SORTING 3

Good Evening

To do list

01. Rearrange the digits to form the smallest no. (count sort)

02. Radix sort

03. Custom comparator

04. Sum of difference of max & min in every subset.

01. Find the smallest number that can be formed by rearranging the digits of given input.

Note !- An array is given that contains all the digits

A = { 1 3 5 2 3 }

O/P = { 1 2 3 3 5 }

A = { 3 4 0 1 1 4 2 }

O/P = { 0 1 1 2 3 4 4 }

Idea 1 → sort the array in ascending order

TC : O(nlogn)

Idea 2 → 0 00...0 | 1 1....1 | 2...2 | 3....3 | 4....4 .... 9...9

count of 0 | count of 1 | count of 2 | count of 3

freq arr of size = 10

freq[i] = freq of i^{th} ele

A = { 1, 3, 8, 3, 2, 6, 5, 3, 8 } → n length

freqarr =

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

```
for (i=0; i<n; i++){
    int val = A[i];
    farr[val]++;
}
```

TC: O(N)
SC: O(1)

```
for (i=0; i<10; i++){
    for (j=1; j≤freq[i]; j++){
        print (i);
    }
}
```

TC: O(n)
SC: O(1)

TC: O(n) SC: O(1) → countsort

1 to 10^9 → Creating the freq arr for this is going to give MLE.

Count sort → Best when range of no. is small

* Countsort on negative no.

$A = \{3, -8, -10, 3, 0, 10, 2, -6, -8\}$

$+10$: 13, 2, 0, 13

farr of size = max − min + 1 = 10 − (−10) + 1 ⇒ 21

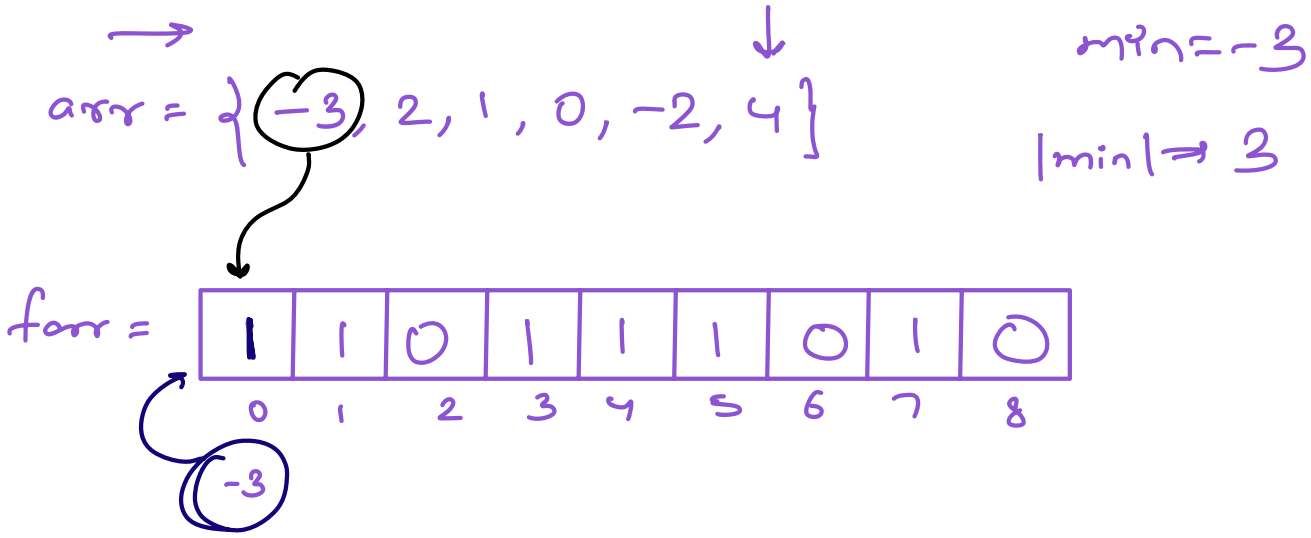
farr =

1	0	2	0	1	0	0	0	0	0	1	0	1	2	0	0	0	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10

freq[i] = freq of [i−10]

```
for (i=0; i<n; i++){
    int val = A[i];
    farr[val + |min|] ++;
}
```

```
for (i= -10 ; i ≤ 10 ; i++){
    for (j=1 ; j ≤ freq[i + |min|] ; j++){
        print (i);
    }
}
```



`j=1; j ≤ freq[i+|min|]`

$i = -3$ $\quad j=1 \; ; \; j \le freq[-3+3]$

$i = -2$ $\quad j=1 \; ; \; j \le freq[-2+3]$

$i = -1$ $\quad j=1 \; ; \; j \le freq[-1+3]$

Range = 1200 to 1700

freq arr = 1700 − 1200 + 1

1200 → 0

1201 → 1

1202 → 2

1200 − min

1201 − min

1202 − min

Smallest no. is mapped to 0th index

Q If any random is given, say n = 368

Unit digit of n = n % 10

10th digit of n = (n/10) % 10 = 6

100th digit of n = (n/100) % 10 = 3

Kth digit of n = (n/10^K) % 10

Q Sort the integer array wrt Kth digit of the number

A = {326, 18, 523} K = 0

O/P = {523, 326, 18}

A = {362, 399, 318} K = 2

O/P = {362, 399, 318}

A = {361, 432, 12, 78, 500, 112, 06} K = 1

O/P = {500, 06, 12, 112, 432, 361, 78}

Idea 1 → Custom comparator based on

$$\left(\frac{n}{10^k}\right) \% 10$$

TC: O(n log n)

Maintain stability then don't use it.

Idea 2 → Sort using count sort

freq[i] = elements where k^{th} index is i

freq[0] = {500, 06}

freq[1] = {12, 112}

freq[2] = { }

freq[3] = {432

freq[4] = { }

freq[5] = { }

freq[6] = {361

freq[7] = 78

freq[8] = { }

freq[9] = { }

* Pseudocode →

A = {361, 432, 12, 78, 500, 112, 06} K=1

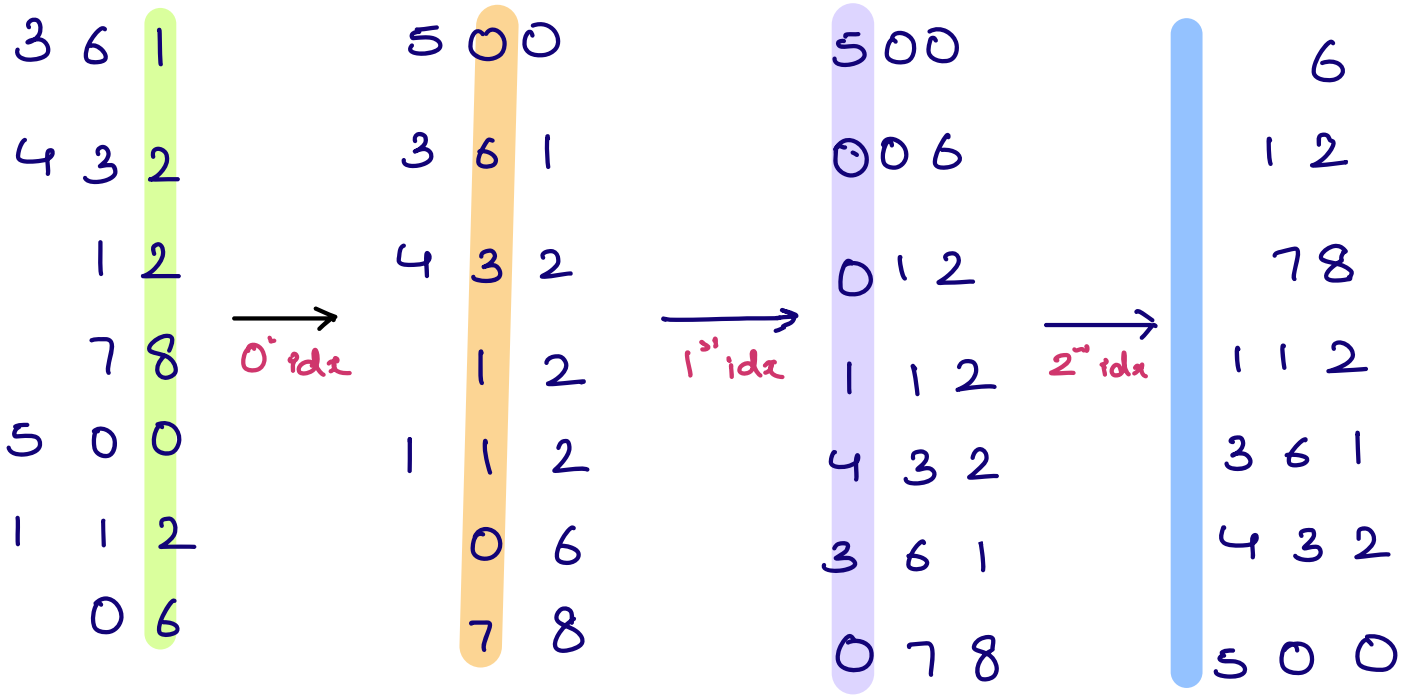
```
countsort on k(arr, k){
    list<list<I>> freq ;

    for (i=0; i<n; i++){
        int val = A[i]
        int idx = (val / 10^k) % 10

        freq.get(idx).add(val):
    }

    for (d=0; d<10; d++){
        list<I> small = freq.get(d)

        for( int k : small){
            print(k);
        }
    }
}
```

freq arr

10:18 pm

Radix Sort → Sorting digit by digit

A = {361, 432, 12, 78, 500, 112, 06}



TC: O(max no. of digits * n)

SC: O(n)

Pseudocode

01. To find the max no. of digits

↳ max no. ⟶ count of digits = c

```
K=0
02. while (c>0){
        countsortonk(arr, K)
        K=K+1
        c--;
    }
```

3

* Custom comparator

01. arr[] = {10, 3, 4, 1, 5}

Arrays.sort(arr); ⟶ 1, 3, 4, 5, 10

* Arrays.sort(arr, new Order()) //Descending

* Arrays.sort(arr, new sortk()); // $\frac{val}{10^k}$ %10

Descending

```
class Order implements Comparator<Integer>{
    public int compare(Integer a, Integer b){
        if (a > b) return -1
        else if (a<b) return 1
        else return 0
    }
}
```

if a before b return -1

else if a after b return 1

else a == b return 0

Ascending order

```
class sortk implements Comparator<Integer>{

    public int compare(Integer a, Integer b)
    {
        int val1 = (a / 10^k) % 10

        int val2 = (b / 10^k) % 10

        if (val1 < val2) return -1:      // True
        else if (val1 > val2) return 1:  // false
        else return 0:
    }
}
```

A = {3, 2, 1, 4}

subarrg = {3, 2, 1, 4} → order & ele should be continous

Subsequences = {3, 2, 4} → order should be maintained

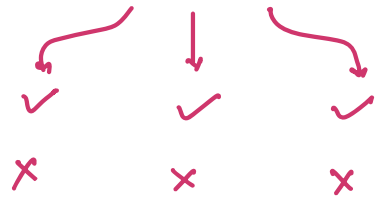
subset = {4, 2, 3} {2, 3, 4} {3, 2, 4} {2 4 3}

{4 3 2}

Q → Find the sum of (max-min) for all subsets of the array

continous & non-continous

A = {3 2 5} =



Total subsets

$= 2^3 = 8$

	max-min
{3}	3 − 3 = 0
{2}	2 − 2 = 0
{5}	5 − 5 = 0
{3, 2}	3 − 2 = 1
{3, 5}	5 − 3 = 2
{2, 5}	5 − 2 = 3
{3, 2, 5}	5 − 2 = 3
{ }	0 − 0 = 0

sum = 9

Brute force approach → Find all the subsets

iterate & find the max-min,

add the diff to sum

TC: $O(n * 2^n)$

Idea 2 → Use contribution technique

$$Ans \Rightarrow \sum_{i=0}^{n} \text{contribution of } A[i] \text{ as max \& min}$$

contribution of 3 = 3 * (max − min) = 3 * (2 − 2) = 0

contribution of 2 = 2 * (1 − 4) = −6

contribution of 5 = 5 * (4 − 1) = 15

9

$$Ans = \sum_{i=0}^{n} A[i] * (\text{subsets where } A[i] \text{ as max} - \text{subsets where } A[i] \text{ as min})$$

arr [ ] = { 3 2 8 7 4 6 }

3: ✓ ✗ | 2: ✓ ✗ | 8: ✗ | 7: ✗ | 4: ✓ ✗ | 6: ✓

subset where 6 is acting as max = 2 * 2 * 1 * 1 * 2 * 1

= 8

arr [ ] = { 3 2 8 7 4 6 }

3: ✗ | 2: ✗ | 8: ✓ ✗ | 7: ✓ ✗ | 4: ✗ | 6: ✓

subsets where 6 is acting as min = 1 * 1 * 2 * 2 * 1 * 1

= 4

subsets where A[i] is max = $2^{\#\text{ smaller ele} < A[i]}$

subsets where A[i] is min = $2^{\#\text{ greater ele} > A[i]}$

arr :- { 3, 2, 8, 7, 4, 6 }

Sort it = { 2 3 4 6 7 8 }
0 1 2 3 4 5

subsets where 6 is max = 2^i

subsets where 6 is min = 2^{n-1-i}

$$Ans = \sum_{i=0}^{n} A[i]\left(\underbrace{2^i}_{x} - \underbrace{2^{n-1-i}}_{y}\right)$$

Arrays.sort(arr)

sum = 0

$x = 2^0$

$y = 2^{n-1-i}$

```
for ( i=0; i<n; i++)
    sum+ = A[i] * (x-y)
    x = x*2
    y = y/2
3
return sum;
```

```
for (i=0; i<n; i++){
    sum+= A[i] * (2^i - 2^(n-i))
3
```

TODO → {duplicate values}