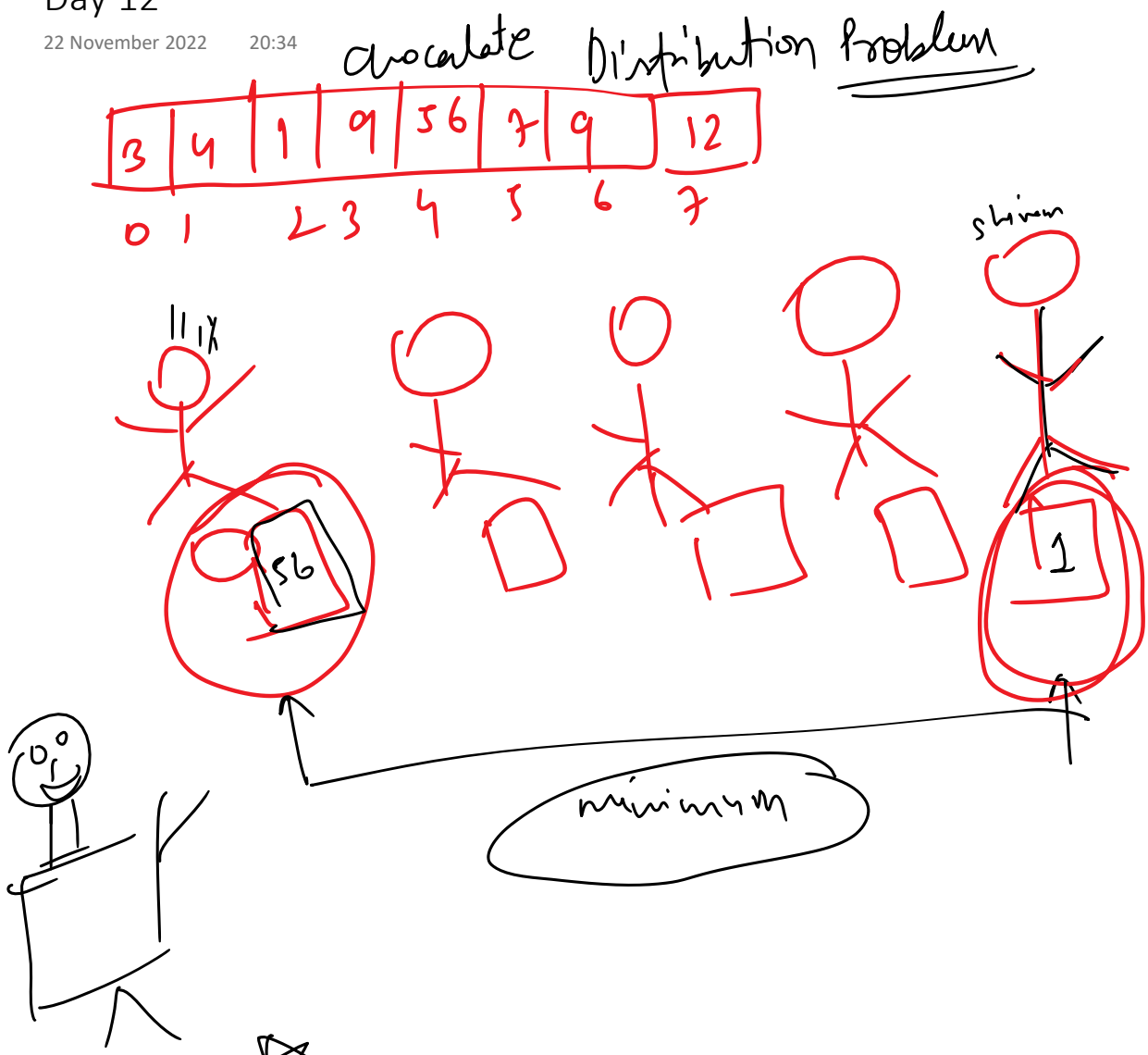
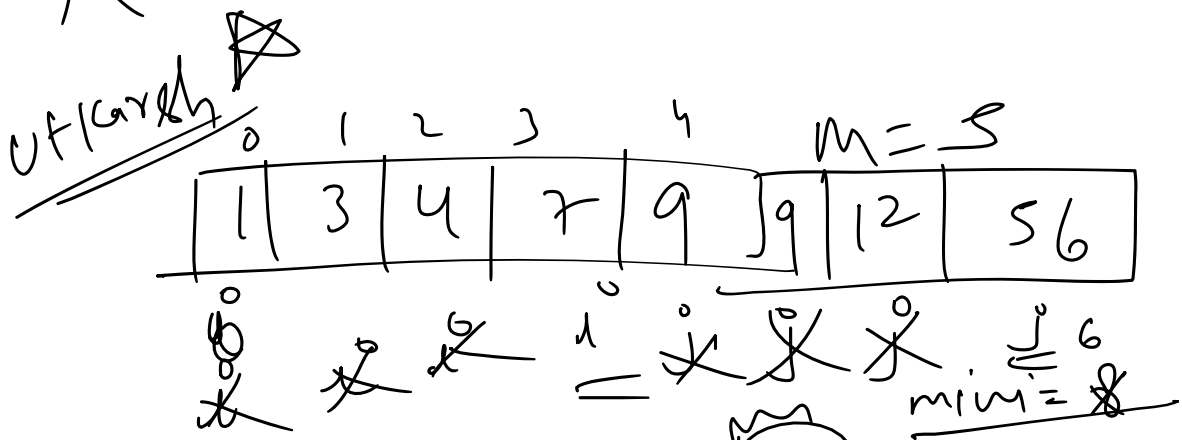
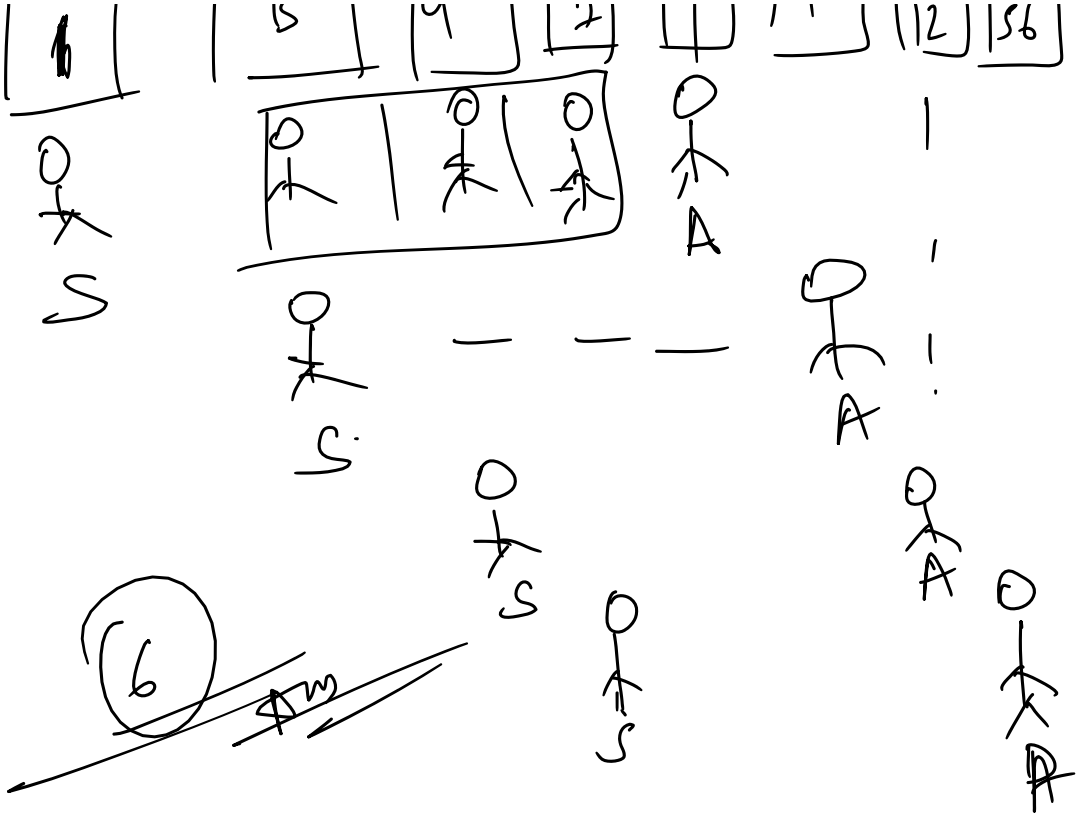
Day 12

22 November 2022 20:34






```
long long findMinDiff(vector<long long> a, long long n, long long m){
   sort(a.begin(),a.end());
   int i = 0,j = m-1;
   long long int mini = INT_MAX;
   while(j < n){
       mini = min(mini,a[j]-a[i]);
       i++;j++;
   }
   return mini;
}
```



```cpp
long long findMinDiff(vector<long long> a, long long n, long long m){
    sort(a.begin(),a.end());
    int i = 0,j = m-1;
    long long int mini = INT_MAX;
    while(j < n){
        mini = min(mini,a[j]-a[i]);
        i++;j++;
    }
    return mini;
}
```

nlogn

n

T.C- O(n log n)

S.C- O(1)

Clue

Smallest subarray with sum > x

A[] = {1, 4, 45, 6, 0, 19}

Smallest subarray with sum greater than x

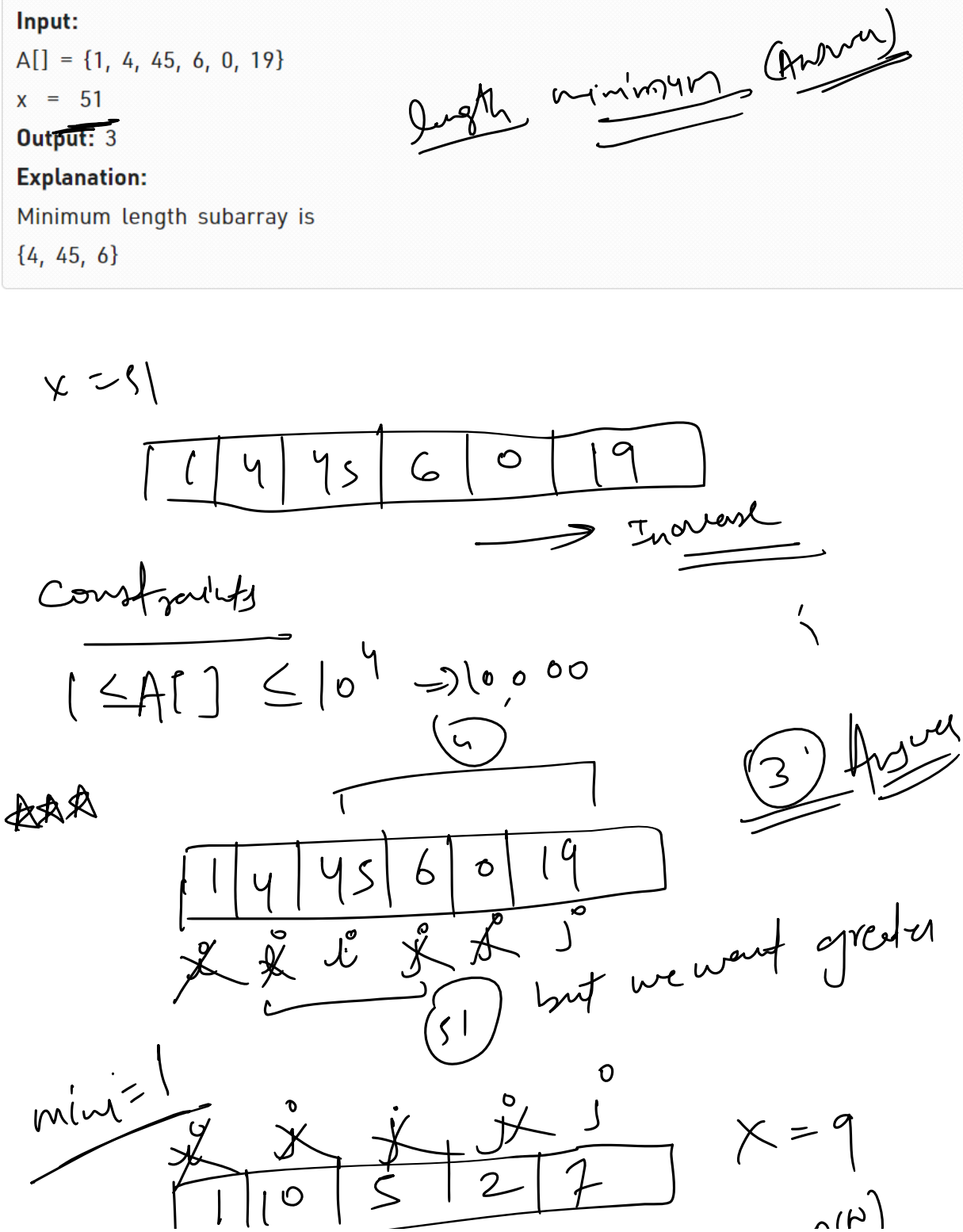
Easy Accuracy: **37.07%** Submissions: **82524** Points: **2**

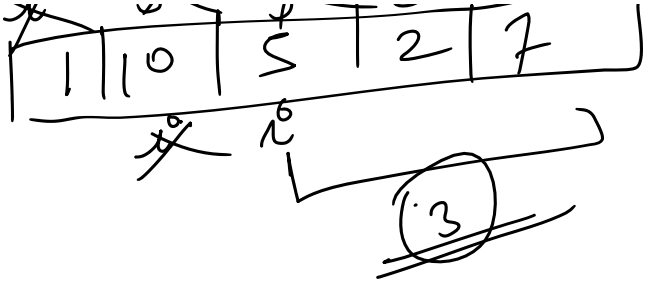
Given an array of integers (A[]) and a number x, find the smallest subarray with sum greater than the given value. If such a subarray do not exist return 0 in that case.

Note: The answer always exists. It is guaranteed that x **doesn't** exceed the summation of a[i] (from 1 to N).

Example 1:

Input:
A[] = {1, 4, 45, 6, 0, 19}
x = 51
Output: 3
Explanation:
Minimum length subarray is
{4, 45, 6}

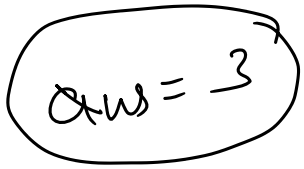




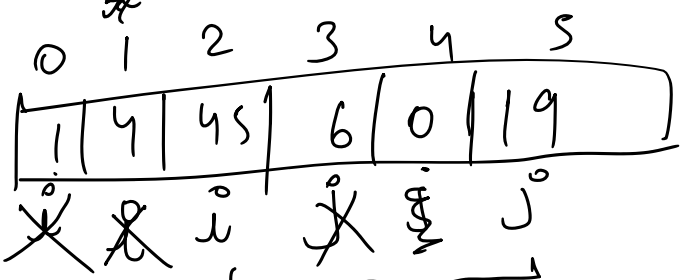
T.C - O(N)
S.C - O(1)

O(N³)

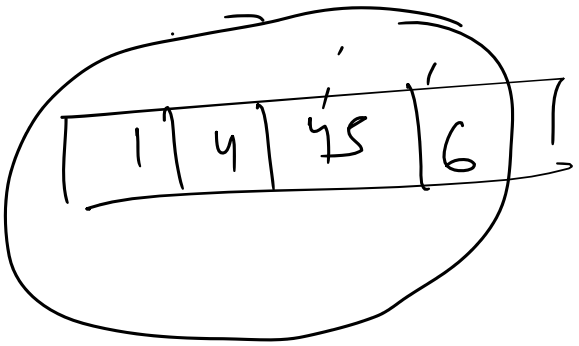
O(N)

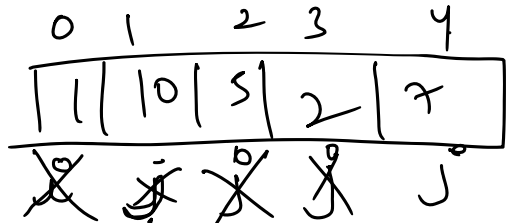


X = 51



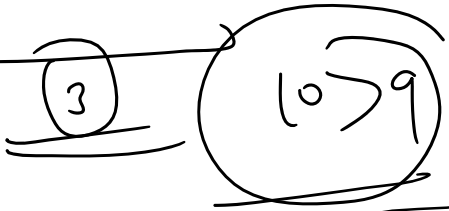
S = 1
S = 5
S = 56
S = 56



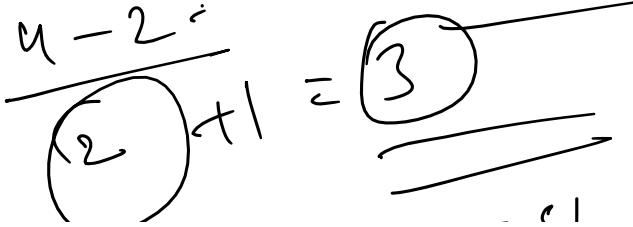


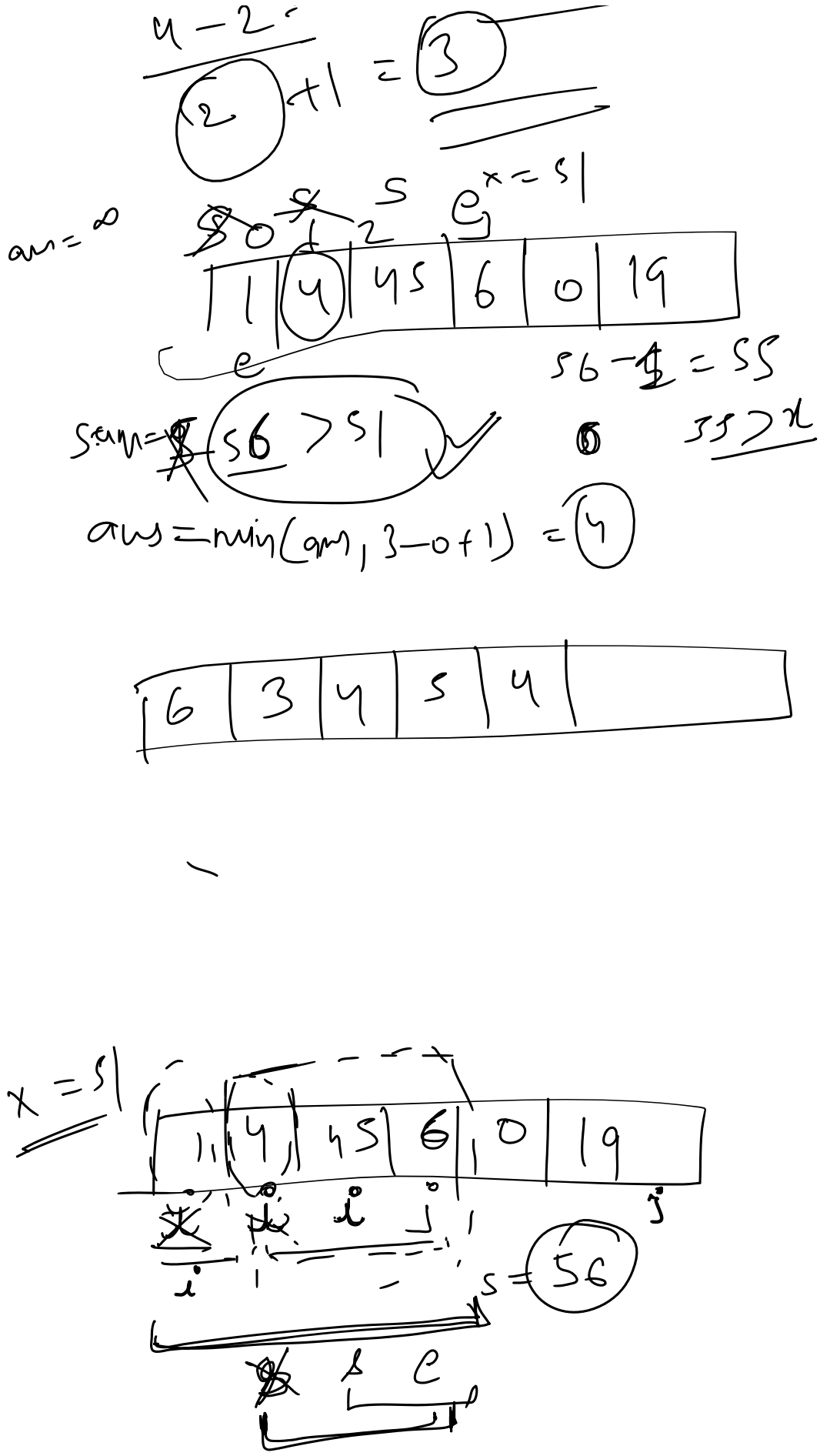
X = 9

S = 1
S = 1+1



ans = X




4-2+1 = 3
ans = ∞
0 1 2
1 4 45 6 0 19
56 - 1 = 55
sum = 56 > 51
55 > x
ans = min(ans, 3-0+1) = 4
6 3 4 5 4
x = 51
1 4 45 6 0 19
i j
s = 56

```cpp
int smallestSubWithSum(int arr[], int n, int x)
{
    // i = subarray
    int subarray_start = 0,subarray_end = 0;
    int sum = 0;
    int ans = INT_MAX;
    while(subarray_end < n){
        sum += arr[subarray_end];
        if(sum > x){
            ans = min(ans,subarray_end - subarray_start + 1);
            while(subarray_start <= subarray_end && sum > x){
                sum = sum - arr[subarray_start];
                ans = min(ans,subarray_end - subarray_start + 1);
                subarray_start++;
            }
        }
        subarray_end++;
    }
    return ans;
}
```

O(N)

T.C - O(N)

S.C - O(1)

Sliding Window

Three way partitioning

Easy Accuracy: **41.58%** Submissions: **99452** Points: **2**

Given an array of size n and a range [**a**, **b**]. The task is to partition the array around the range such that array is divided into three parts.

1) All elements smaller than **a** come first.

2) All elements in range **a** to **b** come next.

3) All elements greater than **b** appear in the end.

The individual elements of three sets can appear in any order. You are required to return the modified array.

Example 1:

Input:

n = 5

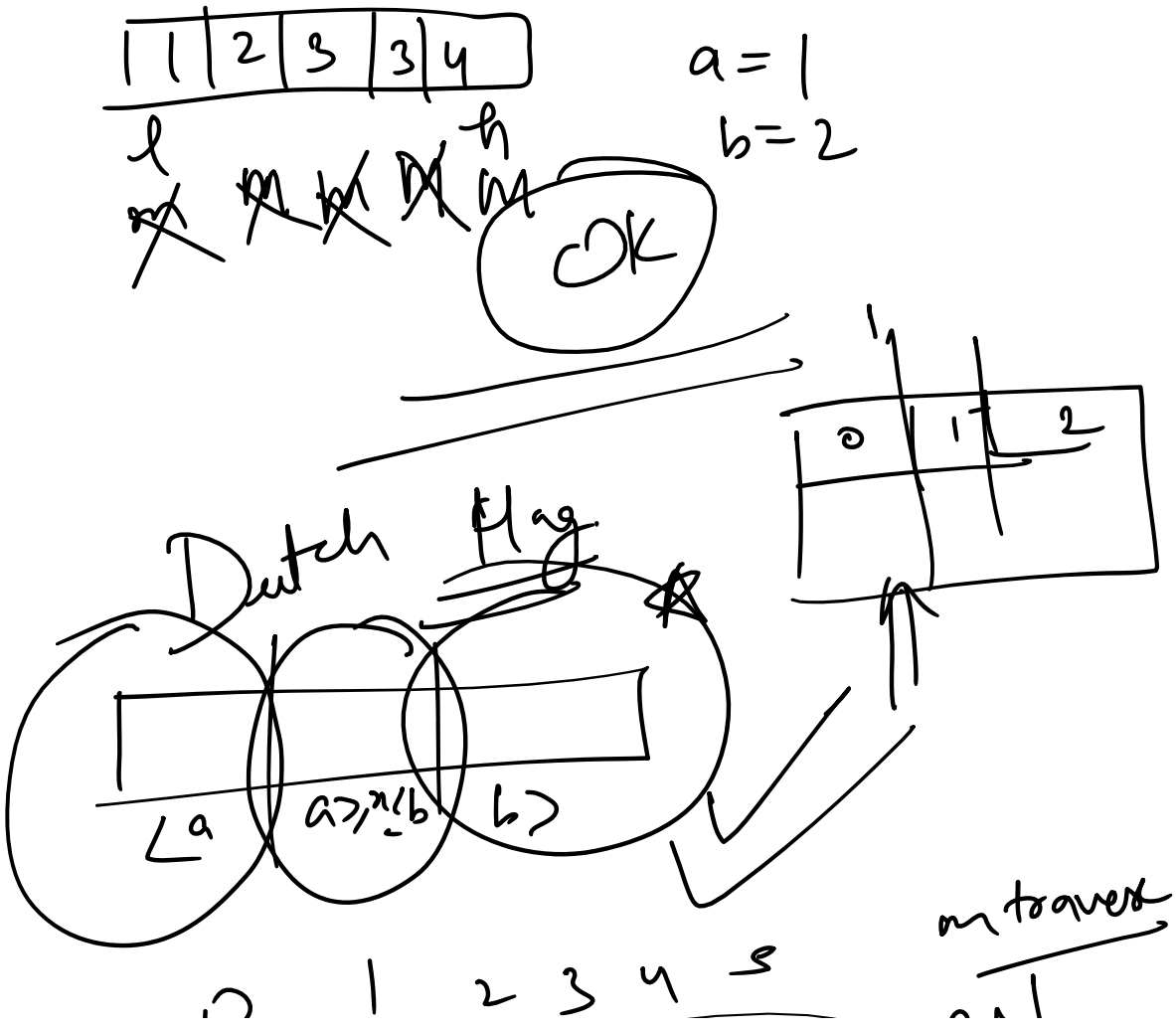
A[] = {1, 2, 3, 3, 4}

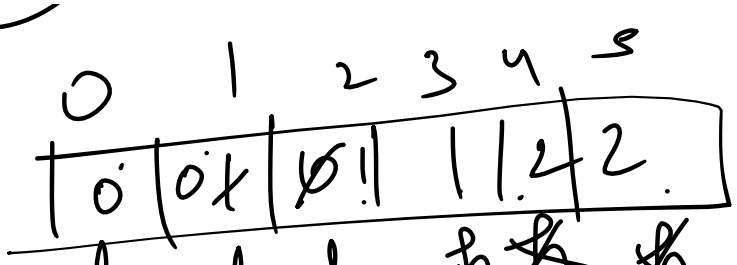
[a, b] = [1, 2]

Output: 1

Explanation: One possible arrangement is:

{1, 2, 3, 3, 4}. If you return a valid





Can 1

0

swap(l, m)

l++ m++

while(m ≤ h)

Can 2

1

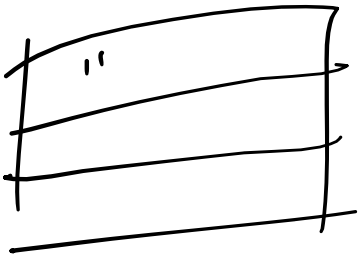
m++

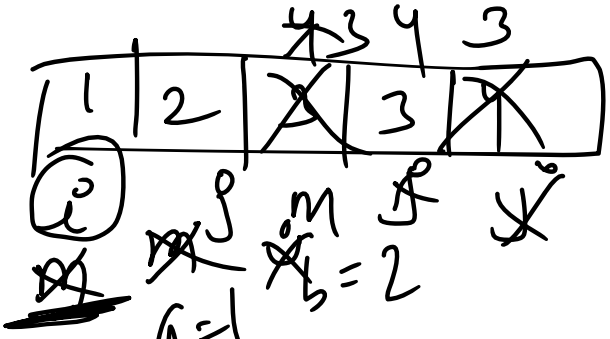
Can 3

2

swap(m, h)

h--





a=1 b=2

set 1

< a

i++ swap(i,m)

m++

set 2

a ≥ m ≤ b

m++

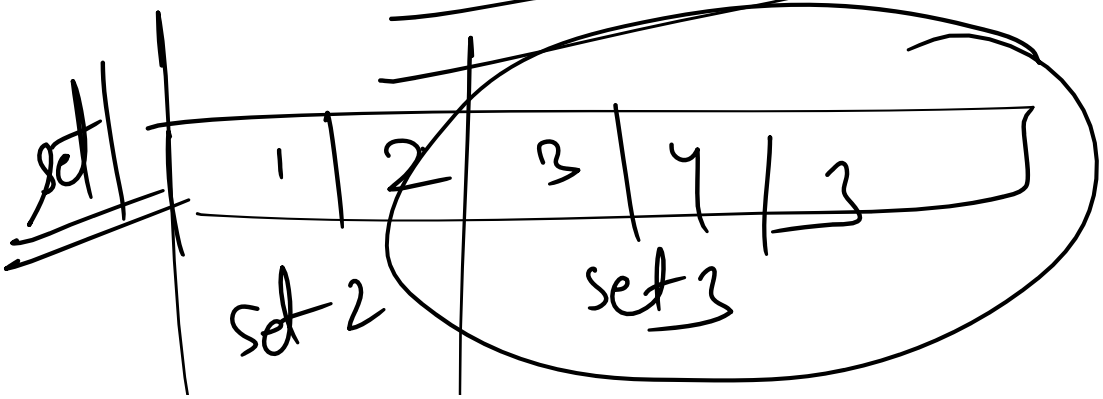
set

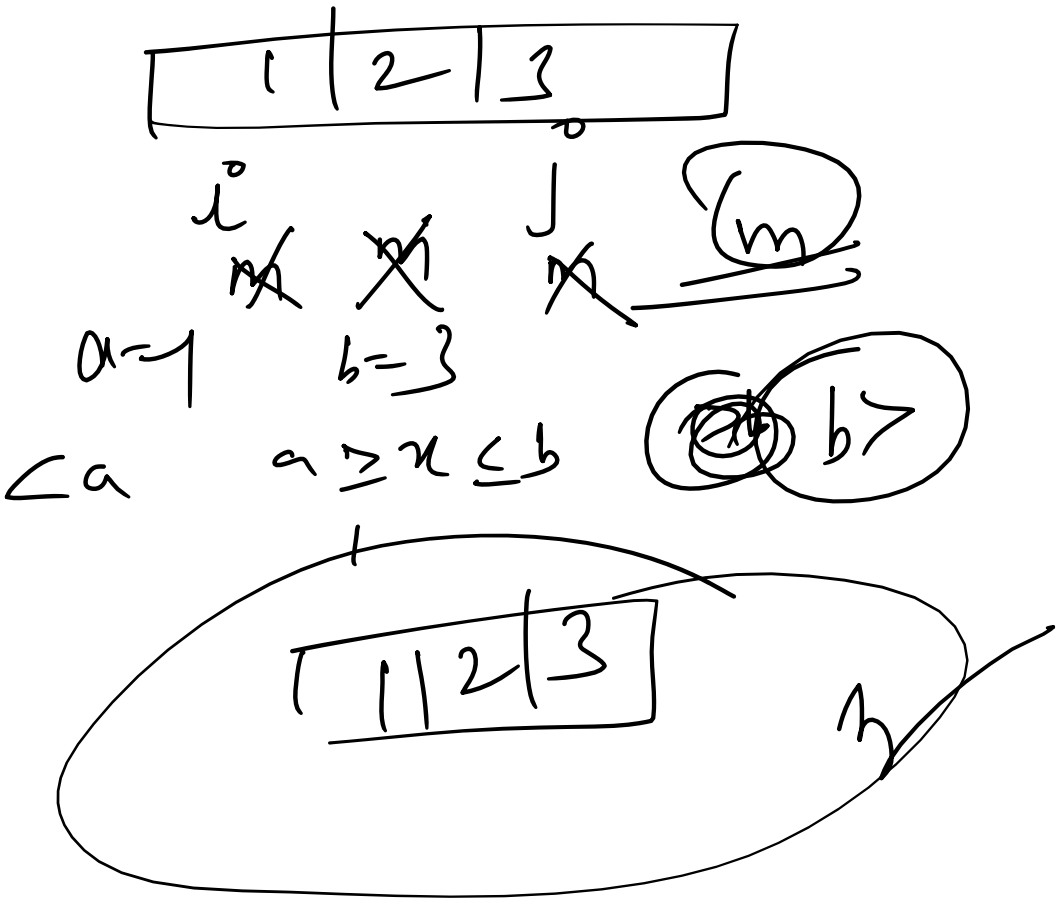
> b

swap(m, j)

j--

while(m ≤ j)

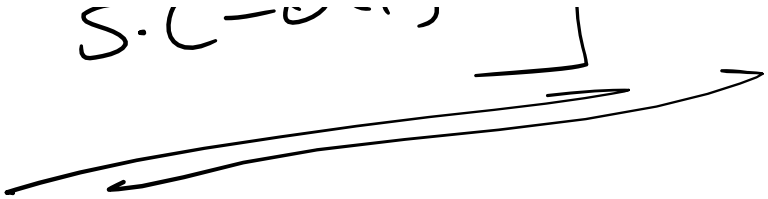





```
void threeWayPartition(vector<int>& array,int a, int b)
{
    int i = 0;
    int m = 0;
    int h = array.size()-1;
    while(m <= h){
        if(array[m] < a){
            swap(array[i],array[m]);
            i++;m++;
        }else if(array[m] >= a && array[m] <= b){
            m++;
        }else{
            swap(array[m],array[h]);
            h--;
        }
    }
}
```

T.C — O(N)

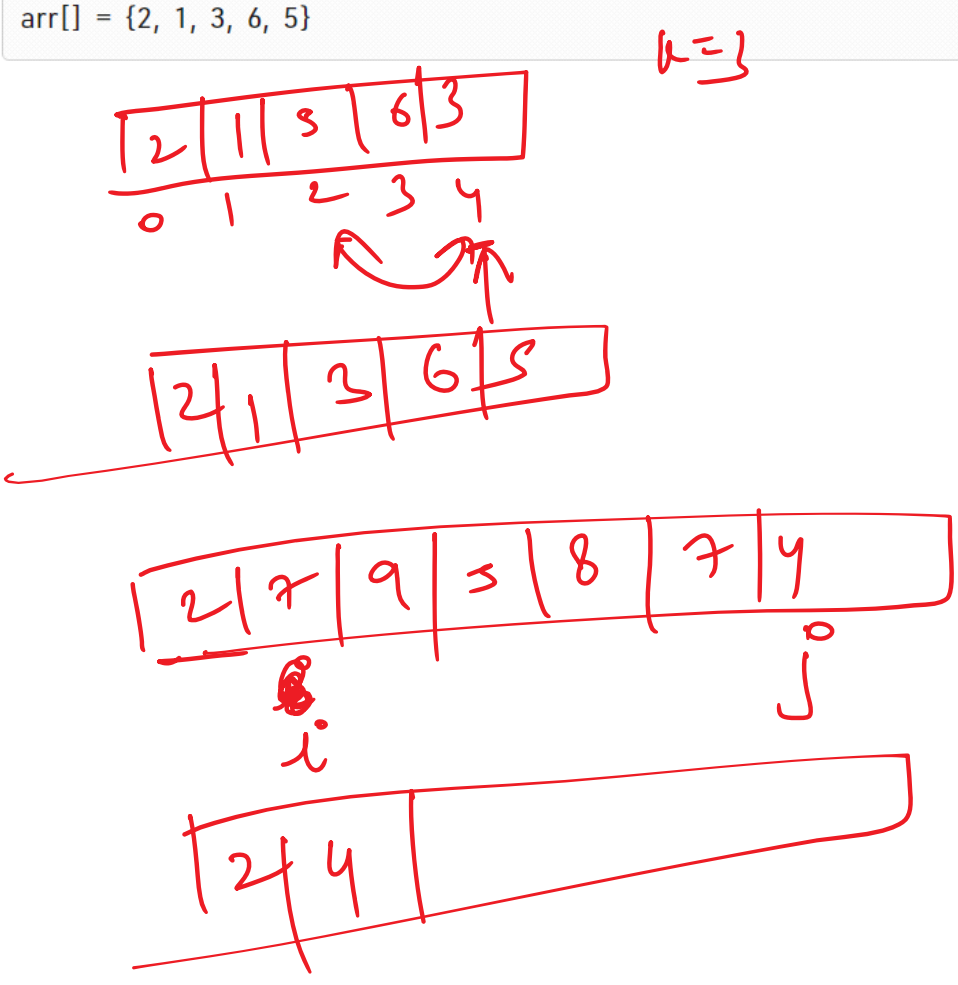
S.C — O(1)

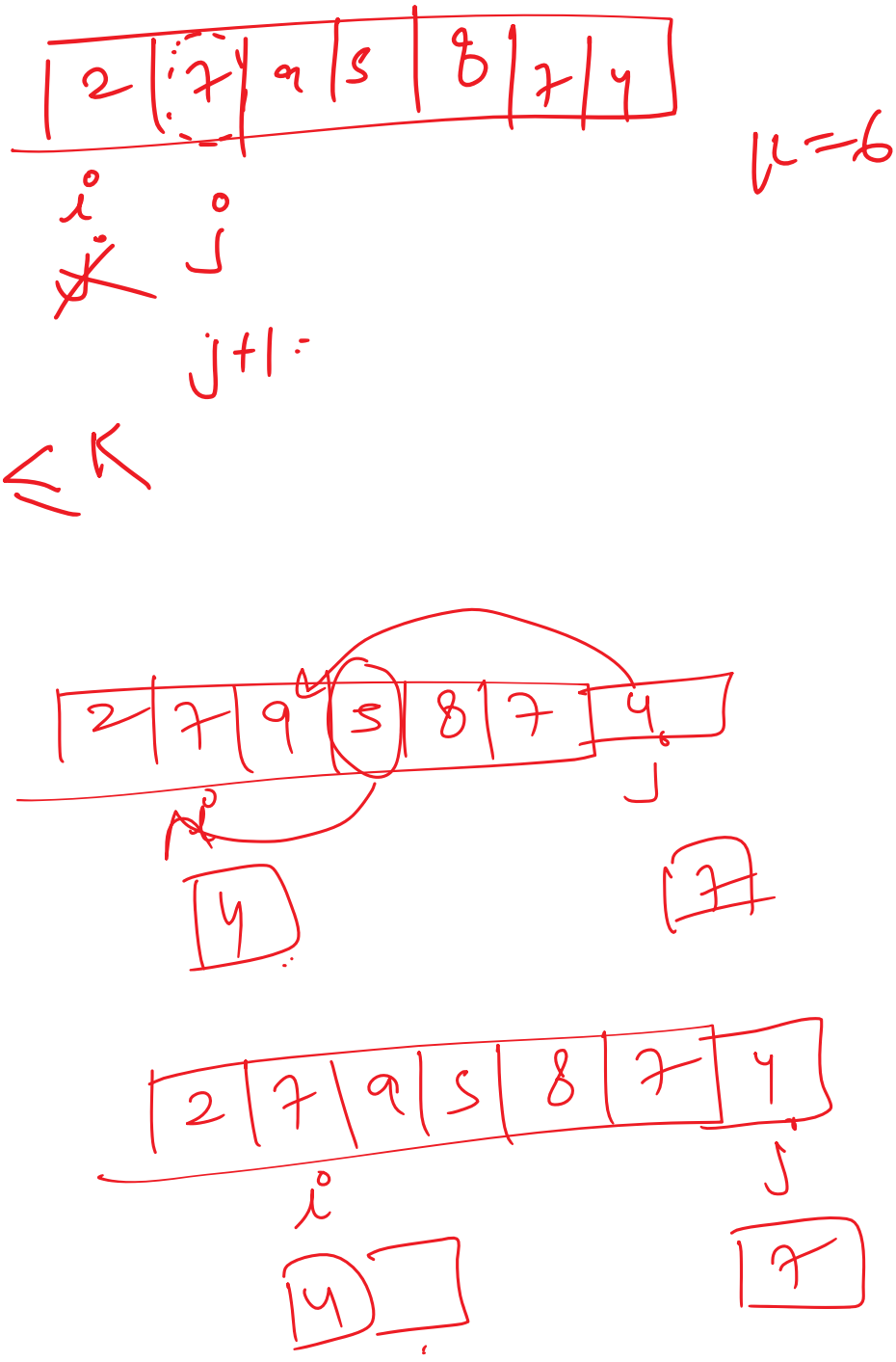


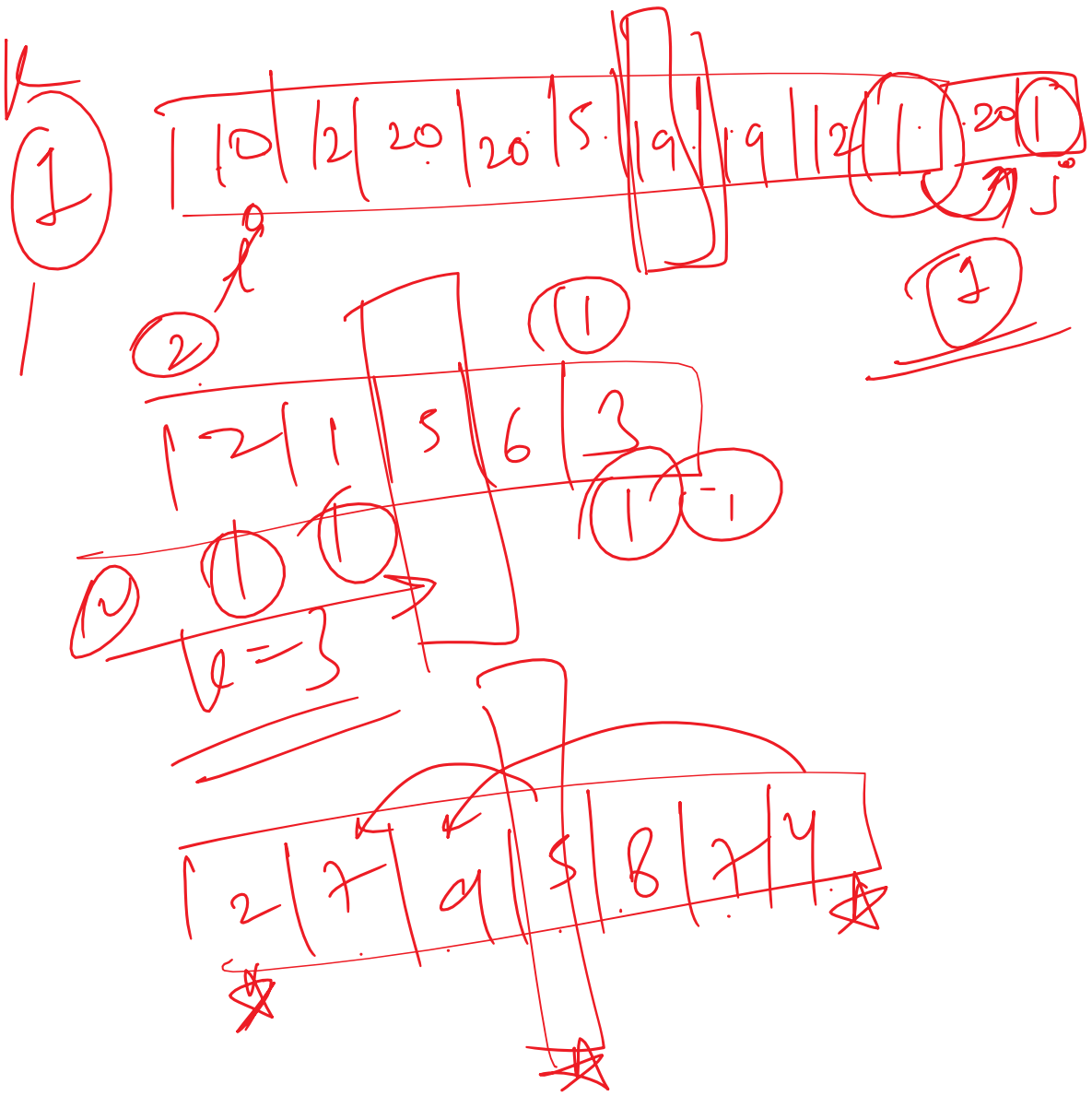
Given an array **arr** of **n** positive integers and a number **k**. One can apply a swap operation on the array any number of times, i.e choose any two index i and j (i < j) and swap arr[i] , arr[j] . Find the **minimum** number of swaps required to bring all the numbers less than or equal to **k** together, i.e. make them a contiguous subarray.

Example 1:

Input :
arr[ ] = {2, 1, 5, 6, 3}
K = 3
Output :
1
Explanation:
To bring elements 2, 1, 3 together,
swap index 2 with 4 (0-based indexing),
i.e. element arr[2] = 5 with arr[4] = 3
such that final array will be-
arr[] = {2, 1, 3, 6, 5}




2 7 9 5 8 7 4
i
j
x=6
j+1
≤ K
2 7 9 5 8 7 4
j
i
4
7
2 7 9 5 8 7 4
i
j
4
7



Using sliding window

Algo

* k = first we calculate all the elements that are less than or equal to k.

* make as our window of size equal to k at each window we

calculate no. of non-favourable element greater than k in array that will be our no. of swaps needed. slide the window over whole array and calculate the minimum non favourable element in a window

```
int minSwap(int arr[], int n, int k) {
   int ws= 0;
   for(int i = 0;i < n;i++){
       if(arr[i] <= k){
          ws++;
       }
   }
   if(ws == 0) return 0;
  int i = 0,j = 0,cnt = 0;
  int mini = INT_MAX;
   while(j < n){
        if(arr[j] > k){
              cnt++;
         }
         if((j- i + 1) == ws){
          mini = min(mini,cnt);
          if(arr[i] > k){
             cnt--;
          }
          i++;
       }
       j++;
   }
   return mini;
}
```

T.C- O(N)

S.C- O(1)