Striver's SDE Sheet.

25/6/21

Q1 solve array 0's, 1's, 2's without sorting algo.

My approach: count 0, 1, 2

add ~~these~~ using 3 ~~fo~~ while loops add first 0's, then 1's and finally 2's. (2 pass)

complexity time = O(n)

space = O(1) (Time: 0.85)

#2

another sorting (but obvious)

#3

Dutch National Flag algorithm

3 pointers = low, mid & high

Initialize = low = 0, mid = 0, high = n-1

[0 --- low-1] = 0

[high+1 --- n] = 2

→ move mid ptr untill it crosses high ptr.

if mid element is 0,

① swap a[mid], a[low].

② low++, mid++

(1 pass)

if mid element = 1 (Time 0.95)

① mid++

Time = O(n)

if mid element = 2 Space = O(1)

① swap a(mid), a(high)

② high--;

Q2. Find missing and repeat number.

my approach.

create another array a[n+1]

traverse given array and increment items of our array.

a[arr[i]]++;

missing = where a[i]=0 & rep = where a[i]=2.

$T = O(2N)$ $S = O(N)$

~~#2 Negation of index~~

nums are 1 to n.

$$\text{sum} = \frac{n \times (n+1)}{2} = S$$

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6} = S^2$$

$$S - (4 + 3 + 6 + 2 + 1 + 1) = 4$$
(5) − (11)

$$x - y = S \longrightarrow x - y = 4 \quad (1)$$

$$S^2 - (4^2 + 3^2 + 6^2 + 2^2 + 1^2 + 1^2) = 24$$

$$x^2 - y^2 = S^2 \longrightarrow x^2 - y^2 = 24 \quad (2)$$

$$(x+y)(x-y) = 24$$

$$x + y = 6$$

$$x = 5, \; y = 1.$$

$$\begin{aligned} x - y &= 4 \\ x + y &= 6 \\ \hline 2x &= 10 \\ x &= 5 \end{aligned}$$

x = missing no, y = repeating

summations might exceed.

approach #3 (using xor operations)

arr[] = {1, 5, 2, 3, 3}

$xr = 0$

$xr \wedge 1 = 1 \wedge 5 = 4 \wedge 2 = 6 \wedge 3 = 5 \wedge 3 = 6$

$6 \wedge 1 = 7 \wedge 2 = 5 \wedge 3 = 6 \wedge 4 = 2 \wedge 5 = 7$

$p = 0.$

3	set	4	not set
3		2	
5		3	
1		2	
rep		set	

$$\begin{array}{r} 100 \\ 100 \\ \hline 100 \end{array} \quad p = 2$$

miss

rep

Date : Page :

Q2

Merge 2 sorted arrays without extra space

#1 with extra space.

→ make array of size $(a_1 + a_2)$

→ add both array and sort them

→ then fill first array and 2nd array

time complexity $O(N \log N) + O(N) + O(N)$

(sort) adding to new array getting elements back to array

#2 without extra space

→ linearly traversing array a_1 and also maintaining pts at smallest element of a_2

→ if $a_2[pts] < a_1[i]$, we swap both the values and sort the 2nd array back

→ like this we'll do until we've done array1 traversing

Time = $O(n \times m)$

Space = $O(1)$

3

GAP

→ Initially takes gap = $\frac{(m+n)}{2}$

1	4	7	8	10

2	3	9

gap = $\frac{5}{2} = 4$

↑ ← gap = 4 →

1	2	3	8	10

4	7	9

gap = $\frac{4}{2} = 2$

↑ ← 2 → ↑

1	2	3	4	7

8	10	9

gap = $\frac{2}{2} = 1$

↑ ← 1 → ↑

end condition

1	2	3	4	7

8	9	10

// Ans

26/6/21

Date : Page :

Q4 Kadane's algo
find maximum subarray sum

#1 brute force approach (O(N³))

#2 Kadane's algo
sum = 0 maxi = a[0]
linear traversal
~~if (sum < 0) → sum = 0~~
maxi = max(sum, maxi) sum < 0 → sum = 0
→ if sum is positive, we don't make value of sum as 0

Time complexity = O(N)

Q5. Merge Intervals
→ ask whether intervals are sorted
#1 brute force approach
→ linearly iterate and check for collisions
→ sort array if not sorted

O(NlogN) + O(N²)

#2

1,3	2,6	8,10	8,9	9,11	15,18	2,4	16,17

1,3	2,4	2,6	8,9	8,10	9,11	15,18	16,17

temp [1, ~~3~~ 6] [8, ~~9~~ 10] [15,18]

res [1, 6] [8, 11] [15,18] Ans

check if next vector's initial val <= final val of 1st vec
if so then merge otherwise push temp vector in result.

Date : Page

Q6. Find duplicate number

~~# there are n distinct numbers and one number repeating from them~~

my approach (applicable when one number is repeating twice)

→ res = 0, xor all the array elements and all the numbers from 1 to n; result is repeating no.

$T = O(N)$
$S = O(1)$

my approach to actual problem

→ maintain bool array of n+1 elements initializing all to false (hashing)

→ iterating over input array.

→ if the ~~index of~~ bool array at index current value at the pointer of input array is false, then we will flip it

→ if its true, then we'll return the value

$T = O(N)$
$S = O(N)$

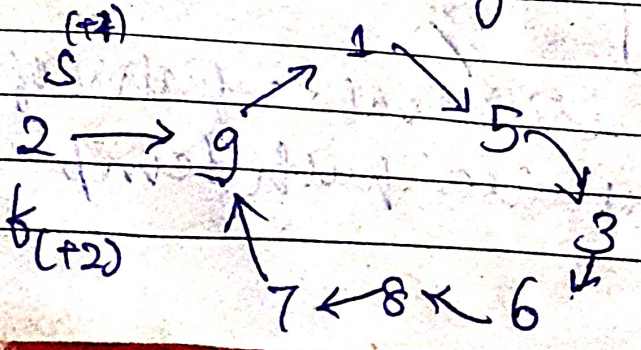
#2. sort the array and check whether any one element is equal to element after it

Time = $O(N \log N)$
Space = $O(1)$

1) Distort the array.

~~#3 hashing~~

#3 linked list cycle method.



a → next ⇒ arr[a]

we maintain slow & fast ptr initially at a[0]

→ fast moves by 2 and slow moves by 1
→ this will be done untill they both collide

Date: Page:

→ one they colide, fast ptr is being sent to a[0] and then both are being set to move untill they both collide
→ colliding num will be ans.

code:

```
slow = a[0];
fast = a[0];
do {
  slow = a[slow];
  fast = a[fast];
} while (slow != fast);
fast = a[0];
while (slow != fast) {
  slow = a[slow];
  fast = a[fast];
}
return fast;
```

27/6/21 set matrix zeros.

if any cell has ~~ma~~ element = 0, makes all the row and column elements containing cell 0.

my approch (brute force approach)

making 2 vectors row & col.

→ traversing matrix once and storing row and col where a[i][j] = 0.

→ making all the row/~~where~~ elements 0 as zero ~~where~~ where i is in row vector

→ same for col.

$O(N^2) + O(N^2) + O(N^2)$

$TC = O(N^2)$

#approach 2.

→ take 2 dummy arrays

→ if $a[i][j] == 0$ $d[i] = 0$ $d[j] = 0$

and afterwards we normally traverse main array, check whether any of values of i & j are 0 in both dummy array, if it is so, then we make element in main array as 0.

$O(M \times n)$

#

Q8. Pascal triangle

my approch.

→ create 2D vector a

→ iterate nested loops $i = 0$ to rows $- 1$ and $j = 0$ to i.

→ create a new vector temp.

→ if $j == 0 \,||\, j == i$, then push 1

else push $a[i-1][j-1] + a[i-1][j]$

→ push temp to the a. $O(N^2)$ $O(N^2)$

~~create~~ find pascal triangle value without creating whole vector

formula ${}^{R-1}C_{C-1}$ $O(N)$ $O(1)$

find a particular row.

find ${}^nC_1$ ${}^nC_2$ ${}^nC_3$ ... ${}^nC_1$

smart way ⇒

```
num = 1
for (int i = 1; i < c ; i++)
    num *= (n - i)
    num /= (i + 1)
```

Date : Page :

Q9. Next permutation

→ inbuilt function: next permutation.

approach 1 : generate all possible combos, linearly iterate through list of all permutation. next to it is an example

approach 2 {
→ linearly iterate from back
→ ~~check~~ find $a[i] < a[i+1]$
→ do linear traversal again ~~and find~~ $a[i] >$ ~~a[i]~~ and find j such that $a[j] > a[i]$
→ swap $a[i]$ & $a[j]$
→ reverse array from $a[i+1:]$.

15432
15234

intuition ~~1 3 5 4 2~~ 13542 ~~14532~~ 14532 ~~13542~~ ~~13245~~

Ans 1 4 2 3 5

Time complexity $O(n) + O(n) + O(n)$
$= O(n)$

Space $= O(1)$

29th June 2021

Q10. count inversion

approach 1 (Brute force)
Time $O(n^2)$ space $O(1)$

approach 2 (Merge sort)
→ modifies the array
Time $O(n \log n)$ space $O(N)$

Date : Page :

Q11. Best time to buy & sell

→ buy & sell on particular day for max profit

[7 1 5 3 6 4]

(i) (j)

j →

→

TLE

brute force approach

Time = $O(N^2)$

Ans 6 − 1 = 5

#2. [7 1 5 3 6 4]

mini = 1

pr = 0 → 4 → 3

min remains same thus pr = 5 − 1 = 4

~~for~~ ~~max~~ pr = 0, mini = INT_MAX;

for(i = 0 to n)

mini = min(mini, prices[i]);

pr = max(pr, prices[i] − mini);

Q12. Rotate matrix.

#1 add n^{th} ~~column~~ row to $(n-r)^{th}$ column of new matrix

T = $O(N^2)$

S = $O(N^2)$

#2 ① Transpose matrix

② reverse each row

T = $O(N^2)$

S = $O(1)$

01/07/21

Q13. Search in 2D matrix.

Leetcode problem that.

→ given 2D matrix & every row is sorted.

→ as well as given target value.

Approach 1

linear traversal in 2D matrix.

~~Time = O(m×n) space = O(1)~~

Approach 2

- traverse each ~~row~~ row and perform binary search on each column element

Time O(N log M)

approach 3 (optimal)

- place ptr at last element of first row
- move left j--
- move down i++
- if ptr moves out of bound, element is not present.

GFG problem

- all the rows as well as columns are sorted
- they might overlap their range.

approach 4

- reshape 2D array in 1D array, it will be sorted (but it will be having more space)
- if any element is found at k ~~next~~ index
- row = $\frac{k}{m}$ & column = k % m
- binary search can be performed ~~at~~ this case

  TC → $\log_2(n \times m)$ SC → ~~O(1)~~

- will find binary search using only indices

code →

```
if (mt.length==0) return false;
int n = mt.length;
int m = mt[0].length;
int l = 0
int h = (n*m) - 1
while (l<=h){
    int mid = (l + (h-l)/2);
    if (mt[mid/m][mid%m] == target) {
        return true;
    if (mt[mid/m][mid%m] < target{
        l = mid+1;
    else
        h = mid-1.
    }
}
return false.
```

$\log(m*n)$

Q.14. Pow(x, n)

#approach 1 brute force approach.
if $n<0$, we find $\frac{1}{x^n}$.

TC = O(N)
SC = O(1)

#approach 2
if $(n\%2==0) \rightarrow x^n = (x \times x)^{n/2}$
else ans = ans × $(x)^{n-1}$

TC = $\log_2 n$

Binary exponentiation

Q.15. Majority element
→ number in array that comes more than n/2 times

#Approach 1 brute force approach $O(n^2)$

Approach 2 use hashmap. $O(n)$ or $O(n\log n)$

Approach 3.

Most voting algo.

~~count = 0~~

el = 0.

linearly iterate over array

	7	7	5	7	5	1	5	7	5	5	7	7	5	5	5	5
cnt	1	2	1	2	1	0	1	0	1	2	1	0	1	2	3	4
elem	7	7	7	7	7	7	5	5	5	5	5	5	5	5	5	5 → Ans.

initial	step	
num = 0 count = 0	if a[i] != num if count = 0 → num = a[i] & count = 1 else count -- else count ++	only applied to element with count more than n/2

Q6 Majority element 2.

→ Same as previous problem but occurs more than n/3.

Approach 1

Brute force approach. time = $O(n^2)$

space = $O(1)$

Approach 2

Hashmap : of number & count. time = $O(n)$ or $O(n \log n)$

space = $O(n)$ → optimization

Approach 3.

Boyer Moore voting algo. (at max we can have least 2 majority elements)

	1	1	1	3	3	2	2	2
num 1 = -1	1	1	1	1	1	1	1	1
num 2 = -1	-1	-1	-1	3	3	3	3	2
c1 = 0	1	2	3	3	3	2	1	1
c2 = 0.	-1	-1	-1	1	2	1	0	1

num 1 = 1, num 2 = 2

check count of both

if > n/3 call both majority.

~~017~~

2/07/21

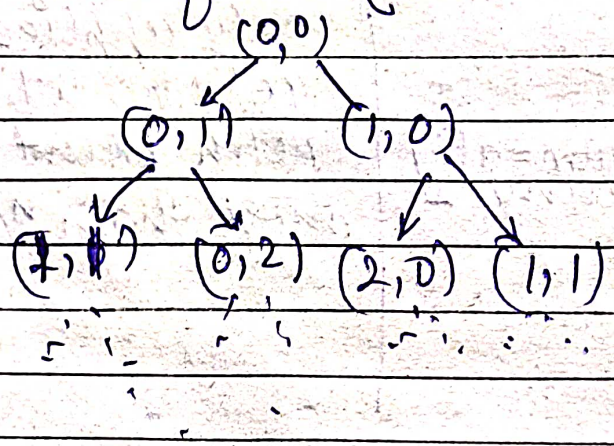
Date Page

017 Unique path.

No. of ways through which robot can reach finish

Approach 1

Brute force (recursive way)



Time complexity

$O(2^m * 2^n)$

Approach 2

Dynamic programming

	0	1	2
0	1	-1	-1
1	1	1	1

← Hash table

```
int countPath(int i, int j, int n, int m,
  vector<vector<int>>& dp){
  if (i==n-1 && j==n-1) return 1;
  if(i>=n || j>=m) return 0;
  if(dp[i][j]!=1) return dp[i][j]
  else return dp[i][j] = countPath(i+1, j, dp) +
                         countPath(i, j+1, dp);
```

$O(n \times m)$

my approach

Top down approach (no recursive calls)

Approach 4

combination

P.C → RRD
DRR
RDR

no. of places
$(m+n-2)$
$= (3+2-2) = 3$ Ans $^3C_2$

formula $^{m+n-2}C_{m-1}$ or $^{m+n-2}C_{n-1}$

Q18. Reverse ~~street~~ pair

number of important reverse pair in an given array.

(i, j) is important reverse pair if $i < j$ & $nums[i] \geq 2 * nums[j]$.

Approach 1 brute force approach

TC → $O(N^2)$
SC → $O(1)$

Approach 2 Merge sort technique.

→ split the problem into 2 halves.

→ we conquer and merge both halves.

40 25 19 12 9 6 2

40 25 19 12 9 6 2

40 25 19 12 9 6 2

40 25 19 12 9 6

→ take left half & right half and 2 ptrs are placed i, j in both array

→ contribution of left half: we will find number of elements in right half that satisfies following condition

$a[i] <= 2 * a[j]$

→ perform merge operation

→ in case we find elements that satisfies condition, we increase counter by 1.

Date : Page :

TC → O(N logN)

SC → O(N)