STACKS & QUEUES

Raj
DATE / /
PAGE

Q Max Sliding Window

Return max no. in window in each traversal

eg:

INPUT array: | 1 | 3 | -1 | -3 | 5 | 3 | 6 | 7 | k = 3

Window 1 :	1, 3, -1	max : 3
" 2 :	3, -1, -3	max : 3
3 :	-1, -3, 5	" : 5
4 :	-3, 5, 3	" : 5
5 :	5, 3, 6	" : 6
6 :	3, 6, 7	" : 7

OUTPUT : [3, 3, 5, 5, 6, 7].

deque Allow insertion and deletion from both ends

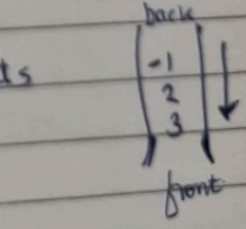
push-back → ← push-front

pop-back ← → pop-front

- front(), back()
- empty(), size

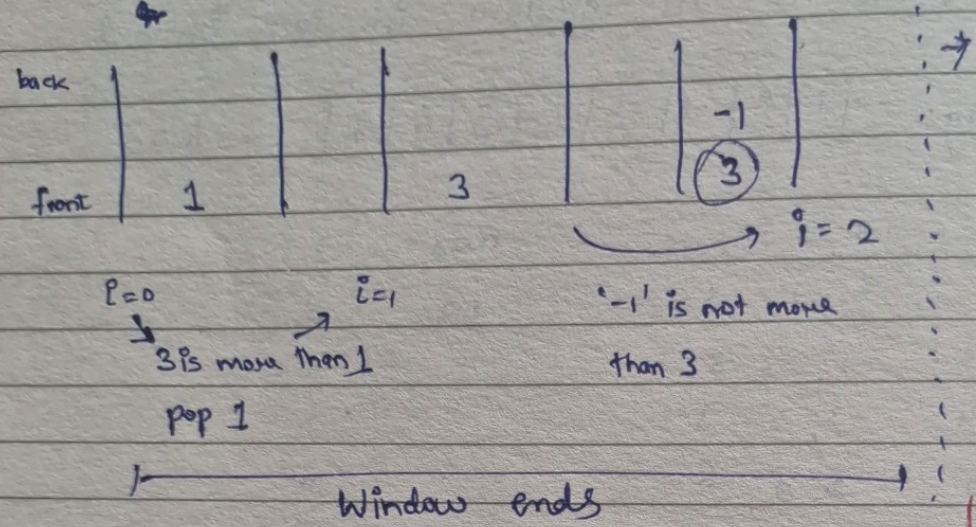
APPROACH:

- We use deque as monotonic stack (store value in dec order)
- We will store elements as size of window



DATE 3/12
PAGE

eg: nums = [1, 3, -1, -3, 5, 3, 7, 1, 6] k=3
0 1 2 3 4 5 6 7 8



→ Add max (front()) when you reach window size.

i=0 → 3 is more than 1 → pop 1

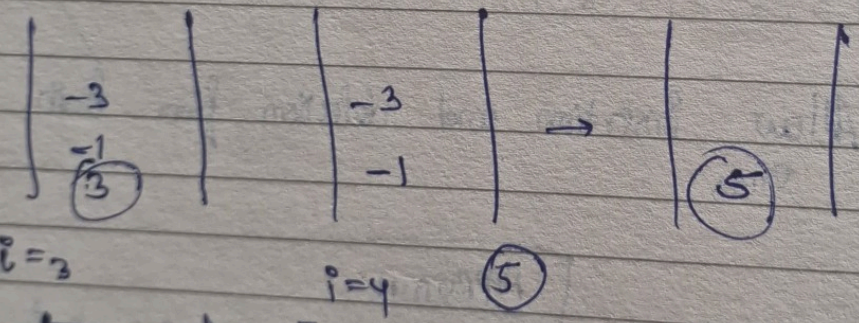
i=1

'-1' is not more than 3

Window ends

We will store indices not value

Now in 2nd window



i=3 → Window over

i=4 → pop front

5

Since 5 is more than all elements then pop all then add 5

i=

Raj
DATE / /
PAGE

```
for ( int i=0 ; i< nums.size(); ++i)
{
```

(1) Remove element from Back while nums[i] > nums[dq.front()]

```
while (!dq.empty() && nums[dq.back()]<nums[i])
{
  dq.pop_back();
}
```

(2) then add that new max value

```
dq.push_back(i);
```

(3) Remove (index) element thats out of window

```
if (!dq.empty() && dq.front()<=i-k)
{
  dq.pop_front();
}
```

(4) if window size is reached then add to the vector.

```
if (i>=k-1)
{
  ans.push_back(nums[dq.front()]);
}
```

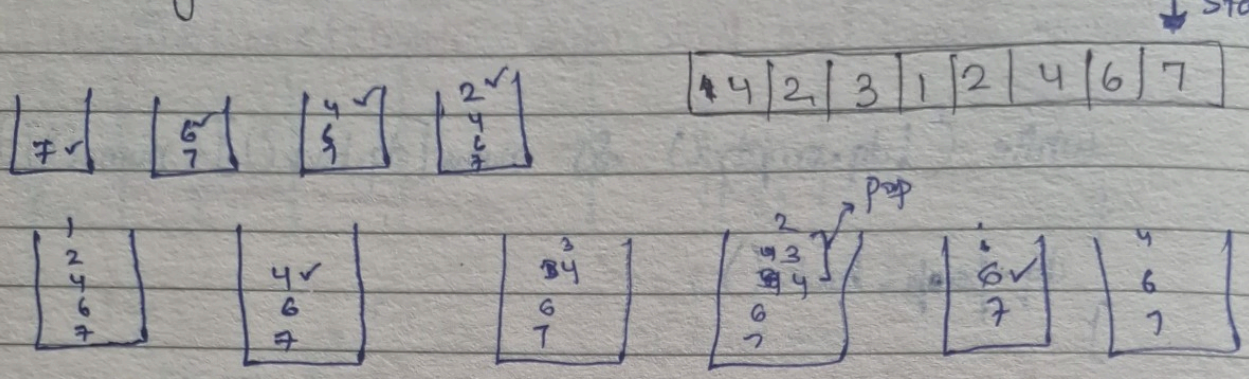
Next Greater Element - I

Raj

DATE / /

PAGE

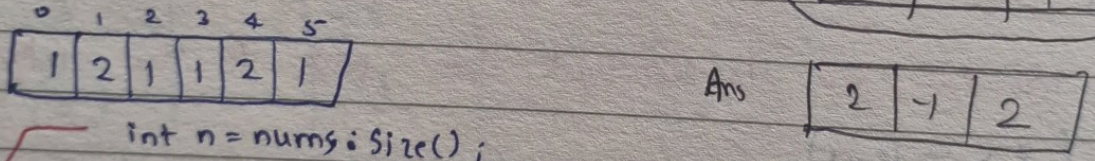
For every element, find index (at right of it) ~~where~~ which is greater than that element.

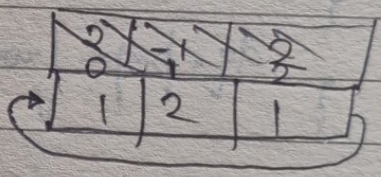


EASY

- If array is circular

We iterate array twice



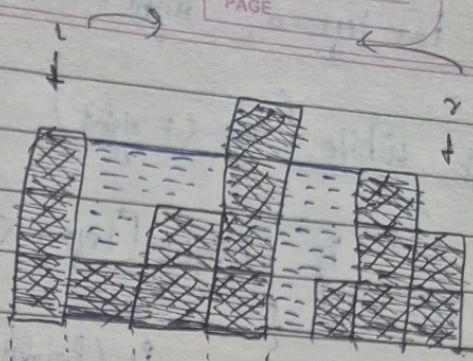


Ans | 2 | -1 | 2 |

```
int n = nums.size();
vector<int> ans (n, -1); // Initialise with "-1"
for (i = 2n-1 ; i >= 0 ; --i)
{  int index = i % n
   while (!st.empty() && st.top() <= nums[index]) st.pop();
   if (!st.empty()) ans[index] = nums[index] st.top();
   st.push(nums[index]);
}
```

TRAPPING RAIN WATER

Raj
DATE / /
PAGE



(1) Make 2 arrays which contain max element from Left & right till index i

Input : 3 1 2 4 0 1 3 2 Units

Output : 8 units

• NOTE: Water can't be stored at extremum

Arrays

Left 3 3 3 4 4 4 4 4

right 4 4 4 4 3 3 3 2

$\min(left[i], right[i]) - a[i]$

Formula: min (left[i] , right[i]) - arr [i]

=

Water Stored at particular Index

Optimized Way

• Water at any index depends on shorter boundary b/t left | right

if (height [left] < height [right])

{

if (height [left] < LeftMax) → We can trap (leftMax - arr[i])

else Update leftMax

}

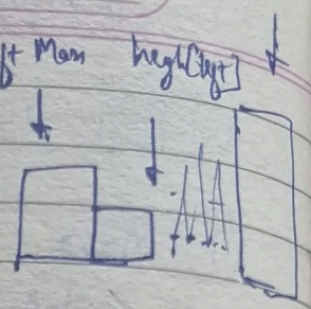
Same with right

DATE / /
PAGE

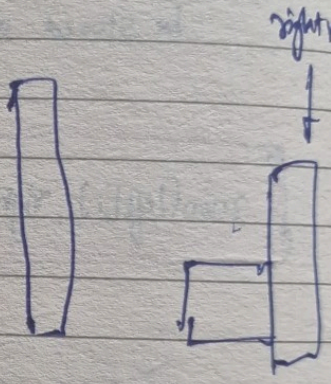
int left=0, right = n-1

CODE

```
while (left <= right)
{
    if (height[left] < height[right])
    {
        if (height[left] < leftMax)
        {
            ans += leftMax - height[left];
        }
        else {
            leftMax = height[left];
        }
    }
    else {
        if (height[right] > rightMax)
        {
            ~~ans+~~ = ~~hmp~~ rightMax = height[right];
        }
        else {
            ans += rightMax - height[right];
        }
```



Possible to trap



Sum of SUBARRAY MINIMUMS

if large value → Mod with 10^9+7

Raj
DATE / /
PAGE

eg: [3 | 1 | 2 | 4]

leg = 109

Subarrays:	Min
3	→ 3
3 1	→ 1
3 1 2	→ 1
3 1 2 4	→ 1
1	→ 1
1 2	→ 1
1 2 4	→ 1
2	→ 2
2 4	→ 2
4	→ 4

Sum of Mins = 17 ✓

Method

no of time this element was minimum → (contributions)

3	1	2	4
×1	×6	×2	×1
3	+ 6	+ 4	+ 4

(17) ✓

How to find each element's contribution

eg:

1	4	6	7	3	7	8	1
0	1	2	3	4	5	6	7

PSC → 0^{th} index

NSC → 7^{th} index

NSE = next smallest element

PS = Previous " "

all the array frm i=1 to i=4, '3' will contribute in it i.e i=1 to 4 will be 3

$7-4=3$, $4-0=4$] $= 3\times4 = 12$

12 contribut" of "3"

* nse / psee are also arrays consisting indices

Raj
DATE / /
PAGE

CODE

nse → findNSE (arr)
psee → findPSEE (arr)
total = 0
mod = (int) (1e9 + 7)
% This after all operation

for (i=0 to n-1)
{
left = i - psee[i] (index)
right = nse[i] - i

*
total += (total + (right * left * (LL) * arr[i]) % mod) % mod

*
// psee = previous Smallest + equal element
equal because we need to handle edge cases of (not to take repeated sequence)

PSEE

vector <int> findPSEE (arr)
{ psee [n] , stack <int> st;
for (i=0 to n-1)
{
while (!st.empty() && arr[st.top()] > arr[i])
{ st.pop();
}
No prev element : put it (-1)
if (st.empty())
{ psee[i] = -1
}
else psee[i] = st.top();

st.push(i);

NSE

vector <int> findNSE (arr)
{
nse [n] , stack <int> st;
for (i = n-1 → 0)
{
while (!st.empty && arr[st.top()] >= arr[i])
{
st.pop();
}
if not next smallest then put it equal to (n)
if (st.empty) : nse[i] = n;
else nse[i] = st.top();

st.push(i);

Sum of Subarray ranges

Raj
DATE / /
PAGE

return sum of ranges of all subarray.

eg: [1 | 4 | 3 | 2]

	ranges
1	1-1=0
1 4	4-1=3
1 4 3	4-1=3
1 4 3 2	4-1=3

	ranges
4	4-4=0
4 3	4-3=1
4 3 2	4-2=2

	ranges
3	0
3 2	1

	ranges
2	0

Sum of ranges = 13

$= \Sigma \left(\Sigma \text{ Sum of Subarr Max.} - \Sigma \text{ Sum of Subarr Mins.} \right)$

↳ did at Back

Remove K digit code

Left to Right

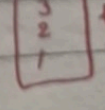
↳ Always remove digit that greater than next digits

If smaller element on top, just push

If larger element (remove that (K--) and add the smaller

or

Keep Stack in Incr order

eg: 

Remove K digits.

Raj
DATE / /
PAGE

Remove k digits to make smallest possible no.

eg: 1432219 k=3

output: 1219

Edge cases.

Remove all no.
k == N
↓
"0"

Initial zeros.
00100
↓
"100"

"12345"
won't be able to remove k elements.

```
for (i=0 to n-1)
{
while (!st.empty() && k>0 &&
 st.top()-'0' > s[i])
 {
    st.pop();
    k--;
 }
 st.push
}
```

If you couldn't remove element then remove last k elements

```
while(k>0){st.pop(); k--;}
if (st.empty()) return "0";
```

Store stack no in string

```
string ans="";
while (!st.empty())
{
} ans = ans + st.top();
```

Remove initial zeroes

```
while (ans.size()!=0 &&
        ans.back()=='0')
{
 ans.pop_back();
}
Reverse ans - string
    reverse(ans)
if(ans.empty()) return "0"
else return ans;
```

Larges

given

M-1

for eve

arr

M2

• U

• We

wh

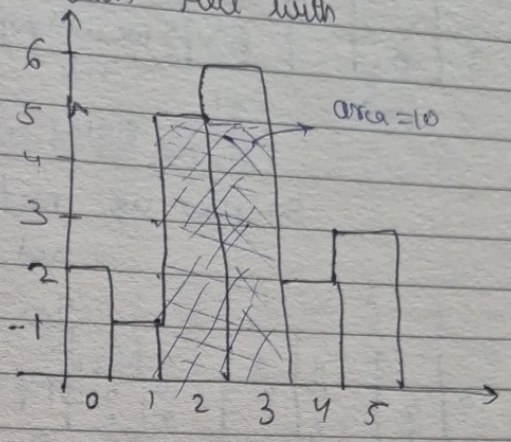
So

Largest Rect in histogram

★★★★★

Raj
DATE / /
PAGE

given heights of histogram bars, return rect with max $ area



(M-1)

for every element

↳ arr[i] x (nse[i] - pse[i-1])

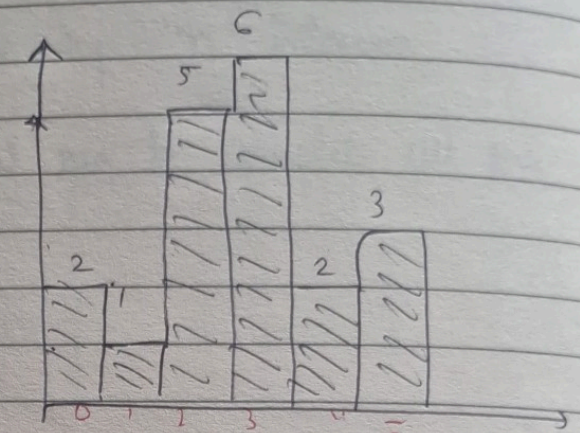
(M2) On fly. Method

- We need value of nse and pse in single traversal
- We will initially compute for pse and whenever Stack element * (top) is greater so we have to pop right while we pop so for that element we know its nse, that nse is reason it is getting popped so.

DID AGAIN AT LAST

~~Top K most frequent elements~~

LARGEST RECTANGLE in HISTOGRAM



ON THE FLY

Problem

arr[i] * (nse[i] - pse[i] - 1)

we have to traverse ~~within~~ whole array

So,

If we traverse Left to Right, we will have knowledge about 'pse'.

For nse,

We can't know in single traversal about 'nse' right?

We Can ! !

0 2

PTO

BRUTE FORCE

```
{
  nse = findNSE(arr)
  pse = findPSE(arr)
  maxi = 0

for (i=0 to n)
{
  maxi = max(maxi, arr[i] * (nse[i] - pse[i] - 1))
}
return maxi
}
```

ON THE FLY

we do same but without use nse/pse, in single traversal.

eg: W

So,

10
2

i=2

0 1 2 3 4 5 6 7

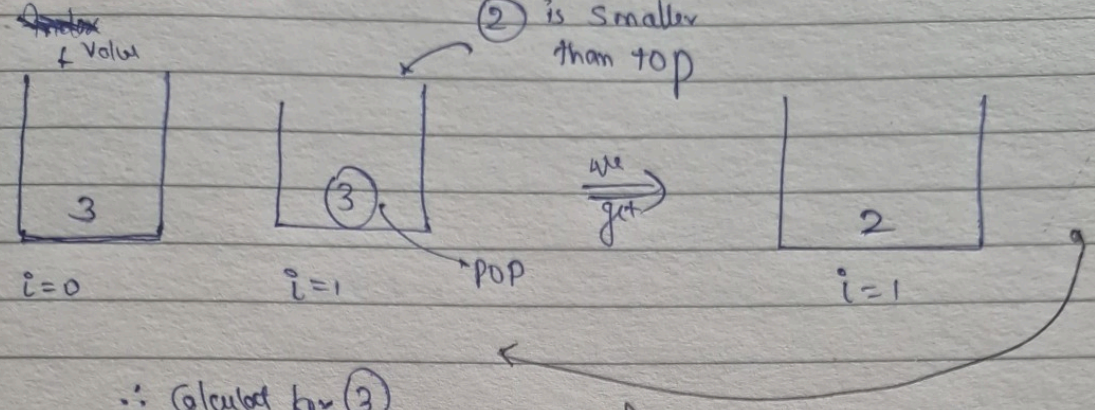
[3, 2, 10, 11, 5, 10, 6, 3]

Raj

DATE / /

PAGE

eg: while calculat^n PSE



∴ Calculat for 3

when we popped 3

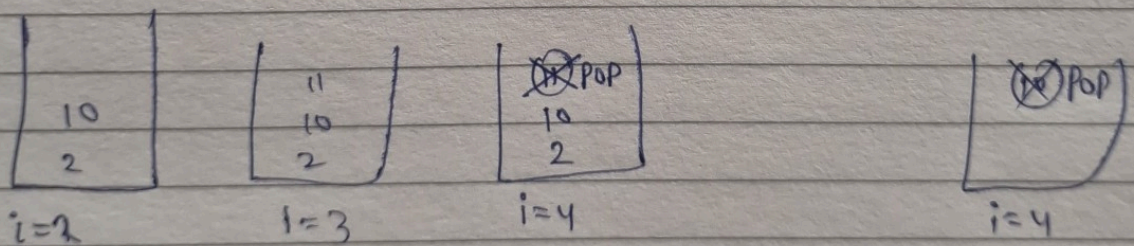
we understood that,

we only pop when smaller element comes

we popped 3 cuz 2 came. ∴ nse of 3 is 2

when we pop, we know it's smaller and is NSE.

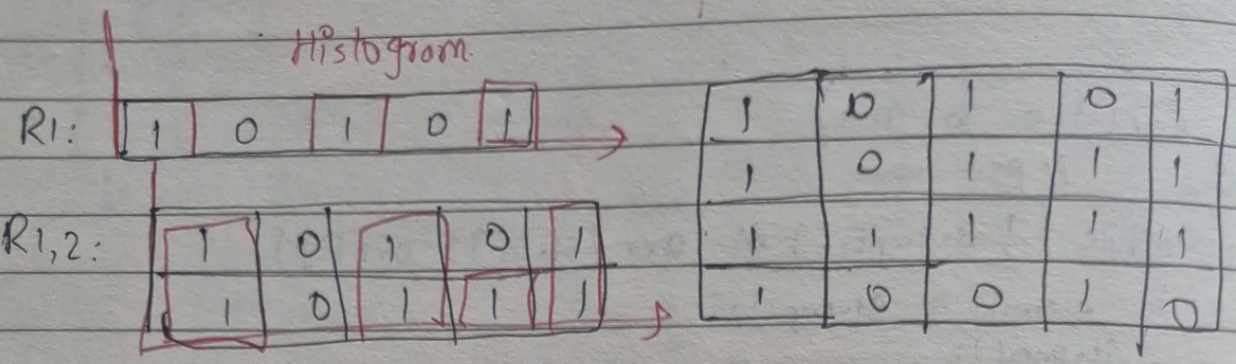
So, i=0 : area = $3 \times (1-(-1)-1)$ = 3



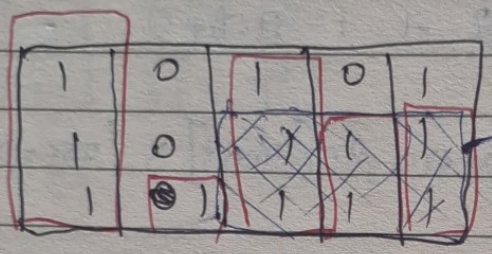
area at i=3 = $11 \times (4-2-1)$ = 11

or (i=2) = $10 \times (4-1-1)$ = 20

Max 1's Rectangle.



R1,2,3:



We use prefix sum to determine array for histogram

1	0	1	0	1
2	0	2	1	2
3	1	3	2	3
4	0	0	3	0

put zero at place of zero since we can't have histogram there.

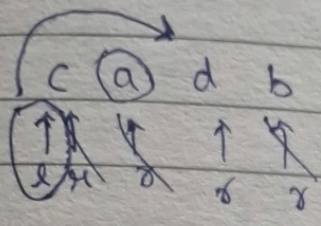
Each row is converted to an array

Longest Substring without duplicates

Raj
DATE / /
PAGE

→ think 2 pointer and Sliding window

eg: c a d b a b



Map

char	index
b	3
d	2
a	1
c	0

- If we encounter any char that is already in map (repeated) will be jump to (index+1) = l

Make sure when you update "l", it must be less than updated value

```
for (r=0 to s.size)
{
  if (m.find(s[r]) != m.end())
  {
    l = max(l, m[s[r]]+1);
  }

  m[s[r]] = r
  maxlen = max(maxlen, r-l+1);
}
```

Max Consecutive 1 with K flips of 0s

Raj

DATE / /
PAGE

Q. find max consecutive "1" if you can flip 'k' zeroes

flip1 flip2

eg: [1,1,0,0,0,1,1,1,1,0]

output: 6 ✓

2 pointer l, r=0

- Find largest subarray with 'k' zeroes
- If we get more zeroes than k then will shift 'l' one by one, and if next is a zero with will dec the zero count until zeroes <= k.

```
for (R=0 to n)
{
    if (arr[i]==0) z++;

    while (z>k)
    {
        if(arr[l]==0) { z--; }
        l++;
    }

    maxLen = max(maxLen, r-l+1);
}
```

* Can be optimized.

q.ins not letting it go beyond maxlen no matter how many zeros we got

Constant length Concept

Raj

DATE / /
PAGE

```
if (z > k)
{
    if (arr[l] == 0) { z-- }
    l++;
}
```

we inc l, r both keeping the length same.

Longest Repeat" char Replacement

Find length of longest substring containing the same letter, you can Replace any letter to any alphabet 'k' times

eg: ABABC k=2
output : 4

eg: AAABBCCD

l r

maxfreq = 0
maxlen = 0

hashmap for freq of each alphabet.

len - maxfreq <= K

A, 1

Raj

DATE / /

PAGE

APP

length - maxfreq <= k

* There is no need to trim the ~~aa~~ s max freq which you reduce the subarray.

Because AAABB 5-3 <= (k) 2 valid

See 3 AAABBCC D 6

6-3 <= (k) 2 X Not valid

So even if you reduce B max freq., it won't contribute in getting longest subarray.

len - maxfreq <= K

5 - (3) <= 2 VALID

Now reducing '3' will never be valid answer

Reducing | Not reduce

6-2 <= 2 X

6-4 <= 2

maxfreq = 3 k = 2

AAABBCC D

len = 5

5-3 <= 2 ✓

∴ It is valid

keep maxlen track

maxlen = 5

maxfreq = 3

AAABBCCD

len = 6

6-3 <= 2 X

(Not valid

TRIM

l MF=2 r

AAABBCC

len = 5

5-2 = 3 X.

maxfreq = 2

AAABBCC

len = 4

4-2 <= 2 ✓

len = 4 < 5

Reducing B maxfreq will never give larger answer.

```
int l=0, r
unordered
for(int r=
{
    m[s[r
maxfreq = ma
int len = r
while (len
{
    m[s[l
    // update
    for(auto
    { maxfreq =
    }
    l++;
}
if( len - ma
{
maxlen = max
}
```

OPTIMIZATIONS

```
int l=0, maxlen = maxfreq=0;
unordered_map <char, int> m;
for(int r=0 to n)
{
   m[s[r]] ++;
maxfreq = max (maxfreq, m[s[r]])
  int len = r-l+1
While (len - maxfreq > k)
{
   m[s[l]] --; (trim subarr)
  // update maxfreq.
  for(auto it : m)
  { maxfreq = max(maxfreq, it.second)
  }
  l++;
}
if ( len - maxfreq <= k)
{
maxlen = max (maxlen, len);
}
```

(1) While loop can be remove by CONSTANT SIZE. (oncept of 2 POINTERS.

→ inc. l by 1 and r by 1 keep max le same unit it becomes valid.

```
if (len - maxfreq > k)
{ m[s[l]]--
   l = l+1 ;
}
```

(2) No need to chang maxfreq.

Becaus. reducig it won't give you bigger maxle.

Binary Subarray of Sum K

Raj

DATE / /
PAGE

Since we have Binary array, we can optimise method of prefix sum

answ = (Subarr of sum <= K) - (Subarr of sum <= K-1) = Subarr with sum K

Subarr of sum <= K

```
if (k<0) return 0;
int l=0, r=0, cnt=0, sum=0;
while (r < nums.size())
{
    sum += arr[r]
    while ( sum > K)
    {
        sum = sum - arr[l]
        l++;
    }
    cnt = cnt + (r-l+1)
}
```

eg: 1 0 1 0 1 K=2

here 3 sub array with sum <= K

→ call this 2 times for K and K-1

then subtract both values

you will get cnt for sum == K

Numb. of Substrs with all 3 chars

Raj

DATE / /

PAGE

Q. given string containg only a, b, c , find no. of substring having all 3 characters (atleast one occurrance)

eg: a b c b a c

Output: 10 : abc, bcba,

last Sum: a = -1 b = -1 c = -1

for eg: b b a c b a → Min. sub string with all 3 char, ending at 'a' Min index among all is c: 3

(end)

So 4 can take,

b b a cba → = 4 → = minIndex (3) + 1 (include itself)

for this index.

for that index possible substr

minIndex + 1

(concept)

minIndex

b b a c b

eg.

2 + 1 = 3 possible for i = 4

```
int ndA, ndB, ndC = -1, cnt = 0;
for (i = 0 to s.size())
{
  If (s[i] == 'a') ndA = i
  If (s[i] == 'b') ndB = i;
  If (s[i] == 'c') ndC = i;

  if (ndA != -1 && ndB, ndC != -1)
{
  int minInd = min (ndA, ndB, ndC);
  cnt += minIndex + 1
```

Longest Substring with atmost K distinct characters

S = [aaabbccd] K = 2

output: aaabb

- Use Map to measure freq. of distinct char
- If map size > K
  move 'i' and reduce freq
  until freq == 0, remove elmnt from map
  Inc 'i'

* // (Subarr with distinct char <= K)
− (Subarr with distinct char <= K-1)

=

Subarr with distinct char exactly equal to K.

eg:
1
2
3

EASY.

Minimum Window Substring. ★★

(s, t) 2 strings of length m, n. return window substring of 's' such that every character in 't' (including duplicates) is included in window.
If not possible return "".

eg: S = "ADOBECODEBANCA" t = "AABC"

output: "BANCA"

Raj

DATE / /

PAGE

```
   i j
   ↓↓
eg: s = d d a a a b b c a        t = abc        len of t = 3
```

① for every 'i' Subtract 'i' from map

If it's not there Then put -1

c, 1
b, 1
a, 1

Char freq

② Whenever the value put existed in map i.e +ve value, we inc the count

Cnt = 0;

So At some point

```
d d a a a b b c a
↑           ↑
l           r
```

cnt = 0 X 2 3 ✓

So here our count == 3 = length of t

So char are included in l to r

d → -2
c, → 0
b, → 0 → -1
a, → 0 → -1 → -2

③ Minimise This, as you move l forward, inc. Their freq in map and if freq of any becomes +ve, inc the count.

```
x x x x x x
d d a a a b b c a
  ↑   ↑   ↑
  l   r   r
```

if count == len of t still (VALID) ✓

Raj
DATE / /
PAGE

total possible characters

```
function(s, t)
{   hash[256] = {0}        l=0, r=0, minlen = 10^9
                           startInd = -1;

    for(i=0 to t.size())
    {                                  } map 't' index
    hash[t[i]]++;
    }
    while(r < s.size())
{
 if(hash[s[r]] > 0)  { cnt = cnt + 1;
                       hash[s[r]]--;
                     }

  while(cnt == t.size()) // we got all 't' char, now
{                          just minimize the string
   if(r-l+1 < minlen)
      {
        minlen = r-l+1;
      } startInd = l;

   hash[s[l]]++;
   if(hash[s[l]] > 0)
    {
      cnt = cnt - 1;
    }
    l++
}
   r = r+1;
}
return startInd == -1 ? "" : s.substr(startInd, minLen);
```