DSA Notes from Striver

Arrays

1) Kadane's algorithm

Problem Statement: Given an integer array arr, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

Approach:
* Traverse through the array.
* Have 2 variables, sum & maxsum
* Keep updating the sum as the sum of elements, and maxsum as the maximum of sum.
* But, when the sum < 0, declare the sum as 0.
* Return maxsum

---

2) Sort an array of 0's, 1's and 2's

Problem statement: Given an array consisting of only 0's, 1's & 2's. Write a program to

in-place sort the array without using inbuilt sort functions (Expected: Single pass-O(N) and constant space)

Eg: ① Input: nums = [2,0,2,1,1,0]

Output: [0,0,1,1,2,2]

② Input: [2,0,1]

Output: [0,1,2]

③ Input: [0]

output: [0]

Approach:-

Initialise low = 0, mid = 0, high = len(nums) - 1

```
while mid <= high:
    if nums[mid] == 0:
        nums[mid], nums[low] = nums[low], nums[mid]
        mid += 1
        low += 1
    elif nums[mid] == 1:
        mid += 1
```

```
elif nums[mid] == 2:
    nums[mid], nums[high] = nums[high], nums[mid]
    high -= 1
```

Eg. dry Run

Initially: [2, 0, 2, 1, 1, 0]

low, mid → 2 ; high → 0

mid = 0

[0, 0, 2, 1, 1, 2] → [0, 0, 2, 1, 1, 2]

low, mid → 0 ; high → 2

low, mid → 2 ; high → 1

[0, 0, 2, 1, 1, 2] ← [0, 0, 2, 1, 1, 2]

low, mid → 2 ; high → 1

[0, 0, 1, 1, 2, 2]

low, mid → 1 ; high → 1

[0, 0, 1, 1, 2, 2]

low, mid, high → 1

3) Stock buy & sell

Problem statement: You are given an array of prices where prices[i] is the price of a given stock on an ith day

You want to maximise your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock. Return the max profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

Examples

Example 1:

Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

Note: That buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

Example 2:

Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.

Approach :- * Take minPrice as a variable to store the minimum price

* Take maxProf as a variable to store the maximum Profit

* Traverse through the prices array, update minPrice and maxProf (as current price - minPrice)

* Return maxProf.

---

4) Merge overlapping subintervals

Problem Statement: Given an array of intervals, merge all the overlapping intervals and return an array of non-overlapping intervals.

Examples

Example 1:
Example 1:

Input: intervals=[[1,3],[2,6],[8,10],[15,18]]

Output: [[1,6],[8,10],[15,18]]

Explanation: Since intervals [1,3] and [2,6] are overlapping we can merge them to form [1,6]
intervals.

Example 2:

Input: [[1,4],[4,5]]

Output: [[1,5]]

Explanation: Since intervals [1,4] and [4,5] are overlapping we can merge them to form [1,5].

Approach :- First sort the array of intervals. Then, create an empty list of intervals. Then,

↘ res

```
i = 0
while i < n:
    j = i + 1
```

```
while j < n and intervals[i][1] >=
intervals[j][0]:
    intervals[i][1] = max(intervals[j][1],
    intervals[i][1])
    j += 1
res.append(intervals[i])
i = j
```

5) Merge 2 sorted arrays without extra space

Given two integer arrays **nums1** and **nums2**. Both arrays are sorted in **non-decreasing** order.

Merge both the arrays into a **single** array **sorted** in **non-decreasing** order.

- The **final** sorted array should be stored inside the array **nums1** and it should be done in-place.
- **nums1** has a length of **m + n**, where the first **m** elements denote the elements of **nums1** and rest are **0s**.
- **nums2** has a length of **n.**

Examples:

Input: nums1 = [-5, -2, 4, 5], nums2 = [-3, 1, 8]
Output: [-5, -3, -2, 1, 4, 5, 8]
Explanation: The merged array is: [-5, -3, -2, 1, 4, 5, 8], where [-5, -2, 4, 5] are from nums1 and [-3, 1, 8] are from nums2

Input: nums1 = [0, 2, 7, 8], nums2 = [-7, -3, -1]
Output: [-7, -3, -1, 0, 2, 7, 8]
Explanation: The merged array is: [-7, -3, -1, 0, 2, 7, 8], where [0, 2, 7, 8] are from nums1 and [-7, -3, -1] are from nums2

Approach :- Consider $i = m-1$, $j = 0$

while $i >= 0$ and $j < n$ and nums1[i] > nums2[j]:

nums1[i], nums2[j] = nums2[j], nums1[i]

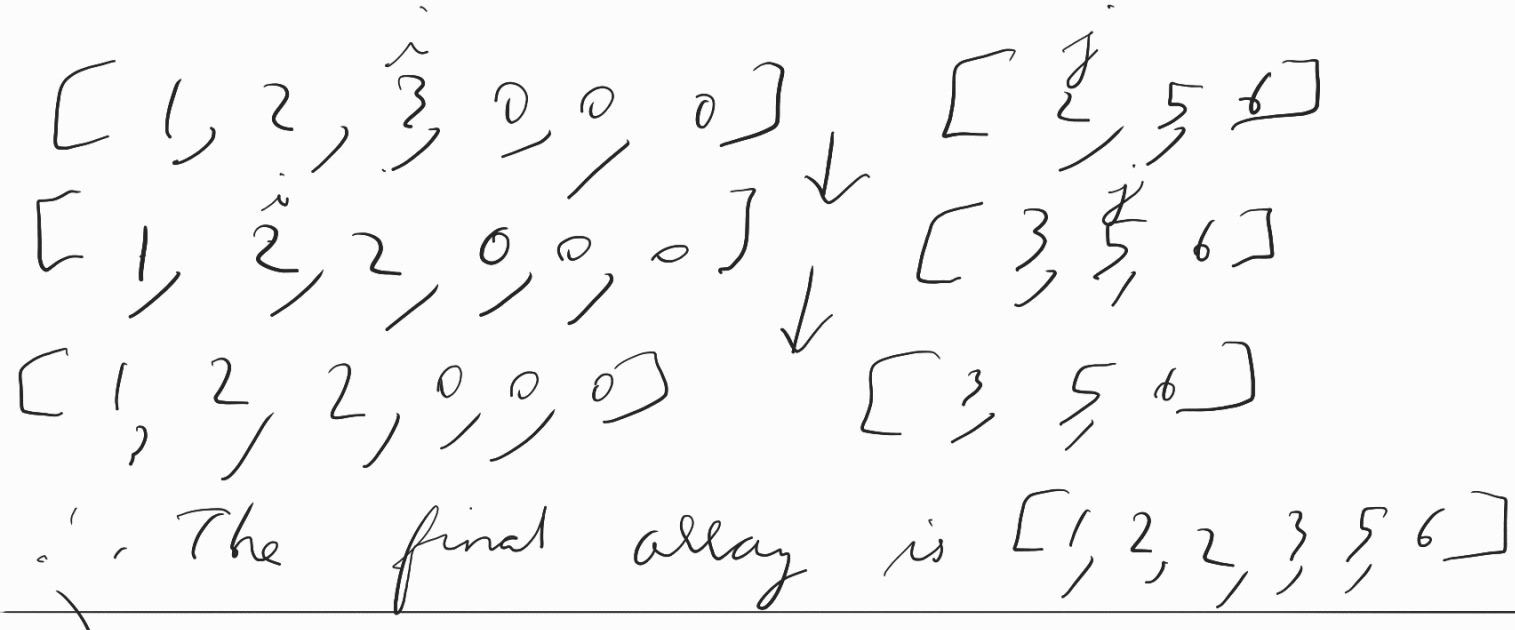
$i -= 1$

$j += 1$

Take nums1 as nums1[:m], and sort

2 arrays. Then join nums1 with nums2.

Eg:- nums1 = [1, 2, 3, 0, 0, 0], m = 3
nums2 = [2, 5, 6], n = 3



∴ The final array is [1, 2, 2, 3, 5, 6]

---

6) Search in a sorted 2D matrix

Problem Statement: You have been given a 2-D array 'mat' of size 'N x M' where 'N' and 'M' denote the number of rows and columns, respectively. The elements of each row are sorted in non-decreasing order. Moreover, the first element of a row is greater than the last element of the previous row (if it exists). You are given an integer ‘target’, and your task is to find if it exists in the given 'mat' or not.

Examples:

Example 1:

Input Format: N = 3, M = 4, target = 8,
mat[] =
1 2 3 4
5 6 7 8

9 10 11 12
Result: true
Explanation: The ‘target’ = 8 exists in the 'mat' at index (1, 3).
Example 2:
Input Format: N = 3, M = 3, target = 78,
mat[] =
1 2 4
6 7 8
9 10 34
Result: false
Explanation: The ‘target' = 78 does not exist in the 'mat'. Therefore in the output, we see 'false'.

Approach

* Convert 2D array into 1D array
* Perform binary search on the 1D array.

---

7) Unique Path

Problem Statement: Given a matrix **m X n**, count paths from left-top to the right bottom of a matrix with the constraints that from each cell you can either only move to the rightward direction or the downward direction.
Examples:
Example 1:

Input Format: m = 2, n= 2

Output: 2
Explanation: From the top left corner there are total 2 ways to reach the bottom right corner:

Step 1: Right ->> Down
Step 2: Down ->> Right

Example 2:

Input Format: m = 2, n= 3
Output: 3
Explanation: From the top left corner there is a total of 3 ways to reach the bottom right corner.

Step 1: Right ->> Right ->> Down
Step 2: Right ->> Down ->> Right
Step 3: Down ->> Right->> Right

Approach: Finding combinations using DP

---

8) Reverse Pairs

Problem Statement: Given an array of numbers, you need to return the count of reverse pairs. **Reverse Pairs** are those pairs where i<j and arr[i]>2*arr[j].

Examples
Example 1:
Input: N = 5, array[] = {1,3,2,3,1)
Output: 2

Explanation: The pairs are (3, 1) and (3, 1) as from both the pairs the condition arr[i] > 2*arr[j] is satisfied.

Example 2:
Input: N = 4, array[] = {3,2,1,4}
Output: 1
Explaination: There is only 1 pair (3 , 1) that satisfy the condition arr[i] > 2*arr[j]

Approach :- * This approach is similar to mergesort.

* Once we divide the array into subchunks, before merging, we have to call this function.

```
def countPairs(A, l, m, r):
    count = 0
    j = m+1
    for i in range(l, m+1):
        while j <= r and A[i] > 2*A[j]:
            j += 1
        count += (j - (m+1))
    return count
```

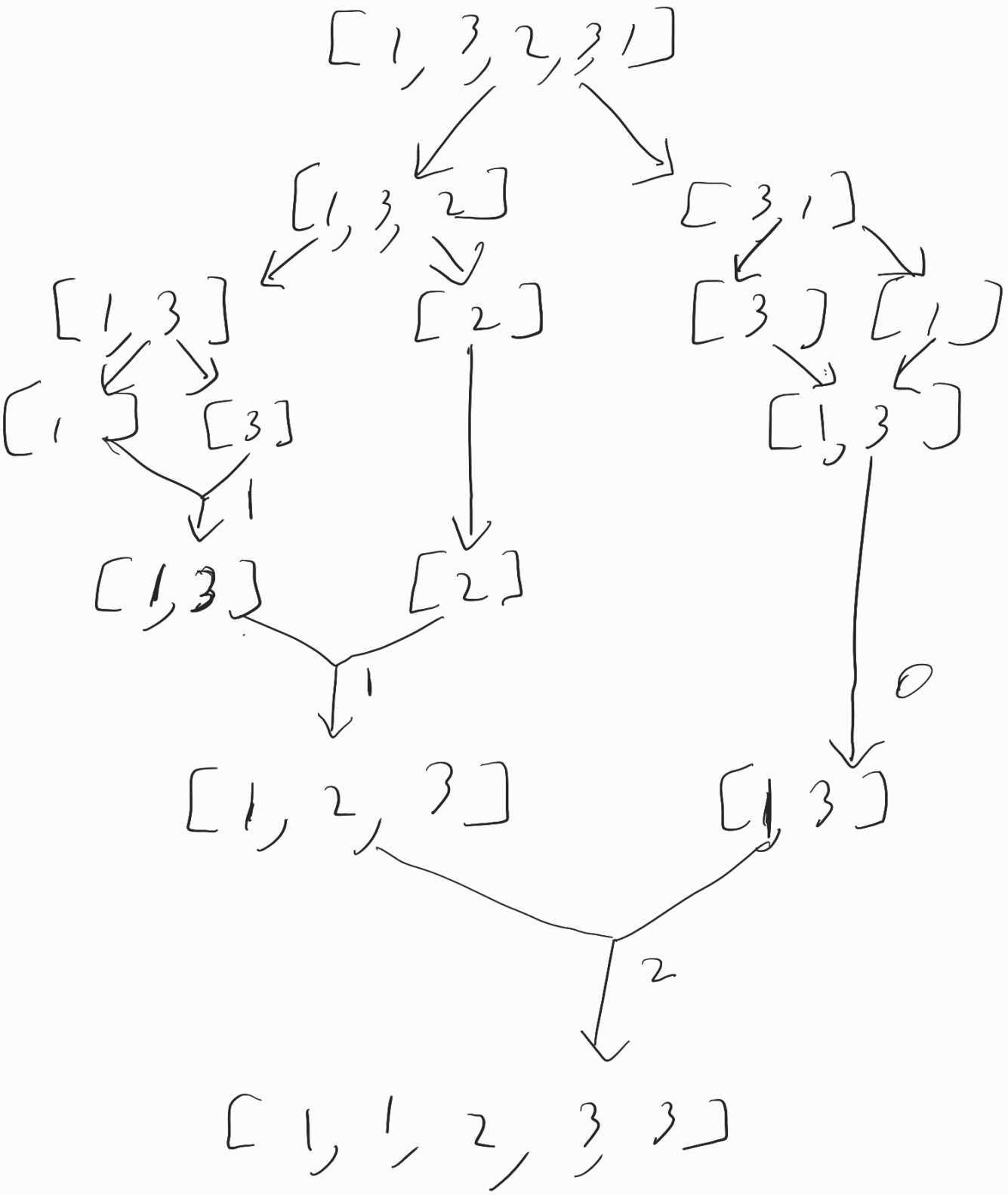
The main mergesort(A, l, r):

```
cnt = 0
if l < r:
    m = (l+r)//2
    cnt += mergesort(A, l, m)
```

```
cnt t = mergesort(A, m+1, r)
cnt t = mergesort(A, l, m, r)
merge(A, l, m, r)
returns cnt
```

Example dry run :-

A = [1, 3, 2, 3, 1]



∴ The final answer is 2.

---

9) 4-sum problem

Problem Statement: Given an array of N integers, your task is to find unique quads that add up to give a target value. In short, you need to return an array of all the unique quadruplets [arr[a], arr[b], arr[c], arr[d]] such that their sum is equal to a given target.

Note:
0 <= a, b, c, d < n
a, b, c, and d are distinct.
arr[a] + arr[b] + arr[c] + arr[d] == target

Examples

Example 1:
Input Format: arr[] = [1,0,-1,0,-2,2], target = 0
Result: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]
Explanation: We have to find unique quadruplets from the array such that the sum of those elements is equal to the target sum given that is 0. The result obtained is such that the sum of the quadruplets yields 0.

Example 2:
Input Format: arr[] = [4,3,3,4,4,2,1,2,1,1], target = 9
Result: [[1,1,3,4],[1,2,2,4],[1,2,3,3]]
Explanation: The sum of all the quadruplets is equal to the target i.e. 9.

Approach :

* We need 4 pointers, i, j, l, r. Sort the array.
* Traverse i from 0 to n-1
* If i > 0 & nums[i] = nums[i-1], then continue.
* Else, run j from i+1 to n.
* Again, if j > i+1 & nums[j] = nums[j-1], then continue
* Lates, initialise l as j+1 & r as n-1.
* While l < r, do the following instructions.

i) If the sum of the numbers at the 4 pointers is equal to the target, then increment l such that the next number is not the current number, and decrement r such that the previous number is not the current number.

ii) If the sum of the numbers at the 4 pointers is less than the target, increment l

iii) If the sum of the number at the 4 pointers is more than the target, decrement r.

Eg dry run:-

arr = [1, 0, -1, 0, -2, 2]

Sorted arr = [-2, -1, 0, 0, 1, 2]
0 1 2 3 4 5

i = 0

j = 1

l = 2, r = 5

sum = -2 - 1 + 0 + 2 = -1 (l++)

l = 3, r = 5

sum = -2 - 1 + 0 + 2 = -1 (l++)

l = 4, r = 5

sum = -2 - 1 + 1 + 2 = 0 ✓

j = 2

l = 3, r = 5

sum = -2 + 0 + 0 + 2 = 0 ✓

j = 3

l = 4, r = 5

sum = -2 + 0 + 1 + 2 = 1

i = 1

j = 2

l = 3, r = 5

sum = -1 + 0 + 0 + 2 = 1

l = 3, r = 4

sum = -1 + 0 + 0 + 1 = 0 ✓

j = 3

l = 4, r = 5

sum = -1 + 0 + 1 + 2 = 2

i = 2

$j = 3$

$k = 4, \; l = 5$

sum = $0 + 0 + 1 + 2 = 3$

∴ The total number of quadruplets are 3.

---

10) Longest Consecutive Sequence

Problem Statement: You are given an array of 'N' integers. You need to find the length of the longest sequence which contains the consecutive elements.

Examples:
Example 1: Input: [100, 200, 1, 3, 2, 4]
Output: 4
Explanation: The longest consecutive subsequence is 1, 2, 3, and 4.

Input: [3, 8, 5, 7, 6]
Output: 4
Explanation: The longest consecutive subsequence is 5, 6, 7, and 8.

Approach :- * First reduce the list of number into a set (by not having duplicate entries)

* Traverse through the elements in st.

* Check if the number one less than

the current element is not there in the set, then initialise the counter as 1. Then, increase the counter, if the next number is there in the set.

* Then after incrementing the counter, then update the longest number variable as the max of the current variable value and the counter value.

Eg: nums = [100, 4, 200, 1, 3, 2]

st = [100, 4, 200, 1, 3, 2]

i = 100
The next number does not exist.

i = 4
3 is there
i = 200.
The next number does not exist

i = 1 cnt = 1 i = 2 cnt = 2	i = 3 cnt = 3 i = 4 cnt = 4	4 is the longest length.

---

11) 3 sum

Problem Statement: Given an array of N integers, your task is to find unique triplets that add up to give a sum of zero. In short, you need to return an array of all the unique triplets [arr[a], arr[b], arr[c]] such that i!=j, j!=k, k!=i, and their sum is equal to zero.

Examples

Example 1:
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation: Out of all possible unique triplets possible, [-1,-1,2] and [-1,0,1] satisfy the condition of summing up to zero with i!=j!=k

Example 2:
Input: nums=[-1,0,1,0]
Output: Output: [[-1,0,1],[-1,1,0]]
Explanation: Out of all possible unique triplets possible, [-1,0,1] and [-1,1,0] satisfy the condition of summing up to zero with i!=j!=k

Approach:- * Sort the numbers

* Take 3 pointers, i, j, k.
* Traverse through i from 0 to n-1.
* Check if the previous number is not as same as the current number
* Take j as i+1 & k as n-1.
* While j < k, calculate the sum of

nums[i], nums[j], nums[k]

* If the sum is equal to 0, then append the triplet in to the answer list, then increment j till the current number is not equal to the previous number, and decrement k till the current number is not equal to the next number.

* If the sum is less than 0, then increment j

* If the sum is more than 0, then decrement k.

Eg Dry run.

nums = [-1, 0, 1, 2, -1, -4]

sorted nums array :- [-4, -1, -1, 0, 1, 2]
(indices 0, 1, 2, 3, 4, 5)

i = 0

j = 1, k = 5
sum = -4 - 1 + 2 = -3 (j += 1)
j = 2, k = 5
sum = -4 - 1 + 2 = -3 (j += 1)
j = 3, k = 5
sum = -4 + 0 + 2 = -2 (j += 1)
j = 4, k = 5

sum = -4 + 1 + 2 = -1

i = 1

j = 2, k = 5

sum = -1 - 1 + 2 = 0 ✓

j = 3, k = 4

sum = -1 + 0 + 4 = 3

i = 2

j = 3, k = 5

sum = -1 + 0 + 2 = 1 (k--)

j = 3, k = 4

sum = -1 + 0 + 1 = 0 ✓

∴ Anslist = [[-1, -1, 2], [-1, 0, 1]]

---

12) Remove duplicates from sorted array

Problem Statement: Given an integer array sorted in non-decreasing order, remove the duplicates in place such that each unique element appears only once. The relative order of the elements should be kept the same.

If there are k elements after removing the duplicates, then the first k elements of the array should hold the final result. It does not matter what you leave beyond the first k elements.

Note: Return k after placing the final result in the first k slots of the array.

Examples

Example 1:
Input: arr[1,1,2,2,2,3,3]
Output: arr[1,2,3,_,_,_,_]
Explanation: Total number of unique elements are 3, i.e[1,2,3] and Therefore return 3 after assigning [1,2,3] in the beginning of the array.

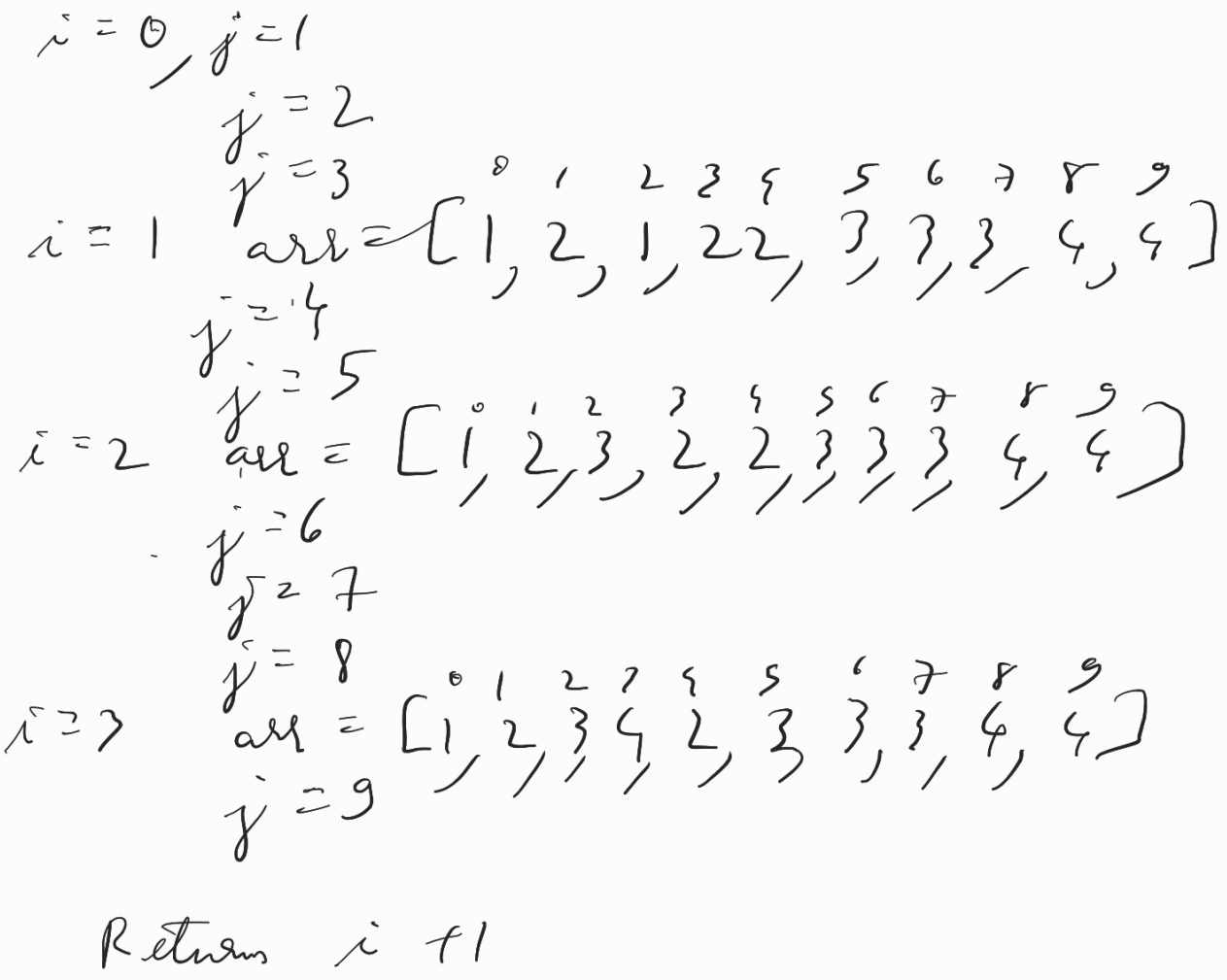
Example 2:
Input: arr[1,1,1,2,2,3,3,3,3,4,4]
Output: arr[1,2,3,4,_,_,_,_,_,_,_]
Explanation: Total number of unique elements are 4, i.e[1,2,3,4] and Therefore return 4 after assigning [1,2,3,4] in the beginning of the array.

Approach :- * Keep 2 pointers, i & j

* i = 0, j from 1 to n-1
* If nums[i] ≠ nums[j], then keep on incrementing i,
* After finding that nums[i] = nums[j] declare that nums[i] = nums[j]
* Return i+1

Ex dry run

arr = [1, 1, 1, 2, 2, 3, 3, 3, 4, 4]
0 1 2 3 4 5 6 7 8 9



---

13) Max consecutive ones

Problem Statement: Given an array that contains **only 1 and 0** return the count of **maximum consecutive** ones in the array.

Examples:

Example 1:
Input: prices = {1, 1, 0, 1, 1, 1}
Output: 3
Explanation: There are two consecutive 1's and three consecutive 1's in the array out of which maximum is 3.

Input: prices = {1, 0, 1, 1, 0, 1}

Output: 2
Explanation: There are two consecutive 1's in the array.

Approach :- * Declare a variable called maxi (to store the max number of ones)

* Traverse through the array.

* If the number is 1 then declare & initialise c as 0, and check if the next numbers are 1. If they are 1, then increment c.

* Finally, when the number is 0 again, update maxi as the max of c and prev max

Eg dry run

nums = [1, 1, 0, 1, 1, 1]
0 1 2 3 4 5

i = 0, maxi = 0
c = 0
c = 1
i = 1
c = 2
i = 2
maxi = 2
i = 3
c = 0

c = 1
i = 4
c = 2
i = 5
c = 3

maxi = 3

---

14) Largest subarray with K sum

Problem Statement: Given an array containing both positive and negative integers, we have to find the length of the longest subarray with the sum of all elements equal to zero.

Examples

Example 1: Input Format: N = 6, array[] = {9, -3, 3, -1, 6, -5} **Result**: 5 **Explanation**: The following subarrays sum to zero: {-3, 3} , {-1, 6, -5}, {-3, 3, -1, 6, -5} Since we require the length of the longest subarray, our answer is 5!

Example 2: Input Format: N = 8, array[] = {6, -2, 2, -8, 1, 7, 4, -10} **Result**: 8 Subarrays with sum 0 : {-2, 2}, {-8, 1, 7}, {-2, 2, -8, 1, 7}, {6, -2, 2, -8, 1, 7, 4, -10} Length of longest subarray = 8

Example 3: Input Format: N = 3, array[] = {1, 0, -5} **Result**: 1 Subarray : {0} Length of longest subarray = 1

Example 4: Input Format: N = 5, array[] = {1, 3, -5, 6, -2} **Result**: 0 Subarray: There is no subarray that sums to zero

Approach :- * Initialise sum as 0, Create a hashmap

* Traverse through the array

* sum t = arr [i]

* If the sum is 0, max length = i + 1
* Else,
i) If sum in hashmap, max length = max (max length, i - hash [sum]).
ii) Else, add the [sum, i] in to the hashmap

* Returns max length

---

15) Count number of subarrays with given xor k

Problem Statement: Given an array of integers A and an integer B. Find the total number of subarrays having bitwise XOR of all elements equal to k.

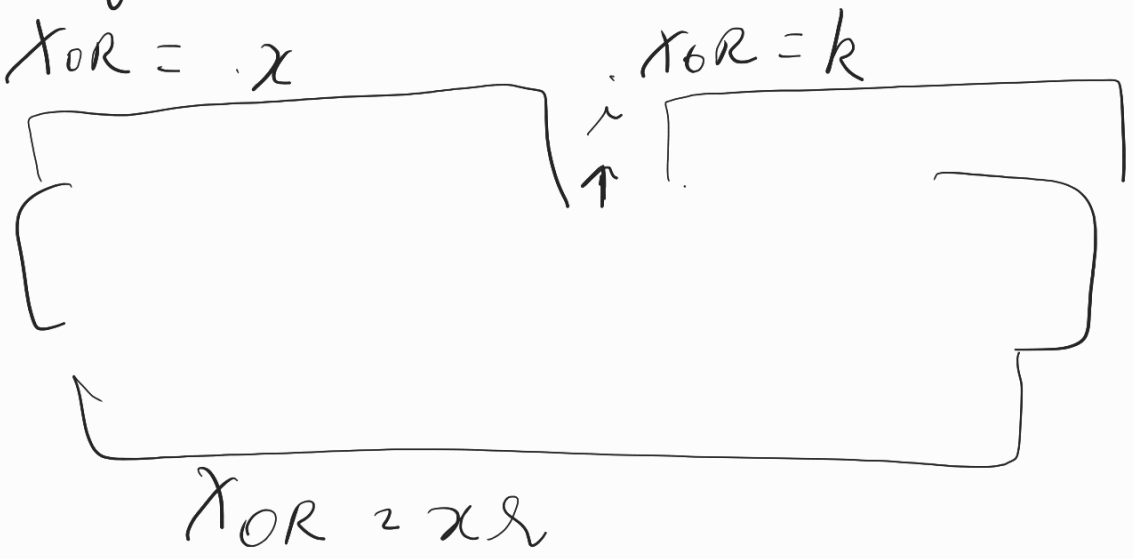
Examples

Example 1:
Input Format: A = [4, 2, 2, 6, 4] , k = 6
Result: 4
Explanation: The subarrays having XOR of their elements as 6 are [4, 2], [4, 2, 2, 6, 4], [2, 2, 6], [6]

Example 2:
Input Format: A = [5, 6, 7, 8, 9], k = 5
Result: 2
Explanation: The subarrays having XOR of their elements as 5 are

[5] and [5, 6, 7, 8, 9]

Approach :- * Imagine that there is an endpoint. The prefix XOR is xr, the prefix index of the elements before any index i is x and that after the index is k.



So, $xr = x \wedge k$

$xr \wedge k = x \wedge k \wedge k$

$\Rightarrow x = xr \wedge k$

* So, our approach should be that for k = 6, we need to see whether there is xr.

* We need to create a hashmap having xr as 0, and an element [xr : 1].

* As we traverse the array, we need to update xr as xr ^ arr[i].

* Later, we need to compute x as xr ^ k.

* If x already exists in the hash, count += hash[x]

* We have to add xr to the hash

* Finally, return the count

Eg dry run

arr = [4, 2, 2, 6, 4]

```
  0 1 0
  1 0 0
-------
  1 1 0
```

xr = 0, hash = {0:1}

i = 0

xr = 4, x = 4^6 = 2, hash = {0:1, 4:1}

i = 1

xr = 4^2 = 6, x = 6^6 = 0, cnt = 1,
hash = {0:1, 4:1, 6:1}

xr = 6^2 = 4, x = 4^6 = 2, cnt = 1
hash = {0:1, 4:2, 6:1}

xr = 4^6 = 2, x = 2^6 = 4, cnt = 1+2 = 3
hash = {0:1, 4:2, 6:1, 2:1}

xr = 2^4 = 6, x = 0, cnt = 3+1 = 4
hash = {0:1, 4:2, 6:1, 2:1}

∴ Final count = 4