ARRAYS AND MATH

Array :- Collection of Homogenous data.
Difference b/w vector & array - vector is dynamic in size.

How to implement a vector using arrays?

We use proper arrays but once these arrays are filled, we create a new array of bigger size & copy all the elements from the prev size to the new array.

- Copying takes O(N) time. How do we deal with the extra time complexity here? Answer - Amortized complexity.

Amortized Complexity

Let's consider a dynamic array with 1 as its intial size. We increase □ ▨ the (array size+1) every time there ~~last~~ ~~placeholder gets filled~~ is no more space left in the array before insertion.

Insertion	Array	Time taken
1	1 ▨	1
2	1 2 □ □ ▨	2
3	1 2 3 □ ▨	1
4	1 2 3 4 □ □ □ □ ▨	4
5, 6, 7	1 2 3 4 5 6 7 □ ▨	1, 1, 1
8	1 2 3 4 5 6 7 8	8
9, 10, ... 15	9 10 11 12 13 14 15 □	1

Consider "Time taken' in the table above.

1, 2, 1, 4, 1, 1, 1, 8, 1 ..... n/2, 1, 1, 1 .....

In the total no. of insertions, (logn) insertions take total of 1 + 2 + 4 + 8 + ... n/2 time.

$1 + 2 + 4 + 8 + \ldots + n/2 = n - 1$ (Geometric progression)

The no. of insertions that could be finished in O(1) = n - logn.

∴ Total time needed $= (n-1) + (n - \log n)$
$= 2n - 1 - \log n$ → ignore as it's a -ve const

∴ The upper bound of insertion $<= O(2n-1)$
$\approx O(n)$ for n insertion

So, after considering amortization, each insertion roughly takes $O(1)$ only.

Do we have to only "Double" the size of the array?
No. We usually have a load factor c. C is determined by the language setting

Python	→	5, 2 ....
C++	→	1.8
Java	→	1.5

<u>Problem</u> Given an unsorted array of integers, find a pair (i, j) such that $|A[i] - A[j]| + |i-j|$ is maximized. $i \neq j$
→ Brute force can find the soln in $O(N^2)$.

<u>Thoughts</u>

$|A[i] - A[j]| + |i-j|$ // Assume $i > j$ always.
// Since we need +ve values or o

Drop the mods since we don't lose generality with $i \geq j$

Case 1: $A[i] > A[j]$

$$A[i] - A[j] + i - j = \underbrace{(A[i] + i) - (A[j] + j)}_{\text{This eq should be maximized}} \quad \text{— (1)}$$

This is possible only when
$A[i] + i$ is max & $A[j] + j$ is min

Case 2: $A[j] \leq A[i]$

$$|A[i] - A[j]| + |i-j| = A[j] - A[i] + i - j$$
$$= \underbrace{(A[j] - j)}_{\text{This should be max}} - \underbrace{(A[i] - i)}_{\text{This should be min.}} \quad \text{— (2)}$$

Find solutions for (1) & (2) & find max b/w 2 values.

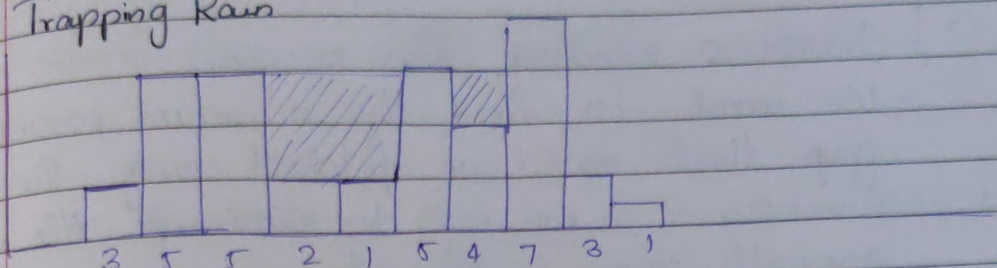
Sol GFG

$x = A[i] + i$

$y = A[i] - i$

return max (maxx - minY , maxY - minx)

Pbm Trapping Rain



Thought

Given array (assume) 3, 5, 2, 1, 5, 4, 7, 3, 1

~~LeftMax = 3, 5, 5, 5, 5, 5, 7, 3, 1~~

~~rightMax~~ rightMax LM = 0, 3, 5, 5, 5, 5, 5, 7, 7

RM =

	5	3	6	7	11	12	0	8
left max	5	5	5	6	7	11	12	12
right max	12	12	12	12	12	8	8	8

- Brute force → For every element, find max on left & right, subract the current height from min(l, r)
  → $O(N^2)$ TC , $O(1)$ SC
- Optimized → Prefix & Postfix array.

for	5	3	6	7	11	12	0	8
prefix	5	5	6	7	11	12	12	12
postfix	12	12	12	12	12	12	8	8

Now, run a for loop, & calculate

rainTrapped += min(postfix[i], prefix[i]) - curr

→ O(N) TC, O(N) SC

Pbm Given an unsorted array, find the maximum difference(gap) b/w 2 consecutive integers IF THE SAME ARRAY WAS SORTED.

Sol Bruteforce → Sort & find (O(NLOGN))

Optimized approach →

- Consider 2 numbers max & min.
- We want to find the minimum possible Max gap that could be fetched using the two numbers ie, we need to minimise the max gap with given n elements.

$$\text{min} \underset{1}{\underline{\quad\quad}} a \underset{2}{\underline{\quad\quad}} b \underset{3}{\underline{\quad}} c \underset{4}{\underline{\quad\quad}} \text{max} \quad [4\ \text{Gaps}]$$

- We can observe that,

$$\text{minMaxGap}(g) = \left\lfloor \left(\frac{\text{max} - \text{min}}{n-1} \right) \right\rfloor \quad // \text{ Ceil. Not floor}$$

- Create buckets with this gap from min all the way till max.
- Now, bucket elements.
- For each consecutive buckets, take difference b/w max of the left & min of right, calculate max.
- However, if all the buckets have exactly one, then ans is $\left\lfloor \frac{\text{max} - \text{min}}{n-1} \right\rfloor$

→ O(N) TC, O(N) SC

NOTE :

Given a particular number x, it should fall in $\left\lfloor \frac{n - \text{min}}{g} \right\rfloor$ bucket.

Example :-

$$\frac{3-1}{3} = \frac{2}{3} = 0$$

$$\frac{8}{2} = 2 \quad \frac{0}{2} = 1$$

$$\frac{9-1}{3} = \frac{8}{3} = 2$$

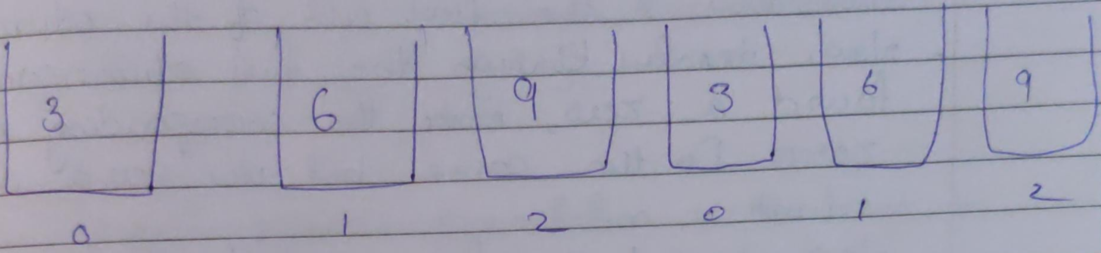
[3, 6, 9, 1]

max = 9 , min = 1

$g = \lceil 8/3 \rceil = ~~2~~\ 3$

minBucket = | 3 | 6 | MAX | maxBucket = | 0 | 1 | MIN |

Min Buckets   Max Buckets



minb	3	6	9
maxb	3	6	9

Homework

Pblm **Set Matrix Zeroes**

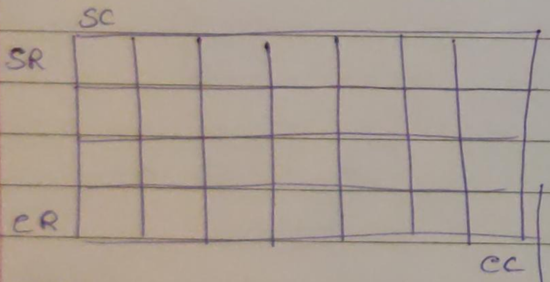
Sol
- first Step is to identify if there are any zeroes already present in the first row & first column.

  Boolean frow = false / true;
  Boolean fcol = false / true;
- Now, iterate through the rest of the elements in rows 1 → n & cols 1 → n.

  If you encounter a zero, mark the first cell of that row & the first cell of the column zero.
- Now, iterate through the first column and if you found a zero, make the corresponding row cells zeroes. Do the same but vice versa with first column.
- Now, based on frow & fcol, decide whether or not to mark every element in them zeroes.

Pblm **Spiral Matrix I**

Sol



- Set startrow, startcolumn, endrow & endcolumn

```
while (SC <= eC && SR <= eR)
  - i: SC → er
        print m[SR][i]
    SC++;
  - i: SR → eR
        print m[i] [eC]
    SR++;
  - if (SR <= eR)
        i : eC → SC :
            print m [eR][i]
  - if (SC <= eC):
        i : eR → SR
            print m [i] [SR]
```

Pblm **Find Next Permutation**

Sol **Approach**

a b c d e f k j i h g

- Find the first decreasing element from the right
- Find the next bigger element to the one found above to its right.
- Swap them both.

a b c d e g k j i h f

reverse this (sort this)

a b c d e g f h i j k

Note

The largest possible permutation is the one in descending order.

The next permutation in lexicographic order would be the sorted ascending order.

ex:- The next permutation of 321 would be 123