Array
Medium

__/__/__

BF (1) 2 sum problem

BF - Basically you keep a i as the current element from the outerloop then add a i to all the elements in the array using the inner loop then mark i again add all ele from i→n (j=i→N)
- then when you get a sum that equals k return its indeces.

Better - Basically you create a map and using a for loop iterate through all the elements
- now check if there is an element in the map that when you add that ele to current ele youll get k (sub = k - currentele)
- if that exists in the map that means you found a pair so return that elements index & current ele index.

OPT - basically you sort the given^(array) and keep two pointers at start & end
- now add the two pointers & if the sum = k then return Yes
- if sum > k you reduce the sum by decreasing right pointer value (right--)
- if sum < k you increase the sum by increasing the right pointer element in the sorted array (left++)

__/__/____

② Sort an array of 0, 1, 2

#BF basically you keep a pointer at the start of the array & a counter to count only till 2 (using while) now iterate through the array & if the current element is equal to the counter then swap it with i & move i you basically start filling up the array starting from 0 → 2

#OPT Dutch national flag algorithm
basically you keep three pointers (low → start of array) (mid → current ele) (high → last of array) so you move/swap the 0s to start of the array & 2s to end of the array by the end of this the 1s are swapped to mid by which the array will be sorted.

③ majority element > n/2

#BF basically for every element you check the whole array to see how many times an element is present once you get an element that's present more than n/2 times then return that element.

__/__/___

Better — just store all elements in a map with its values then just iterate through the map & return the element with value greater than n/2

OPT — keep a candidate & a count increment the candidates count as long as its the same candidate once you get a different cdt you start decrementing the count until it reaches 0 now change the candidate & start the counting again and now the final candidate whos value doesnt drop down to 0 will most likely be the majority element

- now verify that final candidate by getting its total count in the array if its greter than n/2 then returnn that cdt orelse -1.

__/__/____

④ Maxim Sum of a subarray

#BF basically you iterate through the array using two loop i outer loop to & j inner loop so for every i element runs j from i -> n adding all element and at each sum check if the currentsum is bigger or the previous sum is bigger ~~of~~ which is stored in maxi then finally return maxi which has the highest sum of a subarray.

#OPT basically you iterate from the start of the array till the end adding all the elements and while adding if you get a sum that is greater than a previous stored sum then you store the currentsum into maxi but if you get a sum i.e less than 0 then you reset the sum to 0 ~~since~~ and start a new subarray cuz if you keep adding negative numbers it brings down the actual values of future subarrays.
then finally return maxi.

__/__/____

⑤ Stock buying & selling

BF just keep two loops & for every ith element run j = i → N subtracting selling price(a[j]) with cost price(a[i]) which gives the profit & keep track of maximum profit throughout & return the highest profit.

OPT iterate through the array and keep track of the minimum price so far and subtract the current element with the minimum encountered so far the answer gives the profit & keep track of maximum profit & return it.

(in expand)

__/__/____

⑥ Rearrange the array with positives & negatives alternating

#BF Basically just iterate through an array while iterating if the current element is a positive number then just store it in ~~(positive)~~ even indexes of temp array if the current number is negative then store in odd places then just return the temp array. (works only when Pos & negs are equal)

#OPT- basically just create two arrays ~~to~~ ~~(be)~~ read separate all positives to one array & negs to the then iterate through the array & create 2 seperate arrays for both

- now keep track of both Pos & neg array as long as there are same number of elements copy these elements to alternatively to a new array.
- then when one of the array is over that is when ~~pos number becomes~~ pos nos count crosses neg nos count or vice versa you just put in the remaining positive or neg elements whichever is more at the end of the array.

__/__/____

⑦ Next Permutation

BF - basically just generate all possible permutations in order
- Linear search throughthe permutations to find the current given permutation
- then when you find the current permutation then the next permutation will be the answer.

Better- we can just use in built function next_permutation which returns the next permutation

Optimal- basically all youre doing is finding the next smallest number
- so just first find a dip point i.e the first decreasing number encounterd from the right.
- again iterate from right to find a greater element than dip element & swap the two
- now from the dip point to till the end of the array its all higher numbers just reverse so itll give the next permutation

__/__/____

(8) Leaders in an array
an element that's greater than all elements on its right

OPT just start iterating from the back of the array comparing each element to max i if the current ele is bigger than max i that means current ele is bigger than all elements to its right
(this prints from back)

OPT for the above logic store all max is in a vector & then print the vector from reverse so it prints the array of leaders in correct order

and using vector here shows dynamic size allocation

ex: 10, 22, 12, 3, 0, 6 leaders: 6, 12, 22
0 1 2 3 4 5

if you declare array of size N=6 and store only 3 elements in arr[n] for unoccupied spaces it might print garbage values or 0.

but in vector dynamic sizing

__/__/____

⑨ Consecutive sequence of numbers

#BF Basically for every ith element (keeping it as start of the sequence) you iterate through the whole array with inner loop to find the next element if you find the next element you keep that as the current element & find its next number while increasing count & also iterating through the array multiple times
and then at last you just check for the maximum sequence by comparing currentCount & maxi after every i loop.

#Better Basically you sort the whole array now start iterating through the array (1 -> N) & check
- if previousEle + 1 = currentElement - if yes then increase count (this step also ignores duplicates)
- else if the currentEle != prevEle (when the current number is a new number but also not a duplicate) you get the current count & then reset the count.

then one final check for maxi outside the loop

__/__/____

#OPT - I sort all elements into a set
- Iterate through the vector
- for every element you check if there is a previous element if Yes then that ele cant be start of the sequence but if No then that might be a start of seawence
- So from that element you start checking for next element if you find the next element you update count & current element & then repeat the same
- update maxi after every seawence end.

#OPT - Insert all ele into set
- iterate through the vector
- look if current num ele - 1 (i.e previous num) doesnt exist in the set then that means the current num could be start of new seawence so update current num & length.
- now from that current num look for next elements in the set while updating current num & length until next num isnt found
- & keep track of the longest seawence & return the longest

__/__/____

(10) Set matrix zeros.

#BF - iterate through the matrix
- while when you get a 0 mark its row & column elements with -1 by calling row & col mark functions
- now iterate through the whole matrix & convert all -1s to 0s

#Bet - keep 2 tracker array of size n & m to mark row & col no with 0
- iterate through the matrix if current ele is 0 mark it row & col in tracker array
- now iterate again & whichever column or row is marked 1 then set that whole row & col to 0

#OPT - - just use 0th row & col for tracker
- iterate through the matrix & mark in 0th row or col when current ele is 0
- now iterate from 1,1 to end excluding 0th row & col if you get a non 0 element & if its row or col is marked then set that ele to 0
- now for 0th row & col check if it needs to 0d then set it accordingly.

(1) Rotate matrix 90°

#BF - Basically just iterate through the matrix and for each element place it in the ans matrix like for every i keeping column constant youre changing rows.

- so the formula ans[j][n-1-i] = mat[i][j]
- then just print out the new matrix

#OP - for the given matrix get the transpose of the matrix just by iterating the lowerhalf

- then reverse each row of the transposed matrix.

__/__/____

A2Z

Arrays Medium

(13) Print matrix in spiral way

- Print top row then move top boundary down (++)
- print right column then move right boundary left inside (--)

#OP - print bottom row (if within bounds) then move bottom boundary inside (--)

- Print left column (if within bounds) then move the left boundary in

same process goes on till all ele are printed.

** again

(14) basically we iterate through the array adding all ele & checking presum & count as sub array only from the presums

#OP