Sorts

0 Selection Sort

⇒ Select min

13	46	24	52	20	9	
0	1	2	3	4	5	0
9	46	24	52	20	13	step 1
9	13	24	52	20	46	2
9	13	20	52	24	46	3
9	13	20	24	52	46	4
9	13	20	24	46	52	5

⇒
```
for (i=0; i<=n-2; i++)
{ min = i;
  for (j=i; j<=n-1; j++)
  {
    if (arr[j] < arr[min]) {
      min = j;
    }
  }
  Swap(arr[min], arr[i]);
}
```

→ Time Complexity $O\left(\frac{n(n+1)}{2}\right)$

$= O(n^2)$

Best, avg, worst → $O(n^2)$

Date ___/___/___
Page _____

Bubble Sort

→ Compare two elements swap if needed

→ At every round max will set at end of array

0	n-1	4	3	5	1	0
0	n-2					
0	n-3					
0	n-4					
0	1					

```
for ( i=n-1 ; i >=1 ; i--)
{
    for (j=0 ; j<=i-1 ; j++)
    {
        if (a[j] > a[j+1]) ; } No swap happen
            swap                then return
    }
}
```

O(N²) → Worst complexity

O(N) → Best (Sorted)

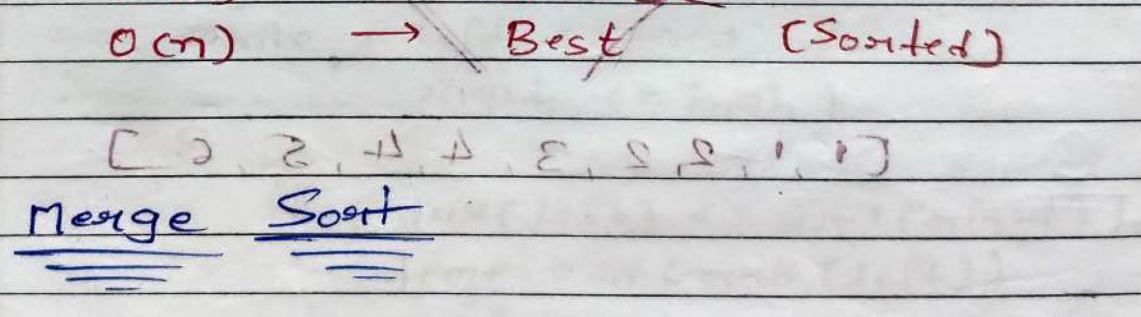
(modification)

Date ____/____/____
Page ____

Insertion Sort

⇒ Take element Place in Correct Position

```
for (i=0; i<=n-1; i++)
{
    j = i;
    while (j>0 && a[j-1] > a[j])
    {
        swap(a[j-1], a[j]);
        j--;
    }
}
```



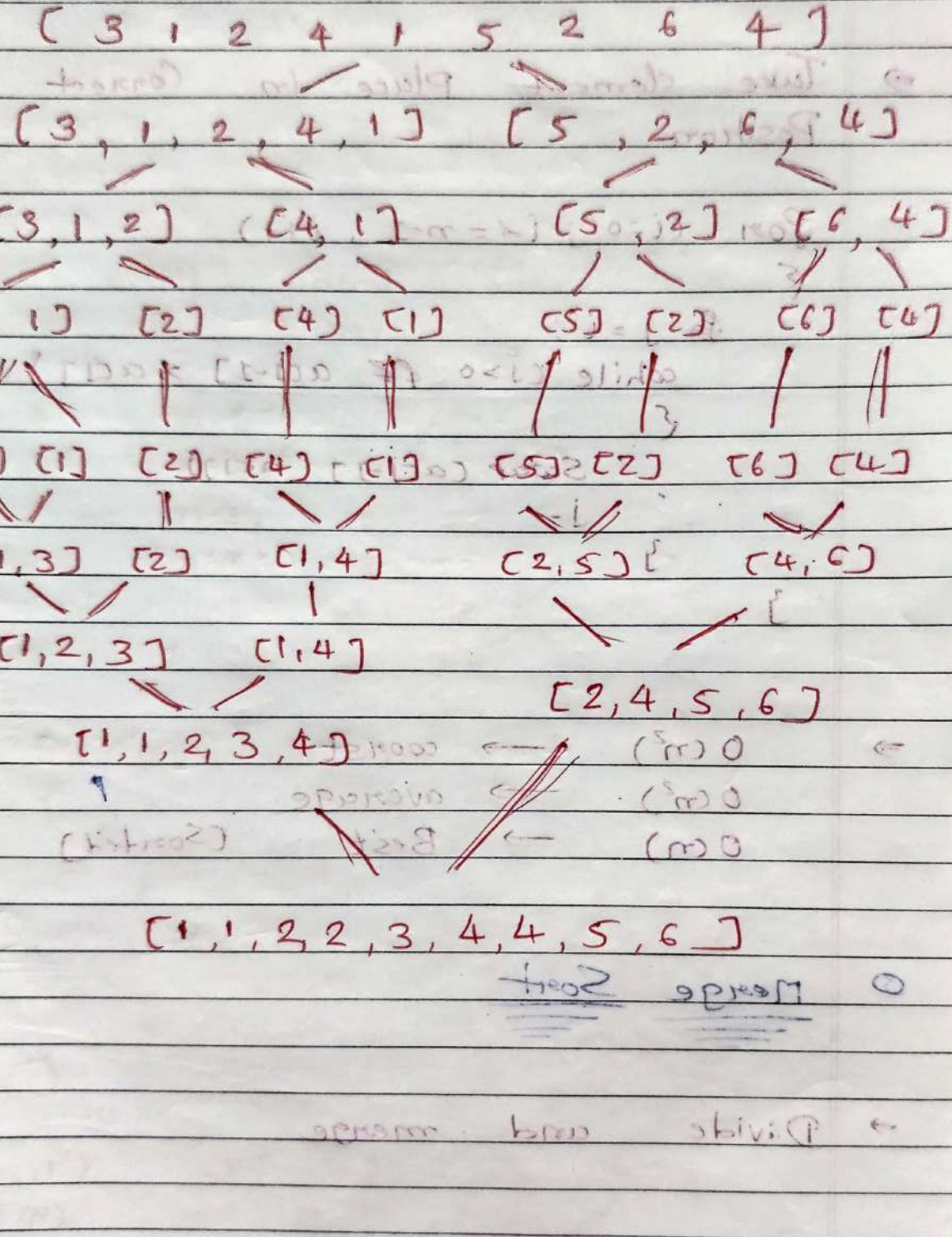
⇒ $O(n^2)$ → worst
$O(n^2)$ → average
$O(n)$ → Best (Sorted)

Merge Sort

⇒ Divide and merge

Date ___/___/___

Page _____


[3 1 2 4 1 5 2 6 4]
[3, 1, 2, 4, 1] [5, 2, 6, 4]
[3, 1, 2] [4, 1] [5, 2] [6, 4]
[3 1] [2] [4] [1] [5] [2] [6] [4]
[3] [1] [2] [4] [1] [5] [2] [6] [4]
[1, 3] [2] [1, 4] [2, 5] [4, 6]
[1, 2, 3] [1, 4] [2, 4, 5, 6]
[1, 1, 2, 3, 4]
[1, 1, 2, 2, 3, 4, 4, 5, 6]

Date ___/___/___
Page ___

merge_sort (arr, low, high) {

{

mid = low + high

merge_sort (arr, low, mid);

merge_sort (arr, mid+1, high);

merge (arr, low, mid, high)

if (low >= high)

return;

}

merge (arr, low, mid, high)

{

temp → []

left = low;

right = mid + 1;

while (left <= mid &&

right <= high)

{

if (arr[left] <= arr[right])

temp.add (arr[left])

left++;

else

temp.add (arr[right])

right++;

}

Date ____/____/____
Page _____

```
while (left <= mid)
{
    temp.add(arr[left]);
    left;
}
while (right <= high)
{
    temp.add(arr[right]);
    right++;
}
for (i = low → high)
{
    arr[i] = temp[i-low]
}
```

O(NlogN) → worst

Quick Sort

Array

Date / /
Page

Second largest

```
largest = arr[0] ,  slargest = INT_MIN

for (int i = 0; i < n; i++) {
    if (a[i] > largest) {
        slargest = largest;
        largest = a[i];
    }
    else if (a[i] < largest && a[i] > slargest) {
        slargest = a[i];
    }
}
return slargest;
}
```

Remove Duplicate

```
i = 0
for (j = 1; j < n; j++)
{
    if (arr[j] != arr[i])
    {
        i++;
        arr[i] = arr[j];
    }
}
return i+1;
```

Left rotate array

⇒ rotate arr with D

[1,2,3,4,5,6,7]

[4,5,6,7,1,2,3]

Revesere (start to D)

Reverse (D+1 to end)

Reverse (start to end)

Union of two Sorted array

arr[] = [1,1,2,3,4,5]

arr[] = [2,3,4,5,6]

⇒ add all to set

remove one by one

Brute force

⇒ Two Pointers

[1, 1, 2, 3, 4, 5]

i

[2, 3, 4, 5, 6]

j

Code

github.com/vinayak501/DSA-CPP/blob/main/union_two_sorted_arrays

Find missing Number

arr[] = [1, 3, 4, 5] N = 5

// 1 N-1 size

ans = 3

Not there blw 1 to N

⇒ Brute for

for every 1 to N

check if (it = exist)

O(N²)

⇒ keep arr[]

make 1 for all first

Iteration

find 0 in second iteration

O(N) + O(N)

O(2N)

⇒ Sum all arr[] elements

Total { Sum of (N ← i = i)
(i ← j = N)

$\frac{N(N-1)}{2}$ − Sum of arr[]

= ans.

O(N)

⇒ Total XOR − Array XOR

O(N)

Date ___/___/___
Page ___

Maximum Consecutive ones

nums[] = [1, 0, 1, 1, 1, 0, 1, 1]
(2, 3, 2)

```
maxi = 0
cnt = 0
for (i = 0 → n)
    if (nums[i] == 1)
        cnt++;
        maxi = max(maxi, cnt);
    else
        cnt = 0
```

max Subarray With Sum K

⇒ Three loop Brute force

```
for (i = 0 → n)
    for (j = i → n)
        for (k = i → j)
```

} O(n^3)

⇒ Two loop Brute force

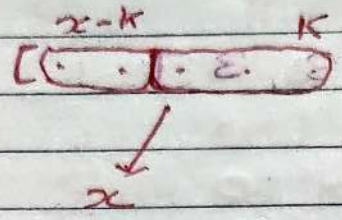
```
for (i = 0 → n)
    for (j = i → n)
        sum += nums[j];
        if (sum == k)
            max = max(max, j - i + 1);
```

} O(n^2)

Date ___/___/___
Page ___

Hashing

Prefix Sum



if there exist a subarray with Sum k as last element

arr [] = [1, 2, 3, 1, 2, 1, 4, 2, 3]

K = 3

map

9, 5	
8, 4	
7, 3	
6, 2	
3, 1	
1, 0	

len = 2

6 - 3 = 3 but len = 1

Don't update

9 - 6 = 3

6 at 2

∴ len for Sum = 3

5 - 2 = 3

∴ update len = 3

TC → O(N log N) + O(N)

SC → O(N)

Date ___/___/___
Page _____

⊙ Two Pointers

[1, 2, 3, 1, 1, 1, 3, 3]
 ↑↑
 i j

⇒ Sum += nums[j]
 j++;
 if (sum >= k)
 Sum -= nums[i]
 i++;

TC → O(2N)

Two Sum

two num in nums such that nums[i] + nums[j] = k

Brute force
 two for loop
 O(N²)

Hashing
[2, 6, 5, 8, 11] target = 14
at 2 search for 12 (14-2)

T(N) → O(N x log N) Sec → O(N)

Date ___/___/___
Page ___

Two Pointer

→ Sort the array

i = start
j = end
Sum > K
j--
Sum < K
i++

O(NlogN) + O(N) = T(c)
O(1) = S(c)

Sort 0's, 1's, 2's

arr[] = [0,1,2,0,1,2,1,2,0,0,0,1]

→ Brute force
Merge sort
T(c) → O(NlogN)
S(c) → O(N)

→ Taking i - Count
Cnt0 = 5
Cnt1 = 4
Cnt2 = 3
Set According

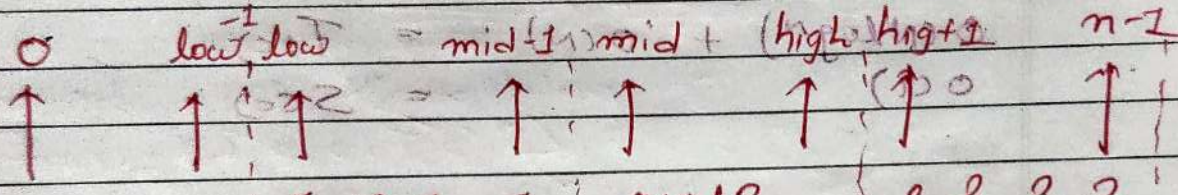
T(c) → O(N) + O(N) = O(2N)
S(c) → O(1)

Date ___/___/___
Page _____

Optimal (Dutch National Flag Algorithm)

arr[] = [0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0]

[0 ... low-1] => 0
[low .... mid-1] => 1
[high+1 .... n-1] => 2



arr[] = [0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0]

mid

high

0 Comes at low low++;
1 Comes at mid mid++;
2 Comes at high high--;

```
a[mid] == 0

    Swap(a[low], a[mid])
    low++;
    mid++;

a[mid] == 1

    mid++;

a[mid] == 2

    Swap(a[mid], a[high])
    high--;
```

T(n) → O(N)
S(n) → O(1)

Majority element

⇒ Brute force
have cnt for all (cnt > n/2)
$O(N^2)$

⇒ Better
Hashing
first Iteration Save
Second Iteration > n/2
O(N) + O(N) + O(NlogN)
O(2N) + O(NlogN)

Date ___/___/___
Page ___

○ Better -2

Sort nums
return nums[n/2];

O(N log N) Sorting only

○ optimal

Moore's Voting Algorithm

arr[] = [7, 7, 5, 7, 5, 1, 5, 7, 5, 5, 7, 7, 5, 5, 5, 5]

el = 7
Cnt = 0 1 2 1 2 1 0

7 7 5 7 5 1

el = 7
other
3 times
3 times

el = 5
Cnt = 0 1 0

5 7

el = 5
Cnt = 0 1 2 1 0

el = 5
cnt = 0 1 2 3 4

Date ____/____/____
Page ____

Rearrange array by sign

arr[] = [3, 1, -2, -5, 2, -4]

↓

arr[] = [3, -2, 1, -5, 2, -4]

Brute force

make two array
pos → Save Positive
neg → Save Negative

for(i=0 → n/2)
    arr[2×i] = pos[i]
    arr[2×i+1] = neg[i]

T(c) = O(N) + O(N)
S(c) = O(N)

Better

[3, 1, -2, -5, 2, -4]
 ↑
 i →

[, , , , ,]
 i

→ if pos ans[i] = nums[i]
          i += 2
  if Neg ans[j] = nums[i]
          j += 2

Next Permutation

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

⇒ make all Permutation
linear Search
Brute force
O(N! X N)

arr[] = [2 1 5 4 3 0 0]

Turn Just greater than 1
figure out break Point

Smaller ← Greater

2 3 remaining in Sorted order

[2 3 0 0 1 4 5]

```
for (i = n-2 → 0) {
    if (a[i] < a[i+1]) {
        ind = i
        break
    }
}
if (ind = -1)
    Reverse;

for (i = n-1 → ind) {
    if (curr[i] > arr[ind])
        swap(a[i], a[ind])
        Break;
    }
}
reverse (arr, ind+1, n-1);
```

ind

[2 3 5 4 1 0 0]

[2 1 5 4 1 0 0]

reverse

→ T(c)

O(N) + O(N) + O(N)

≈ O(3N)

Leaders in An array

⇒ Everything in right should be Smallers

:- leader

Date ___/___/___
Page ___

⇒ Brute force
Take one check all i on right
linear Search
$\approx O(N^2)$

⇒ optimal
at right
if a[i] > max
:- leader

Back iteration

```
maxi = INT_MIN;
for(i >= n-1 -> 0){
    if(a[i] > maxi){
        ans.Push_back(a[i]);
    }
    maxi = max(maxi, a[i]);
}
```

#

Longest Consecutive Sequence

arr[] = [102, 4, 100, 1, 101, 3, 2, 1, 1]

[1, 2, 3, 4]

[100, 101, 102]

⇒ for every element
Search x+1
Cnt = 2
if Count increase
find x+2

Brute force

= O(N²)

```
for (i=0 ; i<N ; i++) {
    x = arr[i]
    cnt = 1
    while (lsCarr, x+1) == true) {
        x = x+1 ;
        Cnt = Cnt+1 ;
```

}

Sort it

[100, 102, 100, 101, 101, 4, 3, 2, 3, 2, 1, 1, 1, 2]

[1, 1, 1, 2, 2, 2, 3, 3, 4, 100, 100, 101, 101, 102]

Date ___/___/___
Page ___

O(NlogN)

```
Sort()
longest = 1
last_smaller = INT_MIN
cnt_curr = 0

for(i=0 → n) {
  if(arr[i]-1 == last_smaller) {
      cnt = 1
      last_smaller = arr[i]
  }
  else if(arr[i] != last_smaller) {
      cnt = 1
      last_smaller = arr[i]
  }
  longest = max(longest, curr)
}
```

Optimal

Put everything in set = x

101	3	99	102
102 X	4 X	100 ↙ (3)	4
		↓	100
0		101	1
1 ↙ (4)		↓	101
)		102	3
2		↓	2
)			
3			
)			
4			

```
n == 0
    return 0;
longest = 1
unordered_set<int> st;
for (i = 0 → n)
    st.insert(a[i]);

for (auto it : st) {
    if (st.find(it-1) == st.end()){
        int cnt = 1;
        int x = it;
        while (st.find(x+1) != st.end) {
            x = x+1;
            cnt = cnt+1;
        }
        longest = max(cnt, longest)
    }
}
return longest;
```

T(c) ≈ O(N) + O(N) ≈ O(2N)

S(c) ≈ O(N)

Date ___/___/___
Page _____

Rotate matrix by 90 degreed

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

→

13	9	5	1
14	10	6	2
15	11	7	3
16	12	8	4

i j		j n-1-i
[0][0]	→	[0][3]
[0][1]	→	[1][3]
[0][2]	→	[2][3]
[0][3]	→	[3][3]

i j		j n-1-i
[1][0]	→	[0][2]
[1][1]	→	[1][1]
[1][2]	→	[2][2]
[1][3]	→	[3][2]

[i][j] → [j][n-1-i]

Transpose matrix

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

→ Rev all Rows

↓

[0][1] → [1][0]
[0][2] → [2][0]
[0][3] → [3][0]
[1][2] → [2][1]

0 → (1 to 3)
1 → (2 to 3)

i → (i+1 to n-1)

```
for( i=0 → n/2) {
    for (j = i+1 → n-i) {
        Swap(a[i][j], a[j][i]);
    }
```

Print Spiral matrix

	0 left	1	2	3 right
0 top	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13 bottom	14	15	16

O(N×N) → S(c)
O(N×N) → T(c)

= [1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10]

~~right~~
top = 0
bottom = 3

```
for (i = left → right)
    a[top][i]
top++;
for (i = top → bottom)
    a[i][right]
right--;
for (i = right → left)
    a[bottom][i]
bottom--;
```

```
for (i = bottom → top)
    a[i][left]
left++;
```

Date ___/___/___

Page _____

Subarray Sum equals K

arr[] = [1, 2, 3, -3, 1, 1, 1, 4, 2, -3]

K = 3

[1, 2]

[1, 2, 3, -3]

[3]

[2, 3, -3, 1]

[3, -3, 1, 1, 1]

[1, 1, 1]

[4, (2+3)]

[-3, 1+1+1, 4, 2, -3]

Brute force

⇒ generate all subarray

Sum, K, i, j

Count++;

```
for (i=0 → n) {
    for (j=i → n) {
        Sum = 0
        for (k=i → j) {
            Sum = Sum + arr[k]
        if (Sum == k)
            cnt++
    }
}
}
```

$\approx O(N^3)$

Date ___/___/___
Page ___

Optimal

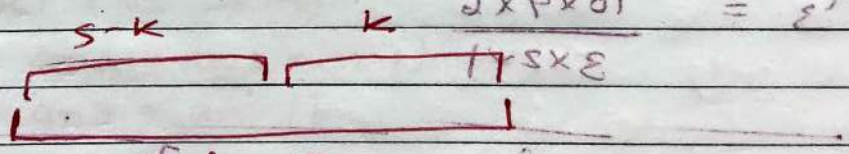
Prefix Sum

arr[] = [1, 2, 3, -3, 1, 1, 1, 4, 2, -3] k = 3

Prefix = 0 1 3 6 3 4 5 6 10 12 9

Cnt = 0

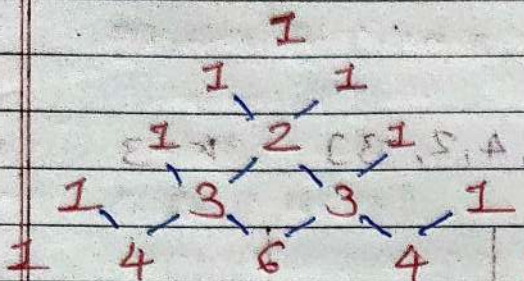
1-3 = -2 ✗		(9,1)
3-3 = 0 ✓	Cnt = 1	(2,1)
6-3 = 3 ✓	Cnt = 2	(10,1)
3-3 = 0 ✓	Cnt = 3	(5,1)
4-3 = 1 ✓	Cnt = 4	(4,2)
5-3 = 2 ✗		(6,2)
6-3 = 3 ✓	Cnt = 6	(3,2)
10-3 = 7 ✗		(7,1)
12-3 = 9 ✗		(0,1)
9-3 = 6 ✓	Cnt = 8	



Prefix (Sum; x<i ; 0=j)

```
map<int, int> mpp;
cnt = 0, Sum = 0;
for (i = 0 → n) {
    Sum += arr[i];
    remove = Sum - k;
    cnt += mpp[remove];
    mpp[Sum]++;
}
return cnt;
```

Pascal Triangle



⇒ given r = row no.
c = col no.
give element at r, c

$$ans = {}^{r-1}C_{(c-1)}$$

$${}^{7}C_{2} = \frac{7!}{2! \times 5!} = \frac{7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1}{2 \times 1 \times 5 \times 4 \times 3 \times 2 \times 1}$$

$${}^{10}C_{3} = \frac{10 \times 9 \times 6}{3 \times 2 \times 1}$$

```
for (i=0 ; i<r ; i++)
{
    res = res*(n-i) ;
    res = res / (i+1) ;
}
return res
```

⇒ Print given row

Nth row → N elements

```
for (c=1 → n) {
    Print (f(n-1, c-1)
}
```

O(N×N)

6th

1 5 10 10 5 1

$\frac{1}{1}, \frac{5}{1}, \frac{5\times4}{1\times2}, \frac{5\times4\times3}{1\times2\times3}, \frac{5\times4\times3\times2}{1\times2\times3\times4}, \frac{5\times4\times3\times2\times1}{1\times2\times3\times4\times5}$

ans = ans × row-col

```
for( i=1 → n)
    ans = ans * (n-i)
    ans = ans / i
    Print (ans);
}
```

O(N)

→ Print all

Generate rows using Second type

for all

Code at

"github / Vinayak501 / CPP-DSA / Pascal_triangle.cpp"

Majority element II

ans[] = [1, 1, 3, 3, 2, 2] , n = 8

$\lfloor \frac{n}{3} \rfloor = \lfloor \frac{8}{3} \rfloor = 2$

[1, 2] appearing more than 2 times

Ans only can (two * one = less) Integers

Brute force

check for all

O(N×N)

Date ___/___/___
Page ___

Better

→ Hashing

Number → Count

ls.add where

Count >= $\frac{n}{3}$

O(N) + O(N)

↑

Check while adding

we can remove this

O(N)

∴ O(N)

Space → O(N)

optimal

[2, 1, 1, 3, 1, 4, 5, 6]

cnt1 = 0 ✗ 0 1 cnt2 = 0 ✗ 2

el1 = 2 el2 = 1

cnt1 = 0, el1 = cnt2 = 0, el2

for (i = 0 → n-1) {

if (cnt1 == 0 && nums[i] != el2) {

cnt1 = 1, el1 = nums[i]

} else if (cnt2 == 0 && nums[i] != el1) {

cnt2 = 1, el2 = nums[i]

}

Date ___/___/___
Page ___

```
else if (el2 == nums[j])
    cnt2++;
else if (el2 == nums[i])
    cnt2++;
else
    cnt1 = 1;
    cnt2 = 1;
```

manual check on
el1, el2

#3Sum

arr[] = [-1, 0, 1, 2, -1, -4]

arr[i] + arr[j] + arr[k] = 0
j != i != k

All unique Triplets

Brute force
Try all elements

```
for (i = 0 → n)
    for (j = i+1 → n)
        for (k = j+1 → n)
```

$O(N^3)$

Date ___/___/___
Page ___

⇒

for eliminating (third loop) $O(N^3)$

for (i=0 → n)
  (for (j=i+1 → n)
    find min (arr[i] + arr[j]);

[br: Hashing

[-1, 0, 1, 2, 1, -4]
 ↑ ↑ ↑→↑→↑
 {j→ j j i

-(-1+0)
= +1 ×

j+1 to 2
store j

= -(-1+1)
= 0 ✓

= -(-1+2)
= -(1)
= -1 ×

= -(-1-1)
= -(-2)
= 2 ✓

2
0

(-1, 1, 0)
(-1, 2, 1)

⇒ After j make set empty
move i
move j from i+1

repeat this till i reaches end

Date ___/___/___
Page _____

```
Set<vector<int>>(&st);
for(int i = 0 -> n) {
    for e
    hashset<int> hashset;
    for (int j = i+1 -> n) {
        this = -(nums[i] + nums[j])
        (if(hashset.find(this)) {
            vector<int> temp
                = {temp[i], temp[j], third}
            sort(temp.begin(), temp.end())
            st.insert(temp);
        }
        hashset.insert(nums[j]);
    }
}
vector<vector<int>> ans(st.);
return ans;
```

⇒ TLE in Leetcode

$O(N^2 * \log N)$

Date ___/___/___
Page ___

Optimal

arr.sort();

[-2 -2 -2 -1 -1 -1 0 0 0 2 2 2 2]
 i j   j j k j k k

-2 + 2 - 2 = -2 ++j
-2 + 2 - 1 = -1 ++j
-2 + 2 + 0 = 0   ✓ j++ k++

move i, j, k Such it's ~~not~~ Same ele again

[-2 -2 -2 -1 -1 -1 0 0 0 2 2 2 2]
 i j   k

for (int i = 0 → n) {
  if (i > 0 && nums[i] == nums[i-1])
    continue;
  j = i+1, k = n-1
  while (k > j) {
    sum =
    if (sum < 0)
      j++;
    else if (sum > 0
      k--;
    else
      temp
      ans.add(temp)
      k--;

Date ___/___/___
Page _____

while (j<k && nums[j] == nums[j+1])
j++;
while (j<k && nums[k] == nums[k-1])

}

O(NlogN) + O(N²)

4Sum

arr[] = [1,0,-1,0,-2,2] target = 0

[-2,-1,0,0,1,2]
i j

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

four nested loops

T(C) O(N⁴)

nums[i] = target - (nums[i] + nums[j] + nums[k]);

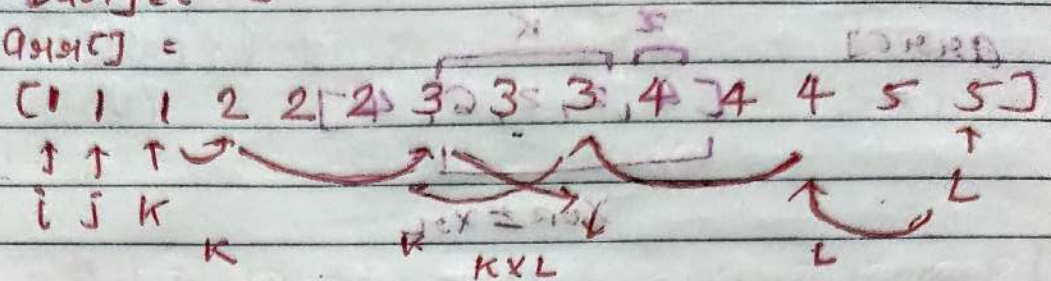
look for nums[i]

Use hashset like Before

Date / /
Page

target = 8

arr[] =



1+1+1+5 = 8 ✓

1+1+2+4 = 8 ✓

1+1+3+3 = 8 ✓

move j when K cross L

[1 1 1 2 2 2 3 3 3 4 4 4 5 5]

Same Process

code

"github.com/vinayak501/4Sum.cpp"

(8,21) (11,8) (2,1)

Subarray With XOR K

⇒ Brute force

all Subarray

XOR == K

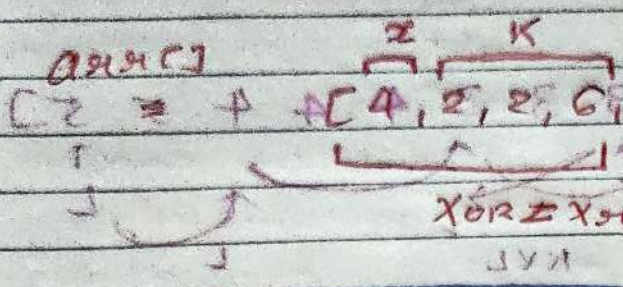
(8,21) (8,21) (11,8) (2,1) (4,5) (4,5) (5,1)

O(N²)

(11,8) (2,1) (4,1)

⇒ Same as Sum K

Date ___/___/___
Page ___



$X \wedge K = XR$

$(X \wedge K) \wedge K = XR \wedge K$

$X = XR \wedge K$

Merge overlapping Subproblems

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

1 2 3 4 5 6 7 8 9 10 11 15 16 17 18

(1,6) (8,11) (15,18)

Brute force

Sort()

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

(1,4) (8,10) (15,18)

(1,6) (8,11)

O(NlogN) + O(N) ≈ O(2N) ≈ T(C)

"github.com/Vinayak501/merge_interval.cpp"

```
Vector<Vector<int>> ans;
Sort(interval);
for(int i=0 → n){
  if(ans.empty() ||
      interval[i][0] > ans.back()[1]){
    ans.push_back(interval[i]);
  }else{
    ans.back()[1] = max(ans.back()[1], Interval[i][1])
  }
return ans;
```

merge two Sorted array without Extra Space

arr1[] = [1, 3, 5, 7] arr2[] = [0, 2, 6, 8, 9]

arr3[] = [0 1 2 3 5 6 7 8 9]

arr1[] = [0, 1, 2, 3]

arr2[] = [5, 6, 7, 8, 9]

Date ___/___/___
Page _____

gap method (shell sort)

[1,3,5,7] [0,2,6,8,9]

4 5

Initial gap = $\lceil 4.5 \rceil$

1 3 5 7 0 2 6 8 9

left right

gap = 5 = $\left\lceil \frac{5}{2} \right\rceil = 3$

1 3 5 7 0 2 6 8 9

left swap swap right swap

1 0 2 7 3 5 6 8 9

gap = $\left\lceil \frac{3}{2} \right\rceil = 2$ while (gap != 1)

Date ____/____/____
Page ____

Find repeating and missing Numbers

arr[] = [4 3 6 2 1 1] n = 6

repeating = 1
missing = 5

⇒ for every Number
(1 + Iteration + 1) —

0 time - missing
1 time - x
2 time - repeating

(i) ——

Brute force

T(n) = O(N²)

⇒ Hashing

First Iteration
Save into hasharray
Second array
Find repeating
Find missing

T(n) = O(N) + O(N)
= O(2N)

S(n) = O(N)

(ii) ——

Date ___/___/___
Page _____

Optimal

$x \rightarrow$ repeating
$y \rightarrow$ missing

arr = [4 3 6 2 1 1] $n = 6$

$(4+3+6+2+1+1) - (1+2+3+4+5+6)$

$= x - y$

$= n - s$

$= -4$

$\boxed{x - y = -4}$ — (i)

$(4^2+3^2+6^2+2^2+1^2+1^2) - (1^2+2^2+3^2+4^2+5^2+6^2)$

$= 1^2 - 5^2$

$= x^2 - y^2$

$= -24$

$x^2 - y^2 = -24$

$(x^2-y)(x+y) = -24$

$x + y = \frac{-24}{-4}$

$\boxed{x + y = 6}$ — (ii)

Date ____/____/____
Page ____

Sum both eqn

$2x = 2$

$\boxed{x = 1}$

$\boxed{y = 5}$

② Xor Method

arr = [4 3 6 2 1 1]

x → repeating
y → missing

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

$= 1 \wedge 5 = 4$

$x \wedge y = 4$

first 1 from right

bitNO = 2

0	1
3	4
2	6
1	4
1	5
1	6
2	
3	

	2	1	0	
	0	0	1	1
^	1	0	1	5
	1	0	0	4

Total xor of
first = 1
second = 5

Date ___/___/___
Page ___

Count Inversion

```
          0  1  2  3  4
arr[] = [5  3  2  4  1]
```

i < j ff a[i] > a[j]

(5,3) (3,2) (2,1) (4,1)
(5,2) (3,1)
(5,4)
(5,1)

} 8

Brute force

```
for (i=0 → n) {
    for (j=i+1 → n) {
        if (a[i] > a[j]) {
            cnt++;
        }
    }
}
```

O(N²)

○ optimal

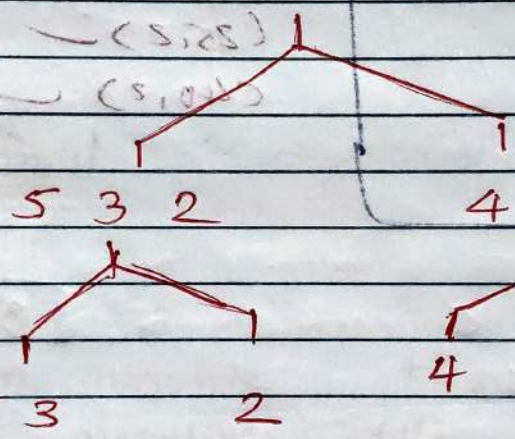
[2, 3, 5, 6] = n
i →

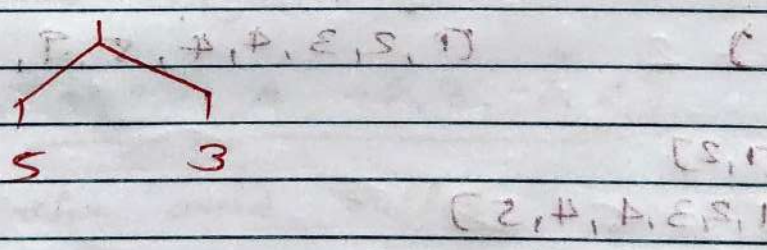
[2, 2, 4, 4, 8] = m
j ←

left > right
2 > 2 X i++
3 > 2 ✓ j++ cnt+=3;

Date:

3 > 2	✓	j++	cnt + 3
3 > 4	✗	i++	
5 > 4	✓	j++	cnt + 2
5 > 4	✓	j++	cnt + 2
5 > 8	✗	i++	
6 > 8	✗	i++	





5	3	5 > 3	+1
35	2	3 > 2	+2
235	1,4	2 > 1	+3
	1		2 > 4 ✗
			3 > 4 ✗
			5 > 4 ✓ +1

T(n) → O(N log N)

Reverse Pairs

arr[] = [40, 25, 19, 12, 9, 6, 2]

i < j ff a[i] > 2 × arr[j]

```
for (i=0 → n-1) {
    for (j=i+1 → n-1) {
        a[i] > 2 × a[j] {
            cnt++;
        }
    }
}
```

O(N²)

(6,2) ✓ (9,6) ✗
(9,2) ✓ (12,6) ✗
(12,2) ✓ 9 > 2×6 ✗
(19,2) ✓ 12 > 2×6 ✗
(25,2) ✓
(40,2) ✓

[6, 13, 21, 25]

[1, 2, 3, 4, 4, 5, 9, 12, 18]

6 → [1,2]
13 → [1,2,3,4,4,5]
21 → [1,2,3,4,4,5,9]
25 → [1,2,3,4,4,5,9,12]

Date:

Maximum Product Subarray

arr[] = [2, 3, -2, 4]

ans = 6

```
for (i=0 → n) {
    for (i=i+1 → n) {
        max(
    }
}
```

OCN²

Optimal

for +ve →

- even negative, +ve
- odd negative, +ve

[3, 2, -1, 4, -6, 3, -2, 6]

Prefix avoid Suffix

- it has Zeros

-2 3 4 -1 0 -2 3 1 4 0 4 6 -1 4

Prefix ~~3 6 -6 -24~~ (-24 x -6)

* github.com/vinayak501/Maximum_Product_Subarray.cpp"

apsara

Date:

Binary Search

0	1	2	3	4	5	6	7
3	4	6	7	9	12	16	17
↑ low			↑ mid				↑ high

target = 6

$mid = \frac{low + high}{2} = \frac{0+7}{2}$

arr[3] = 7 > 6

high = mid - 1

while (high ≥ low)

$mid = \frac{0+2}{2} = 1$

arr[1] = 4 < 6

low = mid + 1

$mid = \frac{2+2}{2} = 2$

arr[2] = 6 = 6 ✓

$O(\log n)$

```
f(arr, low, high, target) {
    if (arr high < low)
        return -1;
    mid = high+low / 2;
    if (arr[mid] == target)
        return mid;
    if (arr[mid] > target)
        f(arr, low, mid-1, target);
    else
        f(arr, mid+1, high, target);
```

Page No.:

lower Bound

→ Smallest index "i" such that

arr[i] >= target

[3, 5, 8, 15, 19] (indices 0 1 2 3 4)

x = 5

ans = 2

x = 9 ans = 3
x = 16 ans = 4
x = 20 ans = 5

[1 2 3 3 7 8 9 9 9 11] (indices 0 1 2 3 4 5 6 7 8 9) x = 1

low mid high

ans = 10, 4, 1, 0

$mid = \frac{0+9}{2} = 4$, 7 >= 1

high = mid - 1

$mid = \frac{0+3}{2} = 1$, 2 >= 1

high = mid - 1

$mid = \frac{0+0}{2} = 0$, 1 >= 1

high = mid - 1 = -1

lower bound = 0

github.com / DSA-CPP / lower_bound.cpp

First last occurrence of X

arr = {2, 4, 6, 8, 8, 8, 11, 13} x = 8

ans = {3, 5}

① Brute force

Linear Search

O(N)

① Optimal

lower_bound → a[ind] >= x
upper_bound → a[ind] > x

[2, 4, 6, 8, 8, 8, 11, 13]
lb up { lb , ub - 1 }
for x = 8

0 1 2 3 4 5 6 7
[2, 4, 6, 8, 8, 8, 11, 13]

x = 10   lb(10) = 11 (6)
x = 14   lb(14) = n (8)

if lb == n || arr[lb] != x
return {-1, -1}

① Optimal
→ Pure Binary Search

Date:

Search in Rotated Sorted array

0	1	2	3	4	5	6	7	8
7	8	9	1	2	3	4	5	6
low				mid				high

arr[] = [7 8 9 1 2 3 4 5 6], x = 1

Identify Sorted half

→ arr[low] < arr[mid]

should be but isn't

left is unsorted

right sorted

1 ∈ [7, 6] ? X false

↑ eliminate right side

low = 0

high = mid - 1 = 4 - 1 = 3

mid = $\frac{0+3}{2}$ = 1

arr[mid] = 8 > arr[low] = 7

left is Sorted

1 ∈ [7, 8] false

eliminate left

arr[high] > arr[mid] should Unsorted

```
f(arr, n, target)
{
  low = 0
  high = n-1
  while(low <= high)
  {
    mid = (low+high)/2
    if (arr[mid] == target)
      return mid
    // left sorted
    if (arr[low] <= arr[mid]) {
      if (arr[low] <= target
          && target <= arr[mid]) {
        high = mid-1;
      else
        low = mid+1;

    // right sorted
    else {
      if (arr[mid] <= target &&
          target <= arr[high]) {
        low = mid+1;
      else
        high = mid-1;
  }
  return -1
}
```

$O(n \log n)$

Date:

Search in Sorted Rotated Array II

arr[] = [7, 8, 1, 2, 3, 3, 3, 4, 5, 6]
target = 1

Same as before But have Duplicated elements
(True, false)

Identifying Sorted half doesn't work here

arr[] = [3 1 2 3 3 3 3 3]

have to Solve
nums[low] = nums[mid] = nums[high]

Condition

Shrink whenever we get this Condition

```
if (arr[low] == arr[mid] &&
    arr[mid] == arr[high])
{
    low++;
    high--;
    continue;
}
```

Find minimum Rotated Sorted array

arr[] = [4, 5, 6, 7, 0, 1, 2]

Linear O(N)

arr[] = [4 5 6 7 0 1 2]
l m h

Identify Sorted half
Sorted may / may not have
Unsorted have minimum

ans = INT_MIN, 4, 0, 2

In Sorted arr[low] is minimum

[4 5 6 7 0 1 2]
l m h

[4 5 6 7 0 1 2]
h
l
m

ans = INT_MAX, 2, 7, 1

[7, 8, 1, 2, 3, 4, 5, 6]
l m h

[7, 8, 1, 2, 3, 4, 5, 6]
l m h

[7, 8, 1, 2, 3, 4, 5, 6]
h
l
m

```
f(arr, n)
{
   low = 0
   high = n-1,
   ans = INT_MAX
   while (low <= high)
   {
      mid = low + high / 2
   if (a[low] <= a[mid]) {
         ans = min(ans, arr[low])
         low = min+1
   else
         ans = min(ans, arr[mid])
         high = mid-1
   }
}
```

O(log n)

```
if (arr[low] <= arr[high])
    direct return arr[low];
```

Find out How many time Array been Rotated

arr[] = [3, 4, 5, 1, 2]

ans = 3

Index of minimum element will be ans

0 1 2 3
[3, 4, 5, 1, 2]

ans = 3

Single element in Sorted array

arr[] = [1 1 2 2 3 3 4 5 5 6 6]

element

arr[i-1] != arr[i] != arr[i+1]

return i;

O(N)

Eliminate i/2

0 1 2 3 4 5 6 7 8 9 10
[1 1 2 2 3 3 4 5 5 6 6]

even, odd → left half (ans - right)

Odd, even → right half (ans - left)

```
f (arr, n)
{
  n==1
    return a[0];
  if (arr[0] != arr[1])
      return arr[0];
  if (arr[n-1] != arr[n-2])
      return arr[n-1];
  low = 1
  high = n-1;
  while (low <= high) {
      mid = low+high/2;
      if( arr[mid] != arr[mid+1] ff
          arr[mid] != arr[mid-1] )
          return arr[mid];

      if((mid%2==1 ff arr[mid-1] ==
                  arr[mid]) {
          low = mid+1;
      or mid%2==0 ff
          arr[mid] == arr[mid+1]
          low = mid+1;
      else
          high = mid-1
  }
  return -1
```

apsara

Date:

Peak element

$-\infty$

arr[] = [1 2 3 4 5 6 7 8 5 1]

l ↑ hi

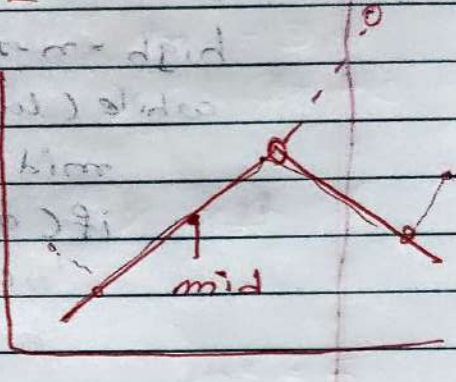
nums[i-1] < nums[i] > nums[i+1]

Assume only 1 Peak

4 < 5 < 6

↑ Peak will be here

Peak can't be on increasing curve




```
f(arr,n)
{
if (n==1) return 0;
if (arr[0] > arr[1]) return 0;
if (arr[n-1] > arr[n-2]) return n-1;
low = 1; high = n-2;
while (low <= high) {
    mid = (low+high)/2;
    check if Peak return mid
    else if (arr[mid] > arr[mid-1])
        low = mid+1
    else if (arr[mid] > arr[mid+1])
        hig = mid-1
}
```

Find in Sqrt of Integer

$n = 25$ $\sqrt{25} = 5$

$n = 30$ $\sqrt{30} = 5$

$n = 35$ $\sqrt{35} = 5$

$n = 36$ $\sqrt{36} = 6$

⇒ Brute force

check for all 1 to n

if in Search

$a \times a > n$

$(a-1)$ = ans

```
for (i = 0 → n) {
    if (i*i <= n)
        ans = i;
    else
        break;
}
return ans;
```

⇒ BS

low = 1 high = 28

mid = 14

$14 \times 14 > 28$

go left

Date ___/___/___
Page ___

low = 1 high = 13

mid = 7

7 × 7 > 28

go left

low = 1 high = 6

mid = 3

3 × 3 < 28

go right

ans = 3

low = 4 high = 6

mid = 5

5 × 5 < 28

ans = 5

low = 6 high = 6

mid = 6

6 × 6 > 28

ans = 5

```
low = 1
high = n-1
while (high >= low) {
    mid = high+low / 2;
    if (mid*mid <= n)
        ans = mid
        low = mid+1;
    else
        high = mid-1;
```

Find m^{th} root of N

$N = 3^3 \quad \therefore m = 27 \quad \sqrt[3]{27} = 3$

$N = 4 \quad m = 69 \quad \sqrt[4]{69} = -1$

⇒ Brute force

⇒ linear Search

low = 1 high = 27

mid = 14

$14 \times 14 > 14 > 27$

go left

low = 1 high = 13

mid = 7

$7 \times 7 \times 7 \times 7 > 27$

go left

low = 1 high = 6

mid = 3

$3 \times 3 \times 3 \times 3 = 81$

return

$T(c) \rightarrow \log_2 n \times \log_2 m$

Koko eating Bananas

Piles [] = [3, 6, 7, 11] h = 8

→ Return the min Integer k such that koko can eat all bananas within h hours

[3, 6, 7, 11]
↑ ↑ ↑ ↑

k = banana/hour

2 hour 3 4 6 hr

= 15 hrs < 8

Now for k = 3

[3, 6, 7, 11]
↑ ↑ ↑ ↑
1 2 3 4

= 20 < 8

Now for k = 4

[3, 6, 7, 11]
↑ ↑ ↑ ↑
1 2 2 3

= 8 hour

Date ___/___/___
Page ___

Brute force

Start from
I → Increase
where Satisfied
return

I to maximum banana

```
for(i=1 → max(piles)) {
    reqTime = fun(arr, i);
}
fun(arr, hourly) {
    totalTime = 0
    for(i=0 → n) {
        totalTime = ceil(arr[i] / hourly)
    }
    return totalTime;
}
if(reqTime <= h)
    return i;
}
```

O(max(piles) * n)

here the same for checking but loop can be replaced by Binary Search

1 6 11
low mid high

check for 6
[3, 6, 7, 11]
1 + 1 + 2 + 2 = 6
8 (>) 6 ← Possible
go left ans = 6

1 3 5
low mid high
check for 3
[3, 6, 7, 11]
1 + 2 + 3 + 4 = 10
10 > 8
go right

4 5
low mid high
check for 4
[3, 6, 7, 11]
1 + 2 + 2 + 3 = 8
8 = 8
possible
4 < 6
ans = 4

Min Days to make m bouquets

bloomDay = [7, 7, 7, 7, 11, 11, 12, 7]

1st flower bloom on 7th day

m = 2 → no. of Bouquets

k = 3 → adjust flowers required

at 13th day all bloomed

Possible but we need minimum.

- Brute force

check for n-1 to 1

pick (min possible)

T(c) → O(maxi - mini + 1) × N

- BS

checking will be same but days will be selected upon BS

T(c) → O(log^(maxi-mini+1) × N)

Date ___/___/___
Page _____

if m*k > bloomDay.size();
    return -1;

low = min(bloo) mid high = max(bloomDay)

7 10 13
Not Possible

11 12 13
Possible ans = 12

11 11 11
not possible

12
end

ans = 12

```
Possible (arr[], day, m, k) {
    cnt = 0; conform = 0;
    for (i=0 → n-1) {
        if (arr[i] <= day) {
            cnt++;
        } else {
            conform += cnt/k;
            cnt = 0;
        }
    }
    conform += cnt/k;
    return (conform >= m);
}
```

```
bs (arr, k, m)
{
    low = min(arr)
    high = max(arr)
    ans = high;
    while (high >= low) {
        mid = low + high / 2;
        if (possible(arr, mid, m, k)) {
            ans = mid;
            high = mid-1;
        else
            low = mid+1;
    }
return ans;
}
```

Find Smallest divisor given a threshhold

arr[] = [1 2 5 9] threshold = 6

div = 4

$\frac{1}{4}$ $\frac{2}{4}$ $\frac{5}{4}$ $\frac{9}{4}$

1 + 1 + 2 + 3 = 7 $\not\leq$ 6

×

Not Possible

Date ___/___/___
Page _____

div = 5

[1 2 5 9]

$\frac{1}{5}$ $\frac{2}{5}$ $\frac{5}{5}$

1 + 1 + 1 + 2 = 5 ✓

Possible

- Brute force
  linear search of Possibility
  for
  1 to max(arr)

- checking will be same
  But on BS

- BS

low	mid	high
1	5	9
	Possible	ans = 5
1	2	4
	Not possible	
3	3	4
	Not Possible	
4	4	4
	Not Possible	
5		4
	end	

ans = 5

```
f (arr, threshold)
{
    low = 1;
    high = max(arr);
    while (high >= low) {
        mid = high + low / 2;
        if (Possible(arr, mid)
                <= threshold) {
            ans = mid;
            high = mid - 1;
        else
            low = mid + 1;
    }
    return ans;
}

possible (arr, mid) {
    for (i = 0 → n-1) {
        Sum += ceil (arr[i] / mid)
    }
    return Sum;
}
```

$T(n) \rightarrow O(\log max \times N)$

Least Capacity to Shir Packages within D Days

days = 5

weight[] = [1 2 3 4 5 6 7 8 9 10]

Ship runs one full per day in 5 days. Ship call Product Min Capacity of Ship needed

Cap = 100

In one day

If Cap = 10

first day → {1, 2, 3, 4}
seco day → {5}
third day → {6}
4th day → {7}
5th day → {8}

{9, 10} remaining

Increase Cap

Cap = 15

1st → {1, 2, 3, 4, 5}
2nd → {6, 7}
3rd → {8}
4th → {9}
5th → {10}

done.

14 Not possible ans = 14

Date ____/____/____
Page ____

min
Capacity = max (weight)
max
Sum of all weight

```
daysReq (wt, cap) {
    day = 1;
    load = 0;
    for (i = 0 → n-1) {
        if (load + wt[i] > cap) {
            day = day + 1;
            load = wt[i]
        else
            load += wt[i];
    }
    return days;
}
```

- Brute force

```
for (i = min(wt) → Sum(wt)) {
    if (dayReq(wt, i) <= day) {
        return i;
    }
}
```

$O((\min(wt) + Sum(wt)) \times N)$

$\approx O(N^2)$

Date ___/___/___
Page _____

BS

low	mid	high
10	32	55
	Possible	ans = 32
10	20	31
	Possible	ans = 20
10	14	19
	Not possible	
~~10~~	~~14~~	~~19~~
14	17	19
	possible	ans = 17
14	15	16
	Possible	ans = 15
14	14	14
	Not possible	
15		14

end

ans = 15

```
f(wt, days)
{
    low = max(wt), high = sum(wt);
    while (low <= high) {
        mid = low + high / 2;
        NoofDays = DaysReq(wt, mid);
        if (NoofDays <= days)
            high = mid - 1;
        else
            low = mid + 1;
    }
    return low;
}
```

Find k^{th} Missing Number

arr[] = [2 3 4 7 11] k = 5

1	2	3	4	5	6	7	8	9	10	11
↑				↑	↑		↑	↑	↑	
1				2	3		4	5	c	
								↑		
								ans		

Brute force

[2 3 4 7 11] k = 5 6 7 8 9

↑ ↑ ↑ ↑ ↑

```
for (i = 0 → n-1) {
    if (arr[i] <= k) k++;
    else break;
}
```

BS

[2 3 4 7 11] k = 5

1 1 1 3 c

11 − 5 = 6 missing

7 − 4 = 3 missing etc

Date ___/___/___
Page _____

0	1	2	3	4
[2	3	4	7	11]
1	1	1	3	6

low mid high

$\frac{0+4}{2} = 2$

1 < 5

low mid high

$\frac{3+4}{2} = 3$

3 < 5

low high mid

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

6 > 5

high low → nearby indexes we got

arr[high] → 7, 8, 9

missing = 3 more = 2

high + more

= 7 + 2

Date ___/___/___
Page ___

Aggresive Cows

N stall

arr[] = [0 3 4 7 10 9] Cows = 4

→ Place 4 cows in such way min distance blw two cows is maximum

arr[] is coordinates of stalls

0 3 4 7 9 10

$c_1 - c_2 - c_3 - c_4$

3 1 3 min = 1

c_1 —(4)— c_2 —(5)— c_3 —(1)— c_4] min = 1

c_1 —(4)— $c_2 - c_3$ —(3)— c_4] min = 3

ans = 3

Sort

→

0	3	4	7	9	10	
c_1	c_2	c_3	c_4			dis = 1 ✓
c_1	c_2		c_3	c_4		dis = 2 ✓
c_1	c_2		c_3		c_4	dis = 3 ✓
c_1		c_2		c_3		dis = 4 X

ans = 3

≈ $O(N^2)$

dis

[1, max-min]

```
bool canWePlace(arr, dist, cows)
{
    cntCows = 1;
    coord = arr[0];
    for (i = 1 → n-1) {
        if (arr[i] - coord) >= dist) {
            cntCows++;
            last = coord = arr[i];
        }
    }
    if (cntCows >= cows) return true;
    return false;
}
```

Brute force

```
for (i = 1; i <= max-min; i++) {
    if (canWePlace(arr, i, cows)) {
        continue;
    }
    else return i-1;
}
```

O(max-min) x N)

(≈ O(N²))

low	mid	high
1	5	10
	Not possible	
1	2	4
	Possible	ans = 2
3	3	4
	Possible	ans = 3
4	4	4
	Not possible	
4	end	3
	ans = 3	

return high;

```
f(arr, cows)
{
   Sort
   low = 0
   high = a[n-1] - a[0];
   while (low <= high) {
      mid = low + high / 2;
      if (canweplace(arr, mid, cows)) {
            ans = mid;
            low = mid + 1;
      else
         high = mid - 1;
   }
return high;
}
```

O(N log N)

Allocate Books

arr[] = [25 46 28 49 24]
Student = 4

Allocate such that maximum no. of Pages Student get is min

25 | 46 | 28 | 49 | 24 = 73
25 | 46 | 28 49 | 24 = 77
25 | 46 | 28 | 49 | 24 = 74
25 | 46 | 28 | 49 | 24 = 71

ans

Keep Pages start from max

49
1 → 25 +46 > 49 X
2 → 46 +28 > 49 X
3 → 28 +49 > 49 X
4 → 49 +24 > 49 X
5 → 24

5 > student (4)
Increase Pages untill it's possible

① Brute force

= start with [max (Pages)
keep Increasing untill
Sum (Pages)

```
for (Pages = Max → Sum)
{
    Cnt Stu = fun (arr, Pages);
    if (Cal student == m) {
        return Pages
    }
}
```

```
fun (arr, Pages)
{
    Stu = 1;
    Pagesstudent = 0;
    for (i = 0 → n-1) {
        if (Pagesstu + arr[i] <= Pages) {
            Pagesstu += arr[i]
        } else {
            Pagestu = 0;
            Stu++;
        }
    }
    return Stu;
}
```

$O((sum - max) \times N)$

$\approx O(N^2)$

Date ___/___/___
Page _____

low	mid	high
49 (5)	110 (2)	172 (1)
	Possible	
49 (5)	78 (3)	109 (2)
	Possible	
49 (5)	63 (5)	77 (3)
	Not possible	
64 (5)	69 (5)	77 (3)
	Not possible	
70 (5)	73 (4)	77 (3)
	Possible	ans = 73
70 (5)	71 (4)	72 (4)
	possible	ans = 71
70	70 (5)	70
	Not possible	
71		70

end ()

ans = 71

return low;

```
while (low <= high) {
    mid = low + high/2
    NoStu = fn (arr, mid)
    if (noStu > m)
        low = mid + 1
    else
        high = mid - 1;
}
return low;
```

≈ O(N log N)

Date ____/____/____
Page ____

Painters Partition / split array largest Sum

arr[] = [10 20 30 40] k = 2

[10] [20 30 40] 90 min Painters
[10 20] [30 40] 70 min
[10 20 30] [40] 60 min

Painter 1 Painter 2

max Subarray Sum should be min

⇒ Same as before

Minimize maximum Distance to Gas Station

arr[] = [1 2 3 4 5] K = 4
place 4 new gas station

= [1 2 3 4 5 6 7 8 9] 1

1 1 1 1 1 1 1 1

[1 1.25 2 2.5 3 3.5 4 4.5 5]

0.5 0.5 = 0.5

Date ___/___/___
Page _____

arr[] = [1 3 5 7] k = 2

[1 3 5 7] ans = 2

2 2 2

arr[] = [1 13 17 23] k = 5

12 4 6

1 7 13 17 23 k = 4

6 6 4 6

1 7 13 17 20 23 k = 3

6 6 4 3 3

1 5 9 13 17 20 23 k = 2

4 4 4 4 3 3

20 to 27 Baki

SKIP

Two Pointers

Sliding Window

Date ___/___/___
Page ___

1. Constant Window

arr[] = [-1 2 3 3 4 5 -1] k = 4

l = 0 r = k-1

Shift window to find max sum of consecutive (4) elements

```
Sum = 7
while (r < n-1)
 {
   Sum = Sum - arr[l]
   l++;
   r++;
   Sum = Sum + arr[r];
   max = max(Sum, max);
 }
return max;
```

2. Longest Substring / Subarray where <Condition>

Longest Subarray With Sum <= k

arr[] = [2, 5, 1, 7, 10] k = 14

Date ___/___/___
Page ___

```
l=0   r=0   sum=0
max=0
while (r<n)
{
    sum = sum + arr[r];
While (sum>k) {
        sum = sum - arr[l]
        l++;
    }
    if (sum<=k)
        max = max(max, r-l+1);
    r++;
}
```

for optimizing :- If we get mx length 4 don't let the window shring more than 3 as we only want ans >= 3.

3. No. of Subarray Where <condition>

4. Shortes / minimum Window. <condition>

Max Points you Can obtain Cards

arr[] = [6 2 3 4 7 2 1 7 1] k = 4

Pick up from start or end

[6 2 3 1] = 12
[6 2 7 1] = 16 ans
[6 2 3 4] = 15

lsum = 0 15 rsum = 0 15
lsum = 11 rsum = 1 12
lsum = 8 rsum = 8 16
lsum = 6 rsum = 9 15
lsum = 0 rsum = 11 11

```
f(nums, k)
{
    lsum = 0, rsum = 0, rind = n-1;
    for(i=0 → k-1) {
        lsum += nums[i]
    }
    maxsum = lsum;
    for(i=k-1 → 0) {
        lsum -= nums[i];
        rsum += nums[rind];
        rind--;
        maxSum = max(maxSum, lsum + rsum);
    }
}
```

Date ___/___/___
Page ___

Longest Substring without Repeating

s = cadbzabcd

ans = cadbz

○ Brute force

Generate all

check

```
for (i = 0 → n-1) {
    hash[256] = {0}
    for (j = i → n-1) {
        if (hash[s[j]] == 1) break;
        len = j - i + 1
        maxlen = max(maxlen, len)
        hash[s[j]] = 1;
    }
}
```

$O(N^2)$ $O(256)$

○ optimal

```
l
s = cadbzabcd
    ↑↑↑↑↑↑↑↑↑
    r
```

a is at 1st index l have to go

l = mpp[s[r]] + 1

b is at 3rd index l have to go

l = mpp[s[r]] + 1

char	index
c	0
a	1
d	2
b	3
z	4

len = r - l + 1

maxlen = 0, 1, 2, 3, 4, 5

```
fun (String s) {
    hash[256] = {-1}
    l = 0    r = 0    maxlen = 0
    while (r < n)
    {
        if (hash[s[r]] != -1) {
            if (hash[s[r]] >= l) {
                l = hash[s[r]] + 1;
            }
        }
        len = r - l + 1;
        maxlen = max(len, maxlen);
        hash[s[r]] = r;
        r++;
    }
    return maxlen;
}
```

O(N) O(256)

Max Consecutive ones III

arr[] = [1 1 1 0 0 0 1 1 1 1 0] k = 2

allowed to make k Zeros to ones

[1 1 1 0 0 0 1 1 1 1 0]

5

6

6

Date ___/___/___
Page _____

For that find longest Subarray with K Zeros

Brute force

Two for loops
all Combination
Containing K Zeros

O(N²)

Optimal

```
0 1 2 3 4 5 6 7 8 9 10
[1 1 1 0 0 0 1 1 1 1 0]
 l r →
```

Zeros = 0 1 2 3 2

len	maxlen
= r - l + 1	= 0
= 0 - 0 + 1	= 1
= 1 - 0 + 1	= 2
= 2 - 0 + 1	= 3
= 3 - 0 + 1	= 4
= 4 - 0 + 1	= 5

l++, l++, l++, l++ Zeros = 2

= 6 - 4 + 1 = 3

= 7 - 4 + 1 = 4 [0 0 0 1 1 1]

= 8 - 4 + 1 = 5

= 9 - 4 + 1 = 6

-6

```
fun (nums, k)
{
    maxlen = 0   l = 0, r = 0, zeros = 0
    while (r < n)
    {
      if (nums[r] == 0)
          zeros++;
      while (zeros > k) {
          if (arr[left] == 0)
              zeros--;
          left++;
      }
      if (zeros <= k) {
          len = r - l + 1;
          maxlen = max(len, maxlen);
      }
      r++;
    }
    return maxlen;
}
```

T(n) = O(N + N) = O(2N) S(n) = O(1)

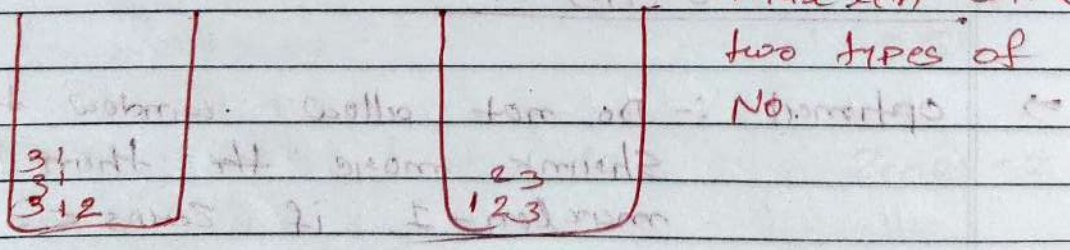
⇒ optimal :- Do not allow window to shrink more than maxlen - 1. if zeros <= k update else not

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

```
func (nums, k) {
    l=0, r=0, zeros=0, maxlen=0
    while (r<n) {
        if (nums[r]==0) zeros++;
        if (zeros > k) {
            if (nums[l]==0) zeros--;
            l++;
        }
        if (zeros<=k) {
            len = r-l+1;
            maxlen update
        }
        r++;
    }
    return maxlen;
}
```

Fruit into Basket

arr[] = [3 3 3 1 2 1 1 2 3 3 4]

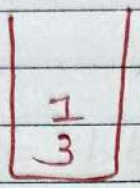


one Basket store only one type

Date ____ / ____ / ____
Page ____

arr[] = [3 3 3 1 2 1 1 2 3 3 4]

Set



Size should be
$x = k, 2$

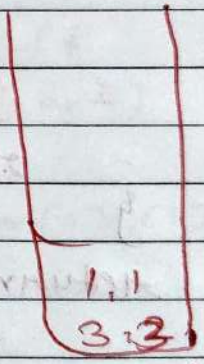
- Brute force
  Double for loop
  $= O(N^2)$ $= O(2)$

- Sliding Window

[3 3 3 1 2 1 1 2 3 3 4]
l,r r

maxLen = 0 1 2 3 4

if any num → 0
remove
move l



num, freq

Date ___/___/___
Page _____

```
fun (arr){
    l=0, r=0, maxlen=0
    mpp
    while ( r < arr.size())
    {
        mpp(arr[r]]++;
        if (mpp.size() >k) {
            while (mpp.size() > k) {
                mpp(arr[l]) (--);
                if (mpp(arr[l]) ==0)
                    mpp.erase(arr[l]);
                l++;
            }
        }
        if (mpp.size() <=k) {
            maxlen = max(maxlen,
                        r-l+1);
        }
        right++;
    }
    return maxlen;
}
```

T(c) → O(N+N)
= O(2N)

S(c) = O(1)

⇒ optimize by stucking window at max

T(c) → O(N)

S(c) → O(3)

Date / /

Date / /
Page

Number of Substring Containing All three chars

S = bbacbca ans = 9

Brute force
two loops
TC. O(N²)

```
fun(S){
  lastSeen[3] = {-1,-1,-1};
  cnt = 0;
  for(i=0 → n)
  {
    lastSeen[S[i]-'a'] = i;
    if(lastSeen[0] != -1 && all are Present)
      cnt = cnt + (1 + min(lastSeen[0],[1],[2]))
  }
  return cnt;
}
```

Date ___/___/___

Date ___/___/___
Page ___

Longest Repeating Character Replacement

S = AABABBA K = 2

= AAAAABA ans = 5

A → 3
B → 2 → Convert this

```
for (i = 0 → n) {
    hash[26] = {0}
    maxf = 0
    for (j = i → n) {
        hash[s[j] - 'a']++ ;
        maxf = max(maxf, hash[s[j]-'a']);
        changes = (j - i + 1) - maxf;
        if (changes <= k)
            maxlen = max(maxlen, j - i + 1)
        else
            Break;
    }
}
return maxlen;
```

≈ O(N²) O(26)

Date / /

Date / /
Page

S = A A A B B C C D K = 2
l r

len - maxf <= K

maxf = 0 1 2

maxlen = 0 1 2 3 4 5

C, 1
B, 12
A, 1 2 3 2

```
fun (String s, k)
{
    l=0, r=0, maxlen=0, maxf=0
    hash[26] = {0};
    while (r < s.size())
    {
        hash[s[r] - 'A']++;
        maxf = max(maxf, hash[s[r] - 'A'])
        while ((r-l+1) - maxf > k)
        {
            hash[s[r] - 'A']--;
            maxf = 0
            for(i=0 -> 25) {
                maxf = max(maxf, hash[i]);
                l++;
            }
            if (r-l+1 - maxf <= k) {
                maxlen = max(maxlen
                    r-l+1);
            }
        }
    }
}
```

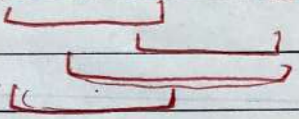
Date ___/___/___

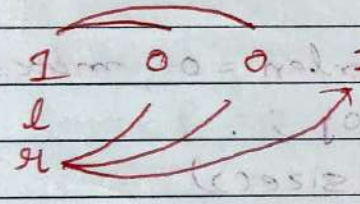
Date ___/___/___
Page _____

$T(c) = O(N+N) \times 26$ $\quad T(c) = O(26)$

Binary Subarray With Sum k

arr [] = [1 0 1 0 1] $\quad$ ans = 4





goal = 2

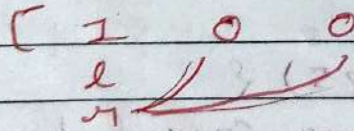
Sum = 0 1 2 3 2

cnt = 0 1 2

Not working

don't know if move l or r

- No. of Subarray where Sum <= goal



goal = 2

Sum = 0 1 2 3 2

cnt = 0 1 + 2 + 3 + 4 + 4 + 5

total 19 Subarray <= goal

Date ___/___/___
Page ___

```
fun ( nums , goal)
{
    l = 0, r = 0, sum = 0, cnt = 0
    while (r < n)
    {
        sum += nums[r];
        while (sum > goal)
        {
            sum = sum - nums[l];
            l++;
        }
        cnt = cnt + (r - l + 1);
        r++;
    }
}

return fun (nums, goal) -
        fun (nums, goal - 1);
```

O(2N) x O(2N) + O(1)

O(4N)

Date ___/___/___

Date ___/___/___
Page ___

Subarray with k diff Integers

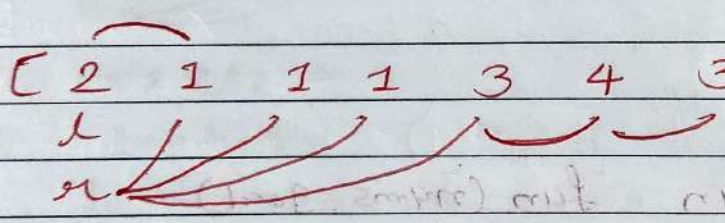
arr[] = [1 2 1 3 4] k=3

ans = 3

Brute force
Two for loops
$O(N^2)$

① optimal Sliding Window

[2 1 1 1 3 4 3 2] k=3



cnt = 12

Same Problem to move r or to move l.

4 → 1
3 → 12
1 → 123
2 → 1

num → freq

Date ___/___/___
Page ___

no. of Subarray where different Integer <=k

[2 1 1 1 3 4 3 2]

cnt = 0+1+2+3+4+5

cnt += r-l+1

4 → 1
3 → 1
1, 1=3
2, 1

```
fun (nums, k)
{
    l=0, r=0, cnt=0, mpp;
    while (r<n) {
        mpp[nums[r]]++;
        while (mpp.size() >= k)
        {
            mpp[nums[l]]--;
            if (mpp[nums[l]] == 0)
                mpp.erase(nums[l]);
            l++;
        }
        cnt += r-l+1;
    }
    return cnt;
}
```

Date ___/___/___

Date ___/___/___
Page ___

return fun(k) - fun(k-1);

T(c) → 2(O(N) + O(N)) → O(4N)

minimum Window Substring

S = ddaaabbcca t = abc

daaa x
aaabbc
Smallest → bca

Smallest Substring Containing all this

ans = bca

Brute force
Two loop
O(N²) + other

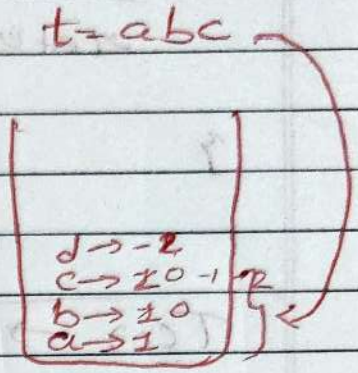
Optimal

Sliding Window

Date ___/___/___
Page ___

s = d d a a a a b b e a

t = a b c



cnt = 0 1 2 3 2 3 2

minlen = 10^9 8 7 6 5 4 4 3

sInd = 1 0 1 2 3 4 5 6

```
fun( s,t)
{
    hash[256] = {0};
    l=0 , r=0
    min = 10^9 , sIn = -1 , cnt=0
    n=s.size(); m= t.size();
    for (i=0→m)
        hash[t[i]++];
    while (r<n){
      if (hash[s[r]]>0)
          cnt = cnt+1;
          hash[s[r]]--;
      while(cnt==m)
          minlen = min(minlen, r-l+1);
          sInd = l;
          hash[s[l]]--;
          if (hash[s[l]]>0){
              cnt=cnt-1;
    }
    r++;
}
```

Date ____/____/____

Date ____/____/____
Page _____

```
}
return sInd == -1 ? "" :
    s.Substring(sInd, minLen);
}
```

$T(n) \rightarrow O(N) + O(N) + O(m)$

$= O(2N) + O(m)$

$S(n) \rightarrow O(256)$