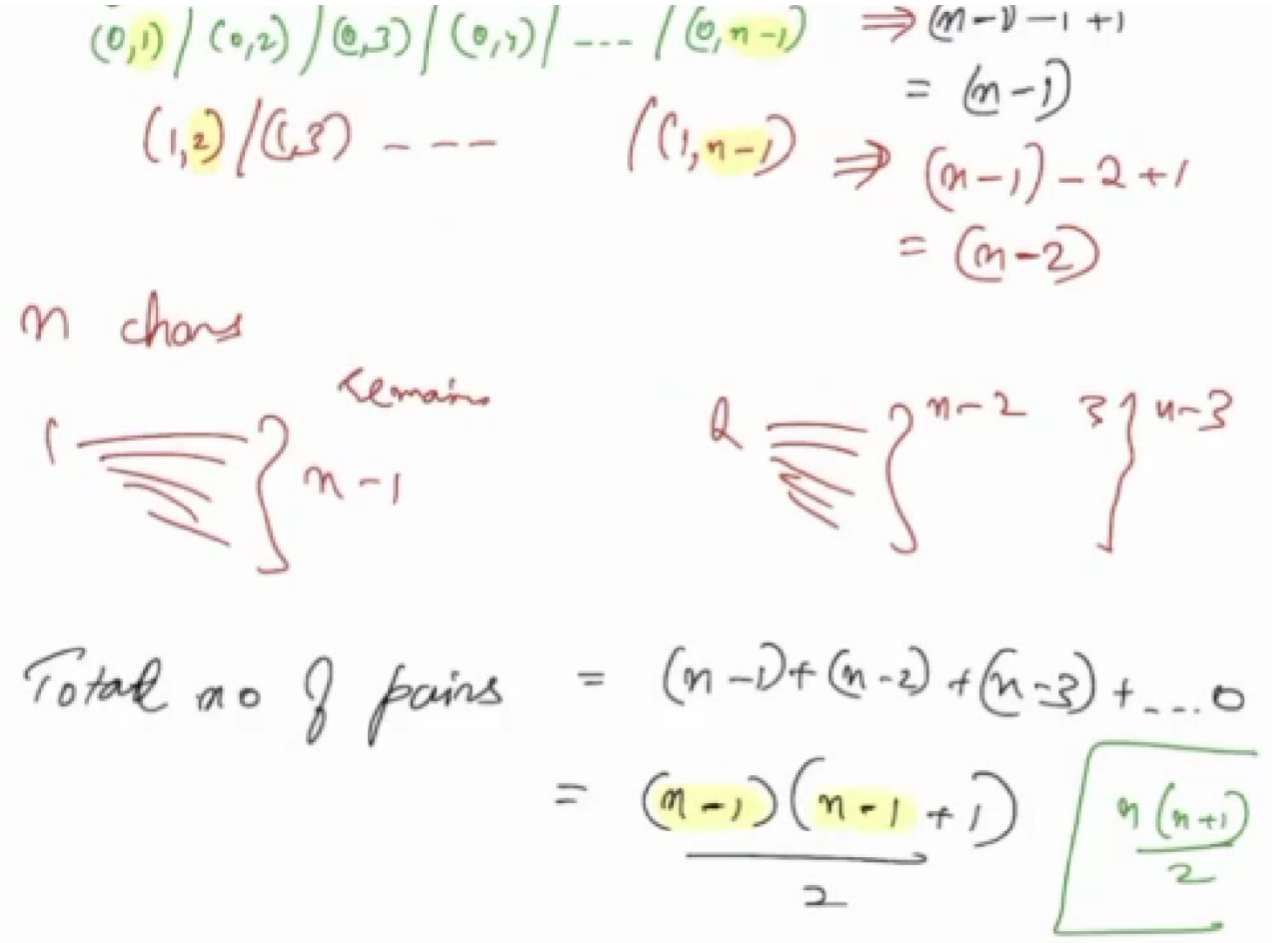
In [ ]:
```
1
```

In [ ]:
```
1
```

Given a string of length N, how many pairs of indices `(i, j)` are there, such that `i < j`?

64 users have participated

A N * (N - 1) / 2 36%



Q1. Special Subsequences "AG" </>

Problem Description

You have given a string **A** having **Uppercase English letters**.

You have to find how many times subsequence **"AG"** is there in the given string.

NOTE: Return the answer modulo **10^9 + 7** as the answer can be very large.

In [49]:
```
A = "asdagsdgaggg"
n = len(A)
count = 0
for i in range(n):
    if A[i] == "a":
        for j in range(i+1,n):
            if A[j] == "g":
                count += 1
print(count)
```

```
13
```

In [48]:
```
def special_subsequences_AG(S):
    n = len(S)

    g_count = n*[0]
    count = 0
    for i in reversed(range(n)):
        if S[i] == "g":
            count += 1
        g_count[i] = count
    print(g_count)
    ans = 0
    for x in range(n):
        if S[x] == "a":
            ans += g_count[x]
    return ans


S = "asdagsdgaggg"
special_subsequences_AG(S)
```

```
[5, 5, 5, 5, 5, 4, 4, 4, 3, 3, 2, 1]
```

Out[48]:
```
13
```

In [50]:
```
def special_subsequences_AG_optim(S):
    n = len(S)
    ans = 0
    gc = 0
    for i in reversed(range(n)):

        if S[i] == "g":
            gc += 1
        if S[i] == "a":
            ans += gc
    return ans


S = "asdagsdgaggg"
special_subsequences_AG_optim(S)
```

Out[50]:
```
13
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Q3. Leaders in an array </>

Problem Description

Given an integer array **A** containing **N** distinct integers, you have to find all the **leaders** in array **A**.

An element is a leader if it is strictly greater than all the elements to its right side.

NOTE: The rightmost element is always a leader.

In [60]:
```
A = [1,7,8,-6,0,3,-7,-10]           # O(n^2)O(1)
n = len(A)
c = 0
for i in range(n):
    leader = True
    for j in range(i+1,n):
        if A[i] <= A[j]:
            leader = False
    if leader:
        c += 1
print(c)
```

```
4
```

In [66]:
```
def leaders_in_array(A):             # most optimised O(n)O(1)
    n = len(A)
    count = 0
    maxnum = -float('inf')
    for i in range(n-1,-1,-1):
        if  A[i] > maxnum :
            count +=1
            maxnum = A[i]
    return count



A = [1,7,8,-6,0,3,-7,-10]
leaders_in_array(A)
```

Out[66]: 4

In [ ]:
```

```

In [64]:
```
A = [1,7,8,-6,0,3,-7,-10]   # O(n)O(n)
n = len(A)

postfixmax = [0]*n
maxele = -float("inf")
for i in range(n-1,-1,-1):
    maxele = max(A[i],maxele)
    postfixmax[i] = maxele

print("PostFix Max Array : ",postfixmax)
print("ans : ", len(set(postfixmax)))

```

```
PostFix Max Array :  [8, 8, 8, 3, 3, 3, -7, -10]
ans :  4
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

Q2. Closest MinMax

</>

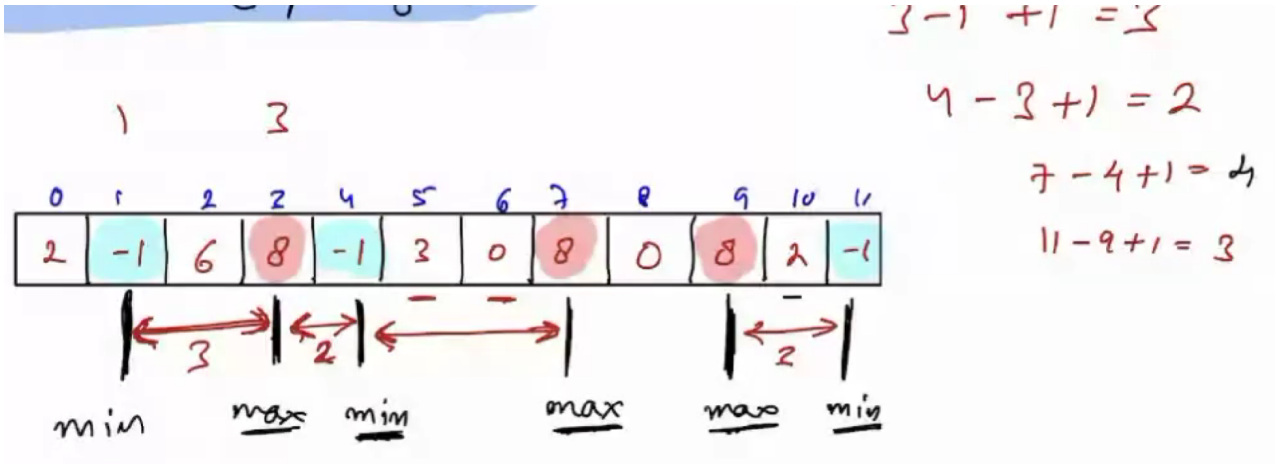
Problem Description

Given an array **A**, find the size of the smallest subarray such that it contains at least one occurrence of the maximum value of the array

and at least one occurrence of the minimum value of the array.

In [80]:
```
A = [2,2,6,4,5,1,5,2,6,4,1]
n = len(A)

minval = min(A)
maxval = max(A)
minlen = float('inf')
for s in range(n):
    for e in range(s,n):
        if minval in A[s:e+1] and maxval in A[s:e+1]:
            l = e-s+1
            if l < minlen:
                minlen = l
print(minlen)
```

3



In [104]:
```
def closest_min_max(A):
    n = len(A)
    minval = min(A)
    maxval = max(A)
    minlen = float('inf')

    imax = None  # assume we have not found any min max
    imin = None     # indexes for max and min updatation on the way

    for i in range(n):

        if A[i] == maxval:    # if we find max , update max index
            imax = i
            if imin != None:           # if we have previously found min
                length = imax - imin + 1           # measure distance
                minlen = min(length,minlen)
        if A[i] == minval:         # if we find min , update min index
            imin = i
            if imax != None:      # if prevouly found max val , measure distance
                length = imin - imax + 1    # b-a+1
                minlen = min(length,minlen)
    return minlen

A = [2,2,6,4,5,1,5,2,6,4,1]

closest_min_max(A)
```

Out[104]: 3

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
1
```

In [ ]:
```
1
```

Q3. Bulbs </> Solved

Problem Description

A wire connects **N** light bulbs.

Each bulb has a switch associated with it; however, due to faulty wiring, a switch also changes the state of all the bulbs to the right of the current bulb.

Given an initial state of all bulbs, find the **minimum number** of switches you have to press to turn on all the bulbs.

You can press the same switch multiple times.

Note: 0 represents the bulb is off and **1** represents the bulb is on.

In [96]:
```
A = [0,1,0,1]
n = len(A)
c = 0
for i in range(n):
    if A[i] == 0:

        for i in range(i,n):
            if A[i] == 0:
                A[i] = 1
            else:
                A[i] = 0
        c += 1


#     print(A)
print(c)

```

```
4
```

In [103]:
```
def bulb(A):
    n = len(A)
    initial = 0
    c = 0

    for i in range(n):
        if A[i] == initial:
            c += 1
            initial  = 1- initial   # change the initial state of bulb
            A[i] = initial
    return c

A = [0,1,0,1]
bulb(A)
```

Out[103]: 4

In [ ]:
```
1
```

In [ ]:
```
1
```

subarrays

In [107]:
```
A = [1,2,3,4,5]

n = len(A)

for i in range(n):
    for j in range(i,n):
        for x in range(i,j+1):
            print(A[x],end = " ")
        print()
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
2
2 3
2 3 4
2 3 4 5
3
3 4
3 4 5
4
4 5
5
```

In [108]:
```
# total number of subarrays : (n*(n+1)) / 2
```

In [ ]:

In [ ]:

Q2. Even Subarrays </> Solved

Problem Description

You are given an integer array **A**.

Decide whether it is possible to divide the array into one or more subarrays of **even length** such that the **first** and **last** element of all subarrays will be **even**.

Return "**YES**" if it is possible; otherwise, return "**NO**" (without quotes).

In [109]:
```
def even_subarray(A):
    if len(A) % 2 == 0 and A[0] %2 ==0 and A[-1] % 2 == 0:
        return "YES"
    return "NO"
```

In [112]:
```
A = [2, 4, 8, 6]
even_subarray(A)
```

Out[112]: 'YES'

In [111]:
```
A = [2, 4, 8, 7, 6]
even_subarray(A)
```

Out[111]: 'NO'

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Q1. Amazing Subarrays </> Solved

You are given a string **S**, and you have to find all the **amazing substrings** of **S**.

An amazing Substring is one that starts with a **vowel** (a, e, i, o, u, A, E, I, O, U).

In [117]:
```
def amazing_subarray_string(A):
    n = len(A)

    vowels =  ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']
    index = len(A)
    ans = 0
    for i in range(n):
        if A[i] in vowels:
            ans += index
        index -=1
    return ans

A = "ABEC"

amazing_subarray_string(A)
```

Out[117]: 6

In [ ]:

In [ ]:

In [ ]:

Q2. Max Sum Contiguous Subarray </> Solved

Asked in: PA YA

Problem Description

Find the **contiguous non-empty** subarray within an array, **A** of length **N,** with the **largest sum**.

In [145]:
```
A = [5,-10,100,-20,15,-10,25,30]

prefix = [A[0]]

for i in range(1,len(A)):
    prefix.append(A[i]+prefix[i-1])
maxsum = -float('inf')
for start in range(len(A)):
    for end in range(start,len(A)):
        if start==0:
            sum = prefix[end]
        else:
            sum = prefix[end]-prefix[start-1]
        if sum > maxsum:
            maxsum = sum
print(maxsum)
```

140

In [143]:
```
def max_sum_subarray(A):
    n = len(A)
    maxsum = -float("inf")

    for i in range(n):
        summ = 0
        for j in range(i,n):
            summ += A[j]
            if summ > maxsum:
                maxsum = summ
                print(A[i:j+1],":",summ)

    return maxsum
```

In [136]:
```
A = [3,-10,100,-20,15,-10,25,30]
max_sum_subarray(A)

```

```
[3] : 3
[3, -10, 100] : 93
[3, -10, 100, -20, 15, -10, 25] : 103
[3, -10, 100, -20, 15, -10, 25, 30] : 133
[100, -20, 15, -10, 25, 30] : 140
```

Out[136]: 140

In [41]:
```
def maxSubarraySum(A):                # Kadane's Algo
    n = len(A)
    maximum = A[0]
    currentMax = A[0]

    for i in range(1,n):
#         print(maximum, currentMax,A[i], A[i]+currentMax)
        currentMax = max( currentMax+A[i], A[i] )

        maximum = max(maximum , currentMax)
    return maximum
A = [1,2,3,-2,5]
maxSubarraySum(A)
```

Out[41]: 9

In [ ]:
```

# max product subarray : kadane's algo
```

In [3]:
```
def maxproduct(arr,n):
    n = len(arr)
    maxproduct = -float("inf")


    current_product = 1
    for i in range(n):
        current_product = current_product * arr[i]
        maxproduct = max(current_product , maxproduct)
        if current_product == 0:
            current_product =1
    current_product = 1
    for i in reversed(range(n)):
        current_product = current_product * arr[i]
        maxproduct = max(current_product , maxproduct)
        if current_product == 0:
            current_product =1
    return maxproduct
A = [3,-10,0,-20,15,0,25,30]
maxproduct(A,len(A))
```

Out[3]: 750

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

Given N array elements how many subarrays of len = k are there ?

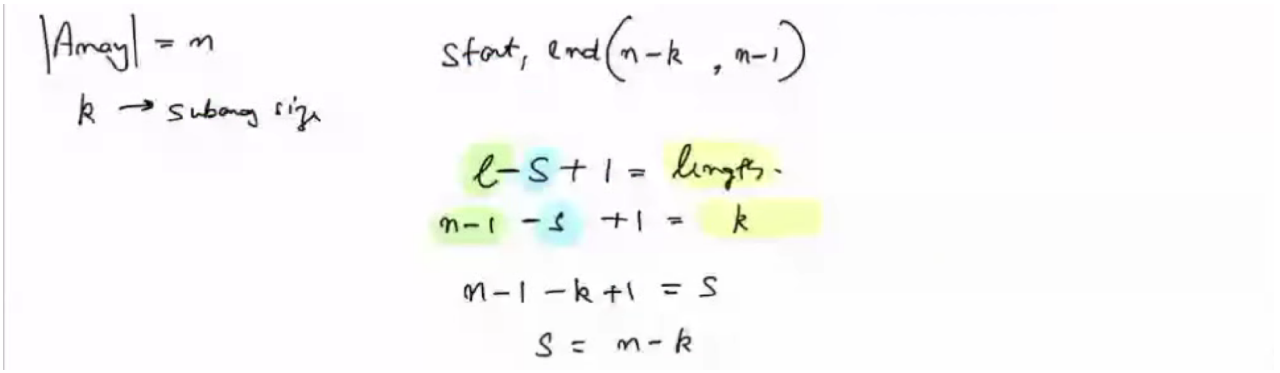
60 users have participated

A	N-k	23%
B	N/k	0%
✓ C	N-k+1	52%
D	N-k-1	25%

Given N array elements and len = k ?
Print last subarray start index and last index

58 users have participated

A	[K, N-1]	14%
✓ B	[N-K , N-1]	21%



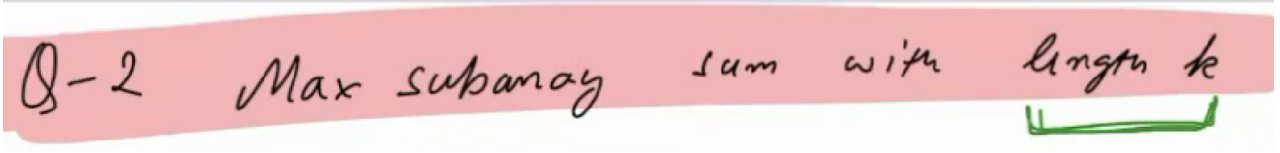
In [ ]: 1

In [ ]: 1

In [ ]: 1

In [ ]: 1

In [ ]: 1



In [149]:

```
def max_k_length_subarray_sum(A,k):
    n = len(A)
    maxsum = -float("inf")
    for s in range(n-k+1):
        e = k + s -1

        summ = 0
        for i in range(s,e+1):
            summ += A[i]

            maxsum = max(maxsum , summ)

    return maxsum


A = [5,-10,100,-20,15,-10,25,30]
k = 6

max_k_length_subarray_sum(A,k)
```

Out[149]: 140

In [ ]:

```
```

In [ ]:

```
```

In [151]:

```
def max_k_length_subarray_sum(A,k):
    n = len(A)
    maxsum = -float("inf")

    prefixsum = [A[0]]
    for x in A[1:]:
        prefixsum.append(x+prefixsum[-1])
    for s in range(n-k+1):
        e = k + s -1

        if s == 0:
            summ = prefixsum[e]
        else:
            summ = prefixsum[e]-prefixsum[s-1]

        maxsum = max(maxsum , summ)

    return maxsum


A = [5,-10,100,-20,15,-10,25,30]
k = 6

max_k_length_subarray_sum(A,k)
```

Out[151]: 140

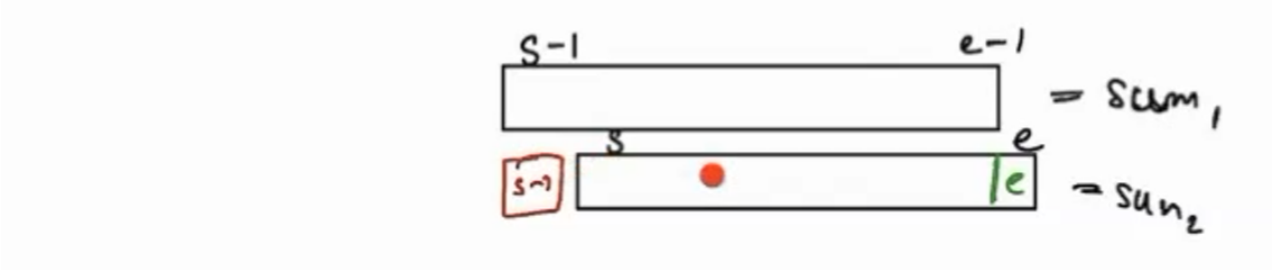
In [ ]:

```
```

In an array A, if the sum of the first subarray of size k is S, what will be the sum of the second subarray of size k?

54 users have participated

A	S - A[0] + A[k+1]	17%
B	S - A[k-1] + A[k]	44%
✓ C	S - A[0] + A[k]	22%
D	S - A[0] + A[k-1]	17%



sliding window

In [19]:
```python
def max_k_length_subarray_(A,k):
    n = len(A)

    maxsum = -float("inf")
    total = sum(A[:k])

    for s in range(1,n-k+1):
        e = k + s -1
        total = total - A[s-1] + A[e]  # current end
                        # previous start
    maxsum = max(maxsum, total)
    return maxsum

A = [5,-10,100,-20,15,-10,25,30]
k = 6
max_k_length_subarray_(A,k)

```

Out[19]: 140

In [ ]:
```python

```

Q3. Subarray with least average </> Solved

Problem Description

Given an array of size **N**, find the subarray of size **K** with the least average.

In [38]:
```python
def k_size_subarray_with_least_average(A,k):
    n = len(A)

    leastavg = float('inf')
    total = sum(A[:k])


    for s in range(n-k+1):
        e = k + s - 1
        length = e-s+1

        total = total - A[s-1] + A[e]

        average = total / length

        if average < leastavg:
            leastavg = average
            starting_index = s
    return starting_index


```

In [39]:
```python
A = [3, 7, 5, 20, -10, 0, 12]
k = 2
k_size_subarray_with_least_average(A,k)
```

Out[39]: 4

In [40]:
```
def k_size_subarray_with_least_average(A,k):    # we dont have to find avg
    n = len(A)

    leastsum = float('inf')
    total = sum(A[:k])


    for s in range(n-k+1):
        e = k + s - 1
        length = e-s+1

        total = total - A[s-1] + A[e]


        if total < leastsum:
            leastsum = total
            starting_index = s
    return starting_index


A = [3, 7, 90, 20, 10, 50, 40]
k = 3
k_size_subarray_with_least_average(A,k)
```

Out[40]: 3

In [ ]:
```

```

In [ ]:
```

```

inverting the problem

Q1. Sum of All Subarrays </> Solved

Problem Description

You are given an integer array **A** of length **N.**
You have to find the sum of all subarray sums of A.
More formally, a subarray is defined as a contiguous part of an array which we can obtain by deleting zero or more elements from either end of the array.
A subarray sum denotes the sum of all the elements of that subarray.

In [22]:
```
A  = [2,1,3]    # carry forward #O(n^2)O(1)

n = len(A)
all_subarray_total = 0
for s in range(n):
    subarraysum = 0
    for e in range(s,n):
        subarraysum += A[e]
        all_subarray_total += subarraysum

print(all_subarray_total)
```

19

In [25]:
```
def sum_of_all_subarrays(A):        # O(n)O(1)
    n = len(A)
    total = 0
    for i in range(n):
        total += A[i] * ((n-i)*(i+1))
              # element * subarray count containing element
    return total
A  = [2,1,3]
sum_of_all_subarrays(A)
```

Out[25]: 19

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
1
```

Q4. Maximum Subarray Easy </> Solved

Problem Description

You are given an integer array **C** of size **A**. Now you need to find a subarray (contiguous elements) so that the sum of contiguous elements is maximum.
But the sum must not exceed **B**.

In [46]:
```
1
2  def max_subarray_Easy(A,B):
3      n = len(A)
4      ans = 0
5      maxsum = -float("inf")
6      for s in range(n):
7          subarray_sum = 0
8          for e in range(s,n):
9              subarray_sum += A[e]
10             if subarray_sum > maxsum and subarray_sum <=B:
11                 maxsum = subarray_sum
12                 ans = maxsum
13     return ans
14 A = [2,1,3,4,5]
15 B = 12
16 max_subarray_Easy(A,B)
```

Out[46]: 12

In [47]:
```
1 A = [2,2,2]
2 B = 1
3 max_subarray_Easy(A,B)
```

Out[47]: 0

In [ ]:
```
1
```

Q1. Counting Subarrays! </> Solved

Problem Description

Given an array **A** of **N** non-negative numbers and you are also given non-negative number **B**.

You need to find the **number of subarrays in A having sum less than B**. We may assume that there is no overflow.

In [70]:
```
1  def counting_subarrays(A,B):
2
3      n = len(A)
4      count = 0
5
6      for s in range(n):
7          subsum = 0
8          for e in range(s,n):
9              subsum += A[e]
10             if subsum < B:
11                 print(A[s:e+1])
12                 count += 1
13     return count
```

In [71]:
```
A = [2,5,6]
B = 10
counting_subarrays(A,B)
```

```
[2]
[2, 5]
[5]
[6]
```

Out[71]: 4

In [72]:
```
A = [1,11,2,3,15]
B = 10
counting_subarrays(A,B)
```

```
[1]
[2]
[2, 3]
[3]
```

Out[72]: 4

In [ ]:
```

```

Q2. Good Subarrays Easy </> Solved

Problem Description

Given an array of integers **A**, a subarray of an array is said to be good if it fulfills any one of the criteria:
1. Length of the subarray is be even, and the sum of all the elements of the subarray must be less than **B**.
2. Length of the subarray is be odd, and the sum of all the elements of the subarray must be greater than **B**.
Your task is to find the count of good subarrays in A.

In [74]:
```
def good_subarray_easy(A,B):

    n = len(A)
    count = 0

    for s in range(n):
        subsum = 0
        for e in range(s,n):
            length = e-s+1
            subsum += A[e]
            if length % 2 == 0 and subsum < B:
                count += 1
            if length % 2 != 0 and subsum > B:
                count += 1
    return count

```

In [ ]:
```

```

In [75]:
```
A = [13, 16, 16, 15, 9, 16, 2, 7, 6, 17, 3, 9]
B = 65
good_subarray_easy(A,B)
```

Out[75]: 36

In [76]:
```
A = [1, 2, 3, 4, 5]
B = 4
good_subarray_easy(A,B)
```

Out[76]: 6

In [ ]:
```

```

In [ ]:
```

```

Q3. Alternating Subarrays Easy </> Solved

You are given an integer array **A** of length **N** comprising of **0's** & **1's**, and an integer **B**.

You have to tell all the indices of array **A** that can act as a center of **2 * B + 1** length 0-1 alternating subarray.

A **0-1 alternating array** is an array containing only **0's** & **1's**, and having no adjacent **0's** or **1's**. For e.g. arrays **[0, 1, 0, 1]**, **[1, 0]** and **[1]** are 0-1 alternating, while **[1, 1]** and **[0, 1, 0, 0, 1]** are not.

In [155]:
```
def alternating_subarray_easy(A,B):

    n = len(A)
    l = 2*B + 1
    centers  = []

    for s in range(n-l+1):
        e = l + s -1

        is_alternating = True
        previous  = None

        for i in range(s,e+1):
            if A[i] == previous:
                is_alternating = False
                break
            previous = A[i]

        if is_alternating:
            centers.append((e+s)//2)
    return centers


```

In [158]:
```
A = [1,0,0,0,1,0,1,0,1,1,1,0,0,1,0,1,0,1,0,1,0,0,0,]
B = 2
alternating_subarray_easy(A,B)
```

Out[158]: [5, 6, 14, 15, 16, 17, 18]

In [ ]:
```

```

In [147]:
```
def alternating_subarray_easy_optim(A,B):
    string = ""
    for x in range(len(A)):
        string += str(A[x])

    l = 2*B+1
    n = len(string)
    ans = []


    for i in range(B,n-B):

        if "00" not in string[i-B:i+B+1] and "11" not in string[i-B:i+B+1]:
            ans.append(i)
    return ans


```

In [157]:
```
A = [1,0,0,0,1,0,1,0,1,1,1,0,0,1,0,1,0,1,0,1,0,0,0,]
B = 2
alternating_subarray_easy_optim(A,B)
```

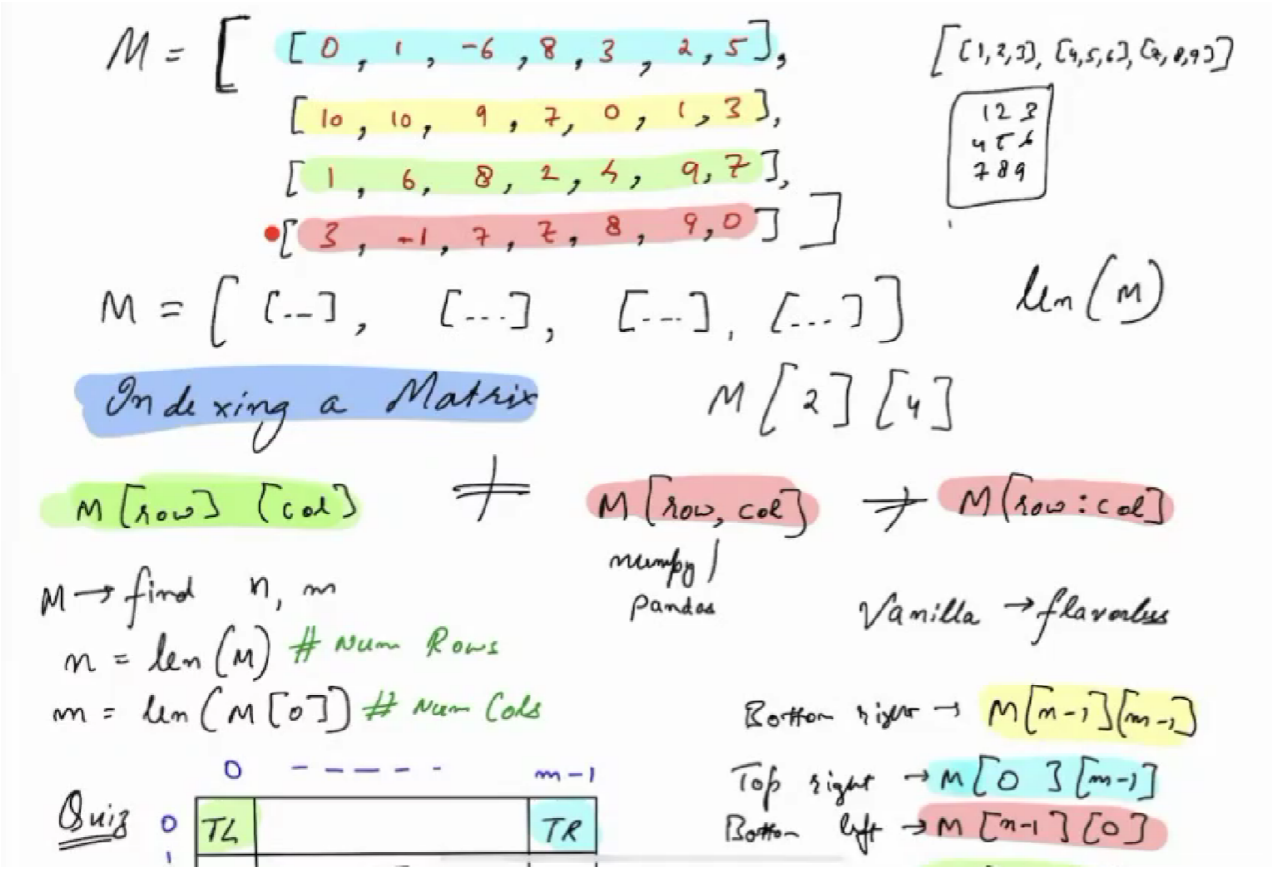
Out[157]: [5, 6, 14, 15, 16, 17, 18]

In [ ]:
```

```

In [ ]:
```

```

2d Matrix

In [ ]:
```

```



In [243]:
```
# row sum | column sum | traversal row wise and column wise
```

You are given a 2D integer matrix A, return a 1D integer vector containing row-wise sums of original matrix.

In [242]:
```
M = [[0,1,2,3,4],
    [9,8,7,4,5],
    [1,5,9,8,4],
    [3,6,2,5,9]]

n = len(M)
m = len(M[0])

for i in range(n):
    rowsum = 0
    for j in range(m):
        rowsum += M[i][j]
        print(M[i][j],end = " ")
    print(": ",rowsum)

```

```
0 1 2 3 4 :  10
9 8 7 4 5 :  33
1 5 9 8 4 :  27
3 6 2 5 9 :  25
```

You are given a 2D integer matrix A, return a 1D integer vector containing column-wise sums of original matrix.

In [178]:
```
for j in range(m):
    colsum = 0
    for i in range(n):
        colsum += M[i][j]
        print(M[i][j],end = " ")
    print(": ",colsum)

```

```
0 9 1 3 :  13
1 8 5 6 :  20
2 7 9 2 :  20
3 4 8 5 :  20
4 5 4 9 :  22
```

In [ ]:
```
1
```

In [ ]:
```
1
```

PRIMARY DIAGOAL

You are given a **N X N** integer matrix. You have to find the sum of all the main diagonal elements of **A**.

Main diagonal of a matrix **A** is a collection of elements **A[i, j]** such that **i = j**.

In [248]:
```
1  M = [[0,1,2,3,4],
2       [9,8,7,4,5],
3       [1,5,9,8,4],
4       [3,6,2,5,9]]
5
6  n = len(M)
7  m = len(M[0])
8  ans = 0
9  for i in range(n):
10     print(M[i][i], end = " ")      # PRIMARY DIAGOAL
11     ans += M[i][i]
12
13 print()
14 print(ans)
```

```
0 8 9 5
22
```

In [ ]:
```
1
```

SECONDARY DIAGONAL

In [219]:
```
1  M = [[0,1,2,3,4],
2       [9,8,7,4,5],
3       [1,5,9,8,4],
4       [3,6,2,5,9]]
5
6  n = len(M)
7  m = len(M[0])
8
9  r = 0
10 c = m-1
11
12 while c >= 0 and r < n:       # secondary digonal
13     print(M[r][c])
14     r += 1
15     c -= 1
16
```

```
4
4
9
6
```

In [226]:
```
1  M = [[0,1,2,3,4],
2       [9,8,7,4,5],
3       [1,5,9,8,4],
4       [3,6,2,5,9]]
5
6  n = len(M)
7  m = len(M[0])
8
9  for r in range(n):
10     print(M[r][n-r],end = " ")
11
```

```
4 4 9 6
```

You are given a **N X N** integer matrix. You have to find the sum of all the minor diagonal elements of **A**.

Minor diagonal of a **M X M** matrix **A** is a collection of elements **A[i, j]** such that **`i + j = M + 1`** (where **i, j** are 1-based).

In [249]:
```
M = [[0,1,2,3,4],
     [9,8,7,4,5],
     [1,5,9,8,4],
     [3,6,2,5,9]]

n = len(M)
m = len(M[0])
ans = 0
for r in range(n):
#     print(M[r][n-r],end = " ")
    ans += M[r][n-r]
print()
print(ans)

```

```
23
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

CREATE EMPTY 2d MATRIX OF n * m SIZE

ASSIGN VALUE TO M

In [213]:
```
M = []
n = 4
m = 8
val  = 0
for i in range(n):
    M.append([])
    for j in (range(m)):
        val += 1
        M[i].append(val)
M
```

Out[213]:
```
[[1, 2, 3, 4, 5, 6, 7, 8],
 [9, 10, 11, 12, 13, 14, 15, 16],
 [17, 18, 19, 20, 21, 22, 23, 24],
 [25, 26, 27, 28, 29, 30, 31, 32]]
```

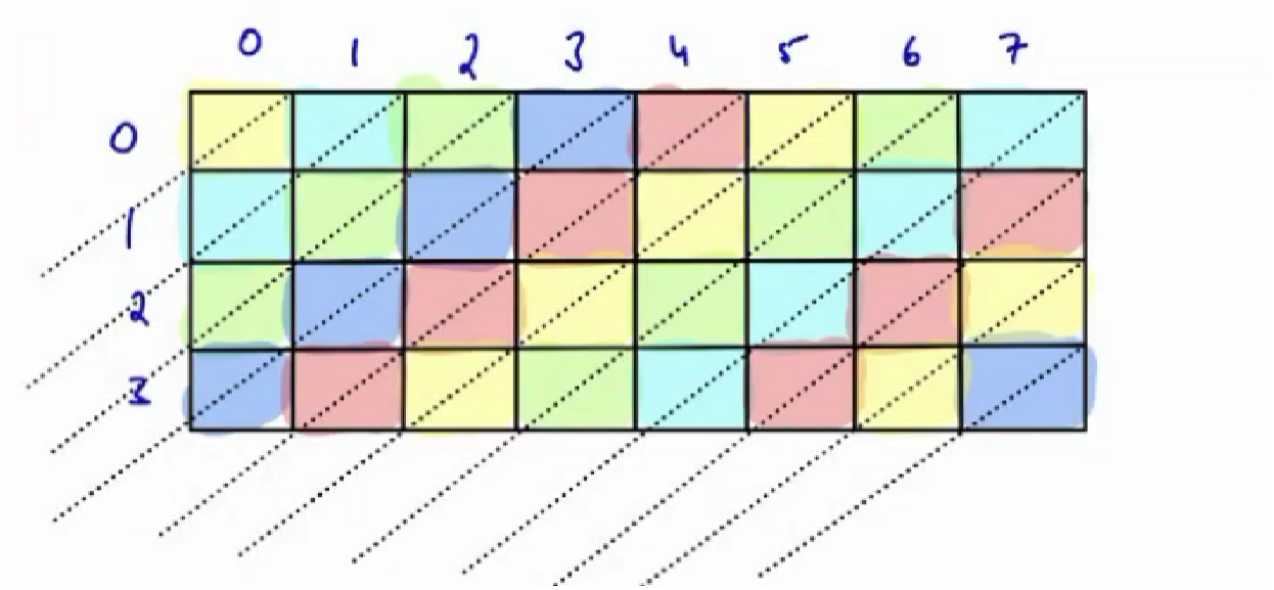
In [217]:
```
M = [[0]*m for _ in range(n)]
val = 0
for i in range(n):
    for j in range(m):
        val += 1
        M[i][j] = val

M
```

Out[217]:
```
[[1, 2, 3, 4, 5, 6, 7, 8],
 [9, 10, 11, 12, 13, 14, 15, 16],
 [17, 18, 19, 20, 21, 22, 23, 24],
 [25, 26, 27, 28, 29, 30, 31, 32]]
```

In [ ]:
```

```

In [ ]:
```

```

Given a rectangular Matrix , N * M. print all the right -->-- left diagonals



Given a matrix of size NxM, how many diagonals will be there?

59 users have participated

A	N*M	2%
B	N*M-1	8%
✓ C	N+M-1	90%
D	N+M	0%

In [240]:

```
M =[[1 ,  2,  3,  4,  5,  6,  7,  8],
    [9,  10, 11, 12, 13, 14, 15, 16],
    [17, 18, 19, 20, 21, 22, 23, 24],
    [25, 26, 27, 28, 29, 30, 31, 32]]
n = len(M)
m = len(M[0])


for c in range(0,m):

    r = 0
    while c >= 0 and r < n:
        print(M[r][c],end = " ")
        r += 1
        c -= 1
    print()
for r in range(1,n):
    c = m-1

    while c >= 0 and r < n:
        print(M[r][c],end = " ")
        r += 1
        c -= 1
    print()
```

```
1
2 9
3 10 17
4 11 18 25
5 12 19 26
6 13 20 27
7 14 21 28
8 15 22 29
16 23 30
24 31
32
```

In [ ]:

```

```

Q5. Matrix Transpose </> Solved

Problem Description

You are given a matrix **A**, you have to return another matrix which is the transpose of A.

NOTE: Transpose of a matrix A is defined as - $A^T[i][j] = A[j][i]$; Where $1 \leq i \leq col$ and $1 \leq j \leq row$

In [255]:
```
def TransposeMatrix(M):
    n = len(M)
    m = len(M[0])

    Mt = [[0]*n for _ in range(m)]

    for r in range(n):
        for c in range(m):
            Mt[c][r] = M[r][c]

    return Mt



M =[[1 ,  2,  3,  4,  5,  6,  7,  8],
    [9,  10, 11, 12, 13, 14, 15, 16],
    [17, 18, 19, 20, 21, 22, 23, 24],
    [25, 26, 27, 28, 29, 30, 31, 32]]
n = len(M)
m = len(M[0])
TransposeMatrix(M)
```

Out[255]:
```
[[1, 9, 17, 25],
 [2, 10, 18, 26],
 [3, 11, 19, 27],
 [4, 12, 20, 28],
 [5, 13, 21, 29],
 [6, 14, 22, 30],
 [7, 15, 23, 31],
 [8, 16, 24, 32]]
```

In [ ]:
```

```

In [259]:
```
M = [[1,2,3],   # only for square matrix
    [4,5,6],
    [7,8,9]]
n = len(M)

for i in range(n):
    for j in range(i,n):
        M[i][j],M[j][i] = M[j][i],M[i][j]
print(M)
```

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

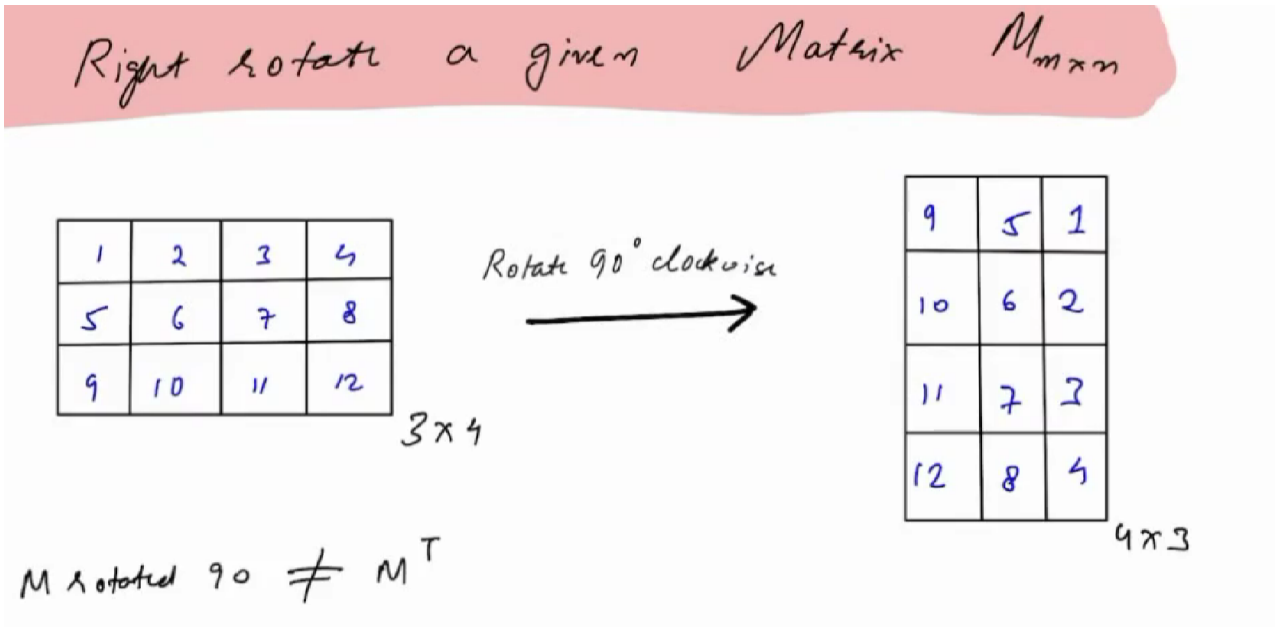
In [ ]:
```

```

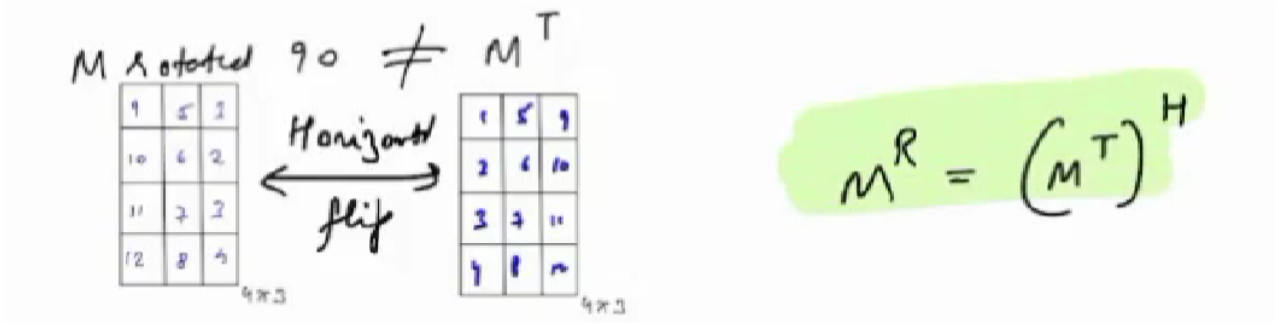
In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

Rotate 90degree clockwise





In [284]:

```python
def FlipHorizontal(M):
    n = len(M)
    m = len(M[0])

    Mh = [[0]*m for _ in range(n)]

    for i in range(n):
        for j in range(m):
            Mh[i][m-j-1] = M[i][j]
    return Mh

```

In [285]:

```python

M =[[1 ,  2,  3,  4,  5,  6,  7,  8],
    [9,  10, 11, 12, 13, 14, 15, 16],
    [17, 18, 19, 20, 21, 22, 23, 24],
    [25, 26, 27, 28, 29, 30, 31, 32]]
FlipHorizontal(M)
```

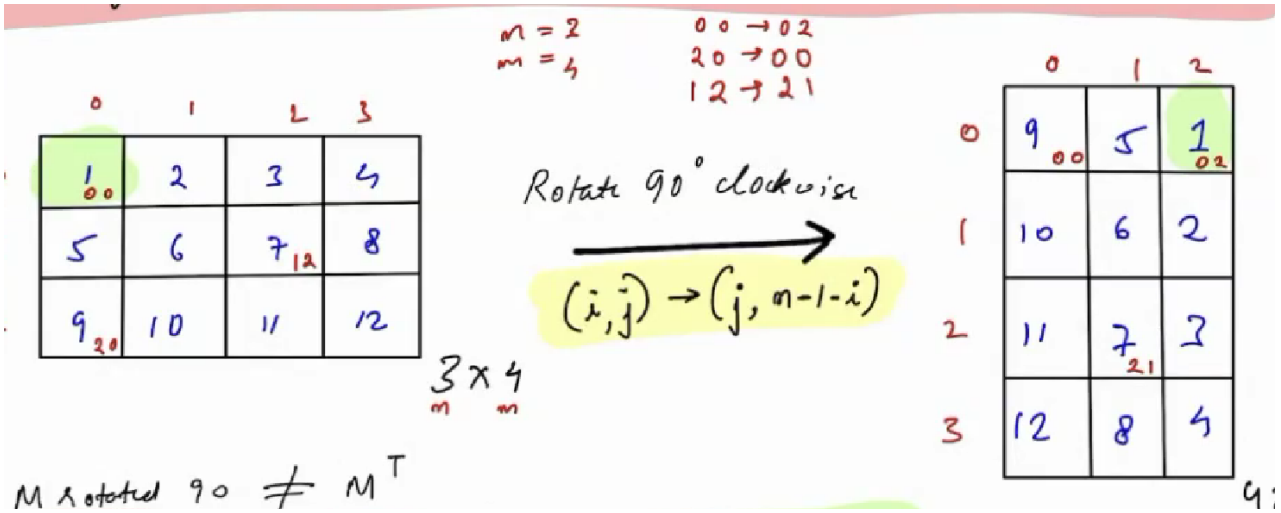
Out[285]:

```
[[8, 7, 6, 5, 4, 3, 2, 1],
 [16, 15, 14, 13, 12, 11, 10, 9],
 [24, 23, 22, 21, 20, 19, 18, 17],
 [32, 31, 30, 29, 28, 27, 26, 25]]
```

In [290]:
```
def rotate90degClockwise(M):

    def TransposeMatrix(M):
        n = len(M)
        m = len(M[0])
        Mt = [[0]*n for _ in range(m)]
        for r in range(n):
            for c in range(m):
                Mt[c][r] = M[r][c]
        return Mt
    def FlipHorizontal(M):
        n = len(M)
        m = len(M[0])
        Mh = [[0]*m for _ in range(n)]
        for i in range(n):
            for j in range(m):
                Mh[i][m-j-1] = M[i][j]
        return Mh

    return FlipHorizontal(TransposeMatrix(M))

rotate90degClockwise(M)
```

Out[290]:
```
[[25, 17, 9, 1],
 [26, 18, 10, 2],
 [27, 19, 11, 3],
 [28, 20, 12, 4],
 [29, 21, 13, 5],
 [30, 22, 14, 6],
 [31, 23, 15, 7],
 [32, 24, 16, 8]]
```

In [ ]:
```

```

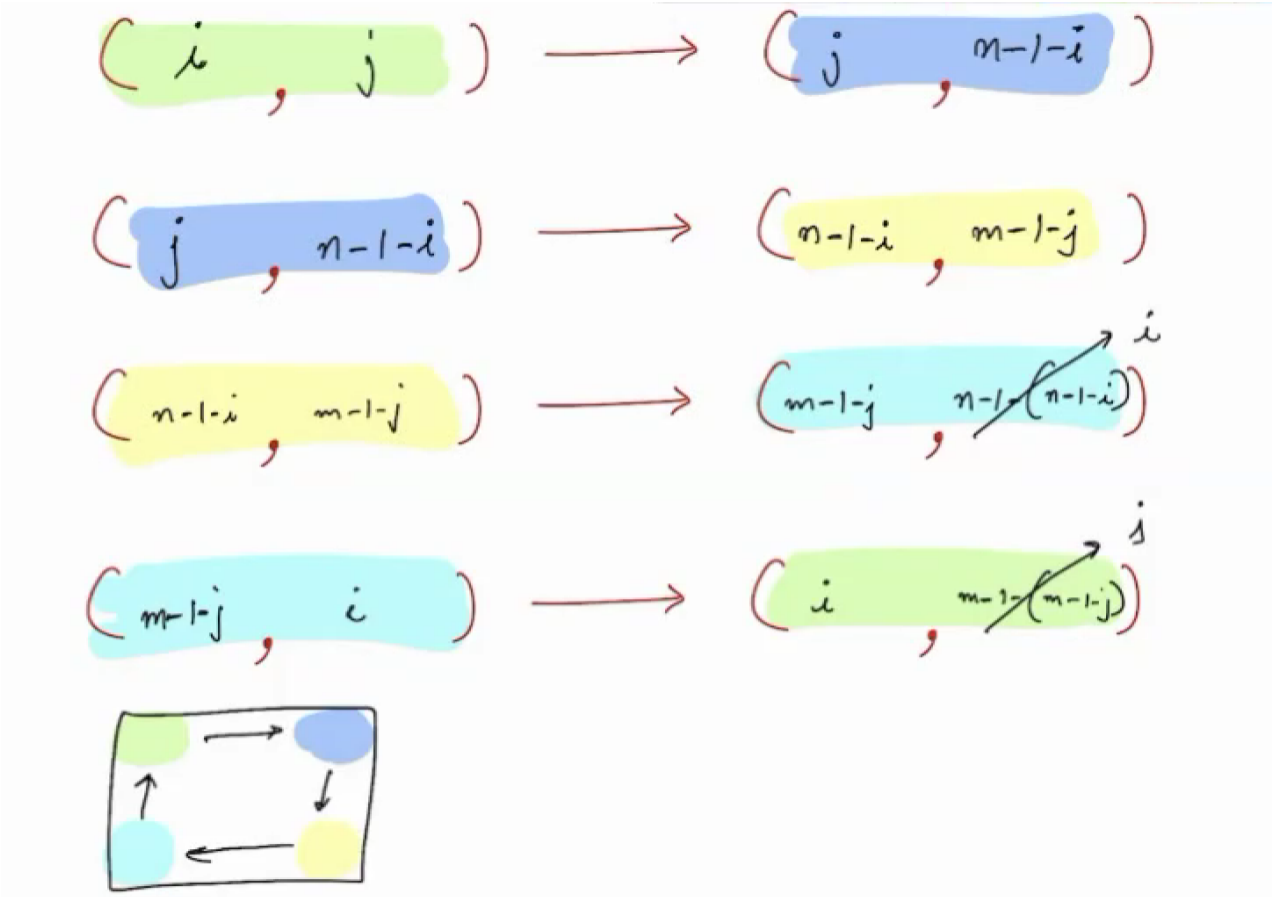


In [9]:
```
def rotate90degClockwise(M):
    n = len(M)
    m = len(M[0])

    Mr = [[0]*n for _ in range(m)]

    for i in range(n):
        for j in range(m):
            Mr[j][n-i-1] = M[i][j]
    return Mr
```

In [10]:
```

M =[[1 ,  2,  3,  4,  5,  6,  7,  8],
    [9,  10, 11, 12, 13, 14, 15, 16],
    [17, 18, 19, 20, 21, 22, 23, 24],
    [25, 26, 27, 28, 29, 30, 31, 32]]
rotate90degClockwise(M)
```

Out[10]:
```
[[25, 17, 9, 1],
 [26, 18, 10, 2],
 [27, 19, 11, 3],
 [28, 20, 12, 4],
 [29, 21, 13, 5],
 [30, 22, 14, 6],
 [31, 23, 15, 7],
 [32, 24, 16, 8]]
```

In [ ]:
```

```



In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

Q3. Matrix Multiplication </> Solved

You are given two integer matrices **A**(having **M X N** size) and **B**(having **N X P**). You have to multiply matrix **A** with **B** and return the resultant matrix. (i.e. return the matrix **AB**).

If $\mathbf{A}$ is an $m \times n$ matrix and $\mathbf{B}$ is an $n \times p$ matrix,

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

the *matrix product* $\mathbf{C} = \mathbf{AB}$ (denoted without multiplication signs or dots) is defined to be the $m \times p$ matrix

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

such that

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj},$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, p$.

In [17]:
```
A = [[0,1,2],
     [3,2,0],
     [1,0,2],
     [0,3,1]]

B = [[1,0,2,3,0],
     [0,1,0,2,3],
     [3,0,1,0,2]]

def multiplyMarixes(A,B):
    m  = len(A)
    n = len(A[0])
    n = len(B)
    p = len(B[0])

    C = [[0]*p for _ in range(m)]

    for i in range(m):
        for j in range(p):
            s = 0
            for k in range(n):
                C[i][j] += A[i][k]*B[k][j]



    return C


multiplyMarixes(A,B)
```

Out[17]:
```
[[6, 1, 2, 2, 7], [3, 2, 6, 13, 6], [7, 0, 4, 3, 4], [3, 3, 1, 6, 11]]
```

In [ ]:
```

```

In [20]:
```
A = [[1, 2],
     [3, 4]]
```

In [22]:
```
B = [[5, 6], [7, 8]]
multiplyMarixes(A,B)
```

Out[22]:
```
[[19, 22], [43, 50]]
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

Print Matrix Boundary

In [63]:
```
n = 5
m = 10
M = [[0]*m for _ in range(n)]

x = 0
for c in range(m):
    for r in range(n):
        M[r][c] = x
        x = x+1
M
```

Out[63]:
```
[[0, 5, 10, 15, 20, 25, 30, 35, 40, 45],
 [1, 6, 11, 16, 21, 26, 31, 36, 41, 46],
 [2, 7, 12, 17, 22, 27, 32, 37, 42, 47],
 [3, 8, 13, 18, 23, 28, 33, 38, 43, 48],
 [4, 9, 14, 19, 24, 29, 34, 39, 44, 49]]
```

In [65]:
```
for c in range(m-1):
    print(M[0][c], end=  " ")
```

```
0 5 10 15 20 25 30 35 40
```

In [66]:
```
for r in range(n-1):
    print(M[r][m-1], end=  " ")
```

```
45 46 47 48
```

In [69]:
```
for c in reversed(range(1,m)):
    print(M[n-1][c], end=  " ")
```

```
49 44 39 34 29 24 19 14 9
```

In [70]:
```
for r in reversed(range(1,n)):
    print(M[r][0], end=  " ")
```

```
4 3 2 1
```

In [ ]:
```

```

In [ ]:
```

```

In [72]:
```
M
```

Out[72]:
```
[[0, 5, 10, 15, 20, 25, 30, 35, 40, 45],
 [1, 6, 11, 16, 21, 26, 31, 36, 41, 46],
 [2, 7, 12, 17, 22, 27, 32, 37, 42, 47],
 [3, 8, 13, 18, 23, 28, 33, 38, 43, 48],
 [4, 9, 14, 19, 24, 29, 34, 39, 44, 49]]
```

In [89]:
```
def print_boundary(M):

    n = len(M)
    m = len(M[0])

    r = 0
    c = 0
    direction = "right"

    while True:

        print(M[r][c], end =  " ")
        if direction == "right":
            if c == m-1:
                direction = "down"
                r = r+1
            else:
                c = c +1

        elif direction == "down":
            if r == n-1:
                direction = "left"
                c = c-1
            else:
                r = r+1

        elif direction == "left":
            if c == 0:
                direction = "up"
                r = r-1
            else:
                c = c-1

        elif direction == "up":

            if r == 1:
                break
            else:
                r = r-1

M = [[0, 5, 10, 15, 20, 25, 30, 35, 40, 45],
     [1, 6, 11, 16, 21, 26, 31, 36, 41, 46],
     [2, 7, 12, 17, 22, 27, 32, 37, 42, 47],
     [3, 8, 13, 18, 23, 28, 33, 38, 43, 48],
     [4, 9, 14, 19, 24, 29, 34, 39, 44, 49]]

print_boundary(M)
```

```
0 5 10 15 20 25 30 35 40 45 46 47 48 49 44 39 34 29 24 19 14 9 4 3 2 1
```

In [ ]:
```

```

In [95]:

```python
def print_boundary_r_c(n , m):

    M = [[0]*m for _ in range(n)]

    r = 0
    c = 0
    direction = "right"
    number = 0
    while True:
        number +=  1
        M[r][c] = number
        if direction == "right":
            if c == m-1:
                direction = "down"
                r = r+1
            else:
                c = c +1

        elif direction == "down":
            if r == n-1:
                direction = "left"
                c = c-1
            else:
                r = r+1

        elif direction == "left":
            if c == 0:
                direction = "up"
                r = r-1
            else:
                c = c-1

        elif direction == "up":

            if r == 1:
                break
            else:
                r = r-1
    return M


print_boundary_r_c(5,7)
```

Out[95]:

```
[[1, 2, 3, 4, 5, 6, 7],
 [20, 0, 0, 0, 0, 0, 8],
 [19, 0, 0, 0, 0, 0, 9],
 [18, 0, 0, 0, 0, 0, 10],
 [17, 16, 15, 14, 13, 12, 11]]
```

In [ ]:

```python

```

Print Spiral

In [105]:

```python
def print_spiral(M):


    n = len(M)
    m = len(M[0])

    r = 0
    c = 0
    direction = "right"
    boundry = 0


    for _ in range(n*m):


        print(M[r][c],end = " ")
        if direction == "right":
            if c == m-1 - boundry:
                direction = "down"
                r = r+1
            else:
                c = c +1

        elif direction == "down":
            if r == n-1 - boundry:
                direction = "left"
                c = c-1
            else:
                r = r+1

        elif direction == "left":
            if c == 0 + boundry:
                direction = "up"
                r = r-1
            else:
                c = c-1

        elif direction == "up":
            if r == 1 + boundry:
                direction = "right"
                boundry += 1
                c = c+1
            else:
                r = r-1



M = [[0, 5, 10, 15, 20, 25, 30, 35, 40, 45],
     [1, 6, 11, 16, 21, 26, 31, 36, 41, 46],
     [2, 7, 12, 17, 22, 27, 32, 37, 42, 47],
     [3, 8, 13, 18, 23, 28, 33, 38, 43, 48],
     [4, 9, 14, 19, 24, 29, 34, 39, 44, 49]]
print_spiral(M)

```

```
0 5 10 15 20 25 30 35 40 45 46 47 48 49 44 39 34 29 24 19 14 9 4 3 2 1 6 11 16 21 26 31 36 41 42 43 38 33 28 23 18 13 8 7 12 17
22 27 32 37
```

In [4]:

```python
def print_spiral(n , m):

    M = [[0]*m for _ in range(n)]

    r = 0
    c = 0
    direction = "right"
    number = 0
    boundry = 0

    for _ in range(n*m):

        number +=  1
        M[r][c] = number
        if direction == "right":
            if c == m-1 - boundry:
                direction = "down"
                r = r+1
            else:
                c = c +1

        elif direction == "down":
            if r == n-1 - boundry:
                direction = "left"
                c = c-1
            else:
                r = r+1

        elif direction == "left":
            if c == 0 + boundry:
                direction = "up"
                r = r-1
            else:
                c = c-1

        elif direction == "up":
            if r == 1 + boundry:
                direction = "right"
                boundry += 1
                c = c+1
            else:
                r = r-1
    return M


print_spiral(5,5)
```

Out[4]:

```
[[1, 2, 3, 4, 5],
 [16, 17, 18, 19, 6],
 [15, 24, 25, 20, 7],
 [14, 23, 22, 21, 8],
 [13, 12, 11, 10, 9]]
```

In [ ]:

```
[[1, 2, 3],
 [8, 9, 4],
 [7, 6, 5]]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Q2. Christmas Trees </> Solved

Stuck somewhere?

Ask for help from a TA and get it resolved.

Get help from TA.

Problem Description

You are given an array **A** consisting of heights of Christmas trees and an array **B** of the same size consisting of the cost of each of the trees (B_i is the cost of tree A_i, where $1 \le i \le size(A)$), and you are supposed to choose **3** trees (let's say, indices p, q, and r), such that $A_p < A_q < A_r$, where $p < q < r$. The cost of these trees is $B_p + B_q + B_r$.

You are to choose **3** trees such that their total cost is minimum. Return that cost.

If it is not possible to choose 3 such trees return **-1**.

Example Input

Input 1:

```
A = [1, 3, 5]
B = [1, 2, 3]
```

Input 2:

```
A = [1, 6, 4, 2, 6, 9]
B = [2, 5, 7, 3, 2, 7]
```

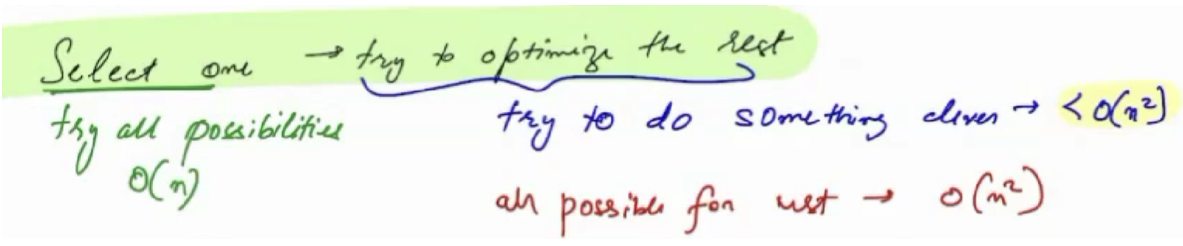
6, 7

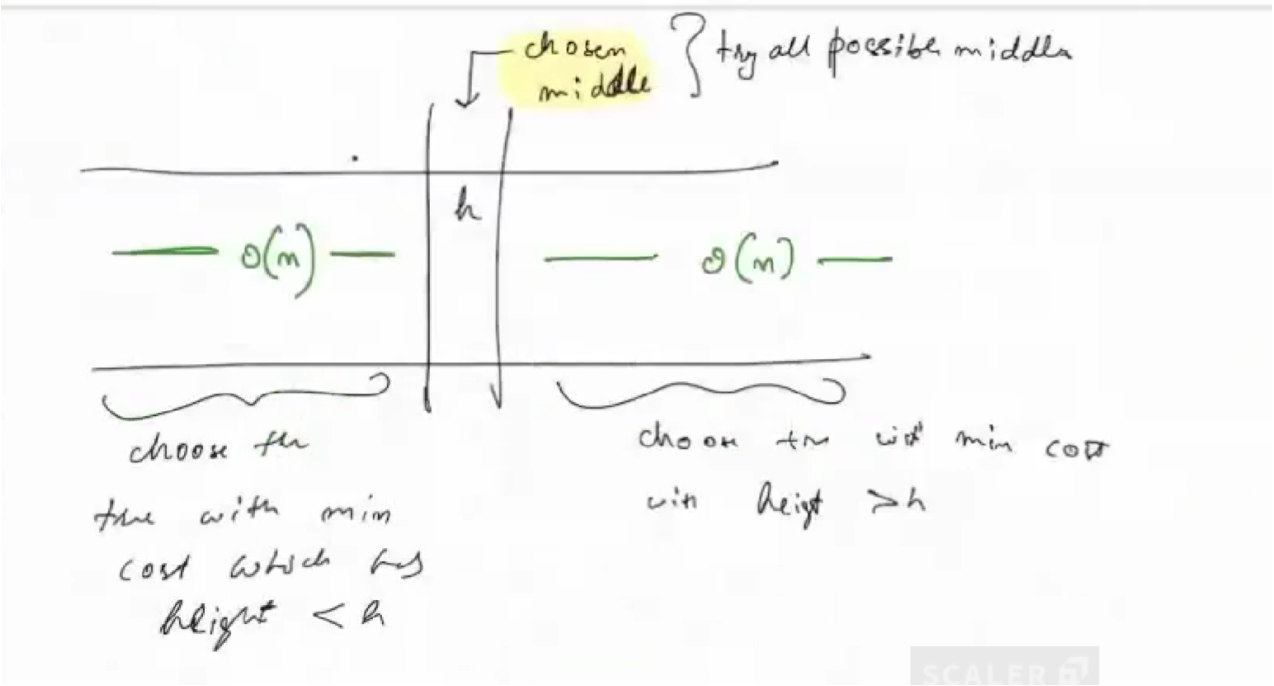
In [ ]:
```
1
```

In [372]:
```
1
```

In [390]:
```
def chistrmas_trees(H,C):   # O(n**3)O(1)

    n = len(C)

    mincost = float("inf")

    for t1 in range(n):
        for t2 in range(t1+1,n):
            for t3 in range(t2+1,n):
                if H[t1] < H[t2] < H[t3]:
                    cost = C[t1] + C[t2] + C[t3]
                    if cost < mincost:
                        mincost = cost
    return mincost

H = [9,7,15,3,6,1,2, 8,2,11,5,13,15,6]
C = [6,4, 1,5,8,5,9,11,6, 3,3, 2, 1,4]
chistrmas_trees(H,C)
```

Out[390]: 6





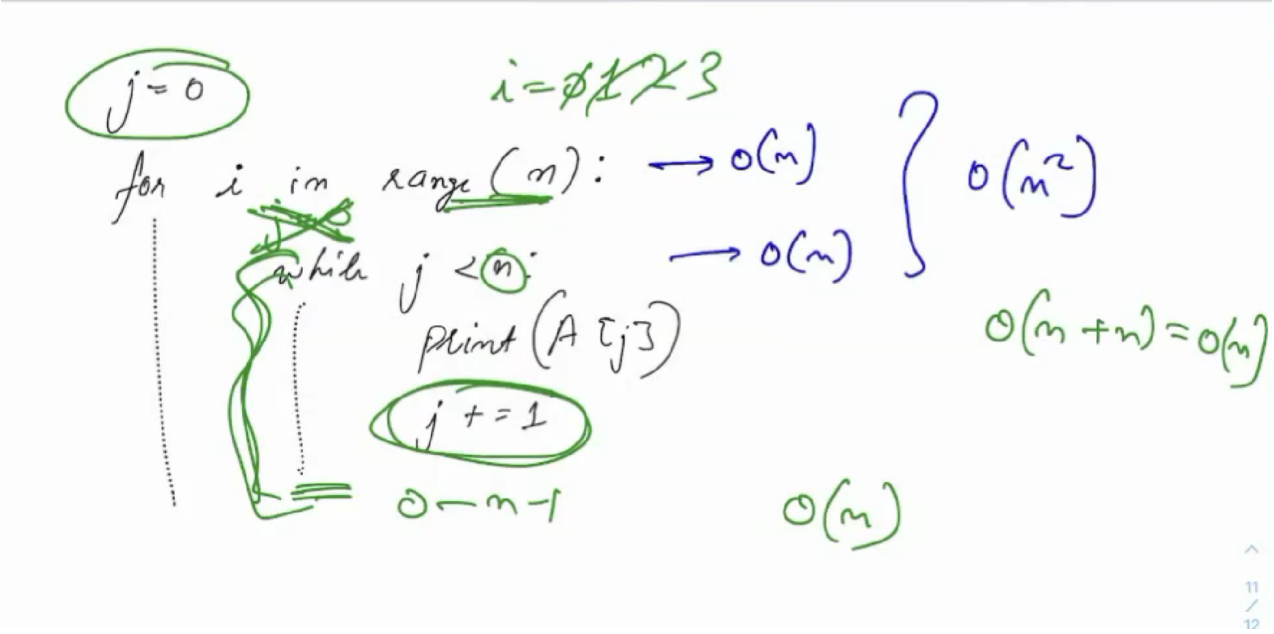
In [402]:
```python
def chistrmas_trees_opt(H,C):

    n = len(C)

    mincost = float("inf")

    for t2 in range(n):
        cost_t2 = C[t2]
        height_t2 = H[t2]

        cost_t1 = float('inf')
        for i in range(0,t2):
            if H[i] < height_t2:
                cost_t1 = min(cost_t1,C[i])

        cost_t3 = float('inf')
        for i in range(t2+1,n):
            if H[i] > height_t2:
                cost_t3 = min(cost_t3,C[i])

        cost = cost_t1 + cost_t2 + cost_t3

        mincost = min(cost,mincost)

    return mincost


H = [9,7,15,3,6,1,2, 8,2,11,5,13,15,6]
C = [6,4, 1,5,8,5,9,11,6, 3,3, 2, 1,4]
chistrmas_trees_opt(H,C)
```

Out[402]: 6

In [ ]:

In [ ]:

In [403]:
```python
# time complexity nice problem
```



In [ ]:
```
1
```

In [ ]:
```
1
```

Bit Manipulation

In [ ]:
```
1
```

```
10 base

xbase10 = [0-1-2-3-4-5-6-7-8-9]









81 base 10 = (a b c d e f g)base2
           = a*2^6 + b*2^5 + c*2^4 + d*2^3 + e*2^2 + f*2^1 + g*2^0
              a*64    b*32   c*16    d*8      e*4     f*2     g*1

             >-----------------------------------------<   |---|
                                 g is the only odd value in the total sum

because  81 is odd : g value is 1 >> g = 81%2 = 1
rest all a,b,c,d,e,f wil be even

80 = a*2^6 + b*2^5 + c*2^4 + d*2^3 + e*2^2 + f*2^1 + g*2^0
g == 1

40 = a*2^5 + b*2^4 + c*2^3 + d*2^2 + e*2^1 + f*2^0
f == 0,

20 = a*2^4 + b*2^3 + c*2^2 + d*2^1 + e*2^0
e == 0

10 = a*2^3 + b*2^2 + c*2^1 + d*2^0
d == 0

5 = a*2^2 + b*2^1 + c*2^0
c == 1

2 = a*2^1 + b*2^0
b == 0

1 = a*2^0
a == 1


ans : 1010001
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [415]:
```
nn = 81

anss = ""
while nn > 0 :

    digit = nn % 2
    anss += str(digit)
    nn = nn // 2

print(int(anss[::-1]))

```

```
1010001
```

In [467]:
```
def dectobin(nn):

    anss = ""
    while nn > 0 :

        digit = nn % 2
        anss = str(digit) + anss
        nn = nn // 2

    return ((anss))
dectobin(81)
```

Out[467]:
```
'1010001'
```

In [418]:
```
bin(81)
```

Out[418]:
```
'0b1010001'
```

In [424]:
```
int(math.log(81,2)+1)   # number of digits in binary string
```

Out[424]:
```
7
```

In [ ]:
```

```

In [434]:
```
def deci_to_oct(nn):

    anss = ""
    while nn > 0 :

        digit = nn % 8
        anss += str(digit)
        nn = nn // 8

    return ((anss[::-1]))
deci_to_oct(98)
```

Out[434]:
```
'142'
```

In [437]:
```
(98)%8
```

Out[437]:
```
2
```

In [432]:
```
((98//8)%8)
```

Out[432]:
```
4
```

In [433]:
```
(98//8)//8
```

Out[433]:
```
1
```

In [ ]:
```

```

In [ ]:
```

```

In [454]:
```
def deci_to_oct(nn,base):
    if base in [0,1]:
        return "base 0 or 1 not valid"
    anss = ""
    while nn > 0 :

        digit = nn % base
        anss += str(digit)
        nn = nn // base

    return ((anss[::-1]))
deci_to_oct(98,5)
```

Out[454]:
```
'343'
```

In [ ]:
```

```

int a = x // random value x is assigned to a
print(a&1)

55 users have participated

A	0	0%
B	1	9%
C	0 if a is odd 1 if a is even	40%
✓ D	0 if a is even 1 if a is odd	49%
E	None of the above	2%

int a = x // random value x is assigned to a
print(a|1)

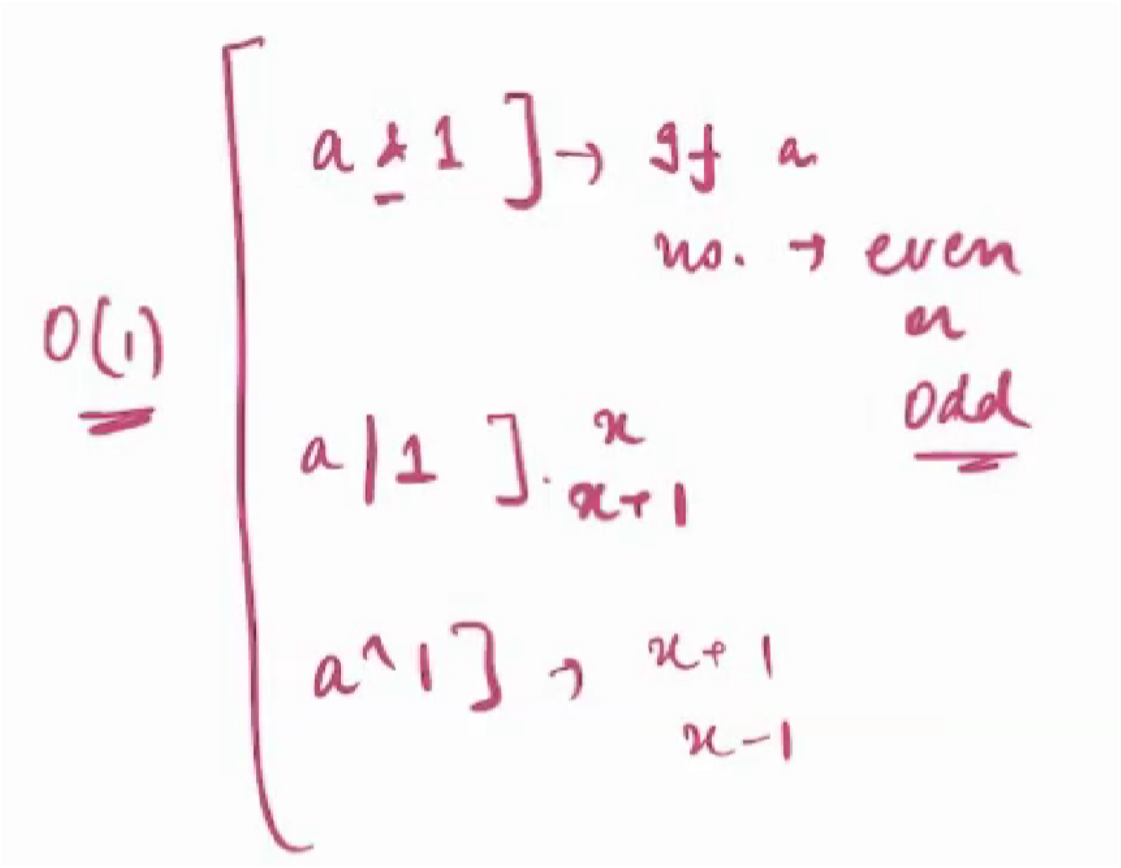
52 users have participated

A	1111111111111.. [32 1's]	4%
B	x+1	12%
C	if a is odd x+1 If a is even x	44%
✓ D	if a is odd x If a is even x+1	40%

int a = x // random value x is assigned to a
print(a^1)

49 users have participated

A	-a-1	0%
B	~a	2%
✓ C	if a is even = x+1 if a is odd = x-1	65%



In [ ]:
```
1
```

In [9]:
```
1 print(bin(52)[2:])
2 print("  ",bin(4)[2:])
```

```
110100
   100
```

In [12]:
```
1 # if we want to turn on/off the bit :
2
3 # 52 & 4 = 4   # will check the bit on or off
4
5
```

In [15]:
```
1 binarytodecimal(9)
```

Out[15]: 9

In [ ]:
```
1 110100  52
2 110000  48
3 110000  48
```

In [ ]:
```
1
```

In [16]:
```
1 bin(9)
```

Out[16]: '0b1001'

In [13]:
```
1  def binarytodecimal(s):   # binary sting
2      s = str(s)
3      i = 0
4      result = 0
5      for d in s[::-1]:
6          result = result+ int(d)*(2**i)
7          i = i+1
8      return result
9  x = str(binarytodecimal(10100))
10 print(x)
```

```
20
```

In [ ]:
```
1
```

Q1. Single Number </> Solved

Asked in: SH TO

Problem Description

Given an array of integers **A**, every element appears twice except for one. Find that integer that occurs once.

NOTE: Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

In [20]:
```
A = [1, 2, 2, 3, 1]

def single_number(A):
    n = len(A)
    ans = A[0]
    for i in range(1,n):
        ans = ans ^ A[i]
    return ans
single_number(A)
```

Out[20]: 3

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [21]:
```
bin(44)[2:]
```

Out[21]: '101100'

In [22]:
```
# check if last bit is on/off :

44&1
```

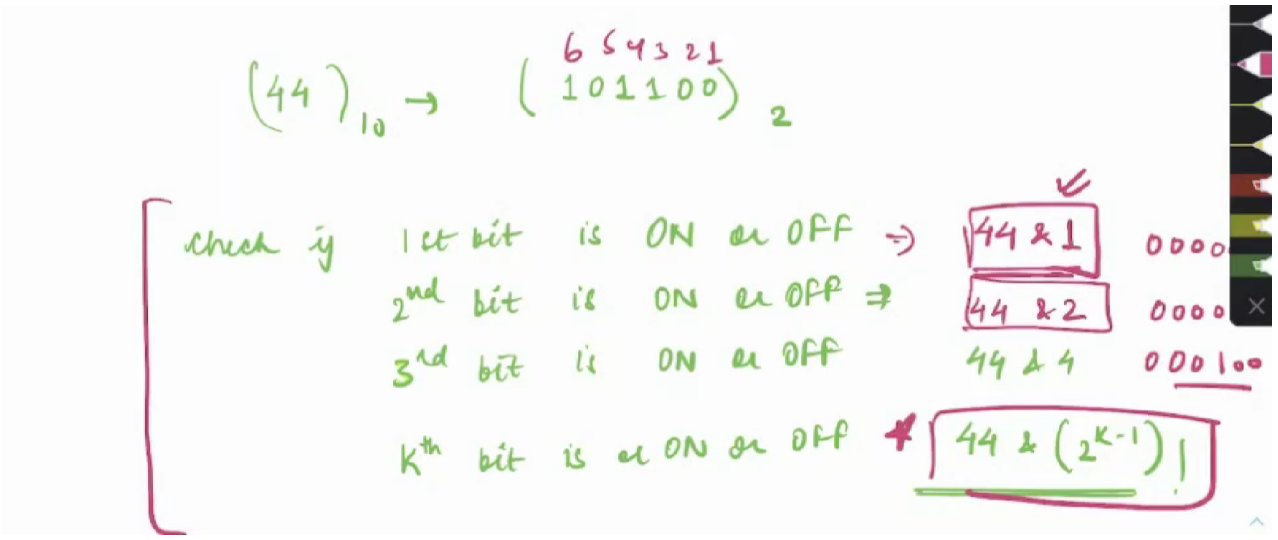
Out[22]: 0

In [23]:
```
# 2nd bit is on or off ? 101100
44 &2
```

Out[23]: 0

In [24]:
```
# 3nd bit is on or off ? 101100
44&4    # ans is not 0 , means its on
```

Out[24]: 4



In [ ]:
```
1
```

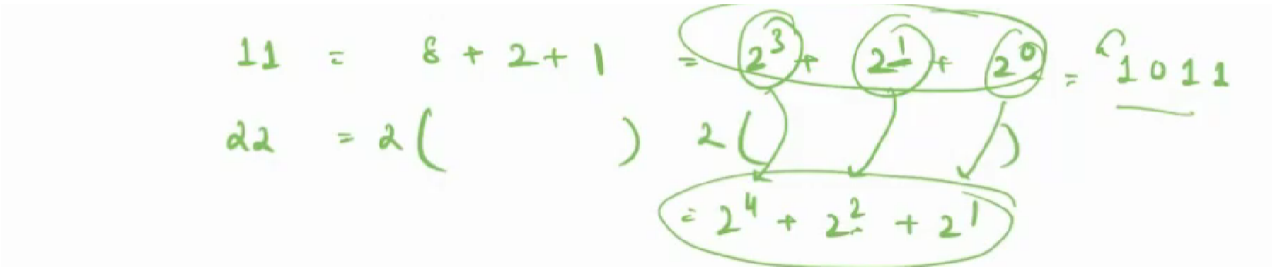
In [27]:
```
1 print(bin(11)[2:])
2 print(bin(22)[2:])
3 print(bin(44)[2:])
4
```

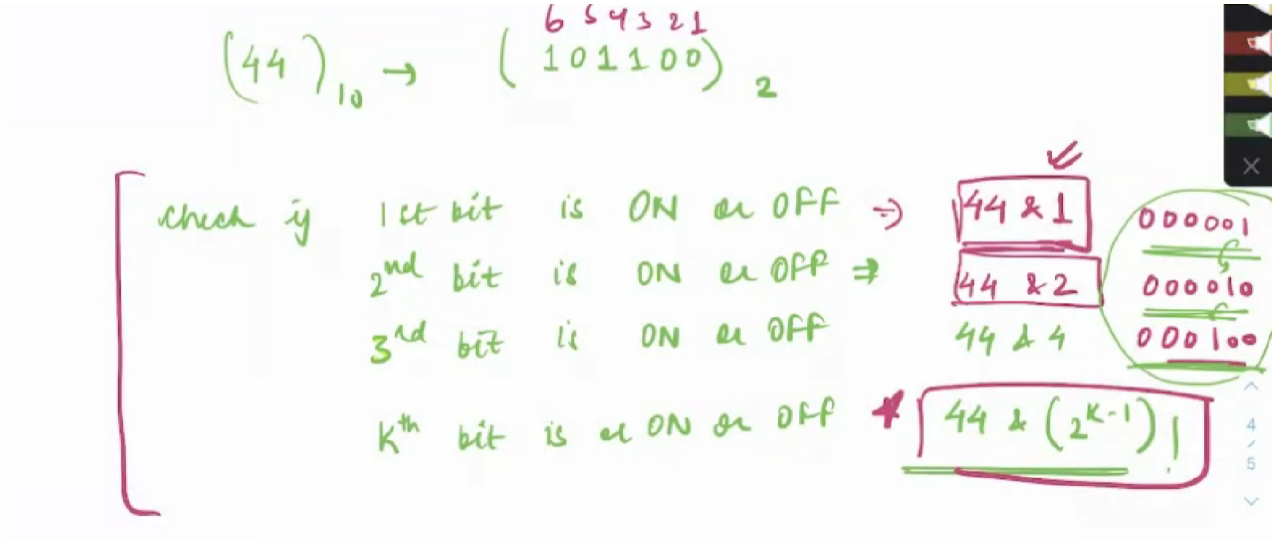
```
1011
10110
101100
```

In [28]:
```
1 print(bin(31)[2:])
2 print(bin(62)[2:])
3 print(bin(124)[2:])
4
```

```
11111
111110
1111100
```

In [ ]:
```
1 # shifting all bits towards left
2
```





In [35]:
```
# shifting operation is very useful and it
#has some correlation with multiplication by 2


print(bin(1)[2:])
print(bin(2)[2:])
print(bin(4)[2:])
```

```
1
10
100
```

In [31]:
```
1<<1
```

Out[31]: (2, '1')

In [41]:
```
3<<1, bin(3)[2:], bin(3<<1)[2:]
```

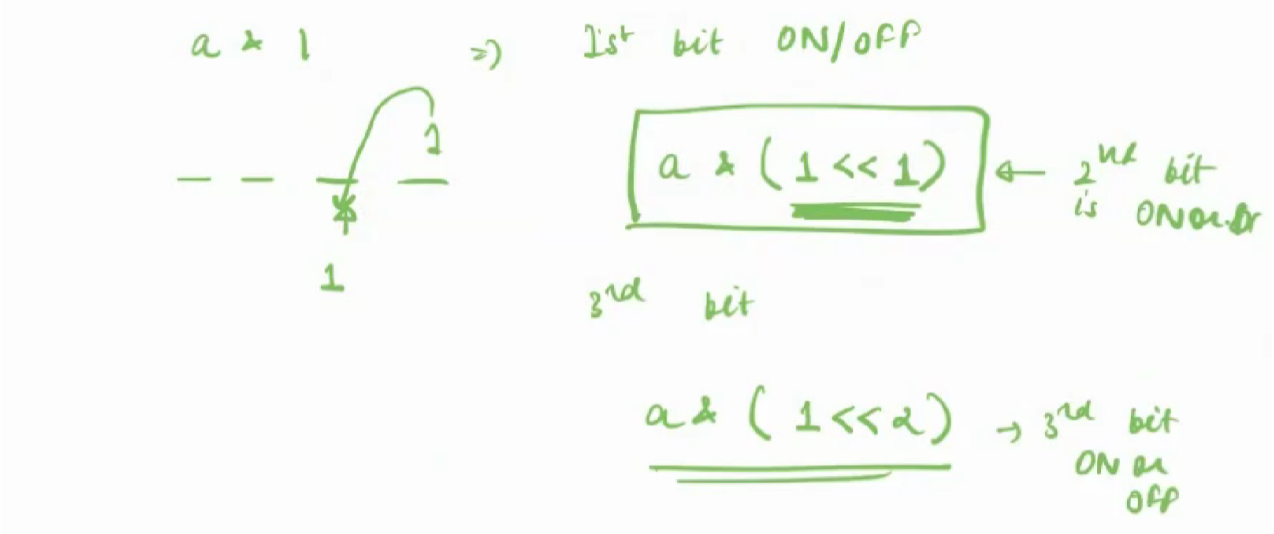
Out[41]: (6, '11', '110')

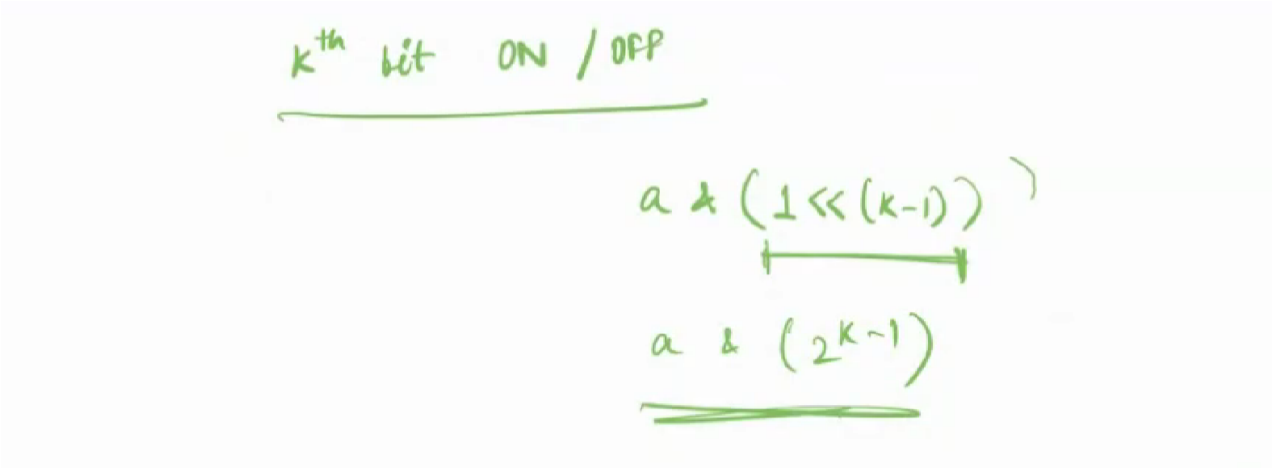
In [42]:
```
3<<2, bin(3)[2:], bin(3<<2)[2:]
```

Out[42]: (12, '11', '1100')

In [ ]:
```

```

In [ ]:
```
#  a << x ==  a*(2^x)
```





In [45]:
```
15<<2, 15*(2**2)  # a = 15, k = 2
```

Out[45]: (60, 60)

In [46]:
```
2<<2
```

Out[46]: 8

In [47]:
```
binarytodecimal(1101)
```

Out[47]: 13

In [48]:
```
13&(1<<3)
```

Out[48]: 8

In [ ]:
```

```

In [51]:
```
# check if ith bit is on or off !

def check_i_bit(num,i):

    if num & (1 << (i-1)) == 0:
        return "OFF"
    return "ON"
```

In [54]:
```
check_i_bit(7,4)
```

Out[54]: 'OFF'

In [55]:
```
bin(7)
```

Out[55]: '0b111'

In [ ]:
```

```

In [56]:
```
44>>2
```

Out[56]: 11

In [ ]:
```
# 101100 ==> 1011
```

In [ ]:
```
# << left shifting is multiplication by  *2
# >> rifgt shifting is integer division //2
```

In [ ]:
```

```

In [57]:
```
44 >> 3, 44//(2**3)
```

Out[57]: (5, 5)

In [59]:
```
29>>2, 29//(2**2)

```

Out[59]: (7, 7)

In [ ]:
```

```

In [60]:
```
binarytodecimal(1011)
```

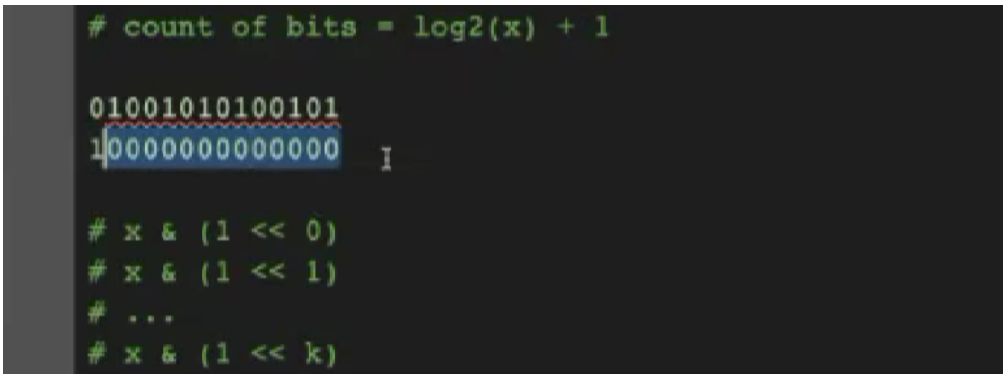
Out[60]: 11

In [62]:
```
11//(2**3),11>>3
```

Out[62]: (1, 1)

In [ ]:
```

```

define a function to calculate number of set bits in binary representation of any number



In [72]:
```
def count_set_bits(num):

    k = 0
    count = 0
    while True:
        if num & (1 << k) != 0:
            count += 1
        k  = k+1
        if num < (1<<k):
            break
    return count
count_set_bits(13)
```

Out[72]: 3

In [83]:
```
def count_set_bitsX(num):

    k = 0
    count = 0
    while k < len(bin(num)[2:]):
        if num & (1 << k) != 0:
            count += 1
        k  = k+1
    return count
count_set_bitsX(654)
```

Out[83]: 5

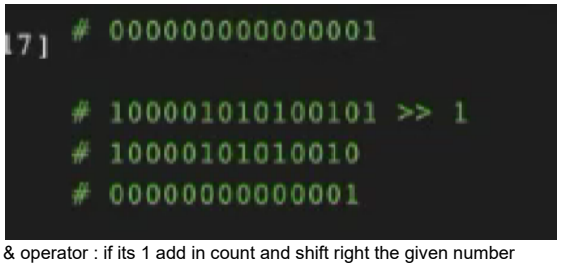
In [81]:
```
bin(654)[2:]
```

Out[81]: '1010001110'

In [ ]:
```

```

In [85]:
```
def count_set_bit_with_rightshift_optr(n):
    cnt = 0
    while n > 0:
        cnt += n & 1
        n = n >> 1
    return cnt
count_set_bit_with_rightshift_optr(654)
```

Out[85]: 5



keepchecking the last bit if its 0 or nt : using & operator : if its 1 add in count and shift right the given number

In [ ]:
```

```

In [ ]:
```

```

Type *Markdown* and LaTeX: α^2

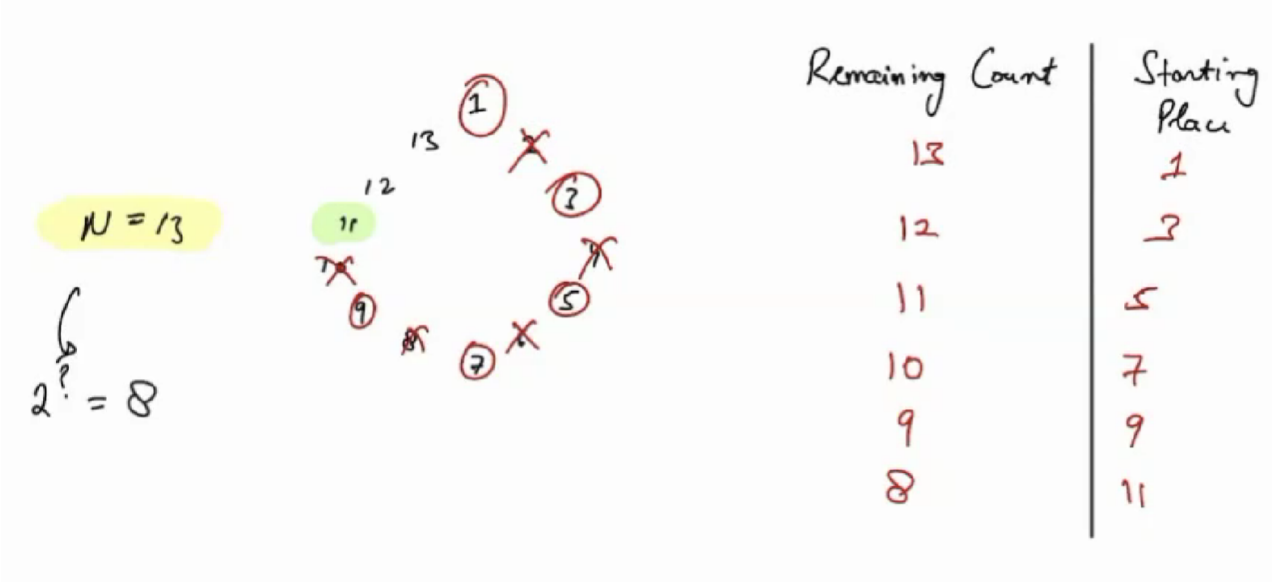
In [ ]:
```

```

In [160]:
```
def people(num):

    a =[i for i in range(1,num+1)]

    p = 0

    while len(a) > 1:
        p += 1

        p = p % len(a)
        del a[p]


    return a[0]
```

In [161]:
```
people(1000)
```

Out[161]: 977

In [ ]:
```

```



In [178]:
```
total_people = 11

def josephus_problm(total_people):
    power = 1
    while power <= total_people:
        power = power * 2
    power = power // 2

    number_of_people_killed = total_people - (power)

    person_left_alive = 2 * (number_of_people_killed) +1

    return person_left_alive
```

In [180]:
```
josephus_problm(1000)
```

Out[180]: 977

In [177]:
```
def josephusProblem(N):

    i = 1
    while i <= N :
        i = i*2
    power = i//2
    C = N - power

    return (2*C+1)
```

In [ ]:
```

```

In [186]:
```
N = 15
i = 1
while i <= N :
    i = i*2
power = i//2
print(power)
```

```
8
```

In [ ]:
```

```

Q1. Majority Element </> Solved

Asked in: a SA

Problem Description

Given an array of size **N**, find the majority element. The majority element is the element that appears more than floor(n/2) times.
You may assume that the array is non-empty and the majority element always exists in the array.

majority elements

In [194]:
```
A = [1,2,3,5,5,1,5,5,5,2,3,4,4,5,5,5,5,5,6,7,8,9,5,5]
n = len(A)
freq_required = n//2
ans = None
for i in A:
    c = 0
    for j in A:
        if i == j :
            c += 1
    if c >= freq_required:
        ans = i
print(ans)
```

```
5
```

In [ ]:
```

```

In [198]:
```
A = [1,2,3,5,5,1,5,5,5,2,3,4,4,5,5,5,5,5,6,7,8,9,5,5]
n = len(A)
A = sorted(A)
freq_required = n//2
ans = None
curr = None
freq = 0
for x in A:
    if x == curr:
        freq += 1
        if freq > freq_required:
            ans = curr
    else:
        curr = x
        ferq = 1
print(ans)
```

```
5
```

In [ ]:
```

```

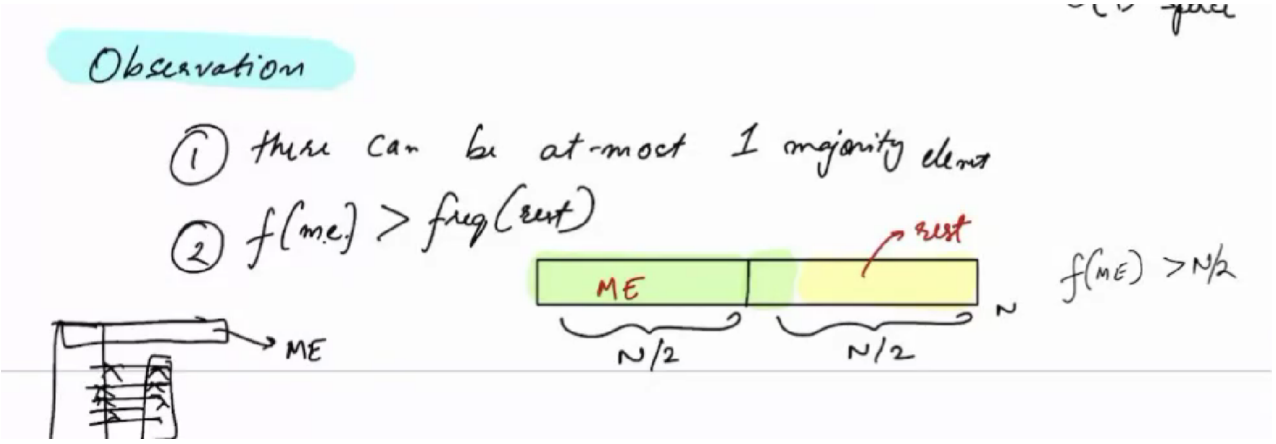
In [200]:
```
from collections import Counter
```

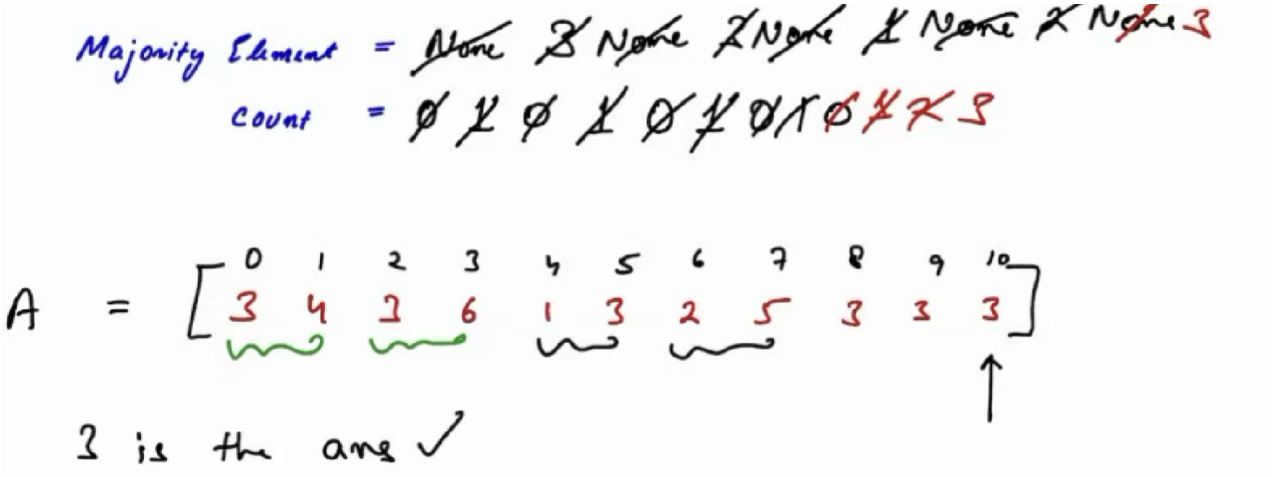
In [209]:
```
A = [1,2,3,5,5,1,5,5,5,2,3,4,4,5,5,5,5,5,6,7,8,9,5,5]
n = len(A)
freq_required = n//2

hmap = Counter(A)
ans = None
for x in A:
    if hmap[x] > freq_required:
        ans = x
print(x)

```

```
5
```

In [210]:
```
hmap
```

Out[210]:
```
Counter({1: 2, 2: 2, 3: 2, 5: 12, 4: 2, 6: 1, 7: 1, 8: 1, 9: 1})
```

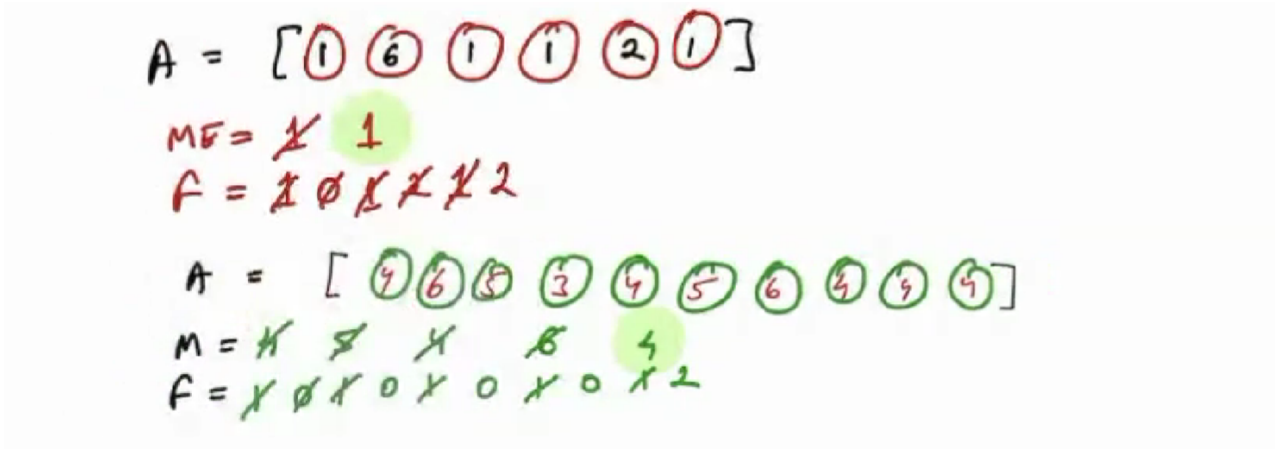


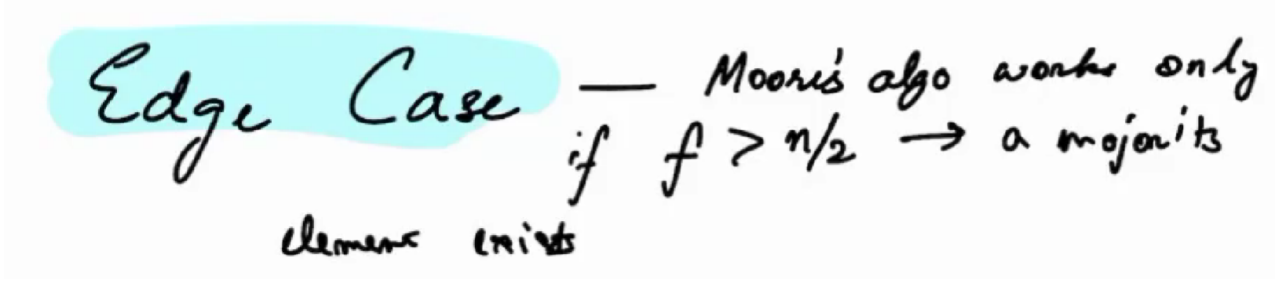


In [293]:

```python
A = [5,5,3,4,5,5,3,5,6,5,5,5,1,3,3,3,5,5,5,3,3,5,5,3,3,5,2,5,3,3,3,5,5,5]
n = len(A)
ans = None
freq_required = n//2


ME = None
count = 0

for i in range(n):

    if ME == A[i]:
        count += 1
        print(A[i],"ME=",ME,"  count = ",count)

    elif count == 0:
        ME = A[i]
        count += 1
        print(A[i],"ME=",ME,"  count = ",count)
    else:
        count -= 1
        print(A[i],"      count = ",count)
print()
print()
print("Majority Element",ME)
f = 0
for i in A:
    if i == ME:
        f += 1
        if f > freq_required:
            ans = i
print()
print(ans)
```

```
5 ME= 5   count =  1
5 ME= 5   count =  2
3       count =  1
4       count =  0
5 ME= 5   count =  1
5 ME= 5   count =  2
3       count =  1
5 ME= 5   count =  2
6       count =  1
5 ME= 5   count =  2
5 ME= 5   count =  3
5 ME= 5   count =  4
1       count =  3
3       count =  2
3       count =  1
3       count =  0
5 ME= 5   count =  1
5 ME= 5   count =  2
5 ME= 5   count =  3
3       count =  2
3       count =  1
5 ME= 5   count =  2
5 ME= 5   count =  3
3       count =  2
3       count =  1
5 ME= 5   count =  2
2       count =  1
5 ME= 5   count =  2
3       count =  1
3       count =  0
3 ME= 3   count =  1
5       count =  0
5 ME= 5   count =  1
5 ME= 5   count =  2


Majority Element 5

5


`
```





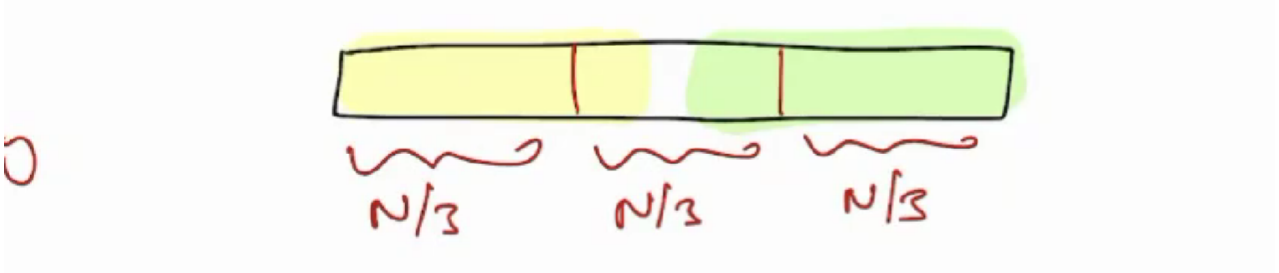
In [ ]:
```
1
```

Q4. N/3 Repeat Number </> Solved

Problem Description

You're given a read-only array of **N** integers. Find out if any integer occurs more than N/3 times in the array in linear time and constant additional space.
If so, return the integer. If not, return **-1**.

If there are multiple solutions, return any one.



In [254]:

```
A = [5,5,3,4,5,5,3,5,6,5,5,5,1,3,3,3,5,5,5,3,3,5,5,3,3,5,2,5,3,3,3,5,5,5]
n = len(A)
ans = None
freq_required = n//3


ME1 = None
ME2 = None

count1 = 0
count2 = 0

for i in range(n):

    if ME1 == A[i]:
        count1 += 1
    elif ME2 ==A[i]:
        count2 += 1

    elif count1 == 0:
        ME1 = A[i]
        count1 += 1
    elif count2 == 0:
        ME2 = A[i]
        count2 += 1

    else:
        count1 -= 1
        count2 -= 1
print()
print()
print("Majority Element",ME1,ME2)

f1 = 0
f2 = 0

for i in range(n):
    if A[i]== ME1:
        f1 += 1
    elif A[i] == ME2:
        f2+= 1
ans1 = None
if f1 > freq_required:
    ans1 = ME1
ans2 = None
if f2 > freq_required:
    ans2 = ME2
print(ans1,ans2)


```

```
Majority Element 5 3
5 3
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Q2. Monkeys and Doors

Solved

Stuck somewhere?

Ask for help from a TA and get it resolved.

Get help from TA.

There are 100 doors, all closed. In a nearby cage are 100 monkeys.

The first monkey is let out and runs along the doors opening every one. The second monkey is then let out and runs along the doors closing the `2nd, 4th, 6th,...` - **all the even-numbered doors**. The third monkey is let out. He attends only to the `3rd, 6th, 9th,...` doors **(every third door, in other words)**, closing any that is open and opening any that is closed, and so on. After all 100 monkeys have done their work in this way, what state are the doors in after the last pass?

Answer with the number of open doors.

> *Answer is an integer.* Just put the number without any decimal places if it's an integer. If the answer is Infinity, output ***Infinity***.
>
> *Feel free to get in touch with us if you have any questions*

In [281]:
```
n = 100

def open_doors_count(n):

    A = [False]* n

    for i in range(1,n+1):
        for j in range(i,n+1,i):
            A[j-1] = not A[j-1]
#         print(i,A)
    c = 0
    for x in A:
        if x == True:
            c += 1
    return c
open_doors_count(100)
```

Out[281]: 10

In [288]:
```
open_doors_count(150)
```

Out[288]: 12

In [ ]:
```

```

In [290]:
```
c = 0
i = 1
N = 150


while i*i <= N:
    c += 1
    i = i+1
print(c)
```

12

In [291]:
```python
def perfect_square(n):

    low = 1
    high = n
    mid = (low + high)//2

    while low <= high:
        mid = (low + high)//2
        if mid * mid > n:
            high = mid - 1

        elif mid * mid < n:
            low = mid + 1
        else:
            return mid
    return high


perfect_square(150)

```

Out[291]: 12

In [ ]:
```python

```

In [ ]:
```python

```

Q5. Find nth Magic Number </> Solved

Asked in: DS AC a

Stuck somewhere?

Ask for help from a TA and get it resolved.

Get help from TA.

Problem Description

Given an integer **A**, find and return the **Ath** magic number.

A magic number is defined as a number that can be expressed as a power of 5 or a sum of **unique** powers of 5.

First few magic numbers are 5, 25, 30(5 + 25), 125, 130(125 + 5), ...

In [298]:
```python
bin(10)[2:][::-1]
```

Out[298]: '0101'

In [301]:
```python
(0*(5*0))+((5**2)*1)+((5**3)*0)+((5**4)*1)
```

Out[301]: 650

In [ ]:
```python

```

In [317]:
```python
def nth_magic_number(n):
    s = bin(n)[2:][::-1]

    p = 1
    ans = 0
    for i in s:
        ans += int(i)*(5**p)
        p += 1

    return ans
```

In [319]:
```
nth_magic_number(3)
```

```
1
1
```

Out[319]: 30

In [ ]:

In [ ]:

In [ ]:

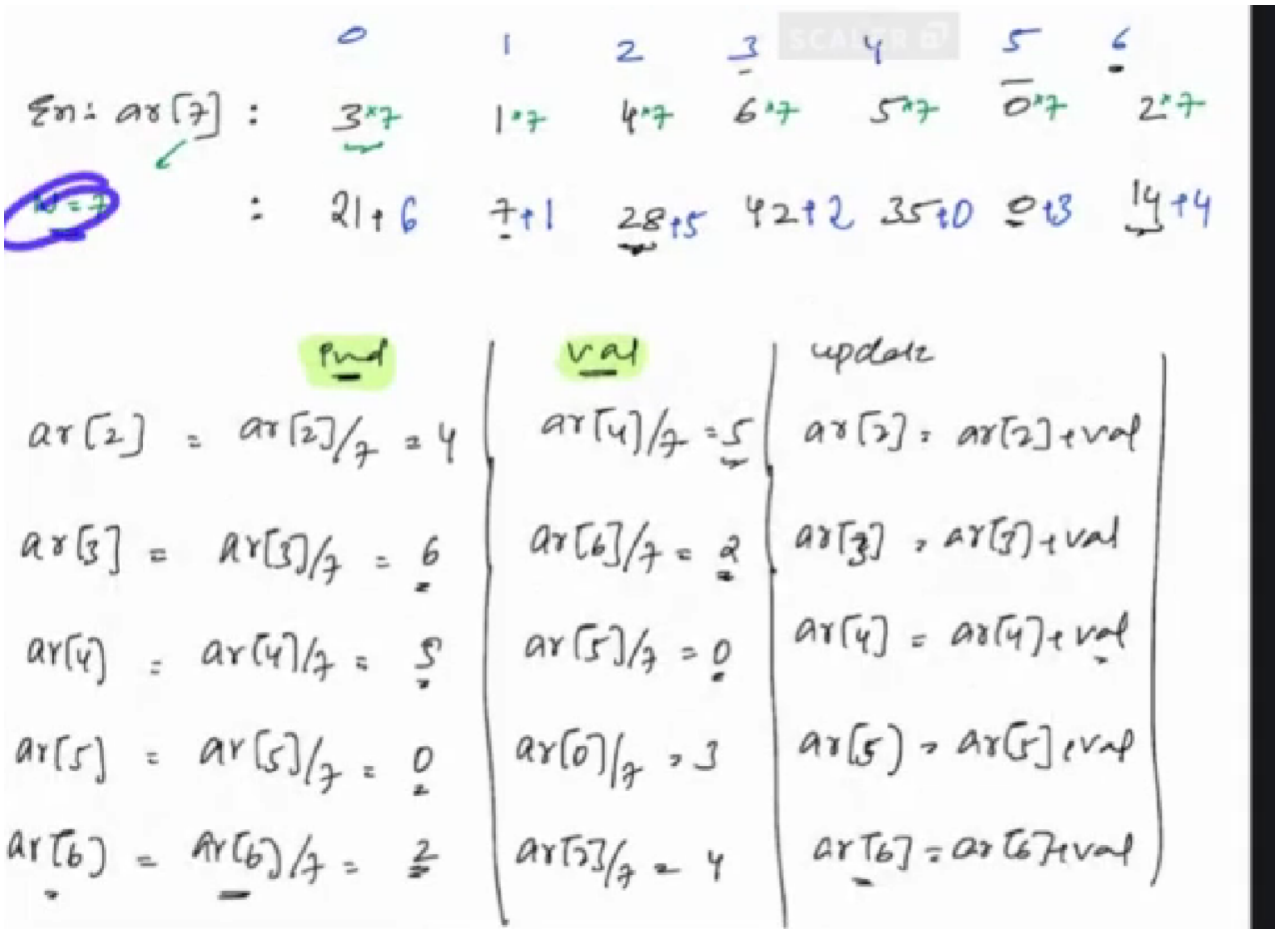
In [ ]:

In [ ]:

In [ ]:

In [ ]:

rearange elements of A array of n integers containing elements from 0 to n-1 , such that A[i]= A[A[i]]



In [357]:
```
A = [3,1,4,7,6,5,0,2]
n = len(A)
```

In [358]:
```
for i in range(n):
    A[i] = n*A[i]

```

In [359]:
```
A
```

Out[359]: [24, 8, 32, 56, 48, 40, 0, 16]

In [ ]:

In [360]:
```
for i in range(n):

    A[i] += A[A[i]//n]//n
```

In [361]:
```
A
```

Out[361]: [31, 9, 38, 58, 48, 45, 3, 20]

In [362]:
```
for i in range(n):
    A[i] = A[i]%n
```

In [363]:
```
A
```

Out[363]: [7, 1, 6, 2, 0, 5, 3, 4]

In [364]:
```
_
```

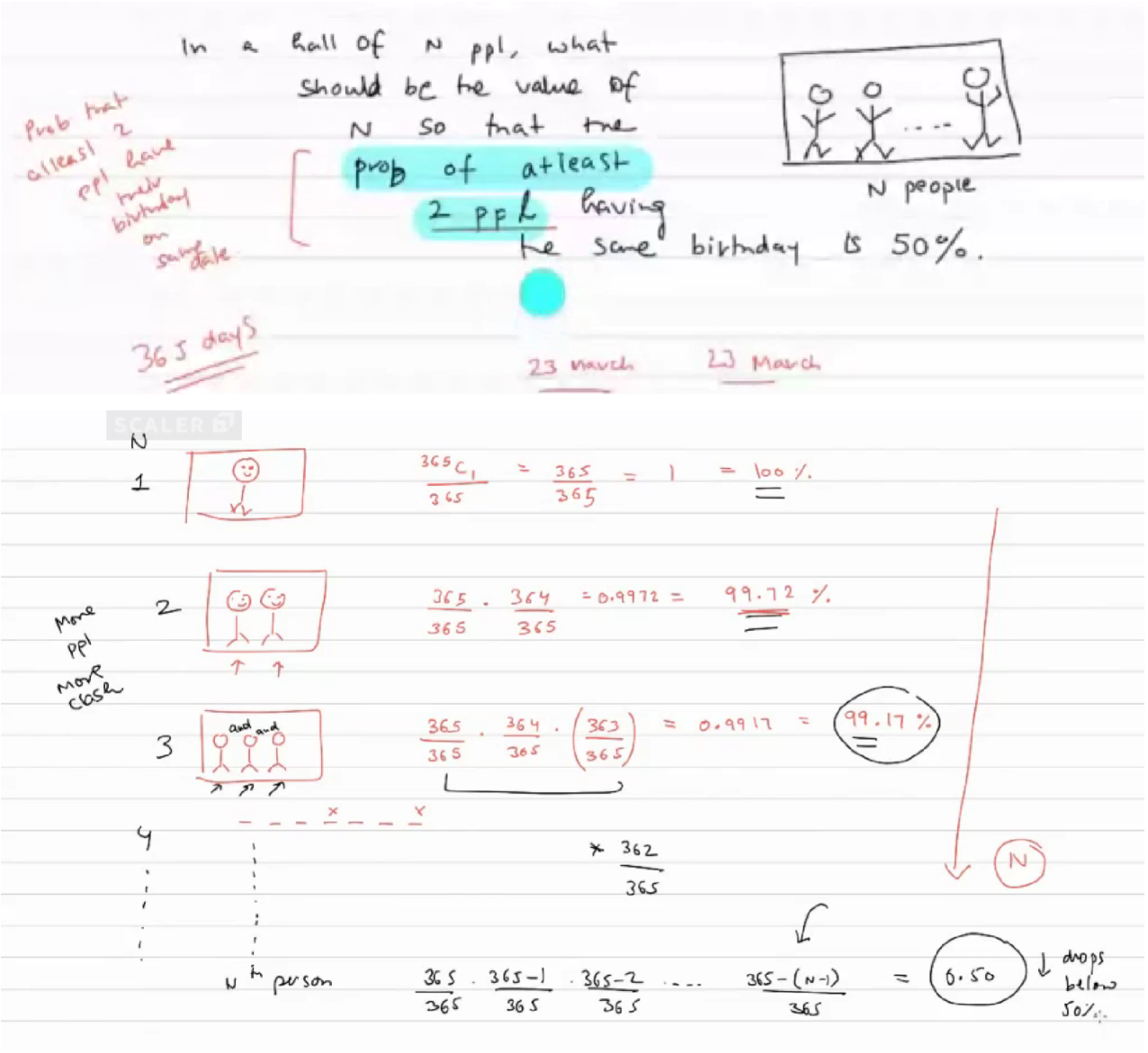
Out[364]: [7, 1, 6, 2, 0, 5, 3, 4]

In [ ]:

In [ ]:

In [ ]:

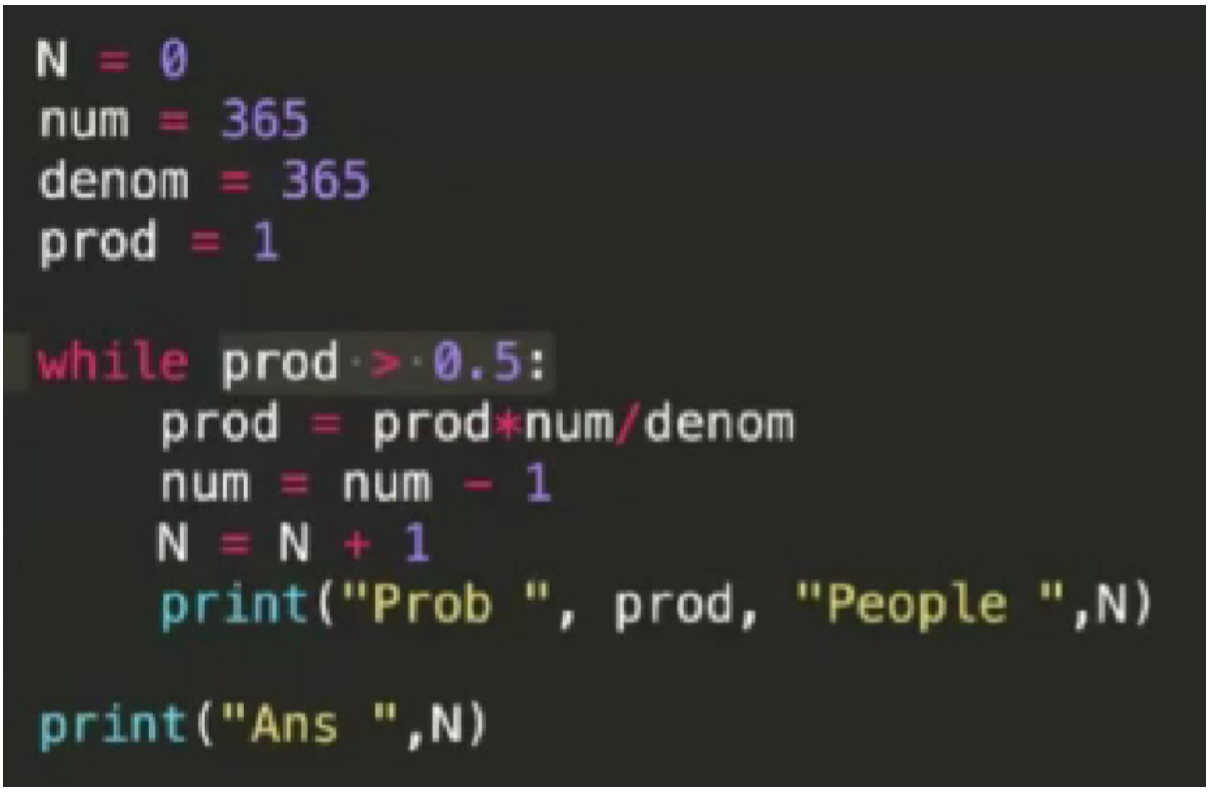
in a hall of N people, what should be value of N (howmany people should be in the hall), such that probability of atleast two people havng the same birthday is >= 50 %.



P(atleast 2 ppl same bday) = p

P (all ppl have diff bday) = 1 − p

```
Prob  1.0 People  1
Prob  0.997260273972602 People  2
Prob  0.9917958341152185 People  3
Prob  0.9836440875334497 People  4
Prob  0.9728644263002063 People  5
Prob  0.9595375163508884 People  6
Prob  0.9437642969040244 People  7
Prob  0.9256647076483308 People  8
Prob  0.9053761661108333 People  9
Prob  0.8830518222889221 People  10
Prob  0.8588586216782667 People  11
Prob  0.8329752111619353 People  12
Prob  0.8055897247675702 People  13
Prob  0.7768974879950266 People  14
Prob  0.7470986802363133 People  15
Prob  0.7163959947471498 People  16
Prob  0.6849923347034391 People  17
Prob  0.6530885821282105 People  18
Prob  0.6208814739684632 People  19
Prob  0.5885616164194198 People  20
Prob  0.5563116648347941 People  21
Prob  0.5243046923374498 People  22
Prob  0.4927027656760144 5 People  23
Ans  23
```



```
N = 0
num = 365
denom = 365
prod = 1

while prod > 0.5:
    prod = prod*num/denom
    num = num - 1
    N = N + 1
    print("Prob ", prod, "People ",N)

print("Ans ",N)
```

In [370]:

```
number_of_people = 0
choose_out_of_days = 365
days_of_year = 365

probability  = 1

while probability > 0.5:
    probability = (probability * choose_out_of_days)/days_of_year
    choose_out_of_days -=  1
    number_of_people += 1
print(number_of_people)
```

```
23
```

In [377]:

```
n = 0
nCr = 365
total = 365
probability = 1

while probability > 0.5:
    probability = (probability* nCr )/total
    nCr -= 1
    n += 1
    print("Probability:",round(probability,4),"  People : ",n)
print(n)
```

```
Probability: 1.0   People :  1
Probability: 0.9973   People :  2
Probability: 0.9918   People :  3
Probability: 0.9836   People :  4
Probability: 0.9729   People :  5
Probability: 0.9595   People :  6
Probability: 0.9438   People :  7
Probability: 0.9257   People :  8
Probability: 0.9054   People :  9
Probability: 0.8831   People :  10
Probability: 0.8589   People :  11
Probability: 0.833   People :  12
Probability: 0.8056   People :  13
Probability: 0.7769   People :  14
Probability: 0.7471   People :  15
Probability: 0.7164   People :  16
Probability: 0.685   People :  17
Probability: 0.6531   People :  18
Probability: 0.6209   People :  19
Probability: 0.5886   People :  20
Probability: 0.5563   People :  21
Probability: 0.5243   People :  22
Probability: 0.4927   People :  23
23
```

In [ ]:

```

```

In [373]:

```

```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

Q2. Elements Removal </> Solved

Stuck somewhere?

Ask for help from a TA and get it resolved.

Get help from TA.

Problem Description

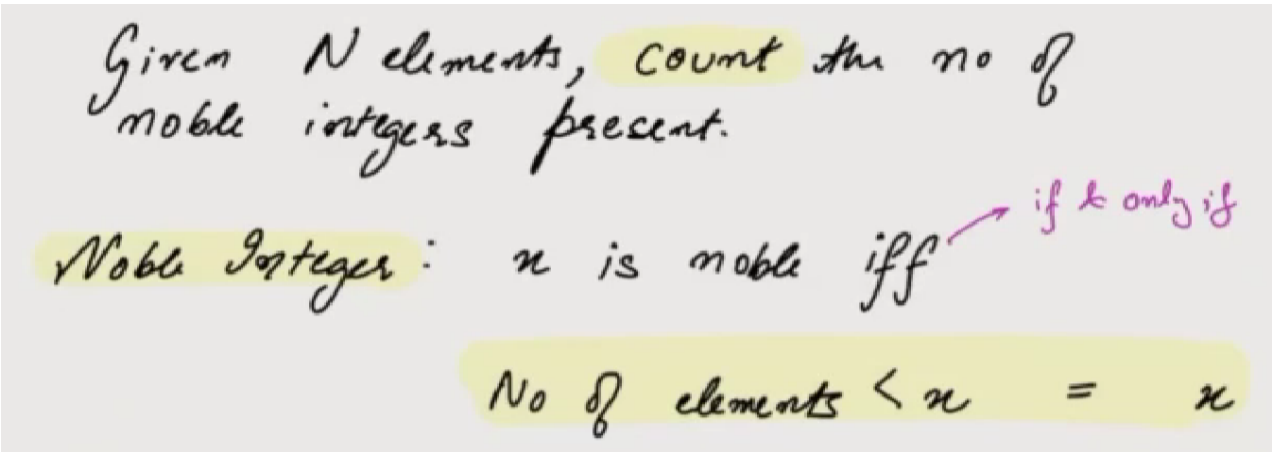
Given an integer array **A** of size **N**. You can **remove** any element from the array in one operation.
The cost of this operation is the **sum of all elements** in the array present **before this operation**.

Find the **minimum cost** to remove all elements from the array.

In [322]:
```
1  A = [3,5,1,-3]
2
3  def elements_removal(A):
4      n = len(A)
5
6      A = sorted(A,reverse = True)
7      x = 1
8      cost = 0
9      for i in range(n):
10         cost += (x*A[i])
11         x += 1
12     return cost
13 elements_removal(A)
```

Out[322]: 2

In [ ]:
```
1
```



In [331]:
```
```

Out[331]: 3

In [361]:
```
A = [-10,1,1,2,2,4,4,4,8,10]

def nobel_elements(A):
    n = len(A)
    c = 0
    for x in A:
        less_number_counts  = 0
        for y in A:
            if x > y:
                less_number_counts += 1
        if less_number_counts == x:
            c += 1
    return c
print(nobel_elements([1,-5,3,5,5,-10,4]))
print(nobel_elements([-10, -5, 1, 3, 4, 5, 10]))
```

```
4
3
```

In [380]:
```
A = [-10,1,1,2,4,4,4,8,10]

nobel_elements(A)
```

Out[380]: 5

In [342]:
```
# if list has no duplicates :
# sort in ascending order : the indexes == less numbers counts

A = [-10, -5, 1, 3, 4, 5, 10]
x = sorted(A)
c = 0
for  i in range(len(x)):
    if i == x[i]:
        c += 1
print(c)
```

```
3
```

In [ ]:
```
```

In [366]:
```
 #     0 1 2 3 4 5 6 7 8 9
A = [-10,1,1,2,2,4,4,4,8,10] # already sorted
     # 0 1 1 3 3 5 5 5 8 9
# in case of duplicates
n = len(A)

count = 0
prev = None
prev_i = None

for i,x in enumerate(A):
    if x == prev:
        lesser_elements = prev_i
    else:
        lesser_elements = i  # if x is not equal to previous element:
                             # then lesser than x elements will be i
        prev = x               # update previous element
        prev_i = i             # update prev_i for in case of duplicte
    if lesser_elements == x :  # if lesser ele == x : add 1 in count
        count += 1
print(count)
```

```
3
```

In [376]:
```
# easy code
```

In [379]:
```
X = [-10,1,1,2,4,4,4,8,10]
A = sorted(X)

n = len(A)
nobel  = 0
lesser = 0
for i in range(n):
    if i > 0 and A[i] == A[i-1]:
        nobel += 1
        nobel -= 1  #  do not increase
    else:
        lesser = i
    if A[i] == lesser:
        nobel += 1

print(nobel)
```

```
5
```

In [ ]:
```
1
```

custom sorting

In [381]:
```
# return sorting array by abs values

nums = [-10,3,5,6,10,4,-5,-44]

from functools import cmp_to_key

```

In [383]:
```
def cmpr_abs(a,b):
    if abs(a) < abs(b):
        return -1
    elif abs(a) > abs(b):
        return 1
    else:
        return 0
```

In [386]:
```
n  = [-10,3,5,6,10,4,-5,-44]

sorted_by_abs_values = sorted(n,key = cmp_to_key(cmpr_abs))
```

In [388]:
```
sorted_by_abs_values
```

Out[388]:
```
[3, 4, 5, -5, 6, -10, 10, -44]
```

In [ ]:
```
1
```

In [389]:
```
# return string array after sorting by length of each strings

strings = ['aaaa','aa','a','aaaaaaaaa','aaaa','bbaabb','aaasaaaa','bab']



absorted = sorted( strings , key = len )

print(absorted)
```

```
['a', 'aa', 'bab', 'aaaa', 'aaaa', 'bbaabb', 'aaasaaaa', 'aaaaaaaaa']
```

In [ ]:
```
1
```

In [390]:
```

strings = ['apple','boy','do','sunny','bay','god','elephant','scaler','interviewbit ']
print("original strings:\n",strings)
print()
lex = sorted(strings)
print("sorted lexicographically: \n",lex)
print()
absorted = sorted( strings , key = len )
print("sorted by length of string :\n ",absorted)
```

```
original strings:
 ['apple', 'boy', 'do', 'sunny', 'bay', 'god', 'elephant', 'scaler', 'interviewbit ']

sorted lexicographically:
 ['apple', 'bay', 'boy', 'do', 'elephant', 'god', 'interviewbit ', 'scaler', 'sunny']

sorted by length of string :
  ['do', 'boy', 'bay', 'god', 'apple', 'sunny', 'scaler', 'elephant', 'interviewbit ']
```

In [ ]:
```
1
```

In [391]:
```
strings = ['apple','boy','do','sunny','bay',
           'god','elephant','scaler','interviewbit ']


def ascii_val_of_string(s):
    ans = 0
    for x in s:
        ans += ord(x)
    return ans
```

In [392]:
```
ascii_val_of_string("apple")
```

Out[392]:
```
530
```

In [394]:
```
sorted_by_ascii_val_strings = sorted(strings,key = ascii_val_of_string)
sorted_by_ascii_val_strings
```

Out[394]:
```
['do',
 'god',
 'bay',
 'boy',
 'apple',
 'sunny',
 'scaler',
 'elephant',
 'interviewbit ']
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [433]:
```
def factor_count(n):
    c = 0
    i = 1
    while i*i <=n:
        if n % i == 0:
            if i*i == n: c += 1
            else: c += 2
        i += 1
    return (c)
factor_count(152)
```

Out[433]: 8

In [435]:
```
items = [1, 2, 5, 10, 15, 16, 6]
from functools import cmp_to_key

sorted_item = sorted(items,key =factor_count)
sorted_item
```

Out[435]: [1, 2, 5, 10, 15, 6, 16]

In [436]:
```
def compare_by_factors(a,b):
    a_count = factor_count(a)
    b_count = factor_count(b)

    if a_count < b_count:
        return -1
    elif a_count == b_count:
        return 0
    else:
        return 1

sortedlist = sorted(items,key = cmp_to_key(compare_by_factors))
print(sortedlist)

```

```
[1, 2, 5, 10, 15, 6, 16]
```

In [437]:
```
sorted(items, key= factor_count, reverse = True)
```

Out[437]: [16, 10, 15, 6, 2, 5, 1]

In [440]:
```
def compare_by_factors(a,b):
    a_count = factor_count(a)
    b_count = factor_count(b)

    if a_count < b_count: return 1
    elif a_count > b_count: return -1
    else:
        if a < b: return -1
        elif a == b: return 0
        else: return 1

X = list(range(20))
print(sorted(X, key = cmp_to_key(compare_by_factors)))
```

```
[12, 18, 16, 6, 8, 10, 14, 15, 4, 9, 2, 3, 5, 7, 11, 13, 17, 19, 1, 0]
```

In [439]:
```
X = list(range(20))


def custom_key(n):
    return (-factor_count(n),n)
print(sorted(X,key = custom_key))
```

```
[12, 18, 16, 6, 8, 10, 14, 15, 4, 9, 2, 3, 5, 7, 11, 13, 17, 19, 1, 0]
```

In [ ]:
```

```

Q3. Largest Number </> Solved

Asked in: a DS

Stuck somewhere?

Ask for help from a TA and get it resolved.

Get help from TA.

Problem Description

Given an array **A** of non-negative integers, arrange them such that they form the largest number.

Note: The result may be very large, so you need to return a string instead of an integer.

In [ ]:
```

```

In [446]:
```
A = [3,30,34,5,9]

from functools import cmp_to_key

def largest_number(A):
    n = len(A)

    def comparator(a,b):
        x = int(str(a)+str(b))
        y = int(str(b)+str(a))

        if x > y :
            return -1
        elif x < y :
            return 1
        else:
            return 0

    x = sorted(A,key = cmp_to_key(comparator))
    ans = ""
    for l in x:
        ans += str(l)
    return int(ans)

largest_number(A)
```

Out[446]: 9534330

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

. Toggle Case </> Solved

Problem Description

You are given a character string **A** having length **N**, consisting of only lowercase and uppercase latin letters.

You have to toggle case of each character of string **A**. For e.g 'A' is changed to 'a', 'e' is changed to 'E', etc.

In [10]:
```python
# toggle the case

def toggle_case(s):
    listring = list(s)

    n = len(listring)

    for i in range(n):

        if 65 <=  ord(listring[i]) <= 90:
            listring[i] = chr(ord(listring[i]) + 32)

        elif 97 <= ord(listring[i]) <= 122:
            listring[i] = chr(ord(listring[i]) - 32)


    return "".join(listring)


s = "Python Is Awesome !! codE?"
toggle_case(s)
```

Out[10]:
```
'pYTHON iS aWESOME !! CODe?'
```

In [ ]:
```
1
```

In [12]:
```python
# toggle the case

def toggle_case(s):
    listring = list(s)

    n = len(listring)

    for i in range(n):

        if (65 <= ord(listring[i]) <= 90)  or (97 <= ord(listring[i]) <= 122): # as 32 is power of a
            listring[i] = chr(ord(listring[i]) ^ 32)                # we can toggle the bit


    return "".join(listring)


s = "Python Is Awesome !! codE?"
toggle_case(s)
```

Out[12]: 'pYTHON iS aWESOME !! CODe?'

In [ ]:
```python

```

In [ ]:
```python

```

In [ ]:
```python

```

In [ ]:
```python

```

In [ ]:
```python

```

In [ ]:
```python

```

In [ ]:
```python

```

In [ ]:
```python

```

In [ ]:
```python

```

Problem Description

Given a string A of size N, find and return the **longest palindromic substring** in A.

Substring of string A is `A[i...j]` where 0 <= i <= j < len(A)

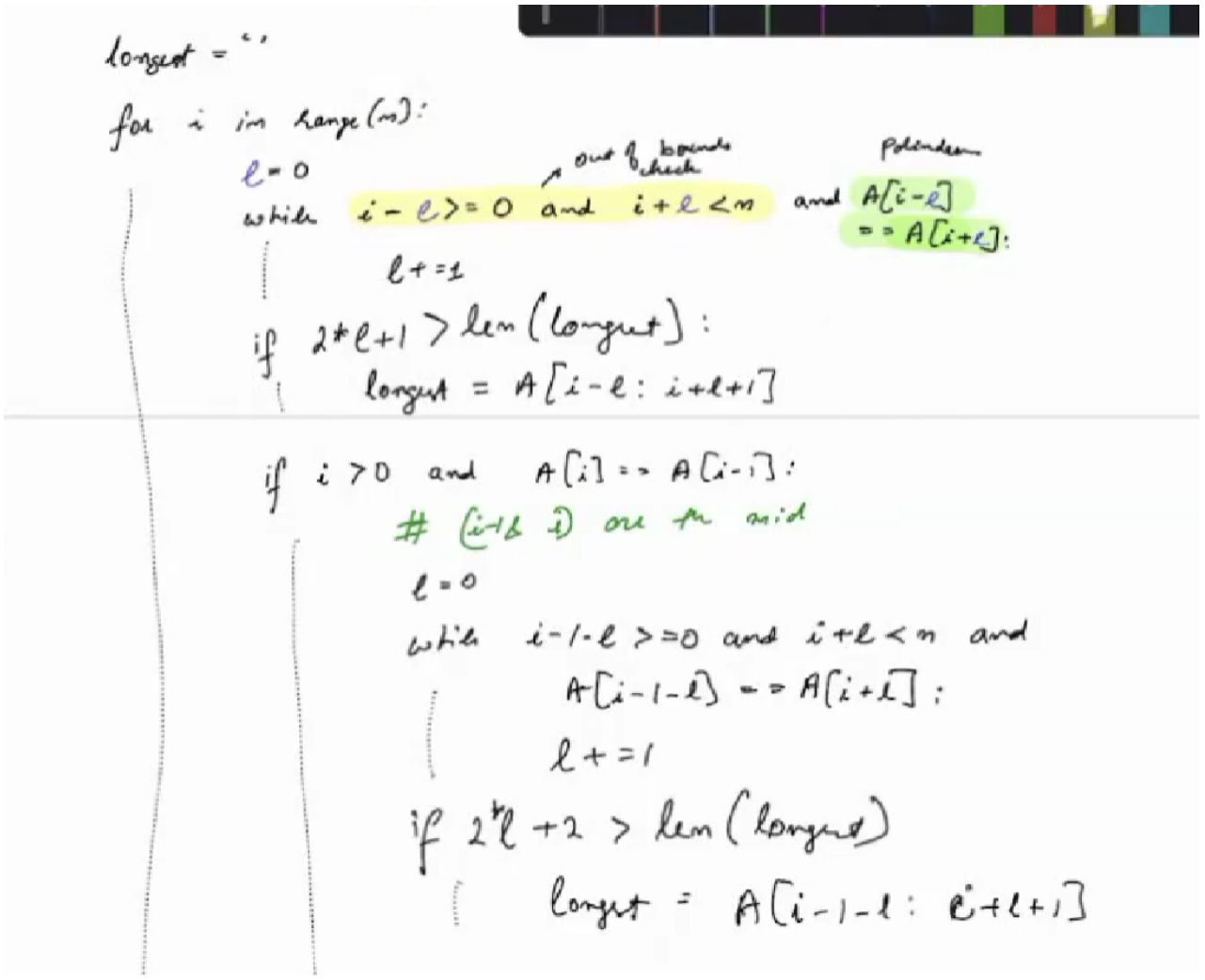
Palindrome string:
A string which reads the same backwards. More formally, A is palindrome if reverse(A) = A.

Incase of conflict, return the substring which occurs first (with the least starting index).

In [65]:
```python
def longestPalindrome(A):
    maxlen = -float('inf')
    n = len(A)
    ans = None
    for s in range(n):
        for e in range(s,n):
            x = A[s:e+1]
            if x == x[::-1]:
                length = e-s+1
                if length > maxlen:
                    maxlen = length
                    ans = x
    return (ans)
```

In [66]:
```python
longestPalindrome("abacaabaxa")
```

Out[66]: 'aba'



In [155]:
```python
def longest_palindormic_substring(A):
    longest = ""
    n = len(A)
    ans = []
    for i in range(n):

        l = 0
        while i - l >= 0 and i+l < n and A[i-l] == A[i+l]:
            l += 1
        if 2*l +1 > len(longest):
            longest = A[i-l+1:i+l]


        elif i > 0 and A[i] == A[i-1]:
            l = 0
            while i-1-l >=0  and i+l < n and A[i-l-1] == A[i+l]:
                l += 1
            if 2*l + 2 > len(longest):
                longest = A[i-l : i+l+1]
    return longest
```

In [156]:
```python
longest_palindormic_substring("abcdcbxaaxbcd")
```

Out[156]: 'dcbxaaxbcd'

In [ ]:
```python

```

In [157]:
```python
longest_palindormic_substring("abbcccbbbcaaccbababcbcabca")
```

Out[157]: 'cbababc'

In [158]:
```python
longest_palindormic_substring("aaaabaaa")
```

Out[158]: 'aaabaaa'

In [ ]:
```python

```

In [ ]:
```python

```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

Q1. Amazing Subarrays </> Solved

You are given a string **S**, and you have to find all the **amazing substrings** of **S**.

An amazing Substring is one that starts with a **vowel** (a, e, i, o, u, A, E, I, O, U).

In [67]:
```
1  def amazing_subarray(A):
2      n = len(A)
3
4      count = 0
5      x = n
6      for i in range(n):
7          if A[i] in ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']:
8              count += x
9          x = x-1
10     return count
```

In [68]:
```
1  amazing_subarray("ABEC")
```

Out[68]: 6

In [ ]:
```
1
```

Problem Description

Find the number of occurrences of **bob** in string **A** consisting of lowercase English alphabets.

In [69]:
```
1  def count_occurrences(A):
2      n = len(A)
3      count =  0
4      for i in range(n-2):
5          if A[i] == "b" and A[i+1] == "o" and A[i+2] == "b":
6              count += 1
7      return count
```

In [70]:
```
1  count_occurrences("bobejhshajboajbobobo")
```

Out[70]: 3

In [ ]:
```
1
```

In [73]:
```python
S = "aabbaa"
def isPalindrom(S):
    i = 0
    j = len(S)-1
    palindrom = True
    while i < j:
        if S[i] != S[j]:
            palindrom = False
            break
        i += 1
        j -= 1

    return palindrom

isPalindrom(S)
```

Out[73]: True

In [75]:
```python
isPalindrom("aasd")
```

Out[75]: False

In [ ]:
```python

```

In [ ]:
```python

```

Q5. Longest Common Prefix </> Solved

Problem Description

Given the array of strings **A**, you need to find the longest string **S,** which is the prefix of **ALL** the strings in the array.

The longest common prefix for a pair of strings **S1** and **S2** is the longest string **S** which is the prefix of both **S1** and **S2**.

Example: the longest common prefix of "abcdefgh" and "abcefgh" is "abc".

```
A = ["abcdefgh", "aefghijk", "abcefgh"]
```

In [159]:
```python
# ans : a


# A = ["abab",
    #  "ab",
    #  "abcd"]

# ans = ab
```

In [179]:
```python
def longestCommonPrefix(A):

    def find_index(A):
        n = len(A)
        m = len(A[0])
        ans = None
        for j in range(len(A[0])):

            ch = A[0][j]

            for i in range(n):
                if j >= len(A[i]):
                    return  j
                if A[i][j] != ch:
                    return j


    return A[0][:find_index(A)]


```

In [180]:
```
longestCommonPrefix(["abab",
                     "ab",
                     "abcd"])
```

Out[180]:
```
'ab'
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

hashmaps

In [184]:
```
A = [1,2,3,1,2,3,4,5,6,4,5,6,7,8,9,7,8,9,9,8,7,
     6,9,5,4,3,2,1,6,5,4,8,7,9,4,8,4,7,4,6,1,3,5,4,6,5,7,4,9,8]

freq = {}

```

In [185]:
```

for i in A:
    if i in freq:
        freq[i] += 1
    else:
        freq[i] = 1
print(freq)


```

```
{1: 4, 2: 3, 3: 4, 4: 9, 5: 6, 6: 6, 7: 6, 8: 6, 9: 6}
```

In [186]:
```
for k in freq:
    print(k,freq[k])


```

```
1 4
2 3
3 4
4 9
5 6
6 6
7 6
8 6
9 6
```

In [187]:
```
for k,v in freq.items():
    print(k,v)


```

```
1 4
2 3
3 4
4 9
5 6
6 6
7 6
8 6
9 6
```

In [188]:
```
sq = {}

for i in range(1,16):
    sq[i] = i*i
sq
```

Out[188]:
```
{1: 1,
 2: 4,
 3: 9,
 4: 16,
 5: 25,
 6: 36,
 7: 49,
 8: 64,
 9: 81,
 10: 100,
 11: 121,
 12: 144,
 13: 169,
 14: 196,
 15: 225}
```

In [189]:
```
sq[10]
```

Out[189]:
```
100
```

In [190]:
```
asd = {i:i*i for i in range(1,16)}
```

In [191]:
```
asd
```

Out[191]:
```
{1: 1,
 2: 4,
 3: 9,
 4: 16,
 5: 25,
 6: 36,
 7: 49,
 8: 64,
 9: 81,
 10: 100,
 11: 121,
 12: 144,
 13: 169,
 14: 196,
 15: 225}
```

In [192]:
```
list(asd.values())
```

Out[192]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225]
```

In [ ]:
```

```

In [ ]:
```

```

ans the queries

In [193]:
```
A = [1,3,4,-9,1,0,3,5,6,3,3,1,9]
Q = [3,4,0,-9,3]
```

In [194]:
```
freq = {}

for x in A:
    freq[x] = freq.get(x,0)+1
print(freq)
```

```
{1: 3, 3: 4, 4: 1, -9: 1, 0: 1, 5: 1, 6: 1, 9: 1}
```

In [196]:
```
for q in Q:
    print(q,freq[q])
```

```
3 4
4 1
0 1
-9 1
3 4
```

better ways to generate dictionaries

In [197]:
```
from collections import defaultdict

freq = defaultdict(int)
for x in A:
    freq[x] += 1
freq
```

Out[197]:
```
defaultdict(int, {1: 3, 3: 4, 4: 1, -9: 1, 0: 1, 5: 1, 6: 1, 9: 1})
```

In [ ]:
```

```

In [198]:
```
from collections import Counter

freq = Counter(A)
freq
```

Out[198]:
```
Counter({1: 3, 3: 4, 4: 1, -9: 1, 0: 1, 5: 1, 6: 1, 9: 1})
```

In [ ]:
```

```

In [ ]:
```

```

In [202]:
```
names = ["Sunny","Abhi","Uttam","Sunny",
         "Srikar","Mehul","Sunny","Urvi","Abhi","Sunny","Prithvi","Bhavya"]

freq = defaultdict(set)

for x in names:
    s = x[0]
    freq[s].add(x)
freq
```

Out[202]:
```
defaultdict(set,
            {'S': {'Srikar', 'Sunny'},
             'A': {'Abhi'},
             'U': {'Urvi', 'Uttam'},
             'M': {'Mehul'},
             'P': {'Prithvi'},
             'B': {'Bhavya'}})
```

In [215]:
```
names = ["Sunny","Abhi","Uttam","Sunny",
         "Srikar","Mehul","Sunny","Urvi","Abhi","Sunny","Prithvi","Bhavya"]

freq = defaultdict(list)

for x in names:
    s = x[0]
    freq[s].append(x)
freq
```

Out[215]:
```
defaultdict(list,
            {'S': ['Sunny', 'Sunny', 'Srikar', 'Sunny', 'Sunny'],
             'A': ['Abhi', 'Abhi'],
             'U': ['Uttam', 'Urvi'],
             'M': ['Mehul'],
             'P': ['Prithvi'],
             'B': ['Bhavya']})
```

In [ ]:
```

```

In [ ]:
```

```

count the distinc elements in array

In [232]:
```
A = [1,3,4,-9,1,55,-77,0,3,5,6,3,3,1,9,91]
n = len(A)
count = 0
for i in range(n):
    is_first = True
    for j in range(i):
        if A[j] == A[i]:
            is_first = False
            break
    if is_first:
        count += 1
print(count)

```

In [233]:
```
print(len(set(A)))
```

```
11
```

In [236]:
```
freq = Counter(A)
```

In [235]:
```
len(freq)
```

Out[235]:
```
11
```

In [ ]:
```

```

Non repeating elements

return first non repeating

In [248]:
```
A= [1,3,4,-9,1,0,3,5,6,3,3,1,9,4,-9,0,5,6,9]
```

In [250]:
```
def return_first_non_repeating_element(A):
    ans = []
    for x in A:
        c  = 0
        for y in A:
            if x == y:
                c+= 1
        if c == 1:
            return x
    return -1

```

In [254]:
```
A= [1,3,2,3,2,1]
return_first_non_repeating_element(A)
```

Out[254]:
```
-1
```

In [253]:
```
A = [1,3,4,-9,1,55,-77,0,3,5,6,3,3,1,9,91]

return_first_non_repeating_element(A)
```

Out[253]:
```
4
```

In [ ]:
```

```

In [ ]:
```

```

In [256]:
```
def return_first_non_repeating_element(A):
    freq = Counter(A)
    for x in A:
        if freq[x] == 1:
            return x
    return -1
```

In [257]:
```
A = [1,3,4,-9,1,55,-77,0,3,5,6,3,3,1,9,91]

return_first_non_repeating_element(A)
```

Out[257]:
```
4
```

In [258]:
```
A= [1,3,2,3,2,1]
return_first_non_repeating_element(A)
```

Out[258]:
```
-1
```

first repeating array

Q2. First Repeating element </> Solved

Asked in: DS GS

Stuck somewhere?
Ask for help from a TA and get it resolved.

Get help from TA.

Problem Description

Given an integer array **A** of size **N**, find the first repeating element in it.

We need to find the element that occurs **more than once** and whose index of the first occurrence is the **smallest**.

If there is no repeating element, return -1.

In [263]:
```
def first_repeating_elemenet(A):
    freq = Counter(A)
    for x in A:
        if freq[x] > 1:
            return x
    return -1
```

In [264]:
```
A = [10, 7, 3, 4, 3, 5, 6,7]
first_repeating_elemenet(A)
```

Out[264]: 7

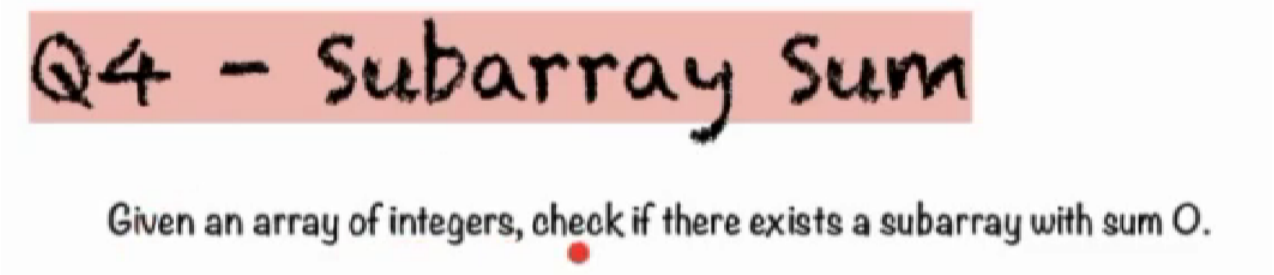
In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:



Q3. Sub-array with 0 sum </> Solved

Asked in: MA PA

Stuck somewhere?
Ask for help from a TA and get it resolved.

Get help from TA.

Problem Description

Given an array of integers **A**, find and return whether the given array contains a non-empty subarray with a sum equal to 0.

If the given array contains a sub-array with sum zero return 1, else return 0.

In [269]:
```
def subarray_sum_0(A):
    n = len(A)
    ans = []
    for s in range(n):
        sm = 0
        for e in range(s,n):
            sm += A[e]
            if sm == 0:
                ans.append(A[s:e+1])
    return ans
```

In [270]:
```
A = [2,2,1,-3,4,3,1,-2,-3]
subarray_sum_0(A)
```

Out[270]: [[2, 1, -3], [-3, 4, 3, 1, -2, -3]]

In [ ]:

In [311]:
```
def subarray_sum_0_(A):
    for i in A:
        if i == 0:
            return 1

    if sum(A) == 0:
        return 1

    seen = set()

    prefix_sum = 0
    seen.add(prefix_sum)
    for x in A:
        prefix_sum +=  x
        if prefix_sum in seen:
            return 1
        seen.add(prefix_sum)
    return 0
```

In [312]:
```
subarray_sum_0_([1,2,3,4,5])
```

Out[312]: 0

In [315]:
```
subarray_sum_0_([0,5])
```

Out[315]: 1

In [314]:
```
subarray_sum_0_([-5,5])
```

Out[314]: 1

In [323]:
```
def subarray_with_sum_0(A):
    n = len(A)

    current_sum = 0
    prefixmap = set()

    for i in range(n):
        current_sum += A[i]
        if A[i] == 0 or current_sum == 0 or current_sum in prefixmap:
            return 1
        else:
            prefixmap.add(current_sum)
    return 0


A = [2,2,1,3,4,-3,-1,2,3]
subarray_with_sum_0(A)
```

Out[323]: 1

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Q1. Common Elements </> Solved

Problem Description

Given two integer arrays, **A** and **B** of size **N** and **M,** respectively. Your task is to find all the **common elements** in both the array.

NOTE:

- Each element in the result should appear as many times as it appears in both arrays.
- The result can be in any order.

In [350]:
```
A = [1, 2, 2, 1]
B = [2, 3, 1, 2]

freqA = Counter(A)
freqB = Counter(B)

intersection = list(set(A).intersection(set(B)))
intersection
```

In [345]:
```
freqA,freqB
```

Out[345]: (Counter({1: 2, 2: 2}), Counter({2: 2, 3: 1, 1: 1}))

In [ ]:
```

```

In [349]:
```
counts = -1
finalans = []
for e in intersection:
    counts = (min(freqA[e],freqB[e]))
    for _ in range(counts):
        finalans.append(e)

print(intersection)
print(finalans)
```

```
[1, 2]
[1, 2, 2]
```

neat code

In [525]:
```
from collections import Counter
def common_elements(A,B):
    freqA = Counter(A)
    freqB = Counter(B)
    intersection = list(set(A).intersection(set(B)))
    print(intersection)
    counts = -1
    finalans = []
    for e in intersection:
        counts = (min(freqA[e],freqB[e]))
        for _ in range(counts):
            finalans.append(e)

    return (finalans)
```

In [526]:
```
A = [1, 2, 2, 1,2,3,5]
B = [2, 3, 1, 2,2]
common_elements(A,B)
```

```
[1, 2, 3]
```

Out[526]: [1, 2, 2, 2, 3]

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
1
```

In [ ]:
```
1
```

Q4. Shaggy and distances </> Solved

Stuck somewhere?

Ask for help from a TA and get it resolved.

Get help from TA.

Problem Description

Shaggy has an array **A** consisting of N elements. We call a pair of distinct indices in that array a special if elements at those indices in the array are equal.

Shaggy wants you to find a special pair such that the distance between that pair is minimum. Distance between two indices is defined as |i-j|. If there is no special pair in the array, then return -1.

In [467]:
```
1  A = [7,1,3,4,1,7,3]
2
3  indexmap = defaultdict(list)
4  for i in range(len(A)):
5      lst = indexmap[A[i]]
6      lst.append(i)
7  print(indexmap)
8
9  ans = []
10 for k,v in indexmap.items():
11     if len(v) == 2:
12         ans.append(v)
13 print(ans)
14 final = []
15 for i in range(len(ans)):
16     for j in range(len(ans[i])):
17         x = ans[i][0]
18         y = ans[i][-1]
19         final.append(abs(x-y))
20 print(min(final))
21
22
```

```
defaultdict(<class 'list'>, {7: [0, 5], 1: [1, 4], 3: [2, 6], 4: [3]})
[[0, 5], [1, 4], [2, 6]]
3
```

In [ ]:
```
1
```

In [425]:
```
1
2
```

In [ ]:
```
1
```

In [468]:
```
A = [7,1,3,4,1,7,3]

indexmap  = defaultdict(list)

for i in range(len(A)):
    lst = indexmap[A[i]]
    lst.append(i)
print(indexmap)
minlen = float('inf')

for x in indexmap:

    for j in range(len(indexmap[x])):
        if len(indexmap[x]) ==2 :
            minlen = min(minlen, abs(indexmap[x][0]-indexmap[x][1]))
    print(x,minlen)

print()
print(minlen)
```

```
defaultdict(<class 'list'>, {7: [0, 5], 1: [1, 4], 3: [2, 6], 4: [3]})
7 5
1 3
3 3
4 3

3
```

In [ ]:
```

```

In [ ]:
```

```

Q5. Largest Continuous Sequence Zero Sum </> Solved

Asked in: MA

Stuck somewhere?

Ask for help from a TA and get it resolved.

Get help from TA.

Problem Description

Given an array **A** of **N** integers.

Find the largest continuous sequence in a array which sums to zero.

In [492]:
```
def longest_con_sq_zero_sm(A):
    n = len(A)
    istart = None
    iend = None
    maxlen = -float('inf')
    for s in range(n):
        sum = 0
        for e in range(s,n):
            sum += A[e]
            length = e-s+1
            if sum == 0:
                if length > maxlen:
                    maxlen = length
                    istart = s
                    iend = e

    if istart == None:
        return []

    return (A[istart:iend+1])
```

In [494]:
```
longest_con_sq_zero_sm([1,2,-2,4,-4])
```

Out[494]:
```
[2, -2, 4, -4]
```

In [ ]:
```

```

In [44]:
```
def lszero(A):
    N = len(A)
    pre = [None]*N
#     print(pre)
    curr = 0
    for i in range(N):
        curr += A[i]
        pre[i] = curr
#     print(pre)

    from collections import defaultdict
    seen_table = defaultdict(list)

    seen_table[0]=[-1]
    longest = -1
    indices = None
    for i in range(N):
        s = pre[i]
#         print(s)
        seen_table[s].append(i)
#         print(seen_table)
        gap = seen_table[s][-1] - seen_table[s][0]
        if gap > longest:
            longest = gap
            indices = (seen_table[s][0]+1,seen_table[s][-1]+1)

    if indices:
        return (A[indices[0]:indices[1]])
    else:
        return ([])
```

In [45]:
```
lszero([1,2,-2,4,-4])
```

Out[45]:
```
[2, -2, 4, -4]
```

In [ ]:
```

```

Q1. K Occurrences </> Solved

Stuck somewhere?

Ask for help from a TA and get it resolved.

Get help from TA.

Groot has N trees lined up in front of him where the height of the i'th tree is denoted by H[i].

He wants to select some trees to replace his broken branches.

But he wants uniformity in his selection of trees.

So he picks only those trees whose heights have frequency K.

He then sums up the heights that occur K times. (He adds the height only once to the sum and not K times).

But the sum he ended up getting was huge so he prints it modulo 10^9+7.

In case no such cluster exists, Groot becomes sad and prints -1.

In [498]:
```
A = [1,2,2,3,3]
k = 2
freq = Counter(A)
```

In [499]:
```
freq
```

Out[499]:
```
Counter({1: 1, 2: 2, 3: 2})
```

In [500]:
```
ans = 0
for i in freq:
    if freq[i] == k:
        ans += i
ans
```

Out[500]:
```
5
```

In [504]:
```python
def getSum( k, A):

    from collections import Counter

    freq = Counter(A)

    lists = freq.values()

    if k not in lists:
        return -1

    ans = 0
    for x in freq:
        if freq[x] == k:
            ans += x
    return ans

getSum(2,[1,2,2,3,3,3,2,5,5,6,6])
```

Out[504]: 11

In [ ]:
```python
```

In [ ]:
```python
```

Q2. Check Palindrome - II </> Solved

Asked in: MS

Stuck somewhere?

Ask for help from a TA and get it resolved.

Get help from TA.

Problem Description

Given a string **A** consisting of lowercase characters.

Check if characters of the given string can be rearranged to form a **palindrome**.

Return 1 if it is possible to rearrange the characters of the string **A** such that it becomes a palindrome else return 0.

In [513]:
```python
def check_palindorm(A):
    x = list(A)
    x

    freq = Counter(x)
    print(freq)

    oddfreq = 0

    for i in freq:
        if freq[i] % 2 != 0 :
            oddfreq += 1
    print(oddfreq)
    if oddfreq <= 1:
        return 1
    return 0
```

In [516]:
```python
A= "abbaee"

check_palindorm(A)
```

```
Counter({'a': 2, 'b': 2, 'e': 2, 'A': 1})
1
```

Out[516]: 1

In [529]:
```python
A= "asdfffwasd"
check_palindorm(A)
```

```
Counter({'f': 3, 'a': 2, 's': 2, 'd': 2, 'w': 1})
2
```

Out[529]: 0

In [ ]:
```
1
```

Q3. Colorful Number </> Solved

Stuck somewhere?

Ask for help from a TA and get it resolved.

Get help from TA.

Problem Description

Given a number **A,** find if it is **COLORFUL** number or not.

If number **A** is a **COLORFUL** number return **1** else, return **0**.

What is a **COLORFUL** Number:

```
A number can be broken into different contiguous sub-subsequence parts.
Suppose, a number 3245 can be broken into parts like 3 2 4 5 32 24 45 324
245.
And this number is a COLORFUL number, since product of every digit of a
contiguous subsequence is different.
```

In [538]:
```
1  Ax = 3245
2
3  def colorful(Ax):
4      A = []
5      while Ax > 0:
6          d = Ax % 10
7          A.append(d)
8          Ax //= 10
9      A.reverse()
10     n = len(A)
11     seen = set()
12     for s in range(n):
13         product = 1
14         for e in range(s,n):
15             product *= A[e]
16             if product in seen:
17                 return 0
18             seen.add(product)
19     return 1
20
21
22 colorful(Ax)
```

Out[538]: 1

In [ ]:
```
1
```

Q1. Subarray with given sum </> Solved

Asked in: FA FA G

Stuck somewhere?

Ask for help from a TA and get it resolved.

Get help from TA.

Problem Description

Given an array of positive integers A and an integer B, find and return first continuous subarray which adds to B.

If the answer does not exist return an array with a single element "-1".

First sub-array means the sub-array for which starting index in minimum.

In [38]:
```
A = [1, 2, 3, 4, 5]         # two pointer technique , starting from index 0
B = 5


def subarray_with_given_sum(A,B):
    n  = len(A)
    p1 = 0
    p2 = 0

    ans_array = []
    total = A[0]
    xx = []
    flag = False

    while p1 < n and p2 < n:

        if total == B:
            flag = True
            break

        elif total < B:
            if p2 +1 == n:
                break
            p2 += 1
            total += A[p2]

        else:
            if p1 + 1 == n:
                break
            p1 += 1
            total = total - A[p1-1]

    if flag == False:
        return -1

#     xx.append(p1+1)
#     xx.append(p2+1)
#     return xx
    for x in range(p1,p2+1):
        ans_array.append(A[x])
    return ans_array




subarray_with_given_sum(A,B)
```

Out[38]:
```
[2, 3]
```

In [ ]:
```

```

In [39]:
```
arr = [1,2,3,4,5,6,7,8,9,10]
n = len(arr)
s = 15
subarray_with_given_sum(arr,s)
```

Out[39]:
```
[1, 2, 3, 4, 5]
```

In [40]:
```
arr = [1,2,3,7,5]
n = len(arr)
s = 12
subarray_with_given_sum(arr,s)
```

Out[40]:
```
[2, 3, 7]
```

In [ ]:
```

```

In [ ]:
```

```

In [547]:
```python
def subarray_with_given_sum_brute_force(A, k):

    n = len(A)
    x = None
    y = None
    for s in range(n):
        sum = 0
        for e in range(s,n):
            sum += A[e]
            if sum == k:
                x = s
                y = e+1

                return A[x:y]
    return -1
A = [1, 2, 3, 4, 5]
B = 5

subarray_with_given_sum_brute_force(A, B)
```

Out[547]: [2, 3]

In [548]:
```python
def subarysm_k_uttam(A,k):

    if sum(A) == k:
        return A

    prefix = [0]
    for x in A:
        prefix.append(x+prefix[-1])
    print(prefix)
                              # a  = b-k in , prefix sum array
    for b in prefix:
        a  = b-k
        if a in prefix:
            index1 = prefix.index(a)
            index2 = prefix.index(b)
    #         print(b-a, prefix.index(a)+1, prefix.index(b))
            return (A[index1:index2])
    return -1
subarysm_k_uttam(A, B)
```

```
[0, 1, 3, 6, 10, 15]
```

Out[548]: [2, 3]

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Q4. Distinct Numbers in Window </> Solved

Asked in: a AC

Stuck somewhere?

Ask for help from a TA and get it resolved.

Get help from TA.

Problem Description

You are given an array of **N** integers, $A_1, A_2, \ldots, A_N$ and an integer **B**. Return the of count of distinct numbers in all windows of size **B**.

Formally, return an array of size **N-B+1** where **i'th** element in this array contains number of distinct elements in sequence $A_i, A_{i+1}, \ldots, A_{i+B-1}$.

NOTE: if **B** > **N**, return an empty array.

In [465]:
```
A = [1,2,1,3,4,5,1,2,3,4,1]
k = 5


def distinct_number_in_window(A,k):
    n = len(A)

    freq = Counter(A[:k])
    print(freq)

    ans = []
    ans.append(len(freq))

    for i in range(k,n):

        freq[A[i]] += 1
        freq[A[i-k]] -= 1

        if freq[A[i-k]] == 0:
            del freq[A[i-k]]

        ans.append(len(freq))



    return ans




distinct_number_in_window(A,k)
```

```
Counter({1: 2, 2: 1, 3: 1, 4: 1})
```

Out[465]:
```
[4, 5, 4, 5, 5, 5, 4]
```

In [466]:
```
A = [1, 2, 1, 3, 4, 3]
B = 3
distinct_number_in_window(A,B)
```

```
Counter({1: 2, 2: 1})
```

Out[466]:
```
[2, 3, 3, 2]
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

given an array , and integer k

pick any k elements such that difference of the sum of k elements and

the sum of remaining elements is minimum

In [138]:

```
def find_min_diff(A,k):
    '''
    given an array  , and integer k
    pick any  k elements such that difference of the sum of k elements and
    the sum of remaining elements is minimum
    '''
    def binarytodecimal(s):   # binary sting
        s = str(s)
        i = 0
        result = 0
        for d in s[::-1]:
            result = result+ int(d)*(2**i)
            i = i+1
        return result
    def count_set_bits_right_shift(x):
        cnt = 0
        while x > 0:
            cnt += x&1
            x = x>>1
        return cnt

    n = len(A)
    totalsbset_of_k = []
    for i in range(2**n):
        subset = []

        s = bin(i)[2:]
        s = s.rjust(n,"0")
        x = 0
        if count_set_bits_right_shift(binarytodecimal(s)) == k:

            for j in s:
                if j == "1":
                    subset.append(A[x])

                x += 1
            totalsbset_of_k.append(subset)

    mindiff = float("inf")
    for x in totalsbset_of_k:
        y = []

        for i in A:
            if i not in x:
                y.append(i)
        diff = abs(sum(x) - sum(y))

        if diff < mindiff:
            mindiff = diff
            print(x,y)

    return (mindiff)


A = [3,-2,4,-1,6,7,2,0]
k = 3
find_min_diff(A,k)
```

```
[7, 2, 0] [3, -2, 4, -1, 6]
```

Out[138]: 1

In [ ]:

```

```

given an array , and integer k

pick any k elements such that difference of the sum of k elements and

the sum of remaining elements is maximum

In [133]:
```
def max_difference_k_size_subsequences(Ax,k):
    # maximise the differnce

    # one side 2 elemenrts , 6 on other

    A = sorted(Ax)
    n = len(A)
    #print(A)

    if k < n/2:

        K_side = A[:k]
        print(K_side)

        rest_side = A[k:]
        print(rest_side)

        ans = abs(sum(K_side) - sum(rest_side))
        return (ans)

    else:
        left = A[k:]
        print(left)

        right = A[:k]
        print(right)

        ans = abs(sum(left) - sum(right))
        return (ans)
```

In [134]:
```
Ax = [-1,3,6,0,8,10,12,100,-60]
k = 7
max_difference_k_size_subsequences(Ax,k)
```

```
[12, 100]
[-60, -1, 0, 3, 6, 8, 10]
```

Out[134]: 146

In [135]:
```
Ax = [1,2,3,4,5,6,7,8]
k = 2
max_difference_k_size_subsequences(Ax,k)
```

```
[1, 2]
[3, 4, 5, 6, 7, 8]
```

Out[135]: 30

In [136]:
```
Ax = [1,2,3,4,5,6,7,8]
k = 6
max_difference_k_size_subsequences(Ax,k)
```

```
[7, 8]
[1, 2, 3, 4, 5, 6]
```

Out[136]: 6

In [ ]:
```

```

In [ ]:
```

```

Q! All subset sum

In [144]:
```
A = [3,-6,7,0]
n = len(A)
def all_subsets_sum(A):
    n = len(A)
    total = 0
    for i in range(2**n):
        subset = []
        s = (bin(i)[2:])
        s = s.rjust(n,"0")
        x = 0
        summ = 0
        for j in s:
            if j == "1":
                summ += A[x]
            x += 1
        total += summ
    return total

all_subsets_sum(A)
```

Out[144]: 32

In [146]:
```
A = [3,-6,7,0]
n = len(A)
summ = 0

for i in A:
    summ += (i*(2**(n-1)))
print(summ)
```

```
32
```

In [ ]:
```
```

In [ ]:
```
```

find the sum of max of all subsequence

In [140]:
```
A = [3,0,6,9]


def sum_of_max_of_subsets(A):
    n = len(A)
    totalsum = 0
    for i in range(1,2**n):
        subset = []
        s = bin(i)[2:]
        s = s.rjust(n,"0")
        x = 0
        maxel = -float('inf')
        for j in s:
            if j == "1":
                maxel = max(maxel,A[x])


            x += 1
        totalsum += maxel
    return totalsum
sum_of_max_of_subsets(A)
```

Out[140]: 102

In [149]:
```

Ax = [9,3,6,0]

A = sorted(Ax)

total_sum = 0

for i in range(len(A)):
    total_sum += A[i]*2**i

print("sum of all maximum number from subsequence :  ",total_sum)

```

```
sum of all maximum number from subsequence :   102
```

find the sum of minimum of all subsequence

In [154]:
```
A = [3,0,6,9]


def sum_of_min_of_subsets(A):
    n = len(A)
    totalsum = 0
    for i in range(1,2**n):
        subset = []
        s = bin(i)[2:]
        s = s.rjust(n,"0")
        x = 0
        minele = float('inf')
        for j in s:
            if j == "1":
                minele = min(minele,A[x])


            x += 1
        totalsum += minele
    return totalsum
sum_of_min_of_subsets(A)

```

Out[154]: 33

In [148]:
```
# for min :
# 9 6 6 0 0 0 3 3 3 3 0 0 0 0

# 0  3  6  9
# 8  4  2  1

Ax = [0,3,6,9]

A = sorted(Ax)

total_sum = 0
x = len(A)-1
for i in range(len(A)):
#     print(i,2**x,A[i])
    total_sum += A[i]*(2**x)

    x = x-1
print("sum of all minimum number from subsequence :  ",total_sum)


```

```
sum of all minimum number from subsequence :   33
```

In [ ]:
```

```

In [ ]:
```

```

Q3. Longest Consecutive Sequence </> Solved

Asked in: a G DI

Problem Description

Given an unsorted integer array **A** of size N.

Find the length of the longest set of consecutive elements from array A.

In [ ]:
```

```

In [218]:
```
def longestConsecutive(Ax):

    A = sorted(Ax)
    n = len(A)
    ans = []
    maxlen = -float('inf')
    for i in range(n-1):
        if A[i+1] == A[i]:
            ans.append(A[i])

        elif A[i+1] == A[i]+1:
            ans.append(A[i])
            ans.append(A[i]+1)
        elif A[i+1] != A[i]:
            ans.clear()
        maxlen = max(len(set(ans)),maxlen)

    if maxlen == -float("inf"):
        return 1
    if maxlen == 0:
        return 1
    return maxlen

Ax = [50,41]
longestConsecutive(Ax)
```

Out[218]: 1

In [289]:
```
longestConsecutive([ 97, 86, 67, 19, 107, 9, 8, 49, 23, 46, -4, 22, 72, 4, 57, 111, 20, 52, 99, 2, 113, 81, 7, 5, 21, 0, 47
```

Out[289]: 6

In [ ]:
```

```

In [283]:
```
def longestConsecutive_optimised(A):
    freqmap = {}
    Amax = max(A)
    Amin = min(A)

    maplist = list(range(Amin,Amax+1))
    for i in maplist:
        if i in A:
            freqmap[i]= True
        else:
            freqmap[i]= False

    anss =[]
    maxlen = -float("inf")
    for i in freqmap:
        if freqmap[i]:
            anss.append(i)
        else:
            anss.clear()

        maxlen = max(maxlen,len(anss))
    return maxlen
```

In [284]:
```
longestConsecutive_optimised([100,200,5,7,9,8,4,625,3,6,10,12,23,14,13,14,15,16,17,18,19,20])
```

Out[284]: 9

In [285]:
```
A = [ 96, 10, 19, -2, 52, 8, 79, 93, 71, 62, 48, 13, 23 ]
longestConsecutive_optimised(A)
```

Out[285]: 1

In [287]:
```
longestConsecutive_optimised([ 97, 86, 67, 19, 107, 9, 8, 49, 23, 46, -4, 22, 72, 4, 57, 111, 20, 52, 99, 2, 113, 81, 7, 5,
```

Out[287]: 6

In [ ]:
```

```

Q2. Diffk II </> Solved

Stuck somewhere?

Ask for help from a TA and get it resolved.

Get help from TA.

Given an array A of integers and another non negative integer k, find if there exists 2 indices `i` and `j` such that `A[i] - A[j] = k, i != j`.

In [301]:
```
A = [1,5,4,1,2]
k = 0
ans = "NO"
n = len(A)
for i in range(n):
    target = A[i]-k

    if target in A:
        ans = "YES"
print(ans)

```

```
YES
```

In [ ]:
```
i - j = k
j = i-k
```

In [309]:
```
def diffpossible(A,k):
    from collections import Counter

    ans = 0
    freq = Counter(A)
    print(freq)
    for x in A:
        if k+x != x:
            if k+x in freq:
                ans = 1
                break
        else:
            if freq[x] > 1:
                ans = 1
                break
    return ans


A = [1,5,4,1,2]
k = 0

diffpossible(A,k)
```

```
Counter({1: 2, 5: 1, 4: 1, 2: 1})
```

Out[309]: 1

First Missing Integer

In [16]:
```
A = [3,4,-1,1]              # O(n**2),O(1)
def first_missing_int(A):
    n = len(A)

    for i in range(1,n+1):
        if i not in A:
            return i




A = [3,4,-1,1,2,6,5]
first_missing_int(A)
```

Out[16]: 7

In [215]:
```
A = [3,4,-1,1]                  # O(n),O(n)
def first_missing_int(A):
    n = len(A)
    from collections import Counter
    freq = Counter(A)
    print(freq)
    for i in range(1,n+1):
        if i not in freq:
            return i



A = [3,4,-1,1,2]
first_missing_int(A)
```

```
Counter({3: 1, 4: 1, -1: 1, 1: 1, 2: 1})
```

Out[215]: 5

In [216]:
```
A = [2,3,-7,6,8,1,-10,15]
first_missing_int(A)
```

```
Counter({2: 1, 3: 1, -7: 1, 6: 1, 8: 1, 1: 1, -10: 1, 15: 1})
```

Out[216]: 4

In [217]:
```
first_missing_int([1])

```

```
Counter({1: 1})
```

In [212]:
```

def firstMissingPositive(A):
    j = 0
    n = len(A)
    for i in range(n):
        if A[i] <= 0:
            A[i],A[j] = A[j],A[i]
            j+=1
    print(A)
    for i in range(j,n):
        print(i)
        if abs(A[i])-1+j < n and A[abs(A[i])-1+j] > 0:
            A[abs(A[i])-1+j] = -A[abs(A[i])-1+j]
    print(A)
    for i in range(j,n):
        if A[i] > 0:
            return i-j+1
    return n - j + 1
```

In [197]:
```
firstMissingPositive([2,3,-7,6,8,1,-10,15])
```

```
[-7, -10, 2, 6, 8, 1, 3, 15]
2
3
4
5
6
7
[-7, -10, -2, -6, -8, 1, 3, -15]
```

Out[197]: 4

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [161]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```