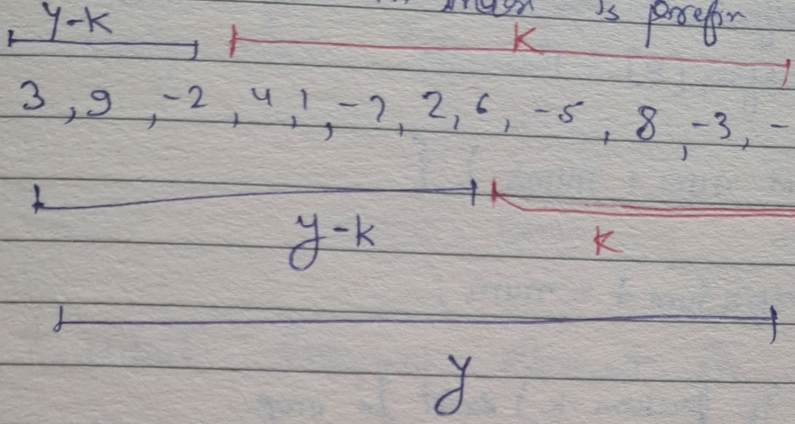
No. of Subarray with Sum = k

Without Brute force

Raj
Page:
Date: / /

Q. Count number of subarray with sum zero

- Prefix sum i : Sum of all element from starting to 'i' th index is prefix sum till 'i'



eg: 3, 9, -2, 4, 1, -7, 2, 6, -5, 8, -3, -7, 6, 2, 1 for k=5

1) Agar PS = y and at 2 places we get sum of (y-k)
∴ Pr total we will have 2 subarray of sum = k

2) Make Prefix sum array

for every PS = y
check for prev elem as y-k

all possible subarray bt them

count = 2
all possible subarrays

3, 9, -2, 4, 1, -7, 2, 6, -5, 8, -3, -7, 6, 2, 1

PS 0, 3, 12, 10, 14, 15, 8, 10, 16, 11, 19, 16, 9, 15, 17, 18

C: 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1

y-k

count of each element of prefix array

- 4, 1
  9, -2; 4, 1, -7
  1, -7, 2, 6, -5, 8
  8, -3

4, 1, -7 6
6, -5, 8 6
-7, 4, 1, -7 ... 6

CODE

Page: Raj
Date: / /

```
int subarraySumK(vector<int>& nums, int k)
{
    unordered_map<int, int> prefixSumFreq;
    int prefixSum = 0, count = 0;
    prefixSumFreq[0] = 1;

    for(int num : nums) {

        prefixSum += num;

        // check if (presum - k) exist in map

        if(prefixSumFreq.find(prefixSum - k) != prefixSumFreq.end())
        {
            count += prefixSumFreq[prefixSum - k]
```

↳ Add count of such prefix sum

```
        // store current prefix sum in map

        prefixSumFreq[prefixSum]++;
    }

    return count;
```

In ARRAY, find min. Sum of K consecutive element

→ Sliding window Method.

Raj
Page:
Date: / /

Q.

-2	10	1	3	2	-1	4	5

K=3

We move fwd by subtract" first element and adding next element

Q Top 4 most frequent element in Array.

eg:

3	1	2	1	2	1	2	3	4	2	4	5	3

Array

Map

KEY	FREQ.
1	3
2	4
3	~~2~~ 3
4	2
5	1

eq. end())

SUDOKU SOLVER (9x9)

Page: Raj
Date: / /

Q. Solve Sudoku (9*9)

- Start from (0,0) and end is (8,8)
- Try out Possible combinat^n
- ans = false, currently at (x, y)

```
if
    ans = already filled, move to next block

else
    for (numbers = 1 to 9)
    {
        if you can place number at (x, y)
            ans = ans or place no. at (x, y) and move to
                  next Block
```

Sudoku Solver

CODE

Raj
Page:
Date: / /

```
void solve(vector<vector<char>>& a) {
    map<pair<int, int>> rows(9);
}
```

```
bool solve(vector<vector<char>>& board) {
{
    int n = board.size();

    for (int row=0; row<n; row++)
    {
        for(int col=0; col<n; col++)
        {
            if (board[row][col] == 0) // cell empty
            { // Try values 1 to 9
                for (int val=1; val<10; val++)
                {
                    if (isSafe(row, col, board, val))
                    {
                        board[row][col] = val;
// recursive call to check further soln is possible with this val
                        bool solPossible = solve(board);
                        if (solPossible)
                        {
                            return true;}
                        // Backtrack
                    else {
                        board[row][col] = 0;
```

```
        }; // If all val are checked
        Return False
    }
  }
}
return True
}

bool isSafe(int row, int col, vector<vector<int>>& board, int val)
{
    for(int i=0; i<n; i++)   // n → board.size()
    {   // Row check
        if(board[row][i] == val)
            return false;
        // col check
        if(board[i][col] == val)
            return false;
        // 3×3 matrix check
        if(board[3*(row/3) + i/3][3*col/3 + i%3] == val)
        {
            return false; }
    }

    Return True;
}
```

POSSIBLE SUBSEQUENCE OF ARRAY

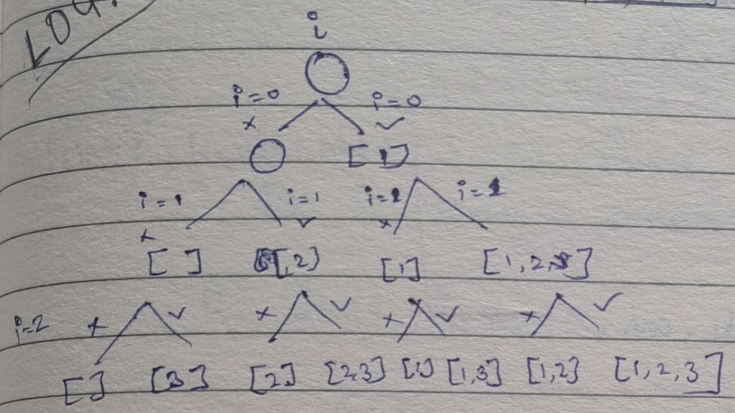
Raj
Page:
Date: / /

- can is discontinious
- order have to be same
-

eg: [1|2|3]

1
1,2
1,2,3
1,3
2
2,3
3
[ ]
3,1 ×

LOGIC



At index i we have two opt^n
1) Include it
2) Not include (exclude)

PSUEDO CODE

Pass By Ref

```
void funct (int index, vector<int> &ds, int arr, int n)
                                  Result
{
    if (index == n)
    {
        for (auto i : ds)
        {
            cout << i ;
        }
        if (ds.size() == 0)
        {
            cout << "{ }"
        }
        return;
    }
```

// This is printing the result stored in vector

Page: Raj
Date: / /

```
// include an element
    ds.push_back(arr[index]);
    Funct(index+1, ds, arr, n);
    ds.pop_back();
// exclude an element
    ds.Funct(index+1, ds, arr, n);
}

int main()
{
    int arr[] = {4, 3, 1, 2}
    vector<int> ds;
    print
        Funct(0, ds, arr, n)

    return 0;
}
```

} Include

Backtrack (exclude)

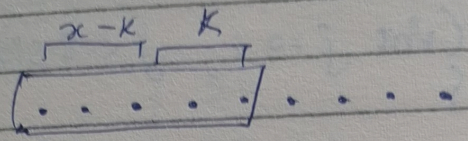
(+ve / -ve elements)

Length of LONGEST SUBARRAY with SUM = K

Page: Raj
Date: / /

Q. given array with +ve/-ve terms, find longest subarray with sum k.

~~Using TWO POINTERS approach~~ PREFIX SUM + HASHING ★★

Concept: 

(array of dots

if I want 'k' in this in subarray so some part has to be x-k, so it need to find if x-k exist in map and till which index

eg: 1 2 3 1 1 1 1 k = 3

Sum = 1, 2+2, 3, 2+3 len = 2 (max)

index = 2 (6) here we need 3=k and remaining part as 6-k = 3

∴ Is there any sub array previously which had sum = 3?

Yes, which is @ index = 1

so from 2 to 1 : len = 1

Sum	Index
	4
7	3
6	2
3	1
1	0

Hashmap

+1 → 7, need sum = 7-3 = 4 (Not found in hash table)

+1 → 8 " " " 5 " " "

index = 5 +1 → 9 need sum = 9-3 = 6 (found at index = 2 ★

So 2 to 5 : len = 3 (MAX)

- Prefix Sum + hashing ⇒ +ve and -ve values in array
- 2 Pointers ⇒ only +ve values

Raj
Page:
Date: / /

Don't store duplicate (sum)s in Map.

★ But if Sum already exist in hashtable then No need to put it again

Code

maxlen = 0, i = 0

unordered map <int, int> m;

key = Sum
Value = index (jis tak woh Sum aaya h)

```
while ( i != V.size() )
{
    sum = sum + V[i]
```

Sum ki jagh XOR (count) { x = x ^ nums[i] }

```
★   if ( sum == k )   // max length is 'i' only
    {
        maxlen = i+1 ;
    }
```

// Sum array ka ek part ka sum is 'X-k', ek part is 'k', where subarr sum = X

```
★   if ( m.find(sum-k) != m.end() )   { sum-k = x^k }
    {
        maxlen = max(maxlen, i - M[sum-k])
    }
```

// only store "sum" wali 'key' if it dosn't already exist

```
★   if ( m.find(sum) == m.end() )
    {
        m[sum] = i ;
    }
}
```

{ No need } m[x]++

Q Subarray with Max Sum in which less x (length = k)

Sliding Window : Not for -ve numbers

```
maxSumSubarray (int arr[], int n, int k, int x)
{
    int sum=0, maxsum=0;
    for (0 to (k-1); i)                ] Initialise window of k length
    { sum+= arr[i]

    if (sum<x) maxsum = max(maxsum, sum);

    for (i=k; i<n; i++)
    {                                  Moving Windgw
        Sum = Sum + arr[i] - arr[i-k];
        if (sum < x)
        {
            maxsum = max (maxsum, sum);
        }
    }

    return maxsum
}
```

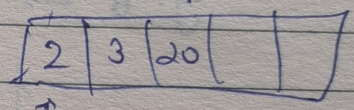
Raj
Page:
Date: / /

NOTE

- In case of +ve and -ve elements in array
  ↳ Best is to use **Prefix Sum + Hashing** ***
- In case of only +ve numbers
  ↳ 2 pointers

Q find Smallest Subarray of Sum > x (only +ve no.)

x = 9




```
for (int i=0; i<n; i++)
{
    Sum = Sum + arr[i];
    while (Sum > x)
    {
        minlen = min(minlen, i-j+1)
        Sum = Sum - arr[j]
        j++
    }
}

return minlen;
```

Sum > x, reduce length of window & record minlen

Q Number formed from Subarray (size = k) which is divisible by 3

Page: Raj
Date: / /

i.e sum of digits of subarray is multiple of 3

{Can be done by | Sliding window (only +ve elements) | Prefix sum + Hashing (+ve/-ve elements both)}

Q Max Perfect numbers in Subarray of size 'k'

- PERFECT NUMBER : PN is equal to its sum of divisor except itself

eg: 6 → divisor = '1' + '2' + '3' → 6. (Perfect)
(Add)

↓

Use Sliding Window. ✓

Q (1,n) no are there in array, exact A = repeat twice, B = missing find A, B

eq (I) Sum of n natural no − Sum of no. of array

eq (II) Sum of sq of n natural no − Sum of sq of element of array

↓

Solve eq (I) : A − B

Solve eq (II) $A^2 - B^2$

Count

↳ whe

ALGORITHMS

Page: Raj
Date: / /

Q Smallest subarray whose sum > x (for +ve/-ve elements)

```
map<int, int> m
int sum=0, minlen = INT_MAX;

for (int i=0; i<n; ++i)
{
    sum += arr[i];
```

sum ki jagah XOR aa gaya toh.

Find prefixSum such that sum - prefixSum > x → PrefixSum < sum-x

```
auto it = m.lower_bound(sum-x)

if ( it != m.begin() )
{
    --it;
```

Move to largest prefixSum which is JUST less than (sum-x)

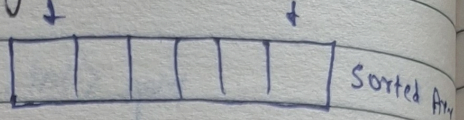
```
    if ( sum - it->first > x)
    {
        minlen = min(minlen, i-it->second);
    }
}

if (m.find(sum) == m.end)
{
    m[sum] = i;
}
}
```

Raj
Page:
Date: / /

Q. 2 Sum. , find 2 element in array whose addⁿ = T

Target



1) Sort array
2) 2 pointers Method

Q 3 Sum. find all triplet that sum = 0
(cannot take same elements more than once)
(return unique triplets)
(order can be any)

approach:

1) Sort the Array ★

initially i, j, k

fix [-2, -2, -2, -1, -1, -1, 0, 0, 0, 2, 2, 2, 2]

Use 2 pointer approach with j, k until they cross ★

When they cross,

Move 'i' to next different element ★

-2, -2, -2, -1, -1, -1, 0, 0, 0, 2, 2, 2, 2]

i j k

then Repeat

Q Max Sum SubArray (Kadane's Algo)

Page: Raj
Date: / /

- we iterate through all array elements, then we add them to "sum" if at any point sum is 0 then we sum = 0, we won't consider that subarrays
- extract max from all values of sum

```
{   for (int i=0; i<n; i++)
    {
        sum = sum + arr[i]
        maxSum = max (maxsum, sum);
        if ( sum<=0)
            sum=0
    }
    retun maxsum;
}
```

Q Rearrange +ve and -ve numbers in array such that they are alternate and ordering remain same

Method -1 : Make 2 array for +ve and -ve elemts (all cases valid)

Method-2) : Make a Map (equal no of +ve/-ve)

lexographically....

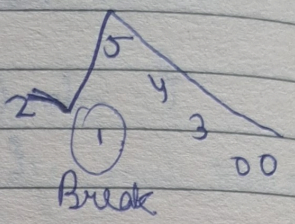
Page: Raj
Date: / /

Q Next Permutation

eg: Permutation of 1,2,3 are {1,2,3} {1,3,2} {2,1,3} {2,3,1} {3,1,2} {3,2,1}

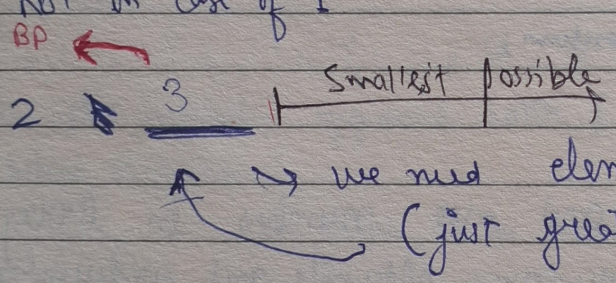
BP ↓ Sort

eg: 2, 1, 5, 4, 3, 0, 0



1) find Break point

- We need it coz, if you take 5 (say) you can never arrange the subarray fm 4 to 0 in such a way it gives a greater value than already existing,
- But not in case of 1



↳ we need element > 1 but smallest (just greater than 1

Now remaining array should be smallest

Steps

1) Breaking Point a[i] < a[i+1]

2) find num > 1 but smallest possible

3) Sort i+1 to n-1

already a exist in STL

already a func exist in STL : next_permutation(A.begin(), A.end());

Page: Raj
Date: / /

ODF:

```
void nextPermutation (vector<int> & nums)
{
    int i = n-2 // Break" Point
```

1) find first index from end which : nums[i] < num[i+1]

```
    while (i>=0 && nums[i] >= nums[i+1])
    {
        i--
    }
```

if such index exist

2) ↳ find next greater element than nums[i] from Right

```
    int j = n-1 ;
    while (j nums[j] <= nums[i])
    {
    } j--;
```

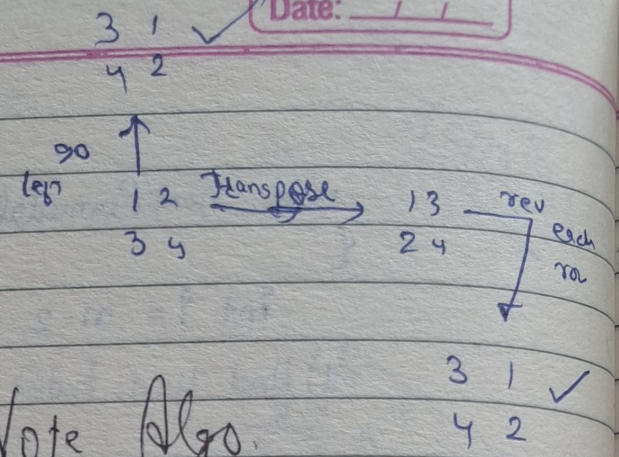
3) Swap nums[i] and nums[j]

4) sort (num.begin + i+1 , num.end());

Raj
Page:
Date: / /

Q Left rotate 90°

1) Take transpose
2) Reverse Row



*BOYER-MOORE-Majority Vote Algo.

Q. Find the integers in array (size: n) which appear more than (n/3) times (OPTIMAL)

Without hashmap

★ We can say there can only be 2 such elements (ALWAYS) ↓ PROOF ⇒ for Maj. elem > n/3 ★★★

eg: n = 8 ⌊n/3⌋ = 8/3 = (int) 2 ⇒ so for Majority element have to be 3 times

floor funct^n

[3, 3, 3, 2, 2, 2, 1, 1]

NOT Possible ← 9 elmnt

1) Brute force
2) Better = hashmap
3) Optimal = Boyer-Moores-Majority vote Algo

Raj
Page:
Date: / /

ele1 , ele2

1) We consider 2 element and their counts c1 and c2 = 0

2) If he we have ele1 in iterat" we inc c1 otherwise dec (same with ele2 and c2)

3) If at a point count = 0, then that element can't be our Majority Because, we are if there is major element (n/3>) then it won't be cancelled in count by other elements

↓
update el1 as nums[i]

CODE

```
int ele1, el2, cnt1=0, cnt2=0;

for (i=0 to n-1)
{
  if (cnt1==0) cnt1=1, el1=nums[i]
  else if (cnt2==0) cnt2=1, el2=nums[i]

  else if (el1==nums[i]) cnt1++;
  else if (el2==nums[i]) cnt2++;

  else { cnt1--; cnt2--; }
}
```

// Now check el1 and el2 have count more that (n/3) or not

```
for(i=0 to n-1){ if (nums[i] == el1) { cnt1++ } }
```

```
if (cnt1==0 && nums[i] != el2)
{ cnt1=1; el1=nums[i]; }
```

modify

```
if (cnt2==0 && num[i] != el2)
{ cnt2=1; el2=nums[i] }
```

Modify:

So That Both el1 / el2 are not pointing to similar elements

```
if (c1>t) ans.push(el1)
```

3 Sum

- find triplet (sorted) whose sum = 0
- No element can be used twice (max)
- order be any

Raj
Page:
Date: / /

Optimal

CODE

```
vector<vector<int>> ans;
vector<int> v = {-2,2,-2,-2,0,-1,0,-1,0,-1,2,2,2};

sort(v.begin(), v.end());

for(int i=0; i<v.size()-2; i++){
{   // handle duplicate cases of 'i'
 * if(i>0 && v[i]==v[i-1]) continue;

    int j = i+1;
    int k = v.size()-1;
    while(j<k)
    {  int sum = v[i]+v[j]+v[k];
       if(sum==0)
         {
Push      ans.push_back(v[i]+v[j]+v[k]);
inc j and dec k   j++; k--;

       // skip duplicate for j and k
       while(j<k && v[j]==v[j-1]) j++;
       while(j<k) && v[k]==v[k+1]) k++;
       }

    else if(sum<0) j++;
    else k++;
}
```

4

CODE

4 Sum

find 4 diff index, sum = K
→
→

Raj
Page:
Date: / /

optimal

1) Sort array

like in 3 sum you fixed "i" here

4 sum : fix "i" and "j"
use 2 pointer in "k" and "l"

- Skip dulicates for i, j, k, l. ☆

CODE

```
for (i=0 ; i<n-3 ; i++)
{
if(i>0 && nums[i]==nums[i-1]) continue;

for (j=i+1 ; j<n-2; j++)
{
if( j>i+1 && nums[j]==nums[j-1]) continue;

int left = j+1;
int right = n-1;

while (left < right)
{
Same
}
```

Merge Overlapping Intervals

Raj
Page:
Date: / /

INPUT: [1,3], [2,6], [8,10], [15,18]
OUT: [1,6], [8,10], [15,18]

1) Sort the matrix Sort(v.begin(), v.end());

```
vector<vector<int>> merge(vector<vector<int>>& v)
{
    vector<vector<int>> ans;
    sort(v.begin(), v.end());

    for(int i=0; i<v.size(); i++)
    {
        if(ans.empty() || v[i][0] > ans.back()[1])
        {
            ans.push_back(v[i]); // Insert Single Interval
        }
        else
        {
            // Swap the max range as higher limit
            ans.back()[1] = max(ans.back()[1], v[i][1]);
        }
    }
```

v.back() → give back element of vector