INDEX

[Queston 1 == Find the duplicate number in an array 2](#_Toc114680606)

[Question 2 == find duplicates in an array2 3](#_Toc114680607)

[Question 3 == sort the arrays of 0,1,2 4](#_Toc114680608)

[Question 4 == find missing and repeating number 5](#_Toc114680609)

[Question 5 == find the maximum sum in a contiguous sub array 6](#_Toc114680610)

[Question 6 == Set matrix zero 7](#_Toc114680611)

[Question 7 == Pascal triangle 8](#_Toc114680612)

# Queston 1 == Find the duplicate number in an array

Given an array of integers nums containing n + 1 integers where each integer is in the range [1, n] inclusive.

There is only **one repeated number** in nums, return this repeated number.

You must solve the problem **without** modifying the array nums and uses only constant extra space.

**Solution**

* The given array is of size N + 1 and the elements are 1 to N so we can conclude that there will be a repeated number for sure
* We also see that they have given that only 1 repeated number exists. We should solve it without using any extra space and modifying array.

**Brute force** == compare each element with every other element using two loops and return the duplicate. TC = O(n2).

**Best approach** == **Linked list cycle method (Floyds hare and tortoise algorithm).**

* Take two pointers hare and tortoise initialized to 0, hare moves 2 steps at a time and tortoise moves one step.

Hare = nums[nums[hare]];

Tortoise = nums[tortoise];

* When they meet each other set the hare again to initial position and move it one step at a time.
* When they meet the 2nd time that will be the duplicate number.

TC = O(n) SC = O(1)

**Now what if there are more than one duplicate elements and you are allowed to modify the array**

**The hare and tortoise method will not work if there exists more the one duplicates.**

# Question 2 == find duplicates in an array2

**Given an array of n elements that contains elements from 0 to n-1, with any of these numbers appearing any number of times. Find these repeating numbers in O(n) and using only constant memory space.**

**Solution**

Here the array elements are from 0 to n-1 so there can be 0 duplicates also and, in the question, they have mentioned that there can be many duplicates also. The advantage is that we can modify our array

**Brute force** == compare each element with every other element using two loops and return the duplicate. TC = O(n2).

**2nd approach** == sort the given array and linearly traverse the array you will find the duplicate element which will be lying adjacently. TC = O(nlogn).

**Best approach** ==

* add the size of the array to the index of the element in the array which means if **the element is two add the size of the array at the 2nd index.**
* Then run a for loop again by dividing the elements by the size of the array if the quotient is greater than or equal to 2 we know that the number is repeated in the original array.so add the index i to the result.
* Check if the result is empty and add -1 if its empty.

# Question 3 == sort the arrays of 0,1,2

Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

**Solution**

**1st approach ==** we would try to sort the array but they have mentioned not to use the sort function in library.

**2nd approach ==** we will keep count variables for 0,1,2 and modify our given arrays with respect to counts value

But it will take 2 for loops means it will take 2n time practically speaking its equal to O(n) but still it takes 2 pases we need to minimize it.

**Best approach == Dutchs national flag algorithm**

**So in this algorithm the final condition should be**

Only 0’s on the left side of low

Only 1’s on the right side of high.

* Assign low and mid as 0 and high as arr.length – 1.
* If mid is 0 swap ( mid and low ) and increment both
* If mid is 1 just increment mid
* If mid is 2 swap (mid and high) and decrement high.
* Run the loop until mid crosses high.

# Question 4 == find missing and repeating number

**You are given a read only array of n integers from 1 to n.**

**Each integer appears exactly once except A which appears twice and B which is missing. Return A and B.**

**Solution**

We cannot modify the array so we cannot sort it.

**1st approach ==** run 2 for loops compare each element with every other element and find the repeating one then find the sum of the given array and subtract it by the repeating element and subtract the arising result with the sum of 0 to n. we will get the missing element. TC = O(n2) SC = O(1).

**2nd approach ==** use hash set u can keep a frequency array to find the missing and repeating element. TC = O(n) SC = O(n).

**Best approach ==**

Method 1 == This method is not suited for java cause only long is available.

* get the sum of 1 to n and arraysum , subtract to get eq1(x-y = 3).

Where x is missing and y is repeating.

* Get the squareSum of 1 to n and arraySquareSum , subtract them to get eq2 (x2 – y2 = 9).
* Now eq2 will be eq2 = eq2 / eq1 (bcoz x2 – y2 can be written as (x+y)(x-y),

Since eq2 = (x+y)(x-y), and x-y = eq1 , therefore eq2 = eq2/eq1).

* Now we got eq2 as x+y and eq1 as x-y so the sum of (eq2+eq1)/2 will be the missing values number.
* And eq2 – missing will get you the repeating numbers.

**Method 2 == Xoring.**

* First keep a integer xorResult initialized with arr[0] and keep on xoring with all the elements in the array.
* Then xor with 1 to n now we got the (x ^ y = ?)
* Now keep a setBit variable and extract the right most bit (MSB) of the variable xorResult.
* Now perform bitwise and for all the array elements with the set bit and if it is set put it to 1 bucket and xor it else put it to another bucket and xor it.
* Next perform bitwise and for all the integers 1 to n with set bit if they are set put it to same bucket and xor it if not put it to another bucket and xor it.
* Now we will get the missing and repeated number in those buckets so do a last check to decide which is repeating and which is missing.
* TC = O(n) ~= 5n SC = O(1).

# Question 5 == find the maximum sum in a contiguous sub array

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

**A subarray is a contiguous part of an array.**

**Solution**

The array also contains negative number and if there is no positive sum you should atleast give the minimum neagative sum

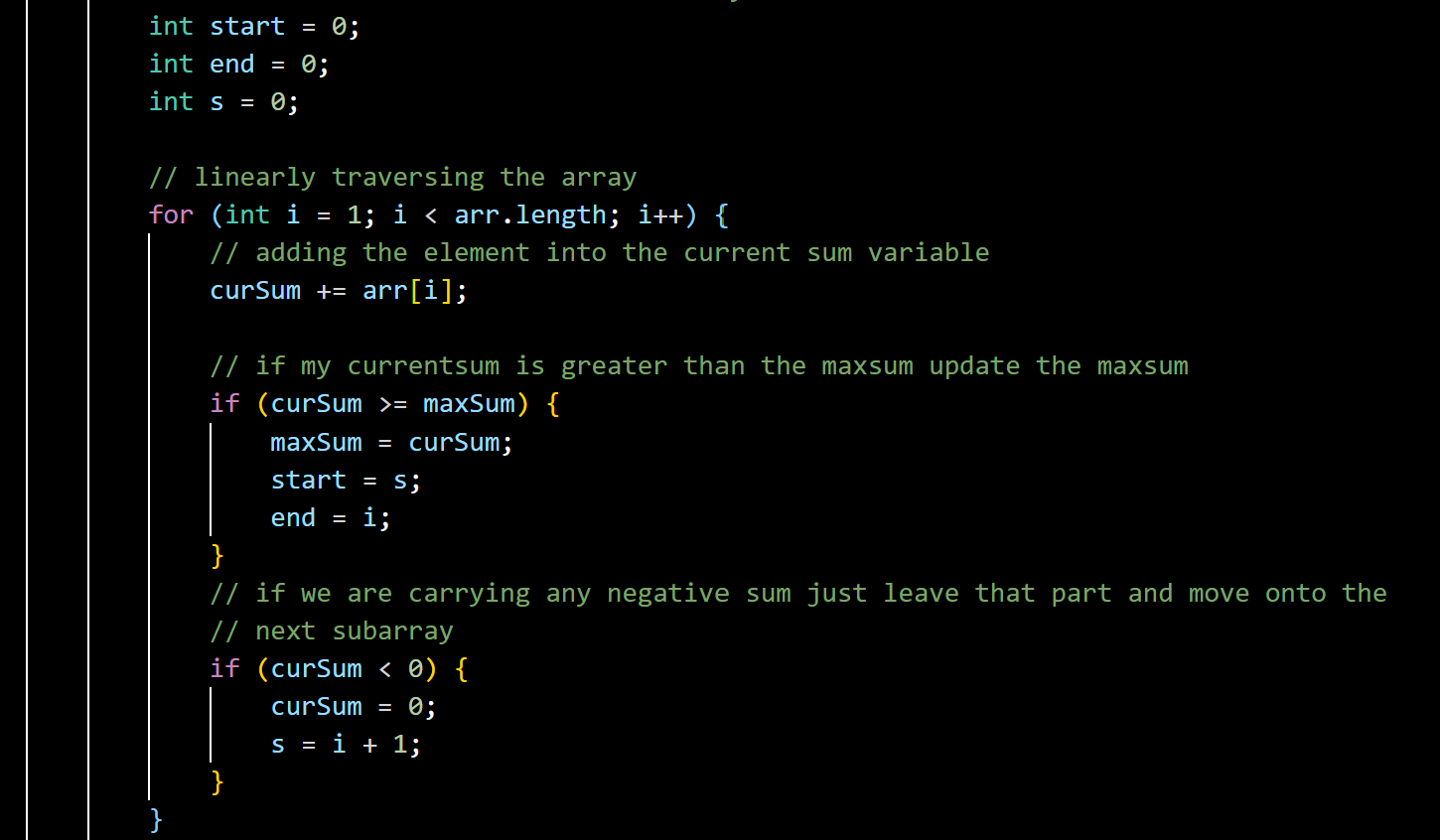
**1st approach** == use 3 for loops to get the sum of all the sub arrays and compare them . TC = O(n3).

**2nd approach** == use 2 for loops and git the maxsum in the sub arrays. TC = O(n2).

**Best approach == Kadane’s algorithm**

* Take a variable maxSum which is initialized with arr[0] because if the array only contains 1 element that should be the maximum sum.
* Take another variable curSum and add the elements (from 1st index) in the array one by one , if the sum is less than 0 , just update it to 0 because we don’t want to carry any negative sum we only want the maximum sum in the sub array.
* If we get the curSum greater than the maxSum update the maximum sum to the current sum.
* After the whole array is iterated we will have the maximum sum.
* TC = O(n).
* SC = O(1).

To return the start and end index of the subarray end will be equal to i when the maxSum gets updated and for start index keep another variable ‘s’ and add i to s whenever curSum is updated to 0 and start will be equal to s when my maxSum gets updated.



# Question 6 == Set matrix zero

**Given an m x n integer matrix matrix, if an element is 0, set its entire row and column to 0's.**

**You must do it in place.**

**Solution**

**Brute force ==** if all the elements in the matrix are on the positive side so we will linearly traverse the matrix and wherever we find zero we will make that whole column and row as a negative number but not at the place where it was already zero, then we will again traverse the matrix and make the negative elements as zero. TC = ( N x M ) x ( N+M ) SC = O(1).

**2nd approach ==** take two dummy array for row and column and linearly traverse the array if you find a zero set the value 0 in both the dummy array at that row and column then after 1st pass, we will have our dummy arrays filled.

Now do a linear traversal of the matrix again and if any of the dummy array is 0 in that position make the array also as 0 at that position.

TC = O(M x N) ~= (M x N) + (M x N) SC = O(M + N)

**Best approach ==** make the 0th row and 0th column as the dummy array itself,

* Keep a col0 variable to get to know that does any of the 0th column element is zero, if it zero mark it in the col0 variable.
* Now traverse the array from the 1st column and If any of the position is zero make your 0th row and that particular column to 0 and make your 0th column and that particular row to 0.
* Which is (if matrix[i][j] == 0) -> matrix[0][j] = 0 and matrix[i][0] = 0.
* Now traverse the matrix from behind which upto 1st column means do a reverse traversal check if any of the 0th row and that column or 0th column and that row is zero if its zero make it also zero.
* Now for the zeroth column if the col0 variable is marked make the zeroth column as zero
* Remember to traverse from back because if you traverse from the beginning my dummy array will be altered so traversing from back will modify the matrix first then will reach our dummy array/
* TC = O(N x M) ~= 2(N x M) SC = O(1).

# Question 7 == Pascal triangle

There can be 3 modifications of this question

1st type == return upto nth row of the pascal triangle.

* Use two for loops in the first one add the rows and in the 2nd one we know that it will run for i times (since the number of elements in a row is equal to its row number in pascal triangle), add the elements using the 2nd for loop.
* Create a list of list of Integers add the rows and add 1 to 0th row and add 1 whenever j = 0 or j = i.
* For the rest of the elements just get the previous array list and add the j-1th index and jth index to get the element.
* TC = O(n2) SC = O(1) (since the returning element is not counted)

2nd type == return only the nth row of pascal triangle.

* This can be done in only O(n) time complexity if solved efficiently.
* Take the 1st element as 1 and keep on multiplying row number and decrement it and divide by column number and increment it
* If they give the row (not the index of the row) just use r-1 C c formula.
* TC = O(n) SC = O(1).

3rd type == return the mth row and nth column element of pascal triangle.

* Just use the nCr formula if they give the row and column just

use r-1 C c-1.

* TC = O(n) SC = O(1).