Agenda:

1. Smallest Number
2. Merge 2 sorted arrays
3. merge sort

Count Sort:

1. Find the smalest number that can be formed by rearranging the digits of the given number in array.
2. Return the smallest number in the form of an array.

$$A = [6 \quad 3 \quad 4 \quad 2 \quad 7 \quad 2 \quad 1]$$
$$= [1 \quad 2 \quad 2 \quad 3 \quad 4 \quad 6 \quad 7]$$

$$A = [4 \quad 2 \quad 7 \quad 3 \quad 9 \quad 0]$$
$$= [0 \quad 2 \quad 3 \quad 4 \quad 7 \quad 9]$$

Approach 1: BF: O(nlogn)

1. Sort the array in accending order

```
A.sort()
```

Approach 2: using hashmap.

1. Cont the freqance with hashmap.
2. iterate over hashmap.keys()
3. update original arrayA and return it.

TC: O(N)

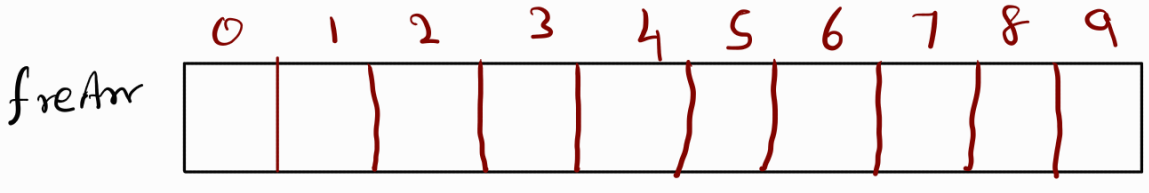
SC: O(N)

problem = hashmap is un-ordered collection of key values.

approch 2: using array of 0-9

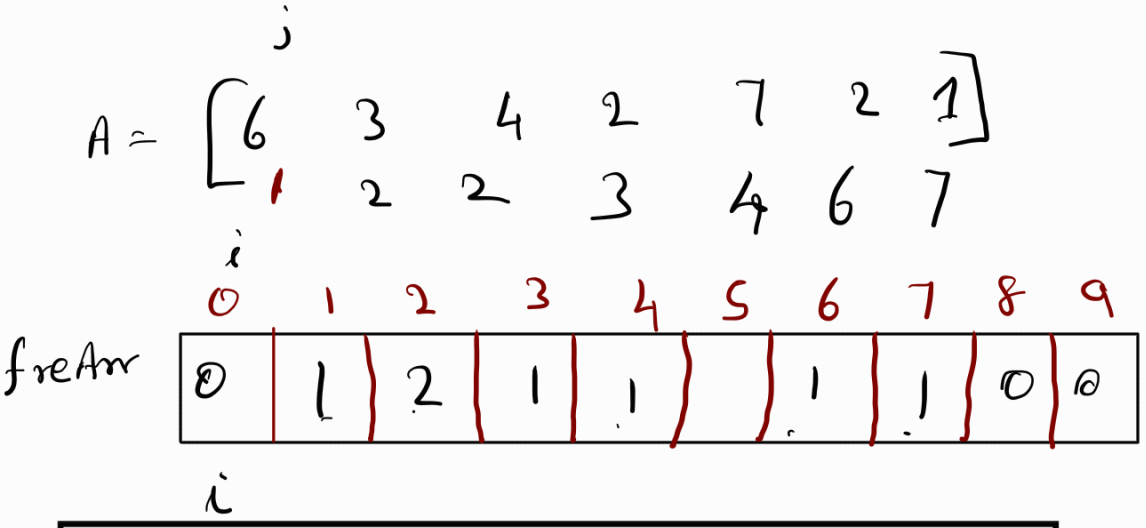
(1) Cout the elements of A and update freqArr.

(2) iterate over the freqArr and updat the original A.




```
freArr = [ ]
for a in A:
    ↓ freArr[a] = freqArr[a] +1
```

(N)




```
i = 0, j = 0
while (i < 10)
    value = freqArr[i]
    if value > 0:
        for k in range(value)
            A[j] = i
            j++
    i++
```

O(10)

(N + 10)

TC: O(N + digits)

SC: O(N)

i = 7

j = 6

(i) ele = 7

value = 1

TC: O(N + digits)

SC: O(digits)

⇒ Count Sort is not suitable for range of $(0, 10^9)$ because array of this size demand to much memory. it is not possible to create array.

⇒ Count sort works well for range of $A[i]$ is $\sim 10^6$

⇒ int require 4 byte

so, 10^9 int require 4 GB of memory

⇒ an array of 10^6 lenght is managable, need 4 MB

10^3 bytes	8 bits = 1 bytes
10^6 bytes	1024 bytes = 1 Kilo bytes
10^9 bytes	1024 KB = 1 Mega bytes
10^{12} bytes	1024 MB = 1 Giga bytes
	1024 GB = 1 tera bytes

1,000,000 = 10^6

1,000,000,000 = 10^9

How to handle negative numbers?

⇒ -9 ... 0 ... 9

⇒ calculate min and max, so freArr can be formed from min → max.

	0	1	2	3	4	5
A =	-2	3	8	3	-2	3

check min = -2

check max = 8

freArr = [0] * (max - min + 1)

8 + 2 + 1 = 11

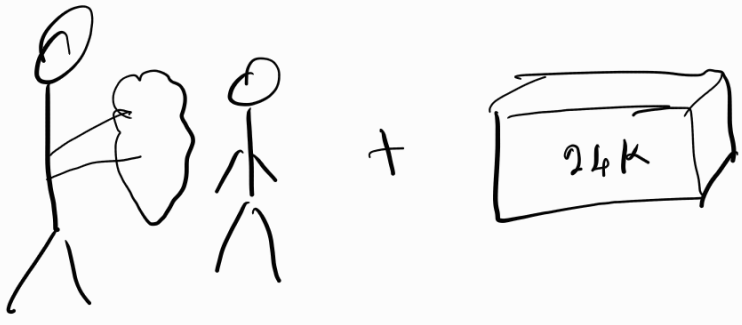
-2	-1	0	1	2	3	4	5	6	7	8
1					2	1	1			1

```
index = 0
for a in range(min, max+1)
    ele = A[index]
    freArr[ele] = freArr[ele] + 1
    index += 1
```

```
i = 0, j = 0   ele = min
while (i < len(freArr)):
    count = freArr[ele]
    if count > 0:
        A[j] = ele
        j += 1
    i++
    ele += 1
```

```
for i in range(len(A))
    ele = A[i]
    index = A[i] - min
    freArr[index] = freqArr[index] + 1
index = 0
range = max - min + 1
for i in range(range):
    ele = i + min
    count = freArr[ele]
    for j in range(count)
        A[index] = ele
    index += 1
```

0 + -2 = -2
1 + -2 = -1
2 + -2 = 0



mari + gold = marigold

Gmail's all Inboxes feature

1. Given two sorted arrays A, B
3. merge them into one.

```
        i
A = [2   4   7   8   12]
        j
B = [3   5   6   7]

ans = [2   3   4   5   6   7   8   12]
```

Approch 1: using Hashmap (dictionary)

1. create freq map having all elements count from A, B.
2. create new array from freq map.

TC: $O(N+M)$ we will not have thing in order

SC: $O(N+M)$

Approch 2: using Array: (Count Sort)

1. same above approach using int[n+m]

=> if N+M length is more than 10^6

=> we can not use this approach.

approach 3: using array

1. Take array of size N+M
2. update the array with elements
3. sort the array
4. return

TC : O(N log N)

SC : O(N+M)

approch 4 :

```
                i
a =  2   4   7   8   12
b =  3   5   6   7
                    j
```

n = 5, m = 4
i = 3
j = 4
k = 7

ans = [2, 3, 4, 5, 6, 7, 7, 8, 12]

Code :

```
def func(A, B):
    n = len(A)
    m = len(B)
    ans = [0] * (n+m)
    i, j, k = 0, 0, 0
    while (i<n & j<m)
        if A[i] < B[j]
            ans[k] = A[i]
            i+=1, k+=1
        else:
            ans[k] = B[j]
            j+=1, k+=1
    while (i<n)
        ans[k] = A[i]
        i+=1
    while (j<n)
        ans[k] = B[j]
        j++
```

i = 0
j = 0
k = 0

```
while (i<n and j<m)
                8     7
    if A[i] > B[j]
        ans.apend(B[j])
        j++, k++
    else
        ans.apend(A[i])
        i++, k++

while(i<n)
    ans.append[A[i]
    i++

while (j<n)
    ans.append(B[j]
    j++
```

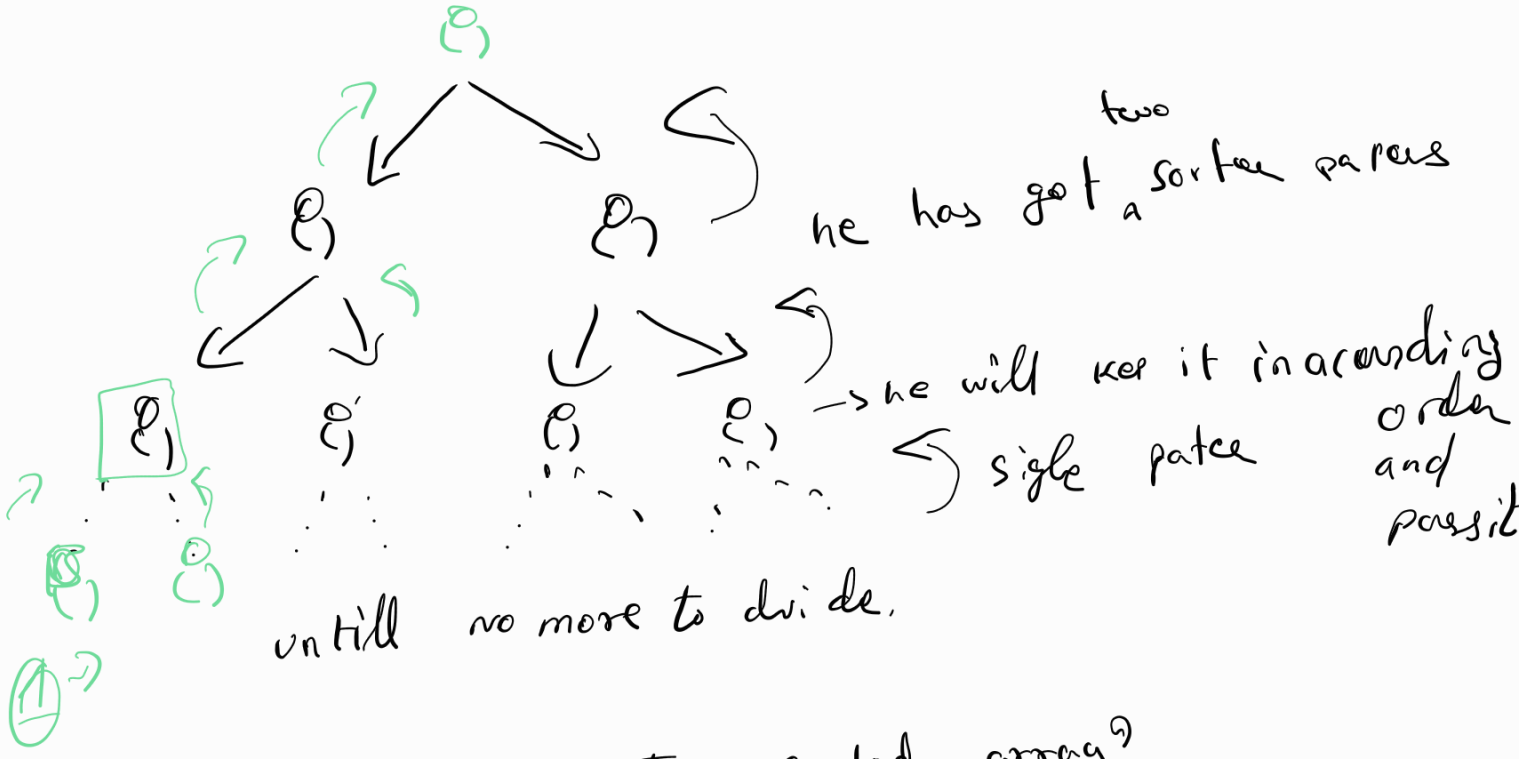
TC : O(n+m)

SC : O(n+m)
⇒ because we take ans array (output space)

merge Sort

uses recursion

Devide
Conquer
Combine



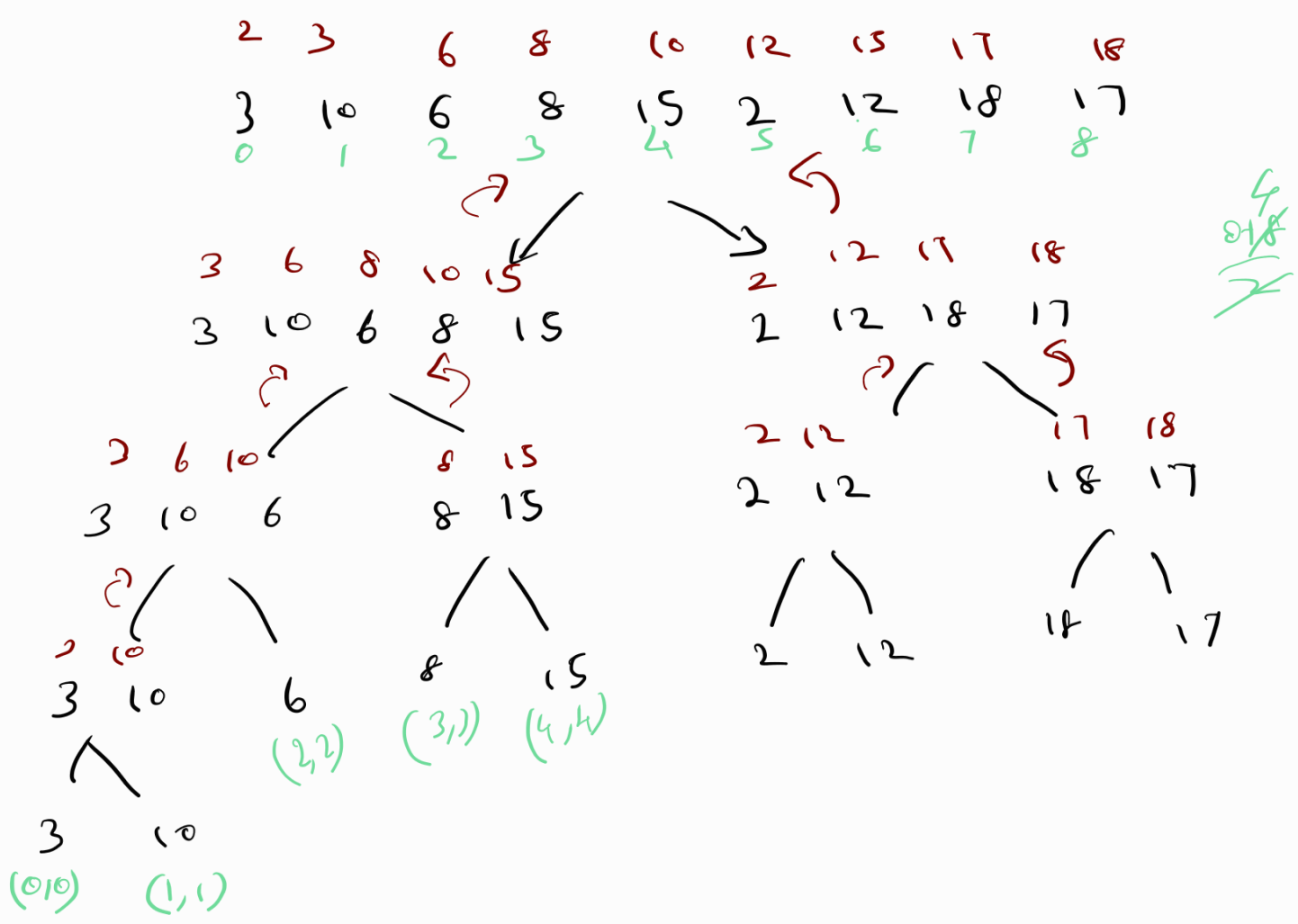
⇒ how to merge two sorted array?

func([2, 3, 4]) = func(

3 1 0 6 8 15 2 12 18

0 1 2 3 4 5 6 7 8

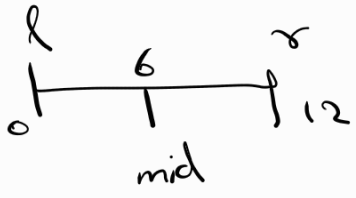
mid = 0+4/2 = 4

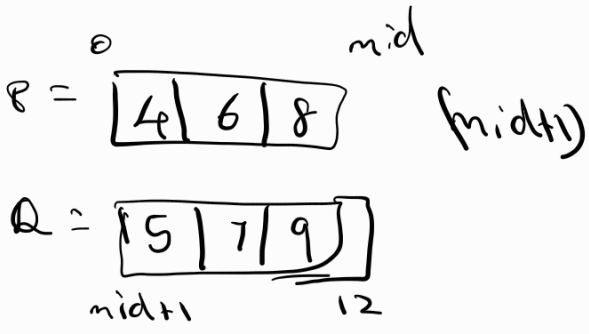


0 N-1

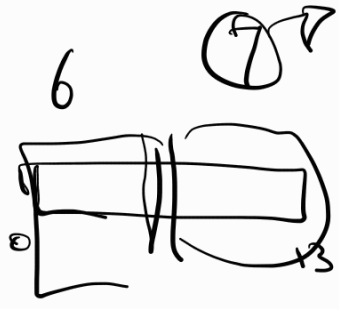
```
def mergeSort(A, l, r)
    # in the end we will have single element
    if (l == r) return # base case
    # getting middle element index.
    mid = (l+r)//2
    # sort the array from l=0 to mid
    mergeSort(A, l, mid)        ----> N/2
    # sort the array from (mid+1) to r
    mergeSort(A, (mid+1), r)    ----> N/2
    # perform merging two sorted array.
    mergefun(A, l, mid, r)      ----> N
```

```
def mergefunc(A, l, mid, r)
    # merging two arry.
    # creating two sepeate array temprarily
To hold the array, while updating sorted
array in A  n1 = (mid - l + 1) # (l, mid)
            n2 = (r - mid) #(mid+1, r)
    P = [0] * n1
    Q = [0] * n2 , i = l, k = 0
    while ( i <= mid)
        P[k] = A[i]
        i += 1 , k += 1
    k = 0 , i = mid
    while ( i <= r)
        Q[k] = A[i]
        i += 1, k += 1
```






```
    # perform sorting and update A
    i, j, k = 0 , 0 , l
    while (i < n1 and j < n2 )
        if P[i] <= Q[j]
            A[k] = P[i]
            i += 1 , k += 1
        else:
            A[k] = Q[j]
            j += 1, k += 1
```



(0+13)//2

0 → 6 → 13

13-6+1

7

```
    # remaining element in array is already sorted.
```

while (i < n1)

↓ A[k] = P[i]

i += 1 , k += 1

while (j < n2)

↓ A[k] = Q[j]

k += 1 , j += 1

return A

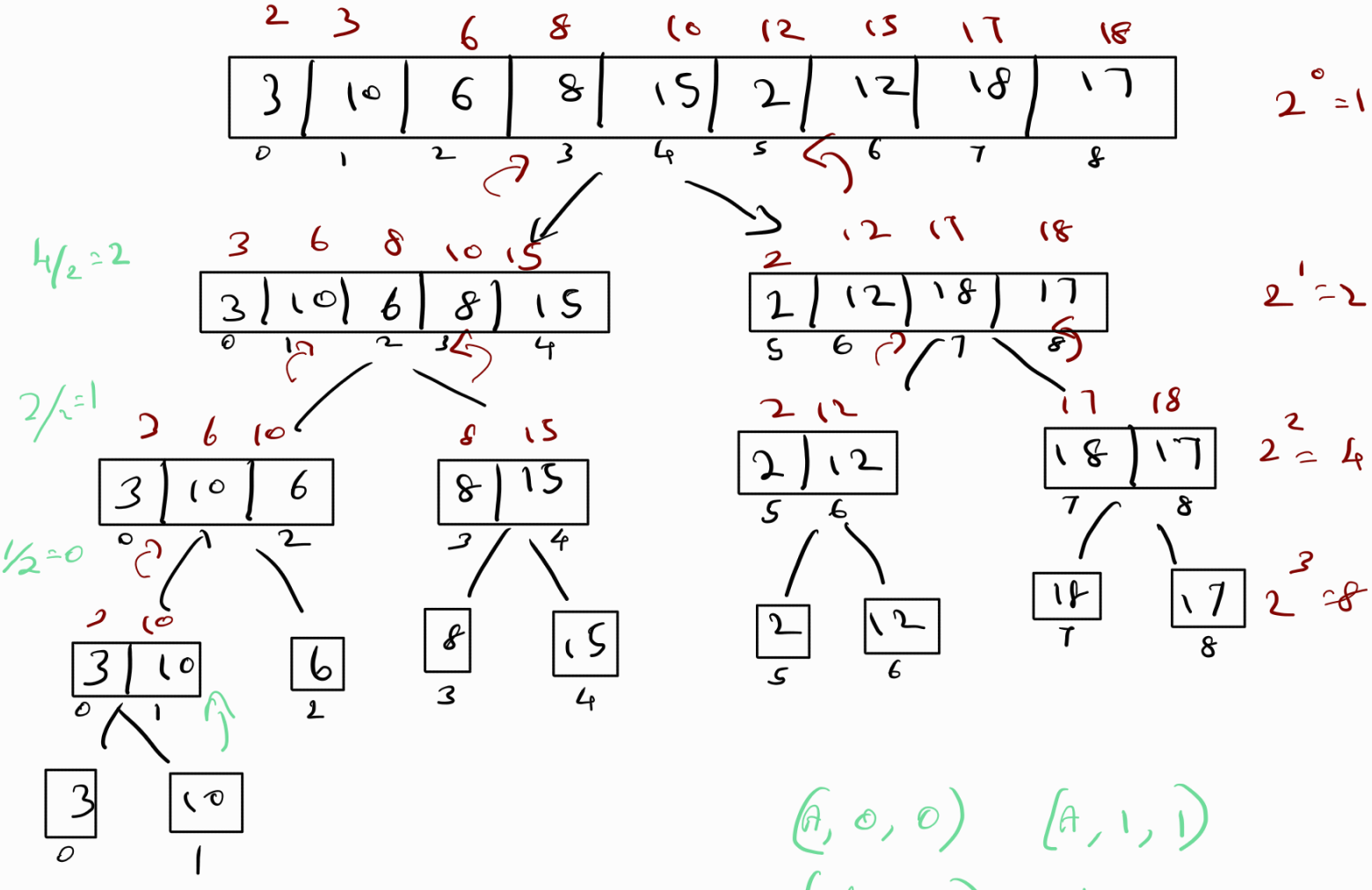
0 +8 = 8/2 = 4

0 n-1

(A, l, r)

mid = r+l//2

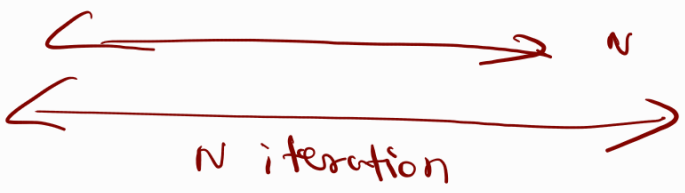
(A, l, mid) (mid+1, r)



(A, 0, 0) (A, 1, 1)

(l == r) return

TODO: Dry run complete merge sort

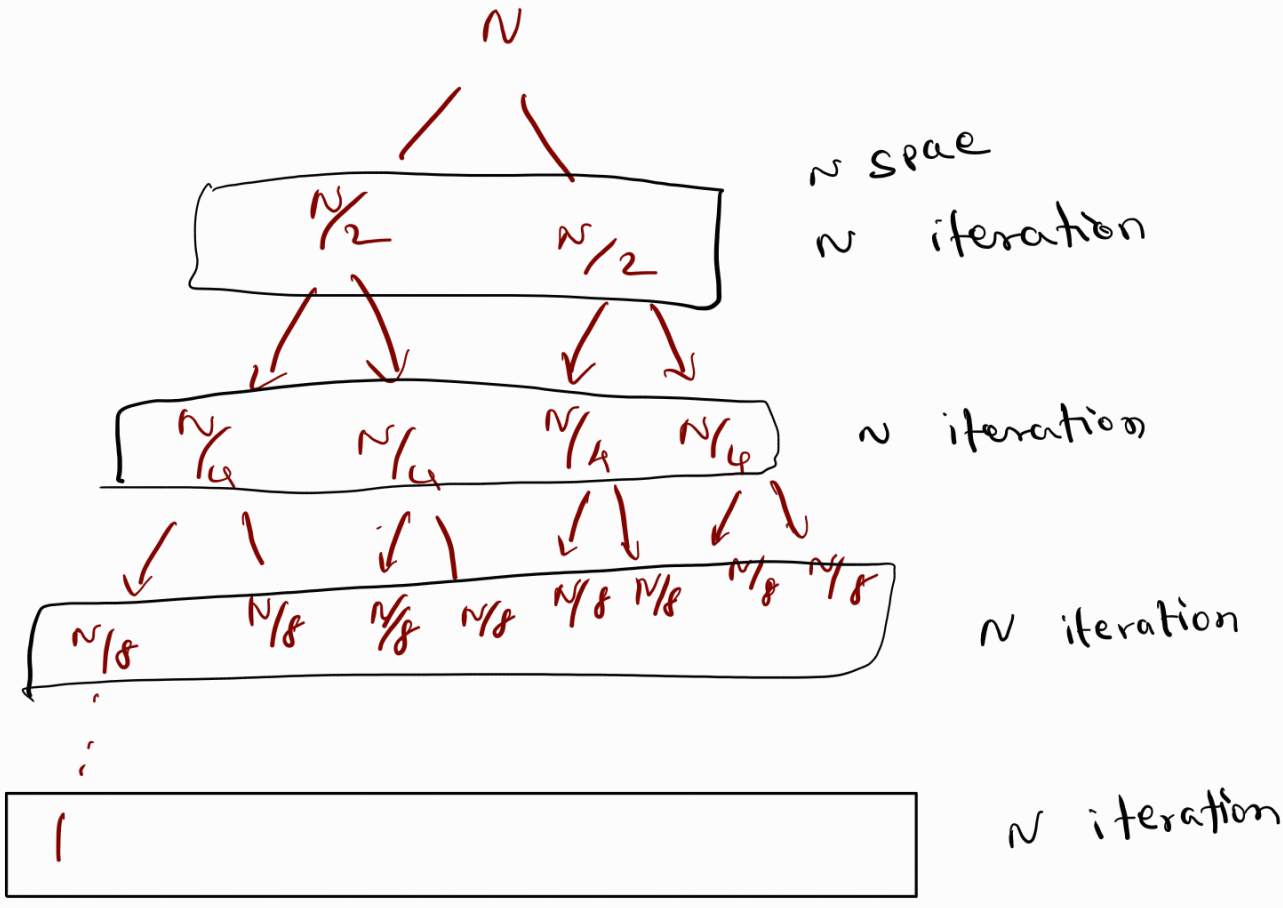


$\Rightarrow$ $\Rightarrow 2^0 + 2^1 + 2^2 + 2^3 \ldots 2^N \Rightarrow \log_2 N$

$\Rightarrow$ For [3] return

for [3|12] Takes 2 iteration

for all horizontally Takes N iteration



N space

N iteration

N iteration

N iteration

N iteration

$\Rightarrow$ For all levels N iteration to merge them

$TC: O(N \log_2 N)$ (function calls x Time per fing)

(depth of function call x Space per func)

(X) we construct array in every level and it gets released

$SC: O(\log N + N)$

Recurence Relation of merge sort

$$T(N) = 2T(N/2) + N$$

- $2T(N/2)$: 2 function calls, each of them $N/2$
- N: N iterations for merging

Substitute: $\Rightarrow$ $T(N/2) = 2T(N/4) + N/2$

$$T(N) = 2\left[2T(N/4) + N/2\right] + N$$

$$T(N) = 4T(N/4) + 2 \times N/2 + N$$

$$T(N) = 2^2 T(N/2^2) + 2N$$

$$T(N/4) = 2T(N/8) + N/4$$

$$T(N) = 4\left[2T(N/8) + N/4\right] + 2N$$

$$= 8T(N/8) + 4(N/4) + 2N$$

$$T(N) = 2^3 T(N/2^3) + 3N$$

Generalization!

based on the pattern if it goes to K steps

$$2^k T(N/2^k) + kN$$

in the end k steps value becomes One

$$\frac{N}{2^k} = 1$$

means, $N = 2^k$

$$k = \log_2 N$$

Substitute:

$$\Rightarrow 2^k T(1) + kN \quad (T(1): \text{constant})$$

$$\Rightarrow N + kN$$

$$\Rightarrow \cancel{N}^{\text{constant}} + \log_2 N \times N$$

TC $\Rightarrow N \log_2 N$

Stable Sort:

⇒ when the relative order of element is not changed,

⇒ then it is called stable sort

A = [6 5 3 5] → 3 5 5 6 } stable sort

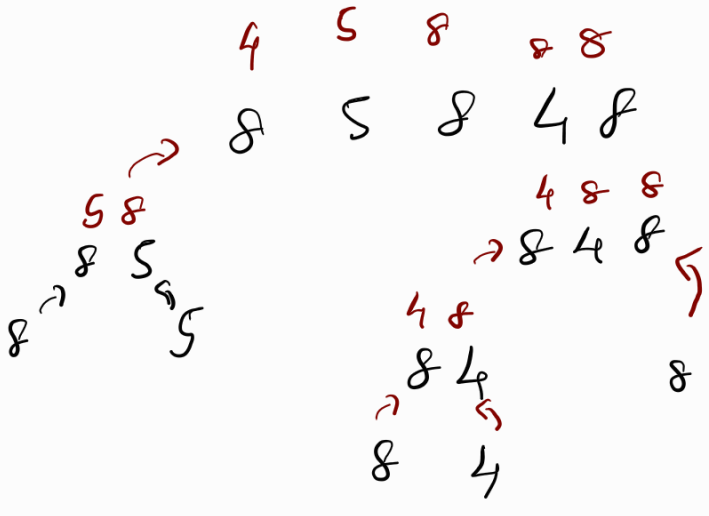
A	8
B	3
C	8
D	4
E	8

After Sort →

B	3
D	4
A	8
C	8
E	8

First come First served

Q. is **mergesort** is stable sort?



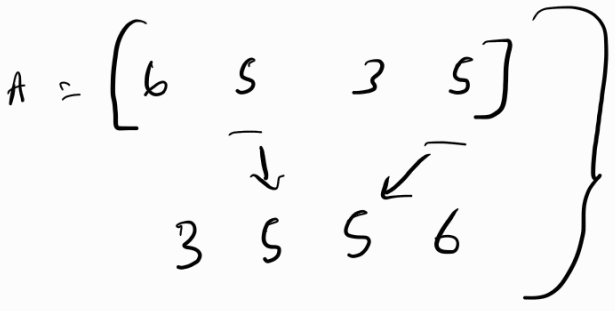
Big YES if P[i] <= Q[j]

Big NO if P[i] < Q[j]

Un-stable Sort:

=> when the order of same element is changed.

=> then it is called un-stable sort



In-place:

when a algo do not take extra space apart from stack space.

is merge sort in-place || Big NO ||

we use

Day 23

1 Merge Two sorted arrays

1. Given two sorted int arrays A and B
2. merge them.

$-2 \times 10^9 \le A[i], B[i] \le 2 \times 10^9$

$1 \le |A|, |B| \le 5 \times 10^4$

ex: A = [4, 7, 9]

B = [2, 11, 19, 22]

j ⟷

[2, 4, 7, 9, 19, 22]

---

2 Inversion Count in an array

1. Given an array of int A

if $i < j$ an $A[i] > A[j]$ then the pair is called an inversion of A.

Find it

ex: A = [1 3 2] (j)

$1 \le |A| \le 10^5$

$1 \le A[i] \le 10^9$

iterate A

↓ for (i, n)

↓ if $A[i] > A[j]$

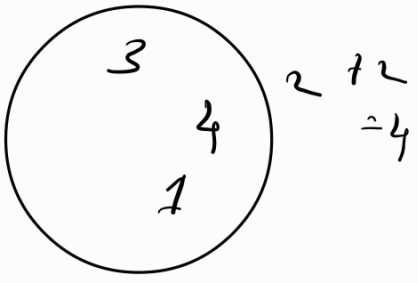
↓ return.

} Brute force $O(N^2)$

1 2 3

ex: [3, 4, 1, 2]

Sorted = 1 2 3 4



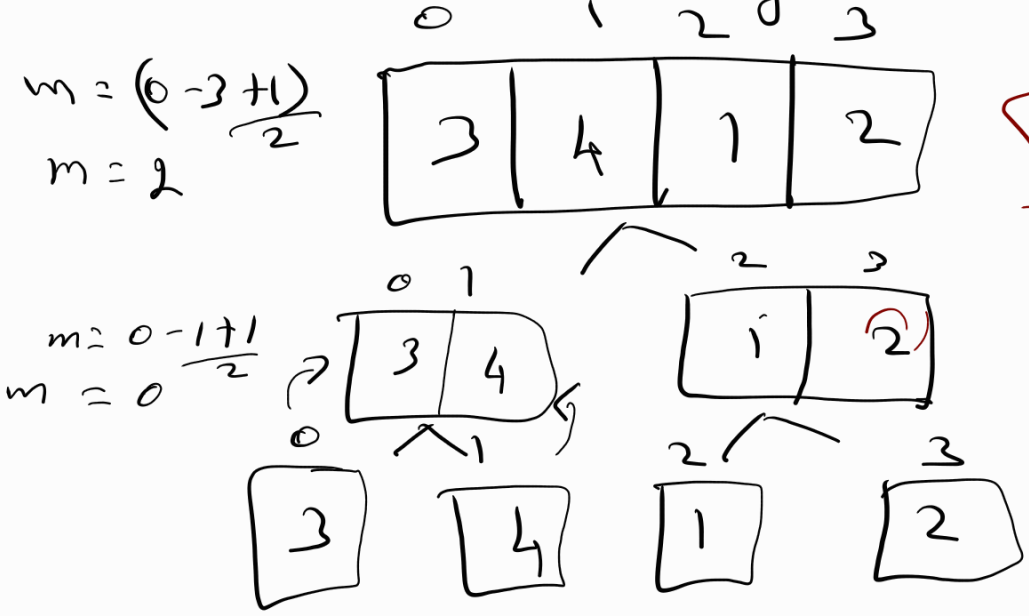
1. BF

~~2. sorting~~

3. merge sort algo

↳ Recursion : function call itself to solve sub problems.

↳ Dividing



1 2 3 4

i < j ✓

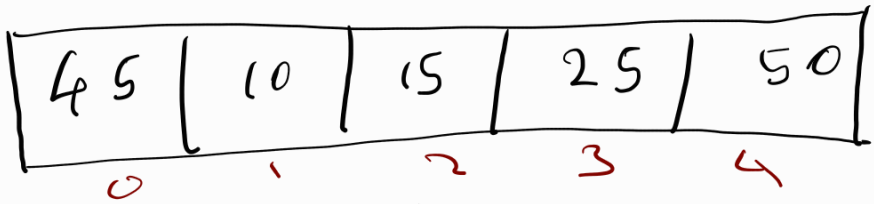
A[i] > B[j] ✓

(N - i)

ans = 2 + 2 = 4

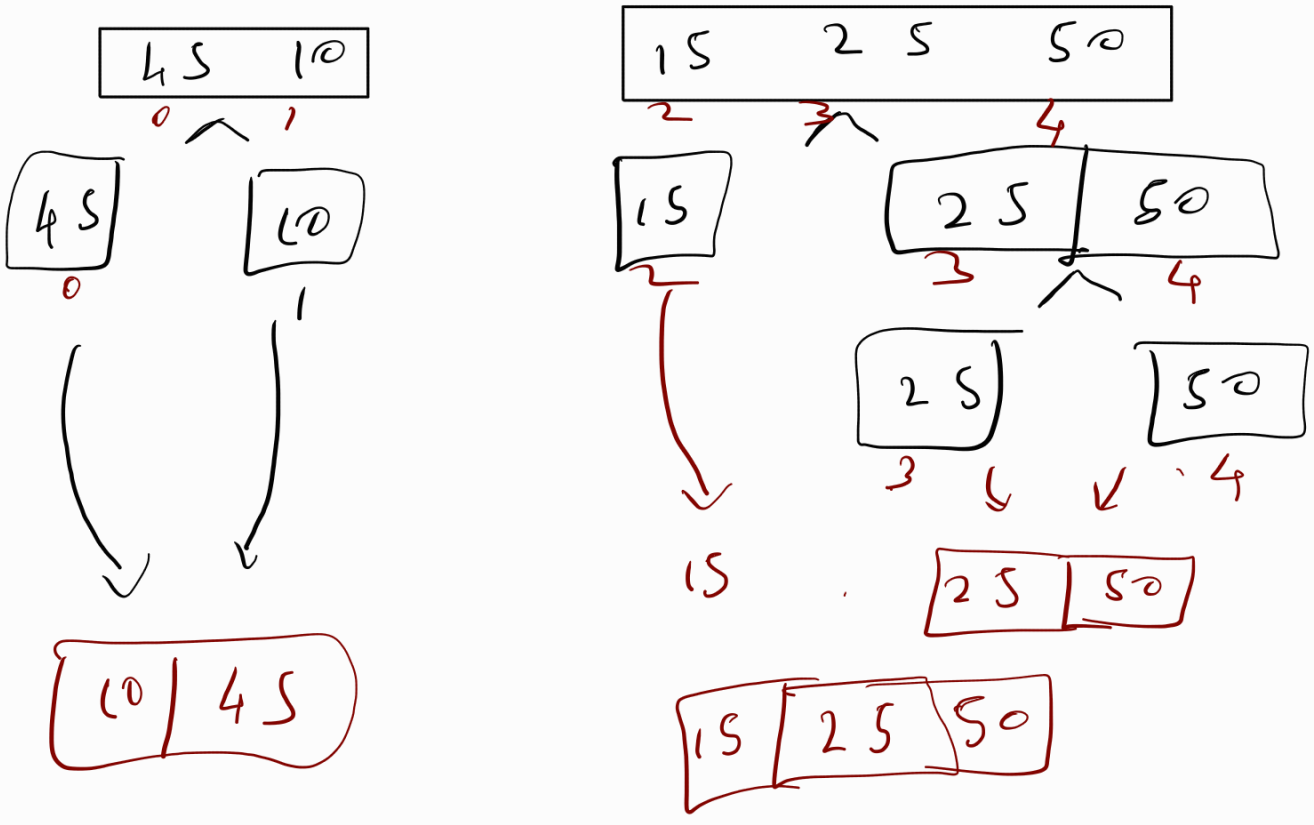
```
mergSort(A, l, r, ans):
    if l==r return
    m = (r-l+1)//2
    mergeSort(A, l, m)
    mergeSort(A, m+1, r)
    ans+= merge(A, l, m, r)
```

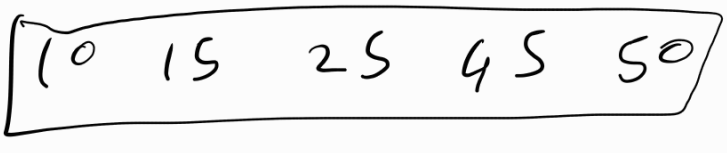
```
merge(A, l, m, r, ans)
    P = A[l : m+1]   # i   n = len(P)
    Q = A[m+1 : r+1] # j   m = len(Q)
    i, j, k = 0, 0, l
    while(i<n and j<m):
        if P[i] <= Q[j]:
            A[k] = P[i]
            k+=1
            i+=1
        else:
            ans+=(n-i)
            A[k] = Q[j]
            j+=1, k+=1
    .
    .
    .
    return ans
```



ans = ~~2~~ 3

m = (4-0+1)//2
= 2





ans = 3

---

③ sort by color:

1. Given array with 0, 1, 2
2. sort them so the same elements are in order 0, 1, 2

Ex: A = [0, 1, 2, 0, 1, 2]

A = [0]

$1 \leq N \leq 10^6$ 1 MB

BF:

default sorting $(N \log N)$

Count Sort $O(N)$ space $= O(N)$

Count all 0
1
2

Create one ans array

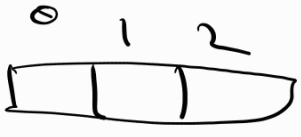
1. Create array of size 3

freqArr = | 0 | 1 | 2 |

2. iterate over A and update frequency.

3. iterate freqArr and update original array

```
n = len(A)
freqArr = [0] * 3
```




```
for a in A:
    freqArr[a] += 1,
k = 0
for i in range(3):
    value = i
```

```
count = freqArr[i]
for j in range(count):
    A[k] = value
    k += 1
return A.
```

Dry run

0 0 1 1 2 2

A = [0, 1, 2, 0, 1, 2] ✓ =>

A = [0]

→

0	1	2
2	2	2

k = 6

TC: O(N)

SC: O(3) => Constant

---

④ merge Sort

1. Given int [] A
2. sort the array using merge sort

ex: A = [1, 4, 10, 2, 1, 5] $1 \leq |A| \leq 10^5$

A = [3, 7, 1] $1 \leq A[i] \leq 10^9$

def mergeSort(A):

---

(5) Smallest Number!

1. Given int, in the form of array.
2. sort and return array
3. $1 \leq N \leq 10^5$

   $0 \leq A[i] \leq 9$

1. create array of size 10
2. iterate over A and update freqency in freq Arrays.

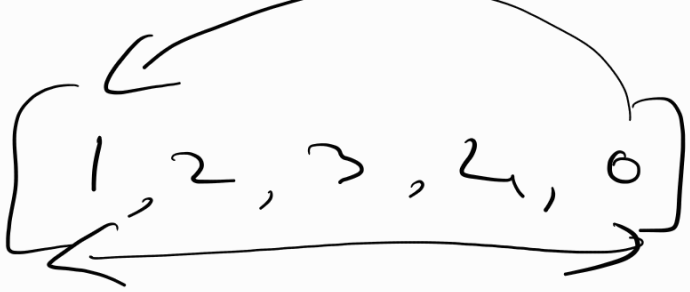
TC: O(N)

SC: O(10) (or) O(digits)

---

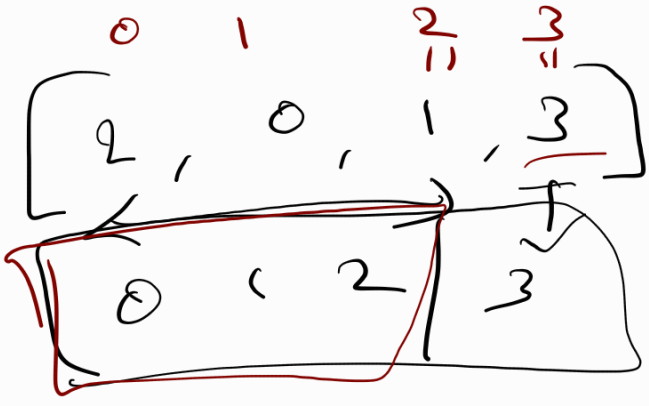
(6) max chunks to make Sorted

1. Given array A
2. that is permutation of $[0, 1, 2, \ldots (N-1)]$
3.

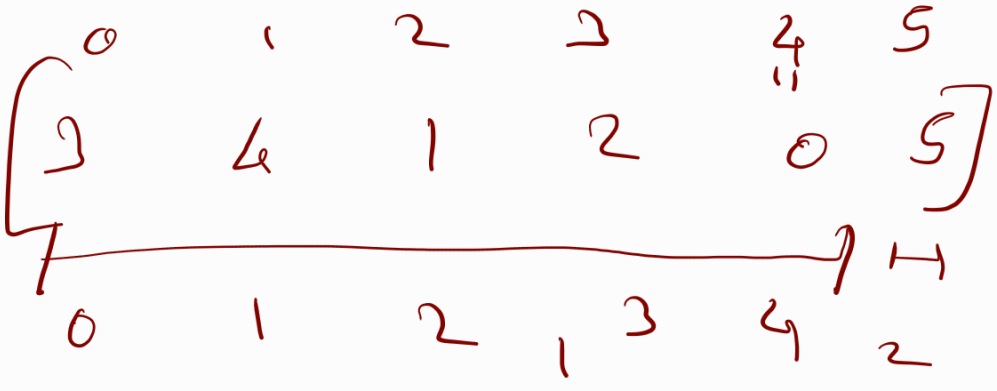
1. 

output = 1

$[0 \to N-1]$

2. 

output = 2



Code!

```
def countChuckOfArray(A):
    N = len(A), ma = 0, ans = 0
    for i in range(N)
        ma = max(A[i], ma)
        if i == ma:
            ans += 1
    return ans
```

TC: O(N) SC: Constant

Count Sort:

Given int[] A
sort this array using Count Sort

[1 3 1]

1. create array of 10^5

freqArr = [0] * 10^5

2. update frequency from A

for a in A:
↓ freqArr[a] = freqArr[a] + 1

3. iterate over freqArr and update A

```
k = 0
for i in range(10^5):
    value = i
    count = freqArr[i]
    for j in range(count):
        A[k] = value
        k += 1
return A.
```

Sort subarray with left and right index

Given int[] A, int B and C

using merge sort:

mid = (l + r)//2

```
[20   30   40   50   10]
 0    1    2    3    4
```

50 | 40 30 20 | 10

(0,0) | (1,3)

m = 0+3//2 => 1

50 | 20 30 40 | 10

m = 1+3//2

50 | 40 30 20 | 10

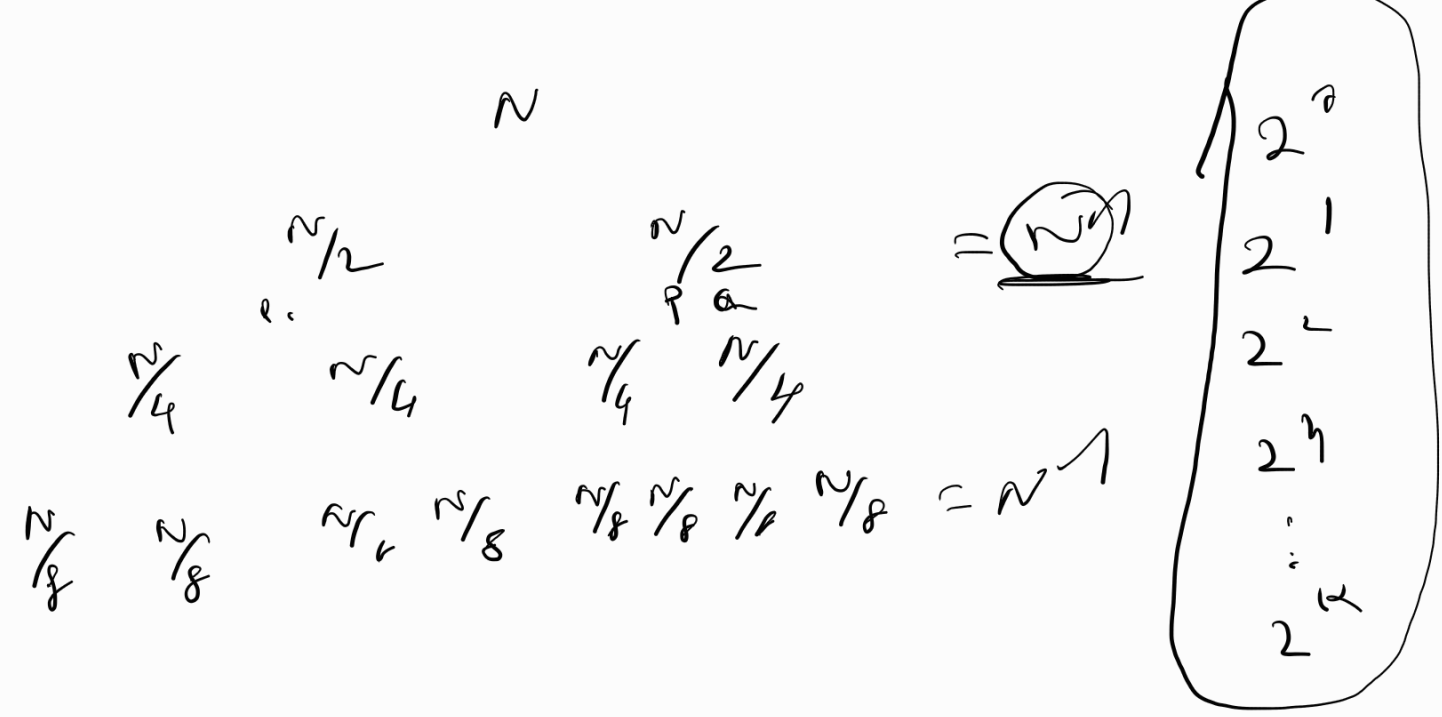
m = 1+2//2 = 1

20 | 30 40 | 10

20 | 40 30 | 10

20 | 40 30 | 10

```
mergeSort(A, l, r):
    if l == r: return
    m = (l+r)//2
    mergeSort(A, l, m)
    mergeSort(A, m+1, r)
    merge(A, l, m, r)
```



$\Rightarrow \log N$

X

$\Rightarrow$ number of function call $\times$ Time complexity

$\Rightarrow_O \log N \times N$

TC $\Rightarrow O(N \log N)$

SC $\Rightarrow O(\log N \times N)$