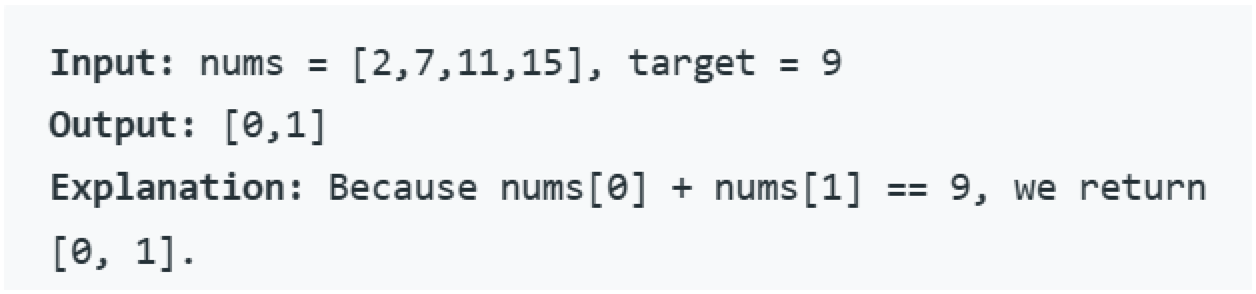
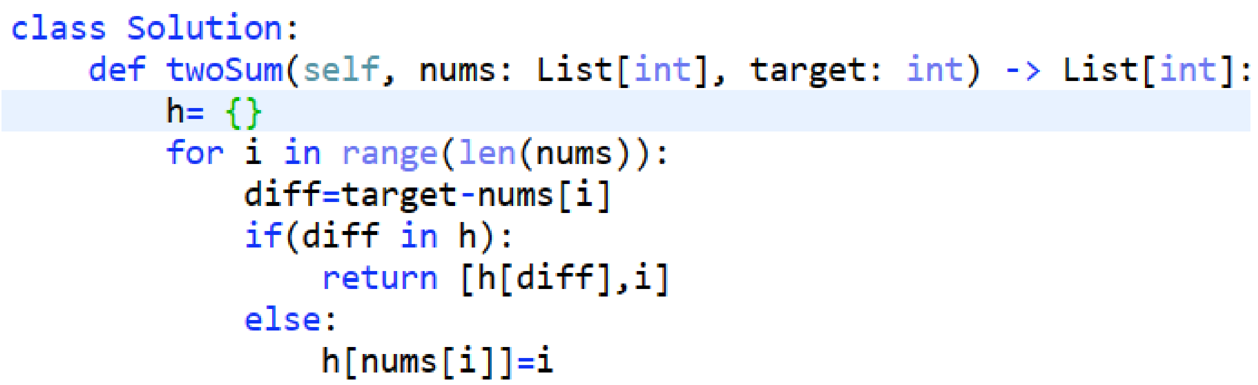
ARRAY

Q1: Two Sum:
Brute force: TC: O(n^2) SC: O(1)
Optimized: **TC:O(n) SC: O(n)**
Take a dictionary and store the val along with its position.
And then take each value and subtract from target and check that difference already in the dictionary or not , if yes then return the pos of that value as well as position for which we get difference.



```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return
[0, 1].
```



```python
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        h= {}
        for i in range(len(nums)):
            diff=target-nums[i]
            if(diff in h):
                return [h[diff],i]
            else:
                h[nums[i]]=i
```

Two pointer approach of striver in two sum which do in O(nlogn) as it first do sort the element. This one is not possible if in ques it is given that find indices of element.

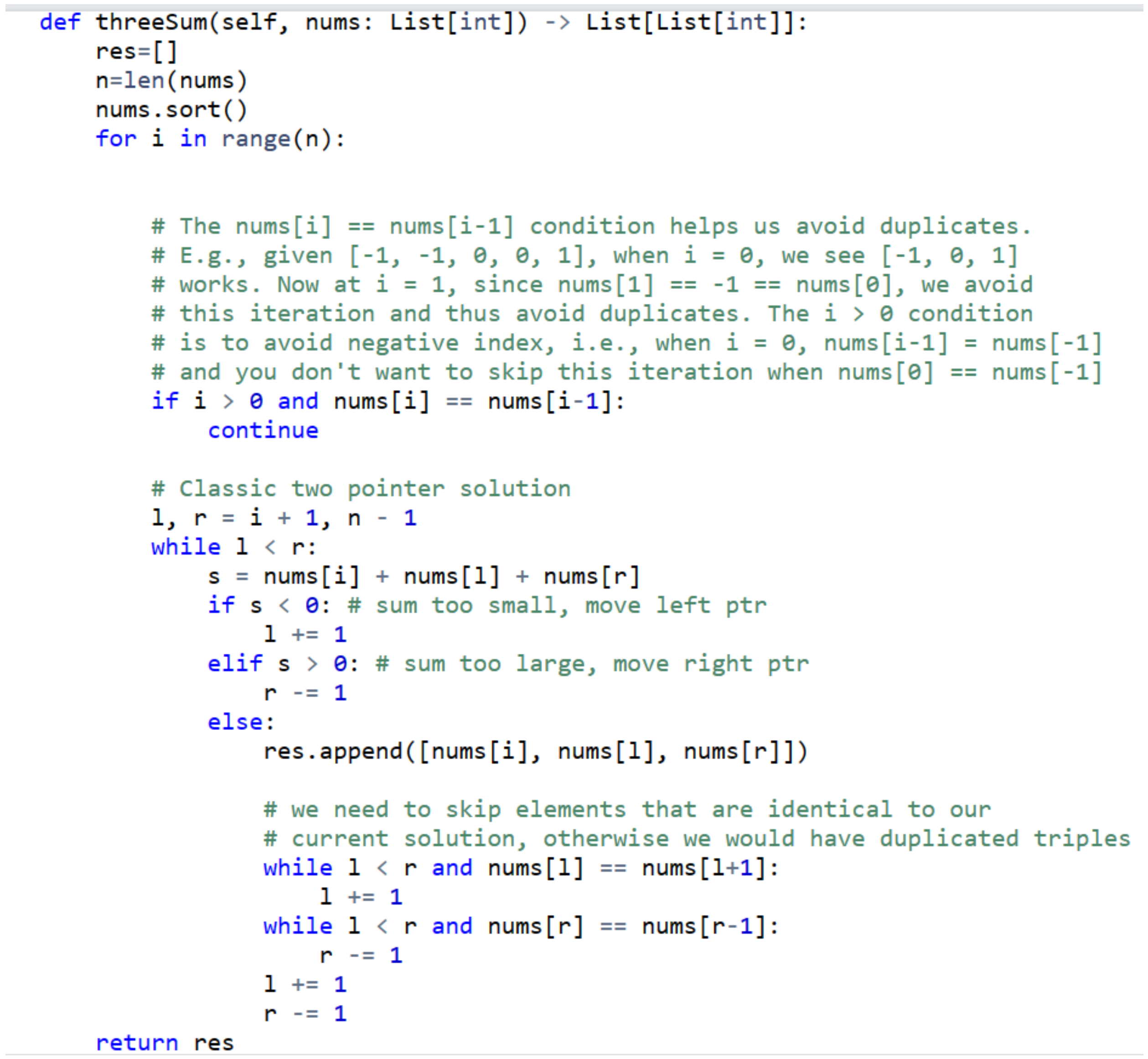
Q2: 3-SUM

Approach optimal: Standard two pointer approach.

First sort the array

TC:O(N^2)

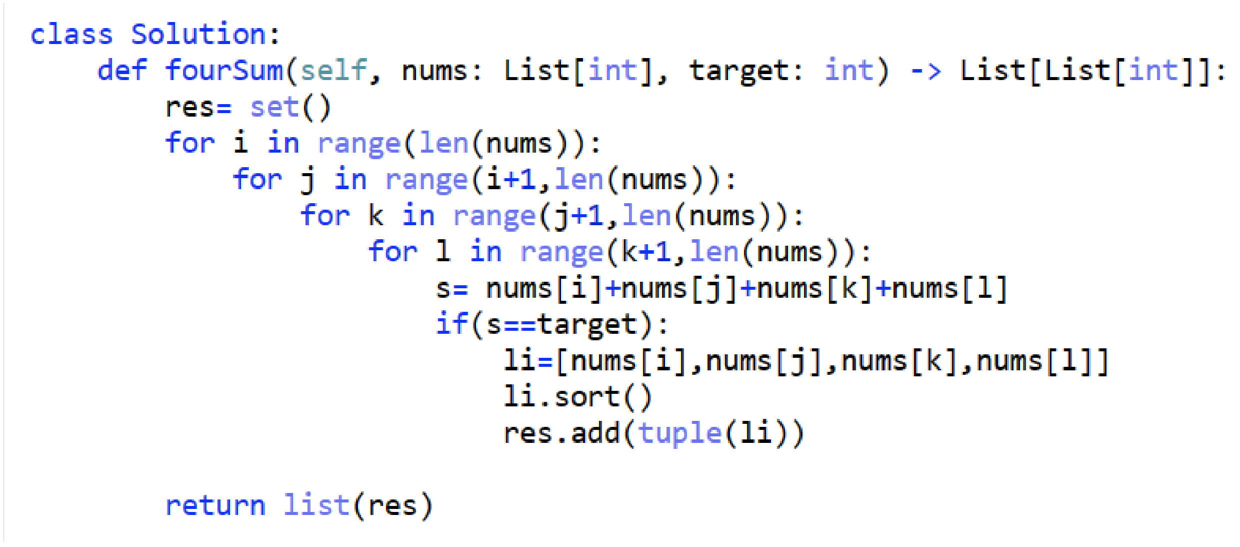
SC:O(no of quad)



```python
def threeSum(self, nums: List[int]) -> List[List[int]]:
    res=[]
    n=len(nums)
    nums.sort()
    for i in range(n):


        # The nums[i] == nums[i-1] condition helps us avoid duplicates.
        # E.g., given [-1, -1, 0, 0, 1], when i = 0, we see [-1, 0, 1]
        # works. Now at i = 1, since nums[1] == -1 == nums[0], we avoid
        # this iteration and thus avoid duplicates. The i > 0 condition
        # is to avoid negative index, i.e., when i = 0, nums[i-1] = nums[-1]
        # and you don't want to skip this iteration when nums[0] == nums[-1]
        if i > 0 and nums[i] == nums[i-1]:
            continue

        # Classic two pointer solution
        l, r = i + 1, n - 1
        while l < r:
            s = nums[i] + nums[l] + nums[r]
            if s < 0: # sum too small, move left ptr
                l += 1
            elif s > 0: # sum too large, move right ptr
                r -= 1
            else:
                res.append([nums[i], nums[l], nums[r]])

                # we need to skip elements that are identical to our
                # current solution, otherwise we would have duplicated triples
                while l < r and nums[l] == nums[l+1]:
                    l += 1
                while l < r and nums[r] == nums[r-1]:
                    r -= 1
                l += 1
                r -= 1
    return res
```

Q3: 4-SUM:

Brute force: simply using 4 for loops



```python
class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        res= set()
        for i in range(len(nums)):
            for j in range(i+1,len(nums)):
                for k in range(j+1,len(nums)):
                    for l in range(k+1,len(nums)):
                        s= nums[i]+nums[j]+nums[k]+nums[l]
                        if(s==target):
                            li=[nums[i],nums[j],nums[k],nums[l]]
                            li.sort()
                            res.add(tuple(li))

        return list(res)
```

NOTE: In python to add the list in set we need to add it using tuple()

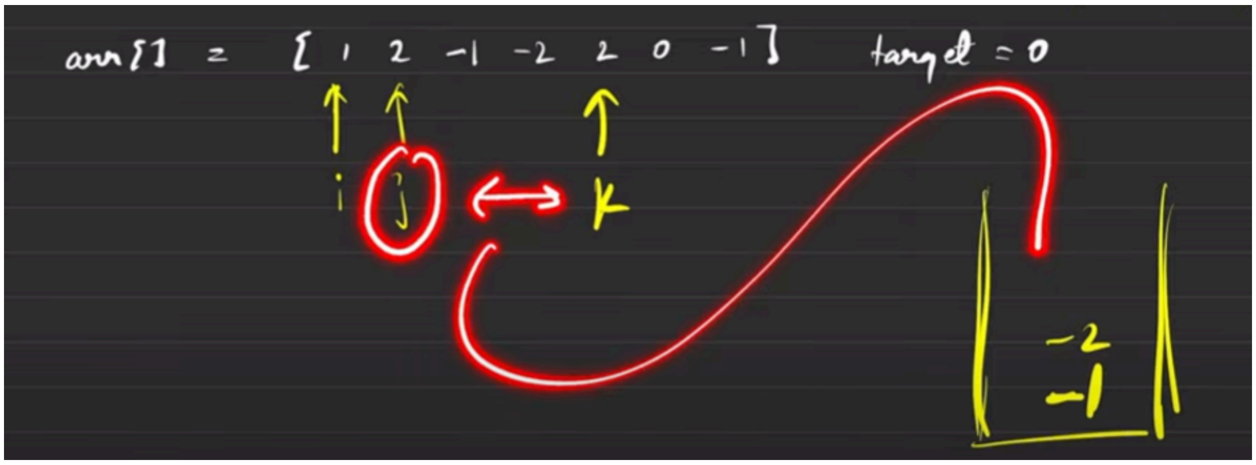
TC: O(N^4)

SC: O(No of quads)*2 (we using 2 as we use it to return the ans)

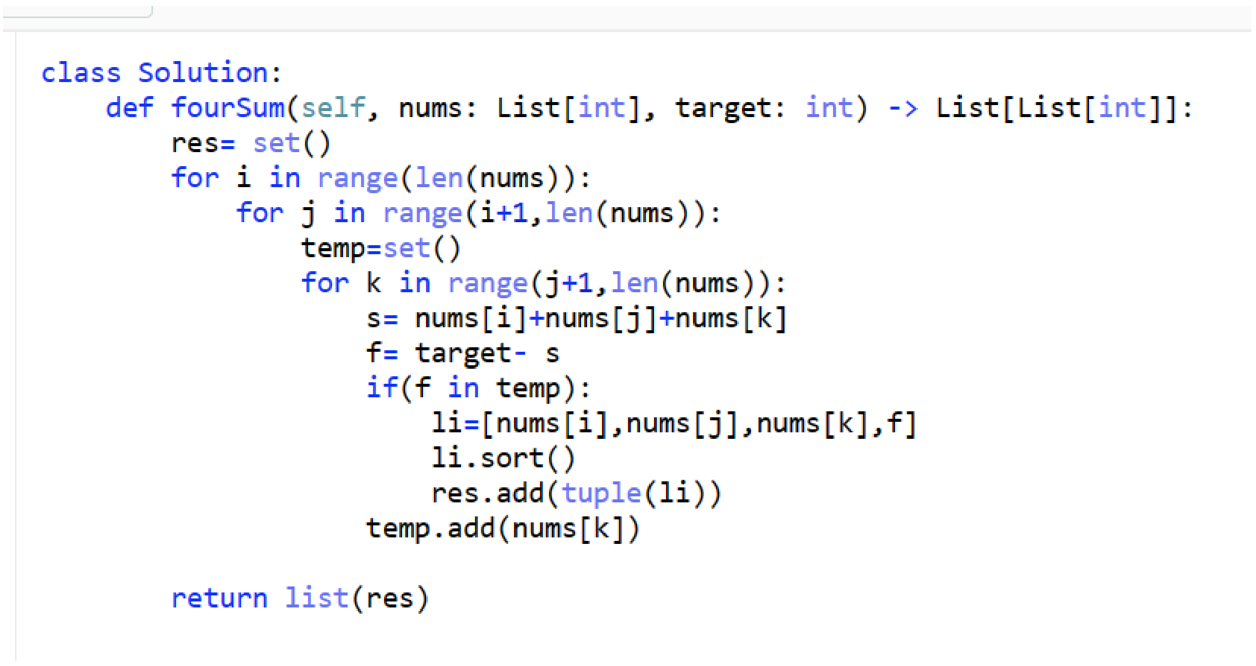
Better approach: Takes O(n^3)log(m)

Sc:O(n) auxiliary space for set used to store nums[k] lets say all element atleast once store in it if all are distinct, so

SC: O(n)* O(quads)*2

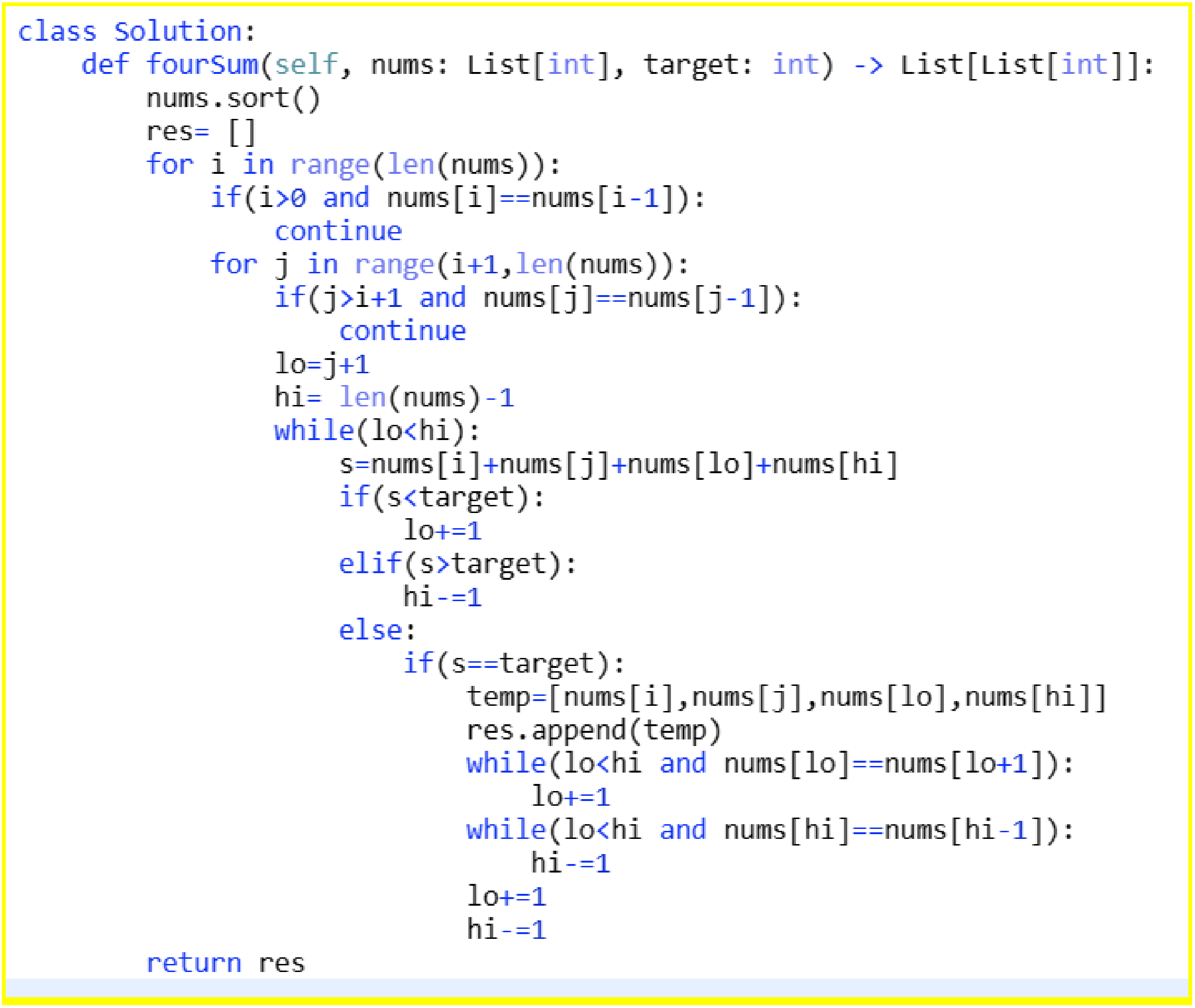


We have to first check the set for the fourth element and then keep the nums[k] in the set if not there to make sure that nums[k] won't come twice.



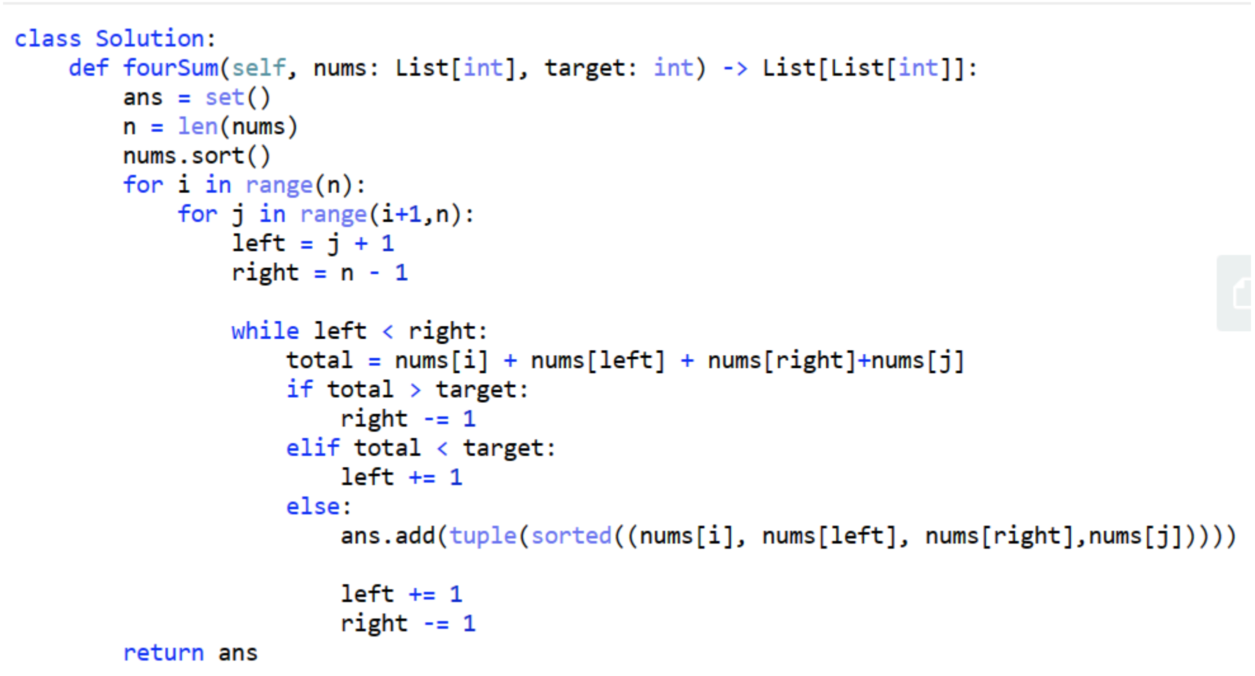
```python
class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        res= set()
        for i in range(len(nums)):
            for j in range(i+1,len(nums)):
                temp=set()
                for k in range(j+1,len(nums)):
                    s= nums[i]+nums[j]+nums[k]
                    f= target- s
                    if(f in temp):
                        li=[nums[i],nums[j],nums[k],f]
                        li.sort()
                        res.add(tuple(li))
                    temp.add(nums[k])

        return list(res)
```

Final Optimized approach: Without using extra space, as well as without using set data structure.
NOTE: Always remember to sort first otherwise this method won't work.
TC: O(n^3)
SC: O(no of quad) we not using *2 as the same is returned as ans here.



```python
class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        nums.sort()
        res= []
        for i in range(len(nums)):
            if(i>0 and nums[i]==nums[i-1]):
                continue
            for j in range(i+1,len(nums)):
                if(j>i+1 and nums[j]==nums[j-1]):
                    continue
                lo=j+1
                hi= len(nums)-1
                while(lo<hi):
                    s=nums[i]+nums[j]+nums[lo]+nums[hi]
                    if(s<target):
                        lo+=1
                    elif(s>target):
                        hi-=1
                    else:
                        if(s==target):
                            temp=[nums[i],nums[j],nums[lo],nums[hi]]
                            res.append(temp)
                            while(lo<hi and nums[lo]==nums[lo+1]):
                                lo+=1
                            while(lo<hi and nums[hi]==nums[hi-1]):
                                hi-=1
                            lo+=1
                            hi-=1
        return res
```

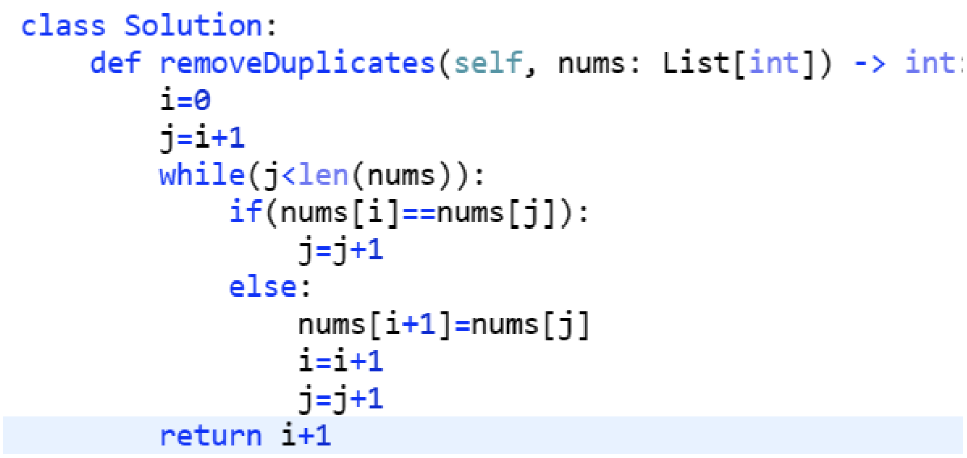
We can also do it like this in writing by using set data structure so we don't have to take care about duplicates.



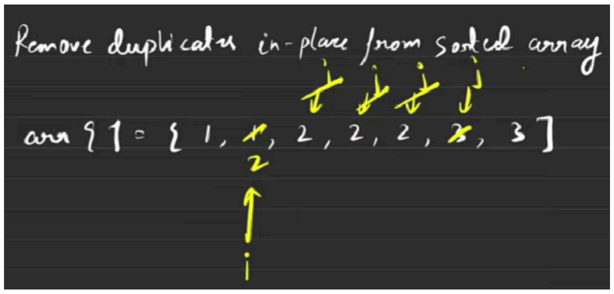
```
class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        ans = set()
        n = len(nums)
        nums.sort()
        for i in range(n):
            for j in range(i+1,n):
                left = j + 1
                right = n - 1

                while left < right:
                    total = nums[i] + nums[left] + nums[right]+nums[j]
                    if total > target:
                        right -= 1
                    elif total < target:
                        left += 1
                    else:
                        ans.add(tuple(sorted((nums[i], nums[left], nums[right],nums[j]))))

                        left += 1
                        right -= 1
        return ans
```

Q5: Remove duplicates from sorted array.

TC:O(n)

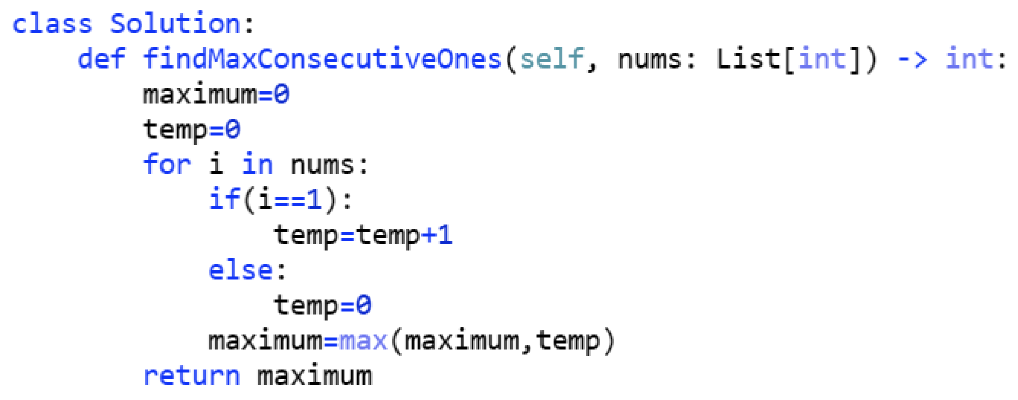


```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int
        i=0
        j=i+1
        while(j<len(nums)):
            if(nums[i]==nums[j]):
                j=j+1
            else:
                nums[i+1]=nums[j]
                i=i+1
                j=j+1
        return i+1
```


Remove duplicates in-place from sorted array
arr[] = { 1, 1, 2, 2, 2, 3, 3]
j
2
i

Q6: Maximum consecutive 1’s

TC:O(N)



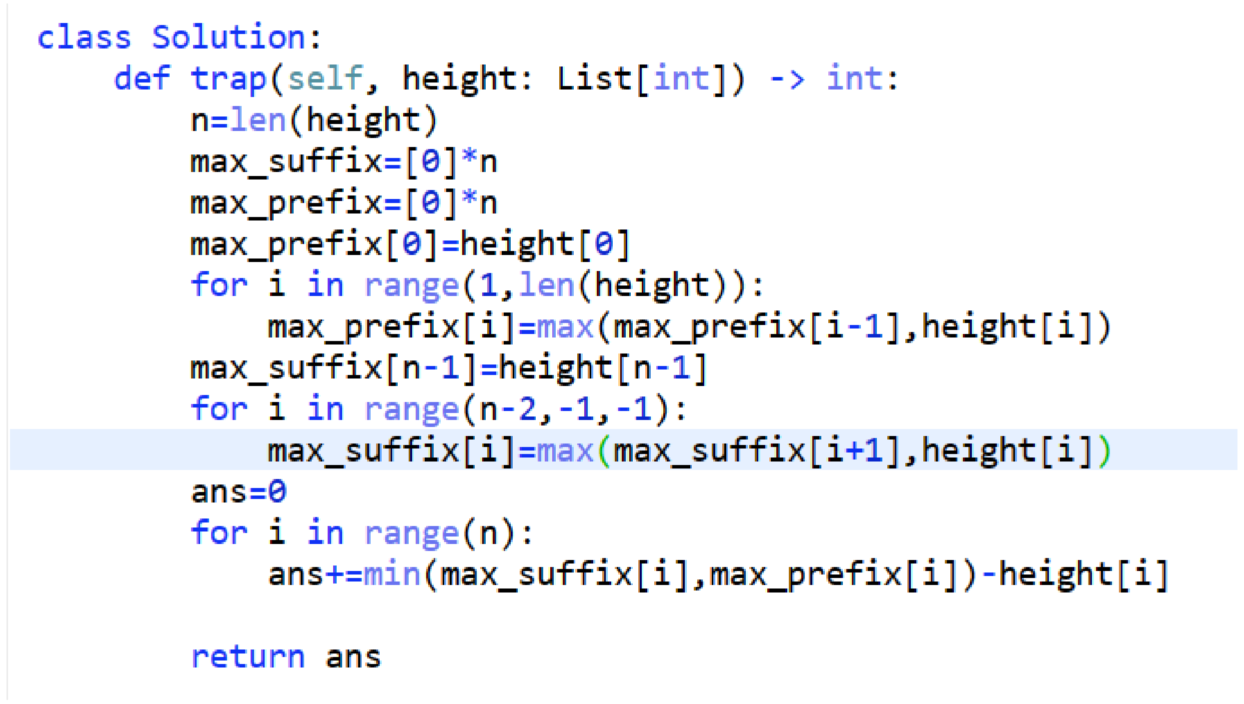
```
class Solution:
    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
        maximum=0
        temp=0
        for i in nums:
            if(i==1):
                temp=temp+1
            else:
                temp=0
            maximum=max(maximum,temp)
        return maximum
```

Q7: Trapping Rain Water:

Better approach: Using max suffix and prefix array

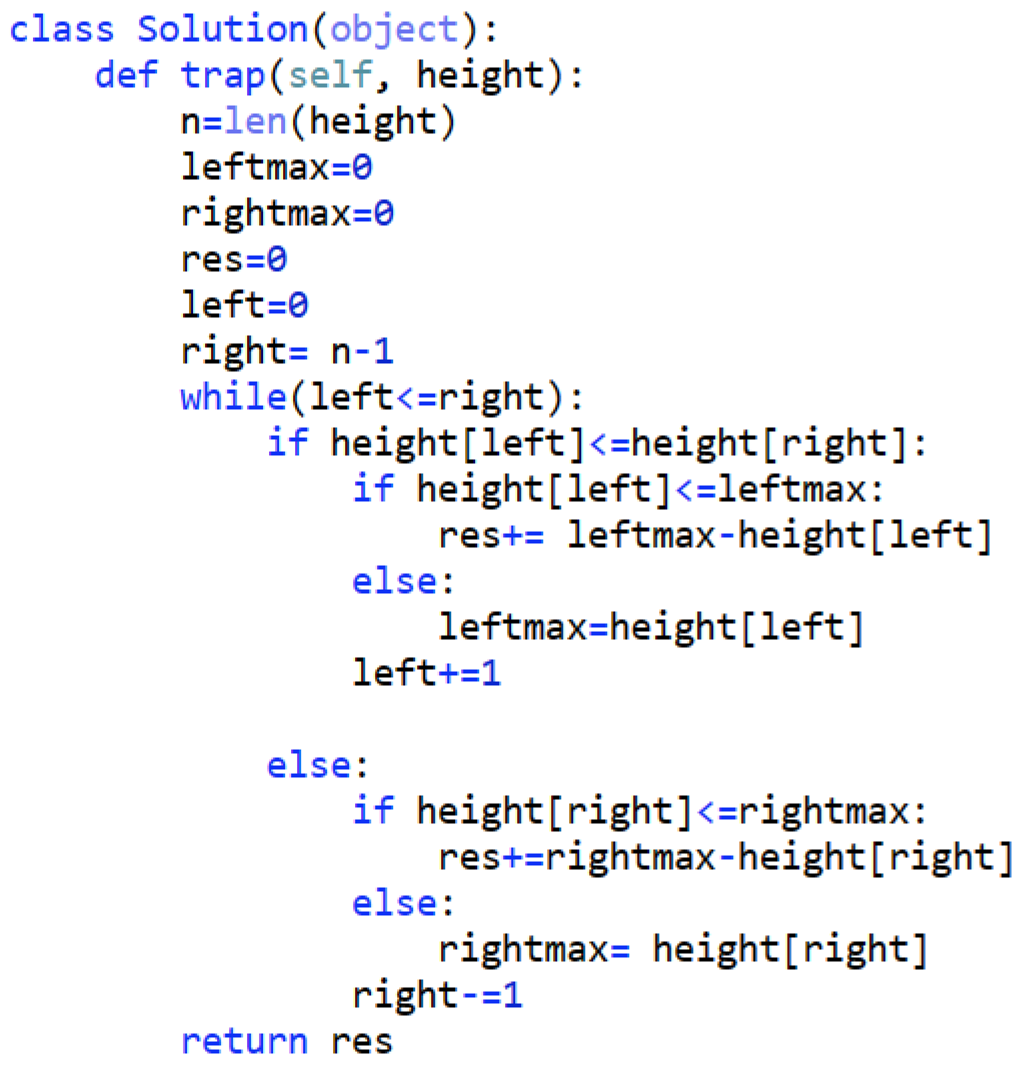
TC: O(N)

SC:O(2N)



```
class Solution:
    def trap(self, height: List[int]) -> int:
        n=len(height)
        max_suffix=[0]*n
        max_prefix=[0]*n
        max_prefix[0]=height[0]
        for i in range(1,len(height)):
            max_prefix[i]=max(max_prefix[i-1],height[i])
        max_suffix[n-1]=height[n-1]
        for i in range(n-2,-1,-1):
            max_suffix[i]=max(max_suffix[i+1],height[i])
        ans=0
        for i in range(n):
            ans+=min(max_suffix[i],max_prefix[i])-height[i]

        return ans
```

Optimal approach: use two pointer approach:



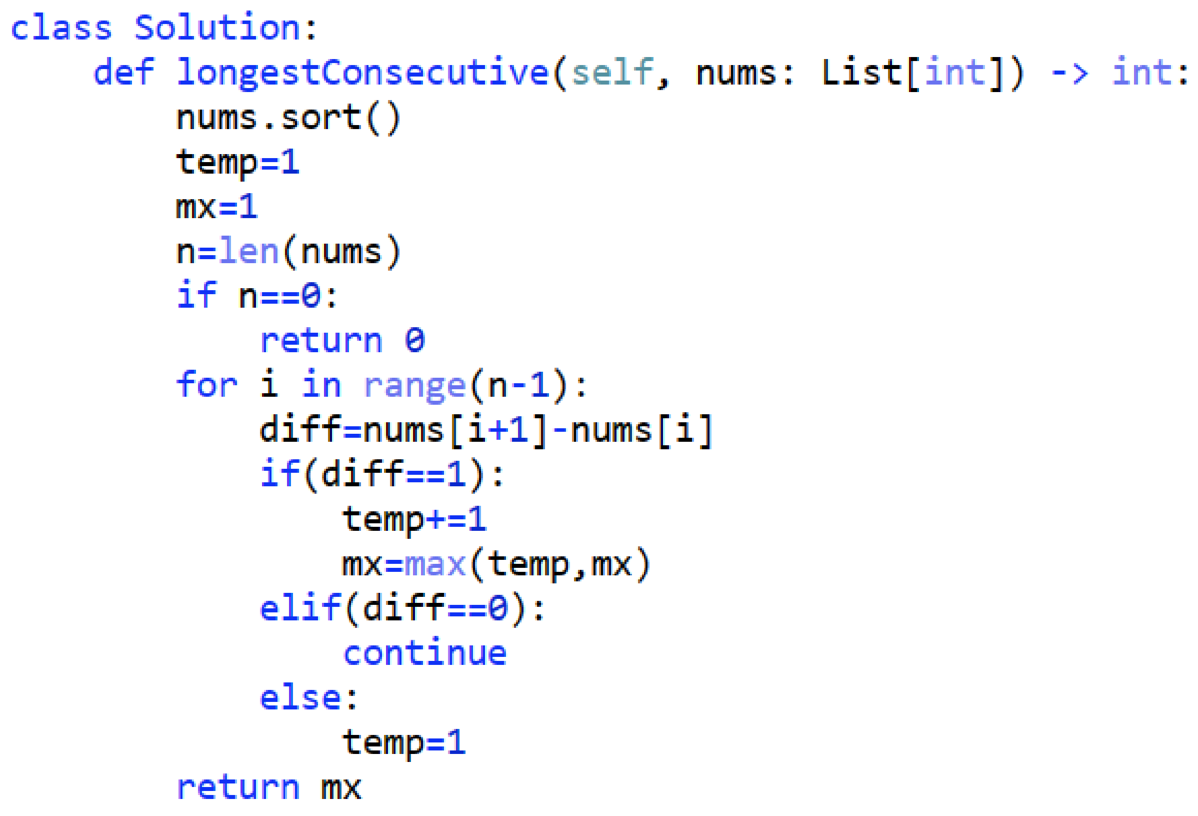
```python
class Solution(object):
    def trap(self, height):
        n=len(height)
        leftmax=0
        rightmax=0
        res=0
        left=0
        right= n-1
        while(left<=right):
            if height[left]<=height[right]:
                if height[left]<=leftmax:
                    res+= leftmax-height[left]
                else:
                    leftmax=height[left]
                left+=1

            else:
                if height[right]<=rightmax:
                    res+=rightmax-height[right]
                else:
                    rightmax= height[right]
                right-=1
        return res
```

TC: O(N)
SC:O(1)

Q: Longest consecutive sequence:

Better solution:

TC: O(nlogn)
SC: O(1)



```
class Solution:
    def longestConsecutive(self, nums: List[int]) -> int:
        nums.sort()
        temp=1
        mx=1
        n=len(nums)
        if n==0:
            return 0
        for i in range(n-1):
            diff=nums[i+1]-nums[i]
            if(diff==1):
                temp+=1
                mx=max(temp,mx)
            elif(diff==0):
                continue
            else:
                temp=1
        return mx
```

Optimal solution: TC:
O(N)
SC: O(N)
Approach: take a set and then iterate over it , lets say 6 is the element then check for 5 if its exists leave that and move forward , and if 5 does not exists then it is the first point from where the consecutive elements starts and then run the loop for 7, 8, 9 exists in set or not if exists then increase the count and store.

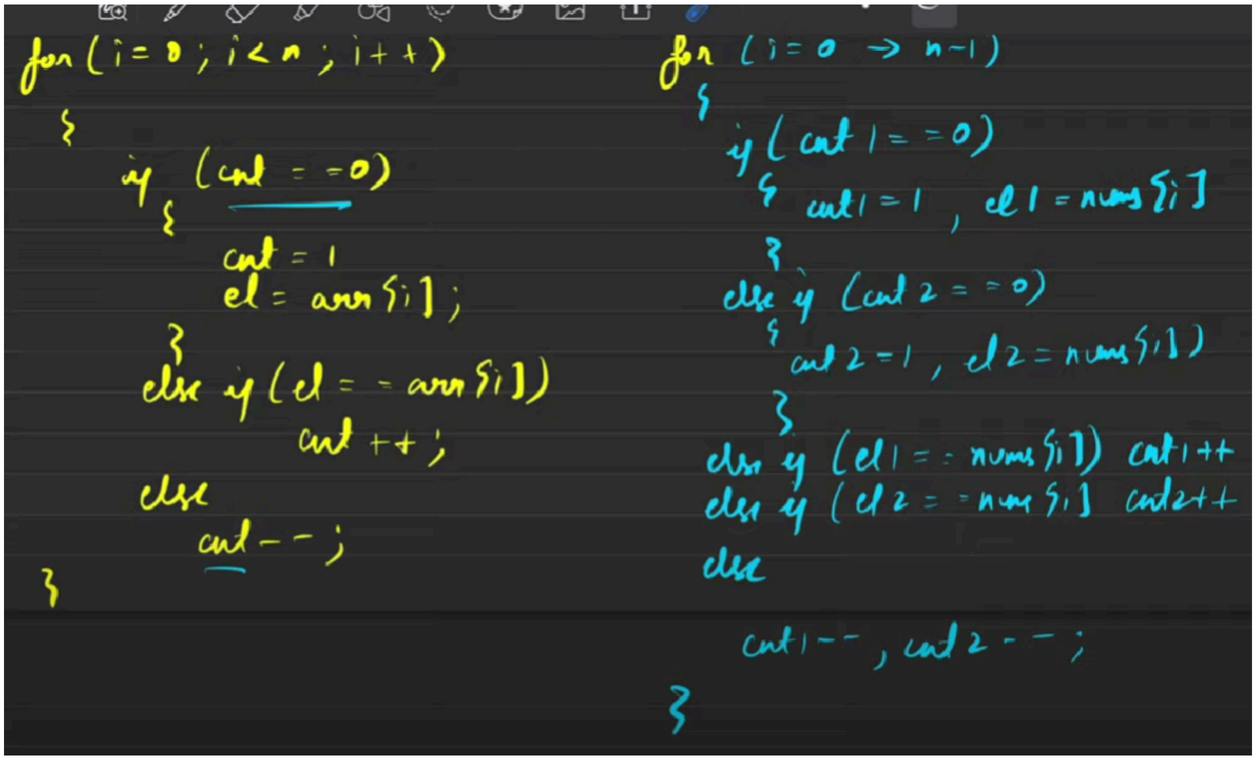
```
class Solution:
    def longestConsecutive(self, nums: List[int]) -> int:
        ans=1
        temp=1
        s=set(nums)
        print(s)
        if(not s):
            return 0
        for i in s:
            c=i-1
            if(c not in s):
                #search for i+1,i+2 etc
                j=i+1
                while(j in s):
                    temp+=1
                    j+=1
            ans=max(temp,ans)
            temp=1
        return ans
```

Q: Search in 2 d matrix:
Given row wise sorted and first col value is always less than or equal to last row last element. So eventually the matrix is sorted overall. So we use binary search for it.

```
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
        n=len(matrix)  #rows
        m=len(matrix[0])   #columns
        low=0
        high=(m*n)-1
        while(low<=high):
            mid= (low+high)//2
            if matrix[mid//m][mid%m]==target:
                return True
            elif matrix[mid//m][mid%m]<target:
                low=mid+1
            else:
                high=mid-1
        return False
```

Q: Find element the occurs greater than n/2 and n/3:
These two questions are quite similar:

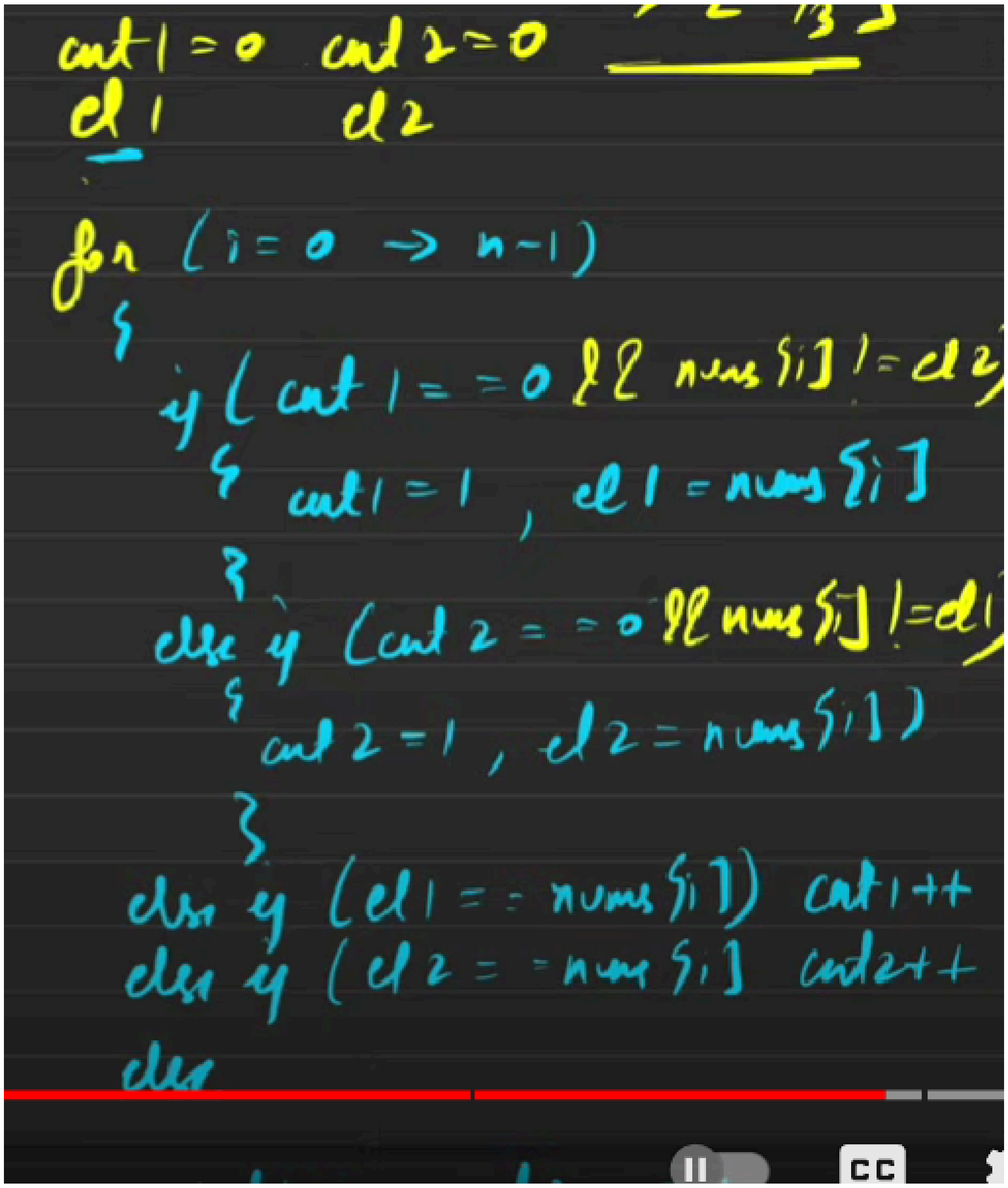
Initially cnt =0 and ele=-1 in both the cases.



```
for(i=0; i<n; i++)
{
    if (cnt == 0)
    {
        cnt = 1
        el = arr[i];
    }
    else if(el == arr[i])
        cnt++;
    else
        cnt--;
}
```

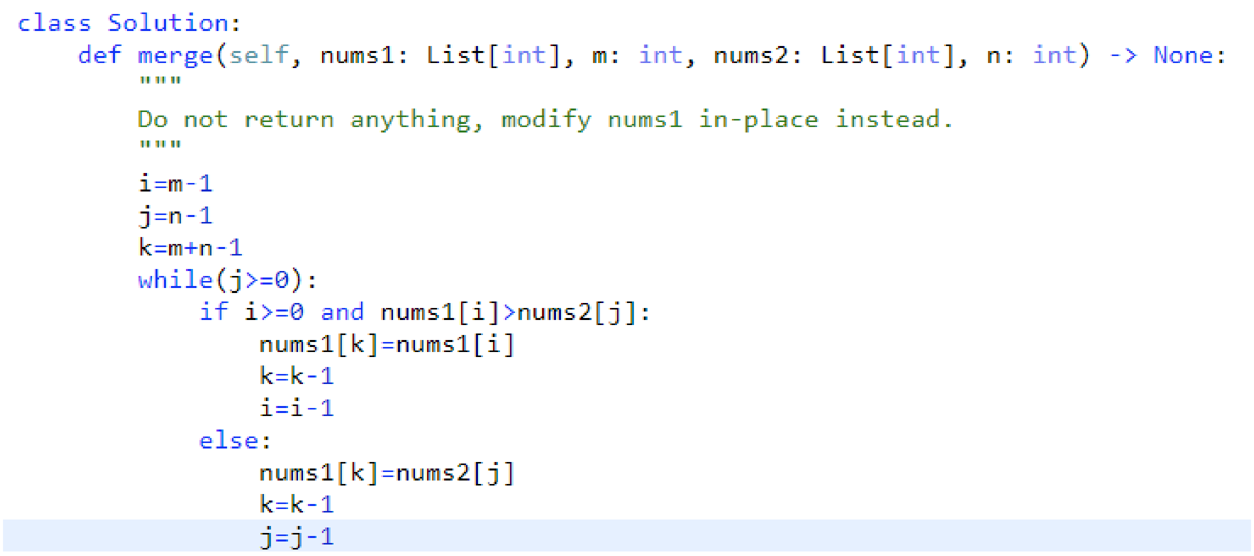
```
for (i=0 -> n-1)
{
    if (cnt1 == 0)
    { cnt1 = 1 , el1 = nums[i]
    }
    else if (cnt2 == 0)
    { cnt2 = 1 , el2 = nums[i])
    }
    else if (el1 == nums[i]) cnt1++
    else if (el2 == nums[i] cnt2++
    else
        cnt1-- , cnt2--;
}
```

Hold on! Wait some extra condition need to be added.

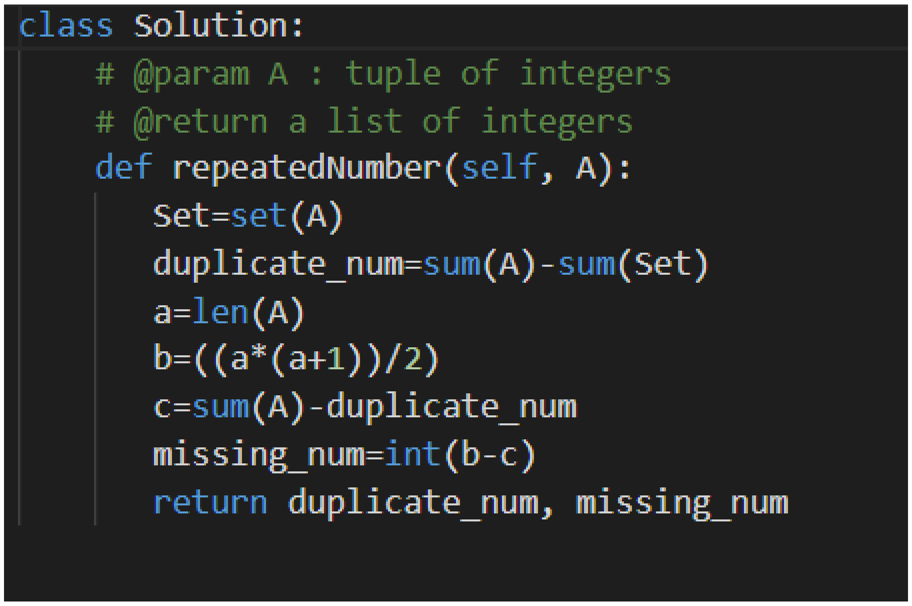


```
cnt1 = 0    cnt2 = 0
el1         el2

for (i = 0 -> n-1)
{
    if (cnt1 == 0 && nums[i] != el2)
    {  cnt1 = 1, el1 = nums[i]
    }
    else if (cnt2 == 0 && nums[i] != el1)
    {
       cnt2 = 1, el2 = nums[i])
    }
    else if (el1 == nums[i]) cnt1++
    else if (el2 == nums[i] cnt2++
    else
```

Q Merge 2 sorted array:


```python
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        """
        Do not return anything, modify nums1 in-place instead.
        """
        i=m-1
        j=n-1
        k=m+n-1
        while(j>=0):
            if i>=0 and nums1[i]>nums2[j]:
                nums1[k]=nums1[i]
                k=k-1
                i=i-1
            else:
                nums1[k]=nums2[j]
                k=k-1
                j=j-1
```

Repeat and Missing Number Array


```python
class Solution:
    # @param A : tuple of integers
    # @return a list of integers
    def repeatedNumber(self, A):
        Set=set(A)
        duplicate_num=sum(A)-sum(Set)
        a=len(A)
        b=((a*(a+1))/2)
        c=sum(A)-duplicate_num
        missing_num=int(b-c)
        return duplicate_num, missing_num
```

Unique path: This is solved by dp:

```
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        def f(i,j):
            if i==0 and j==0:
                return 1
            if i<0 or j<0:
                return 0
            if dp[i][j]!=-1:
                return dp[i][j]
            up=0
            left=0
            if i>0:
                up= f(i-1,j)
            if j>0:
                left= f(i,j-1)
            dp[i][j]=up+left
            return dp[i][j]
        dp=[[-1]*n for i in range(m)]
        return f(m-1,n-1)
```

Q: longest consecutive subsequences:

```
class Solution:
    def longestConsecutive(self, nums: List[int]) -> int:
        s=set(nums)
        maxi=0
        if not s:
            return 0
        for i in s:
            temp=1
            if (i-1) in s:
                continue
            #if i-1 not in set then start from it and
            #find all i+1,i+2 upto where element is present and count it.
            else:
                j=i+1
                while(j in s):
                    j=j+1
                    temp+=1
                maxi= max(temp,maxi)
        return maxi
```

Q: Longest substring without any repeating char:

Better approach: O(2N) time , O(n) space

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        n=len(s)
        st= set()
        if n==0:
            return 0
        maxi=float('-inf')
        l=0
        for r in range(n):
            if s[r] in st:
                while(l<r and s[r] in st):
                    st.remove(s[l])
                    l+=1
            st.add(s[r])
            maxi=max(maxi,r-l+1)
        return maxi
```

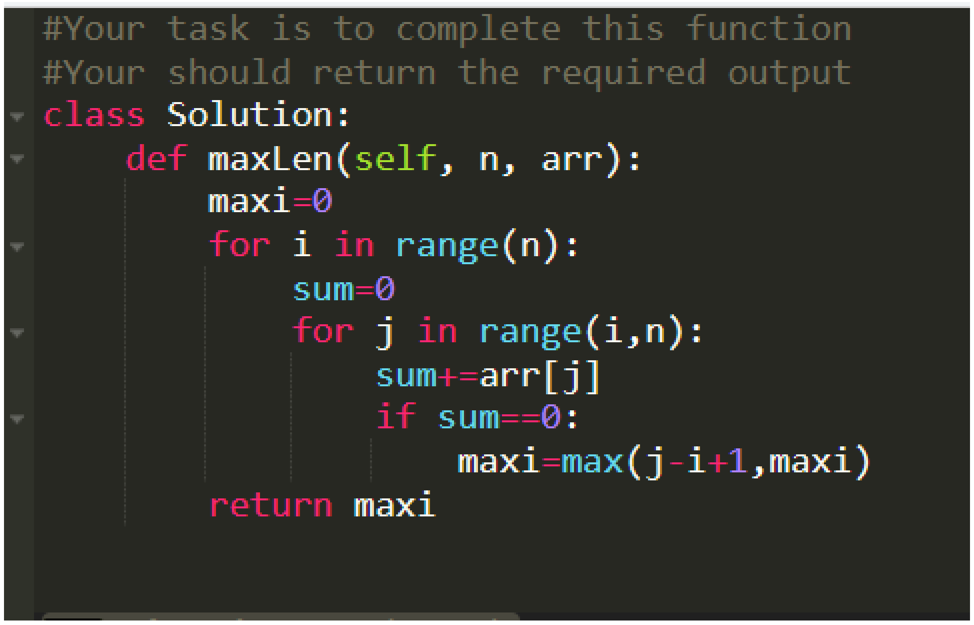
Optimal approach use O(n) tc and O(n) sc:

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        n=len(s)
        st={}
        if n==0:
            return 0
        maxi=float('-inf')
        l=0
        for r in range(n):
            if s[r] in st and st[s[r]] >= l:
                l=st[s[r]]+1
            st[s[r]]=r
            maxi=max(maxi,r-l+1)
        return maxi
```

Q: Largest subarray with sum 0:

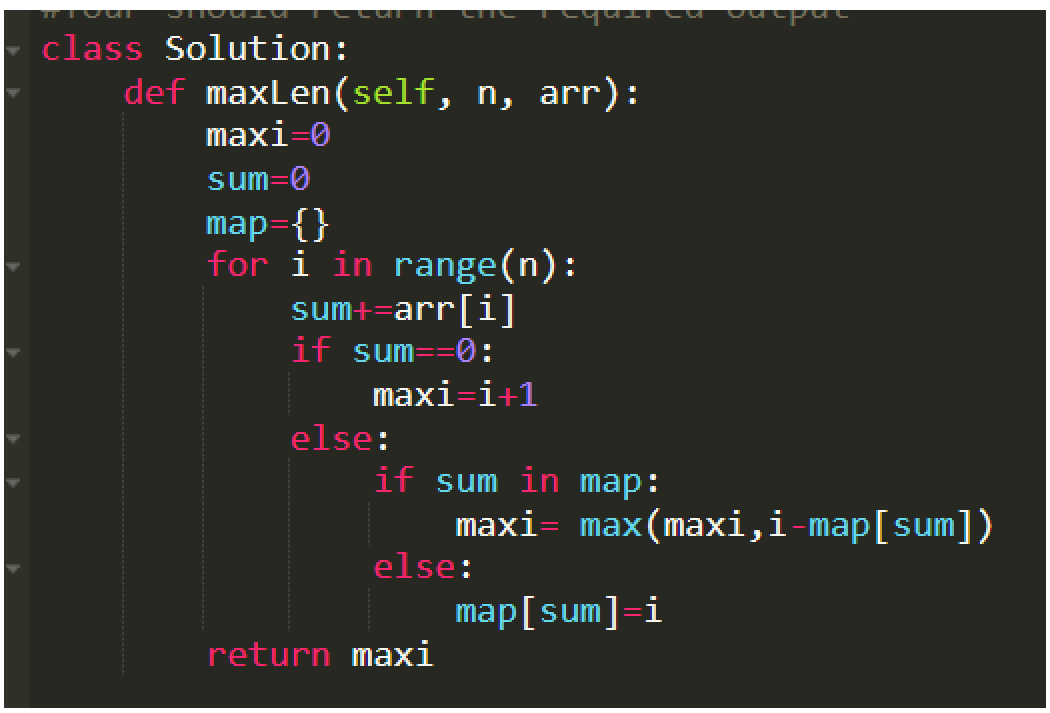
Brute force:

TC:O(n^2) sc: O(1)



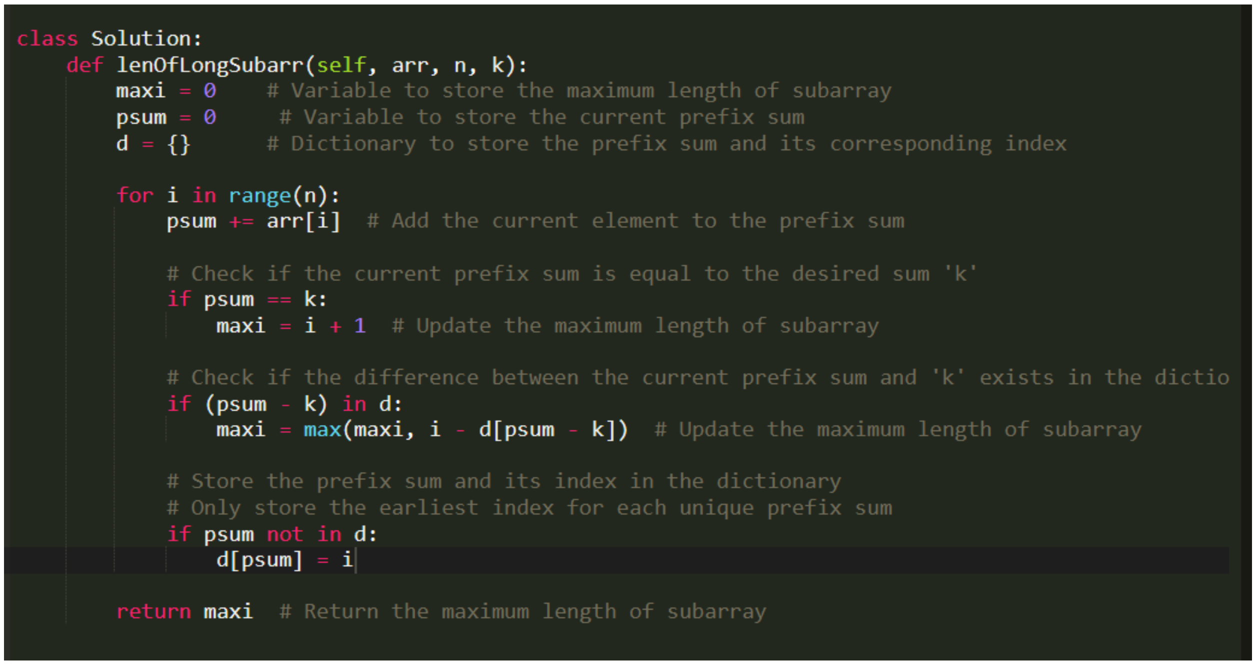
```python
#Your task is to complete this function
#Your should return the required output
class Solution:
    def maxLen(self, n, arr):
        maxi=0
        for i in range(n):
            sum=0
            for j in range(i,n):
                sum+=arr[j]
                if sum==0:
                    maxi=max(j-i+1,maxi)
        return maxi
```

Optimal approach:



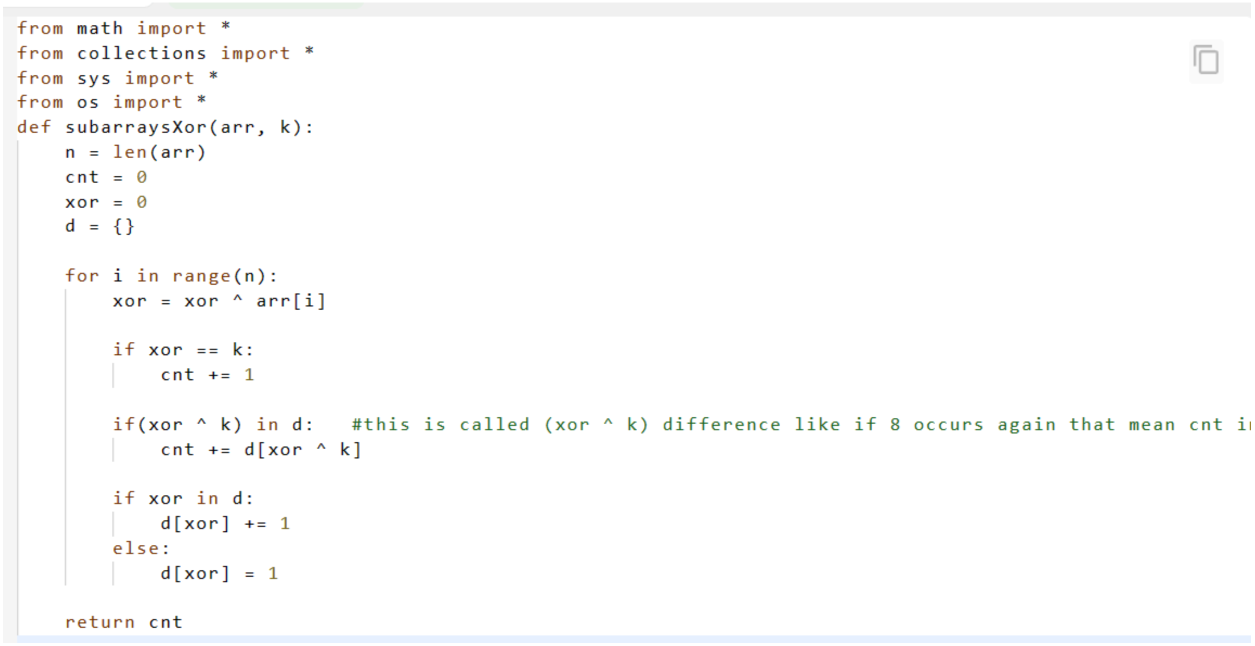
```python
#Your should return the required output
class Solution:
    def maxLen(self, n, arr):
        maxi=0
        sum=0
        map={}
        for i in range(n):
            sum+=arr[i]
            if sum==0:
                maxi=i+1
            else:
                if sum in map:
                    maxi= max(maxi,i-map[sum])
                else:
                    map[sum]=i
        return maxi
```

Largest subarray with sum k:



```python
class Solution:
    def lenOfLongSubarr(self, arr, n, k):
        maxi = 0    # Variable to store the maximum length of subarray
        psum = 0     # Variable to store the current prefix sum
        d = {}      # Dictionary to store the prefix sum and its corresponding index

        for i in range(n):
            psum += arr[i]  # Add the current element to the prefix sum

            # Check if the current prefix sum is equal to the desired sum 'k'
            if psum == k:
                maxi = i + 1  # Update the maximum length of subarray

            # Check if the difference between the current prefix sum and 'k' exists in the dictio
            if (psum - k) in d:
                maxi = max(maxi, i - d[psum - k])  # Update the maximum length of subarray

            # Store the prefix sum and its index in the dictionary
            # Only store the earliest index for each unique prefix sum
            if psum not in d:
                d[psum] = i

        return maxi  # Return the maximum length of subarray
```

Q: SUBARRRAY WITH XOR K:

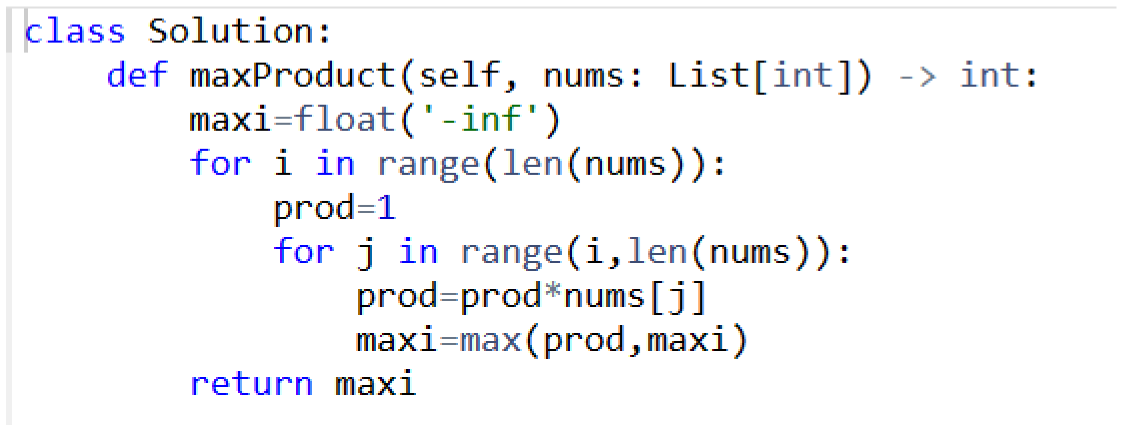


```python
from math import *
from collections import *
from sys import *
from os import *
def subarraysXor(arr, k):
    n = len(arr)
    cnt = 0
    xor = 0
    d = {}

    for i in range(n):
        xor = xor ^ arr[i]

        if xor == k:
            cnt += 1

        if(xor ^ k) in d:   #this is called (xor ^ k) difference like if 8 occurs again that mean cnt i
            cnt += d[xor ^ k]

        if xor in d:
            d[xor] += 1
        else:
            d[xor] = 1

    return cnt
```

Q: MAXIMUM PRODUCT SUBARRAY:

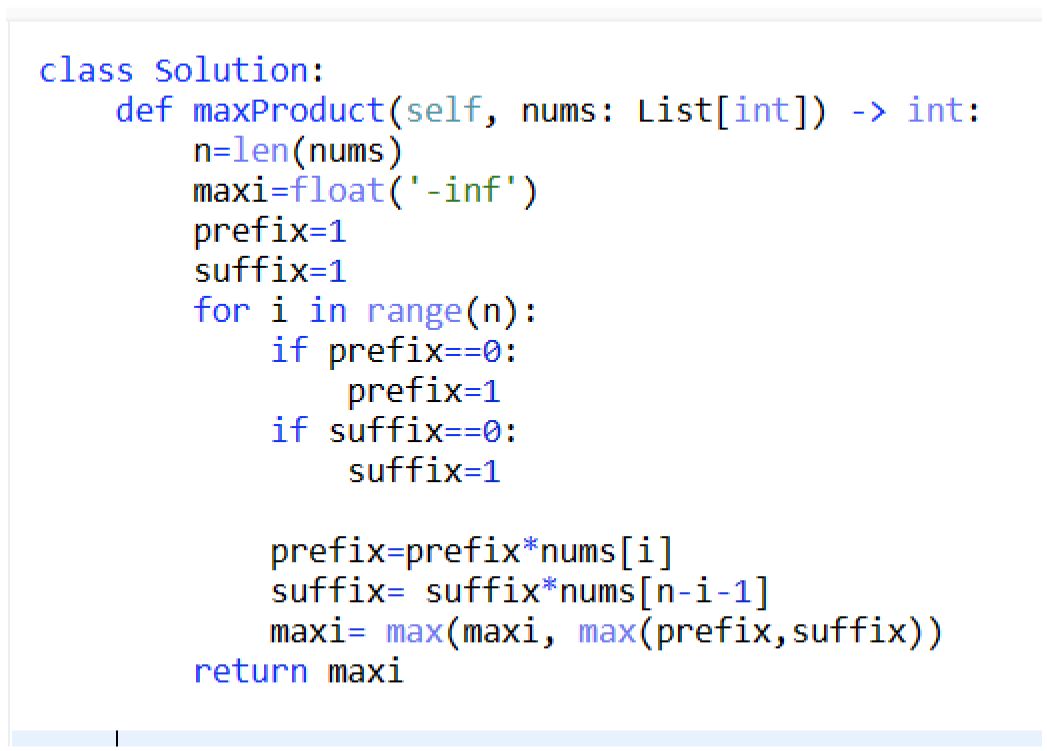
Brute force:

O(n^2)



```python
class Solution:
    def maxProduct(self, nums: List[int]) -> int:
        maxi=float('-inf')
        for i in range(len(nums)):
            prod=1
            for j in range(i,len(nums)):
                prod=prod*nums[j]
                maxi=max(prod,maxi)
        return maxi
```

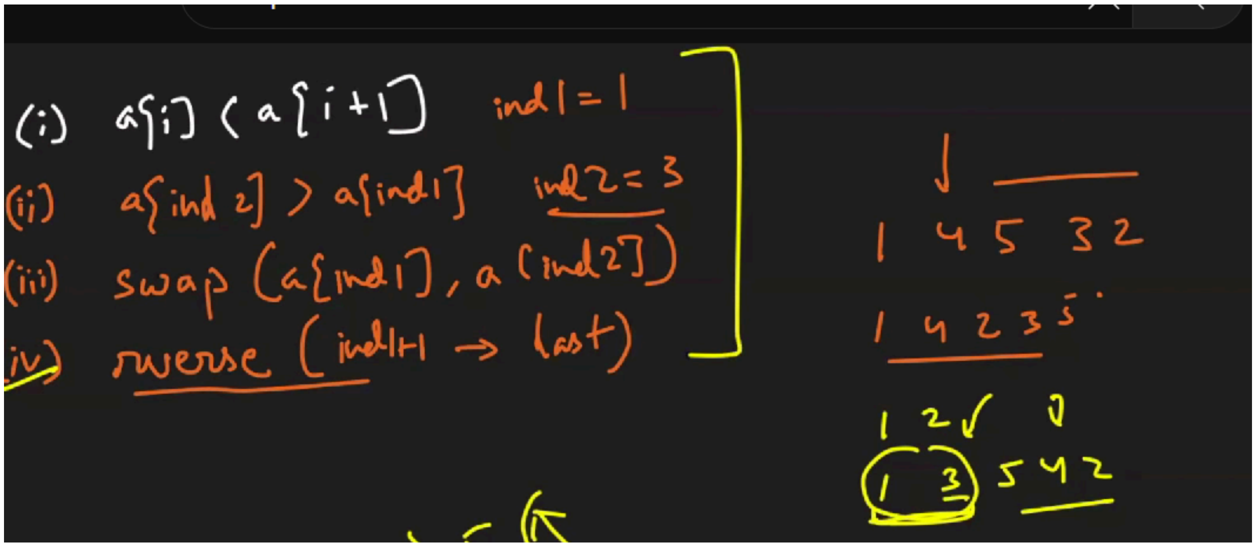
Optimal approach using prefix and suffix product.: O(n)



```python
class Solution:
    def maxProduct(self, nums: List[int]) -> int:
        n=len(nums)
        maxi=float('-inf')
        prefix=1
        suffix=1
        for i in range(n):
            if prefix==0:
                prefix=1
            if suffix==0:
                suffix=1

            prefix=prefix*nums[i]
            suffix= suffix*nums[n-i-1]
            maxi= max(maxi, max(prefix,suffix))
        return maxi
```

Q:Next Permutation:

O(n)


(i) a[i] < a[i+1]   ind1 = 1
(ii) a[ind2] > a[ind1]   ind2 = 3
(iii) swap (a[ind1], a[ind2])
iv) reverse (ind1+1 → last)
1 4 5 3 2
1 4 2 3 5
1 2
1 3 5 4 2

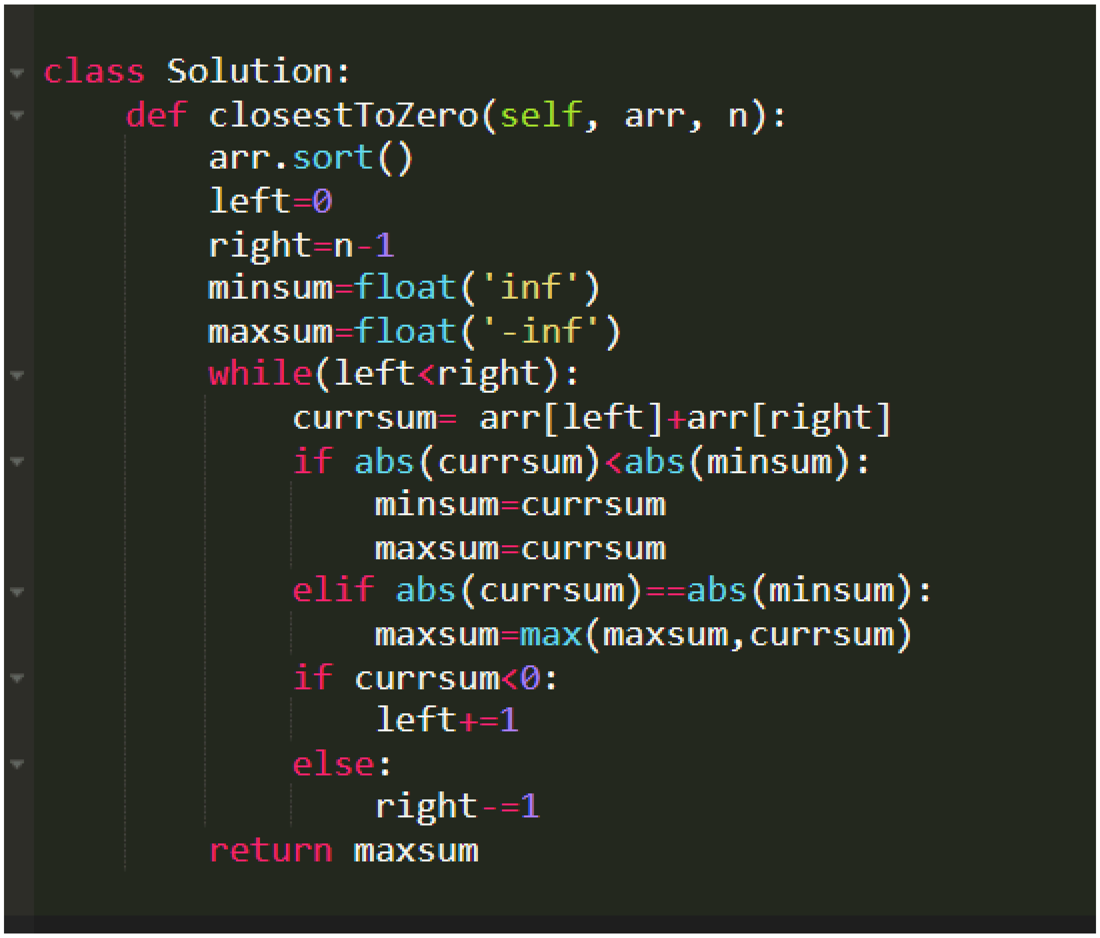
```python
class Solution(object):
    def nextPermutation(self, nums):
        n = len(nums)
        ind1 = -1
        ind2 = -1
        for i in range(n - 2, -1, -1):
            if nums[i] < nums[i + 1]:
                ind1 = i
                break
        if ind1 == -1:
            nums.reverse()
        else:
            for i in range(n - 1, ind1, -1):
                if nums[i] > nums[ind1]:
                    ind2 = i
                    break
            nums[ind1], nums[ind2] = nums[ind2], nums[ind1]
            nums[ind1 + 1:] = reversed(nums[ind1 + 1:])
```

Q: Sum of two element close to zero:

Intuition: sort the array

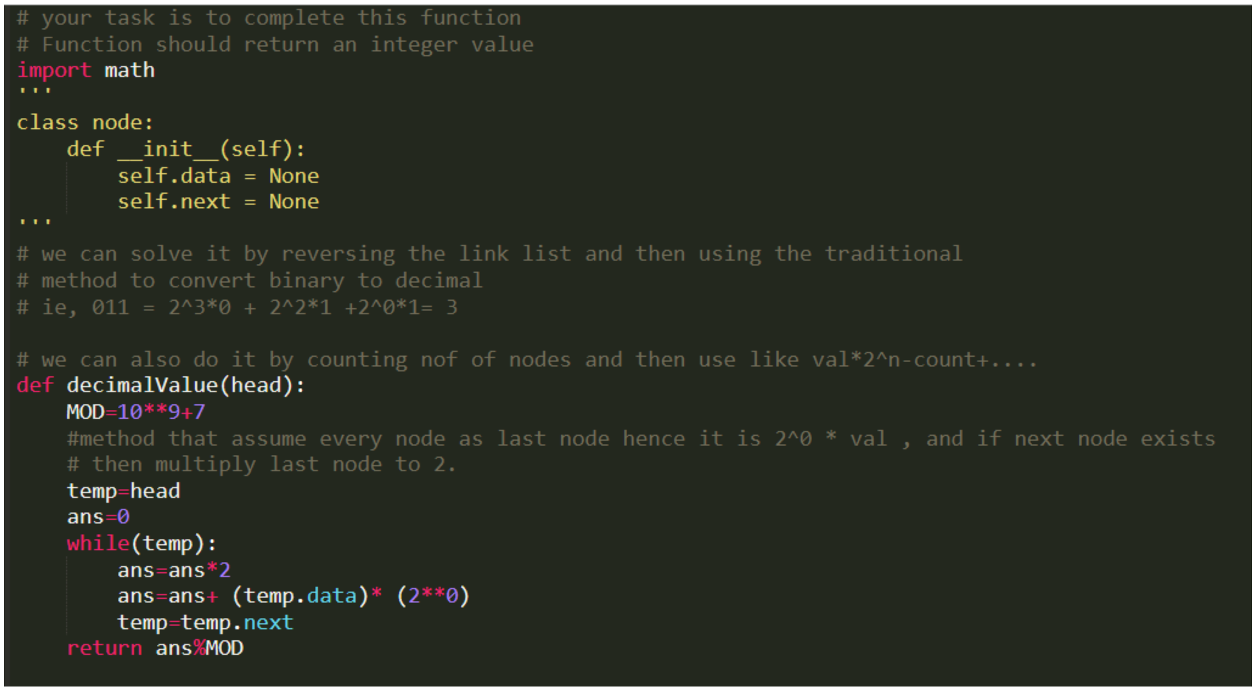
In that if two element who give absolute sum same , then save the positive value. Like if (-5,4) gives -1 and (-3,2) gives 1 we return 1, for this we here use to keep track of maxsum

EX: [-5,-3,-2,2,4]
TC: O(nlogn)



```python
class Solution:
    def closestToZero(self, arr, n):
        arr.sort()
        left=0
        right=n-1
        minsum=float('inf')
        maxsum=float('-inf')
        while(left<right):
            currsum= arr[left]+arr[right]
            if abs(currsum)<abs(minsum):
                minsum=currsum
                maxsum=currsum
            elif abs(currsum)==abs(minsum):
                maxsum=max(maxsum,currsum)
            if currsum<0:
                left+=1
            else:
                right-=1
        return maxsum
```

Q: Decimal equivalent of binary linklist:



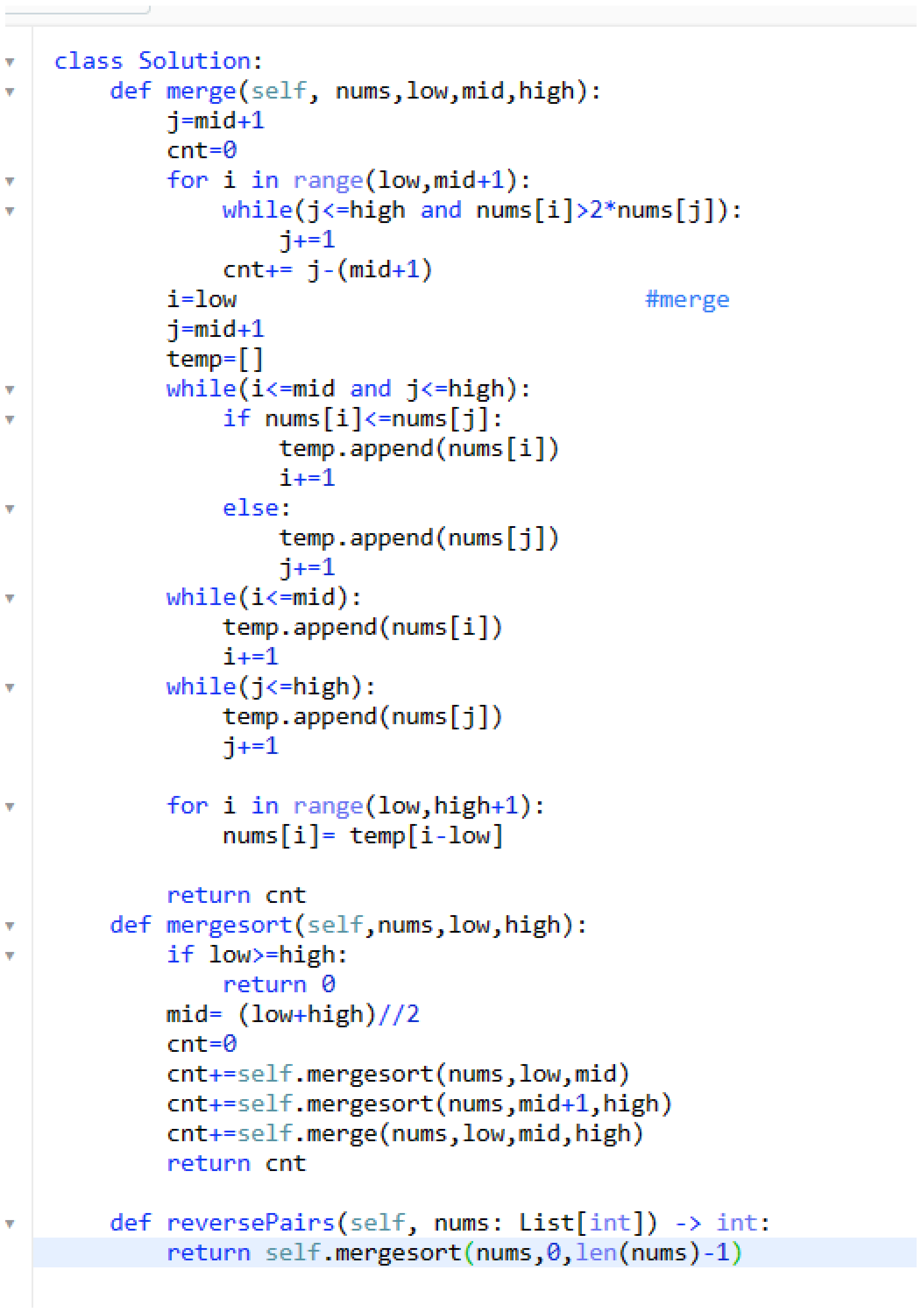
```
# your task is to complete this function
# Function should return an integer value
import math
'''
class node:
    def __init__(self):
        self.data = None
        self.next = None
'''
# we can solve it by reversing the link list and then using the traditional
# method to convert binary to decimal
# ie, 011 = 2^3*0 + 2^2*1 +2^0*1= 3

# we can also do it by counting nof of nodes and then use like val*2^n-count+....
def decimalValue(head):
    MOD=10**9+7
    #method that assume every node as last node hence it is 2^0 * val , and if next node exists
    # then multiply last node to 2.
    temp=head
    ans=0
    while(temp):
        ans=ans*2
        ans=ans+ (temp.data)* (2**0)
        temp=temp.next
    return ans%MOD
```

Count inversion pairs:

```
def merge(skill, low, mid, high):
    # Initialize the pointer for the right half of the array
    j = mid + 1
    # Initialize the inversion count for this merge step
    cnt = 0

    # Count inversions for each element in the left half of
the array
    for i in range(low, mid + 1):
        while j <= high and skill[i] > 2 * skill[j]:
            j += 1
        # Calculate the number of inversions
        cnt += j - (mid + 1)

    # Merge the two halves while sorting the elements
    temp = []
    i = low
    j = mid + 1
    while i <= mid and j <= high:
        if skill[i] <= skill[j]:
            temp.append(skill[i])
            i += 1
        else:
            temp.append(skill[j])
            j += 1
    while i <= mid:
        temp.append(skill[i])
        i += 1
    while j <= high:
        temp.append(skill[j])
        j += 1

    # Copy back the merged and sorted elements to the original
array
    for i in range(low, high + 1):
        skill[i] = temp[i - low]

    # Return the number of inversions counted in this merge
step
    return cnt


def mergeSort(skill, low, high):
    # Base case: if the array size is 1 or less, it's already
sorted
    if low >= high:
        return 0

    # Find the middle index
    mid = (low + high) // 2

    # Initialize the inversion count for this mergeSort call
    inv = 0

    # Recursively sort and count inversions for the left and
right halves
    inv += mergeSort(skill, low, mid)
    inv += mergeSort(skill, mid + 1, high)

    # Merge the two sorted halves and count the inversions
during merge
    inv += merge(skill, low, mid, high)

    # Return the total number of inversions for this call
    return inv


def team(skill: [int], n: int) -> int:
    # Call mergeSort to sort the array and count the
inversions
    return mergeSort(skill, 0, n - 1)
```

Team Contest - Coding Ninjas = Same as inversion pairs

Reverse pairs is same as inversion pairs



```python
class Solution:
    def merge(self, nums,low,mid,high):
        j=mid+1
        cnt=0
        for i in range(low,mid+1):
            while(j<=high and nums[i]>2*nums[j]):
                j+=1
            cnt+= j-(mid+1)
        i=low                             #merge
        j=mid+1
        temp=[]
        while(i<=mid and j<=high):
            if nums[i]<=nums[j]:
                temp.append(nums[i])
                i+=1
            else:
                temp.append(nums[j])
                j+=1
        while(i<=mid):
            temp.append(nums[i])
            i+=1
        while(j<=high):
            temp.append(nums[j])
            j+=1

        for i in range(low,high+1):
            nums[i]= temp[i-low]

        return cnt
    def mergesort(self,nums,low,high):
        if low>=high:
            return 0
        mid= (low+high)//2
        cnt=0
        cnt+=self.mergesort(nums,low,mid)
        cnt+=self.mergesort(nums,mid+1,high)
        cnt+=self.merge(nums,low,mid,high)
        return cnt

    def reversePairs(self, nums: List[int]) -> int:
        return self.mergesort(nums,0,len(nums)-1)
```