Lecture: Count and Radix sort

Agenda

- Count sort
- Radix sort
- Max and min diff in subsets.

Qu1 find smallest number that can be formed by rearranging the digit of an arr[]

arr[i] ⟶ digit from 0 to 9.

Ex: arr[] = [1 3 5 2 3]

ans: 1 2 3 3 5.

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

ans: 0 1 1 1 2 3 5 5.

Approach1 Array sort in inc^r manner.

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

↓ sort()

arr: [0 1 1 1 2 3 5 5]

TC: O(nlogn)

SC: O(1)

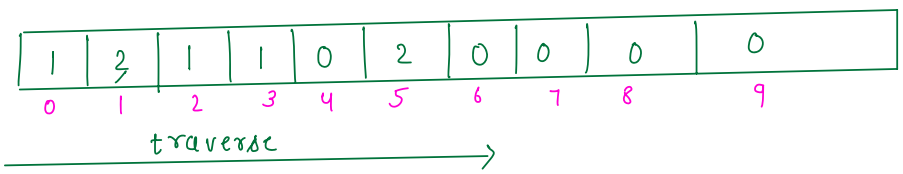
Approach2 Expected TC: O(n).

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

final op: 0 1 1 1 2 3 5 5

freq freq freq

freq[10] =



0 1 1 1 2 3 5 5 Ans

```
void getSmallestNumber(int[] arr) {

    1. create freq array.

    int[] freq = new int[10];

TC: O(n) ——— for(int el: arr) {

        freq[el] ++;

    }

    ——— for(int i=0; i<=9; i++){

        —— for(cnt=1; cnt<=freq[i]; cnt++) {

            print(i);

        }

    }

}
```

TC: O(n) + O(n + 10) ≃ O(n+10) ≃ O(n)

SC: O(10) = O(1)

size of arr — digits

↓

Count sort

Why TC ≠ O(n*10) but O(n+10)

```
for(i=1; i<=10; i++) {

    for(j=1; j<=n; j++) {

        print(ok);
    }
}
```

i	j	# itr
1	n	n
2	n	n
3	n	n
⋮	⋮	⋮
10	n	n

10 times

10 * n

freq[10] =

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

i	j (cnt)
0	1
1	3
2	1
3	1
4	0
5	2
6	0
7	0
8	0
9	0

n => len(arr)

O(n + 10)

Count sort: Sorting on the basis of freq of numbers.

↳ TC: $O(n + R)$

n ↑ len(arr) — R ↑ digits. [range of digits]

SC: $O(n)$.

Issues of count sort:

arr[n] Constraint: $n = 10^5$

$0 <= arr[i] <= 10^9$

sort arr

$n \log n$.

$10^5 * \log_2 10^9$

$10^5 * 32$

$3.2 * 10^6$

count sort

freq[10^9]

$O(n + R)$

$O(10^5 + 10^9)$ >>> $3.2 * 10^6$.

Count sort does not work very well if the range of numbers are very large

Summarise

TC: $O(R+n)$

if $R <= n$

TC: $O(n)$

if $R > n$ (or) $R > n \log n$

TC: $O(R)$.

Will count sort work for long[] arr? No [Range of long is around 10^{18}]

$N = \overset{2}{3}\overset{1}{6}\overset{0}{8}$

0th digit :- 8 → 368 % 10

1st digit : 368 % 100 = 68 [wrong]

$\left(\frac{368}{10}\right)$ % 10 = 36 % 10 = 6

2nd digit : $\left(\frac{368}{100}\right)$ % 10 = 3 % 10 = 3

⋮ ⋮

ith digit : $\frac{n}{10^i}$ % 10

Qu Sort the integer arr[] w.r.t kth digit of a number.

arr[] = [326, 18, 523, 13] k = 0

ans: [523, 13, 326, 18]

Brute force: Arrays.sort() with comparator.

```
void sort(int[] arr, int k){
Arrays.sort(arr, new Comparator<>() → {
```

search it

```
    int compare(int a, int b){
        int kth_a = a / 10^k % 10
        int kth_b = (b / 10^k) % 10
        if (kth_a < kth_b){
            return -1;
        } else if (kth_a > kth_b){
            return 1;
        }
        return 0;
    }
}
```

TC: O(n log n)

SC: O(1)

Approach 2

arr[] = [361 , 432 , 12 , 78 , 500 , 112] k = 1

ans: 500 , 12 , 112 , 432 , 361 , 78

kth digit :- [0 , 9]

freq[10][ ] ;

freq[0] :- 500 , 1st kth digit of 500 is 0

freq[1] :- 12 , 112 ⇒ 1st kth digit of 12 is 1, 112 is 1

freq[2] :-

freq[3] : 432

freq[4]

freq[5]

freq[6] : 361

freq[7] : 78

freq[8]

freq[9]

traverse

500

12

112

432

361

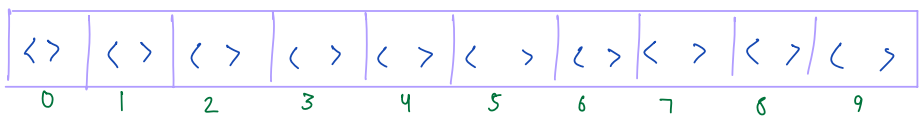
78

```
void sortOnKthDigit(int[] arr, int K) {
    1. List<List<Integer>> freq = new ArrayList<>();
    2. for(i = 0; i <= 9; i++) {
           freq.add(new ArrayList<>());
       }
    for(int el: arr) {
        int kth_digit = (el / 10^K % 10);
        freq.get(kth_digit).add(el);
    }
    for(List<Integer> list: freq) {
        for(int el: list) {
            print(el);
        }
    }
}
```

SC: O(10+n) ⇒

TC: O(10) →

freq: after line 2.



After L2 → freq.get(3).add(5) ⇒ <5>

After Line 1 → freq.get(3).add(5) : Exception

for(int el: arr) { — O(n)

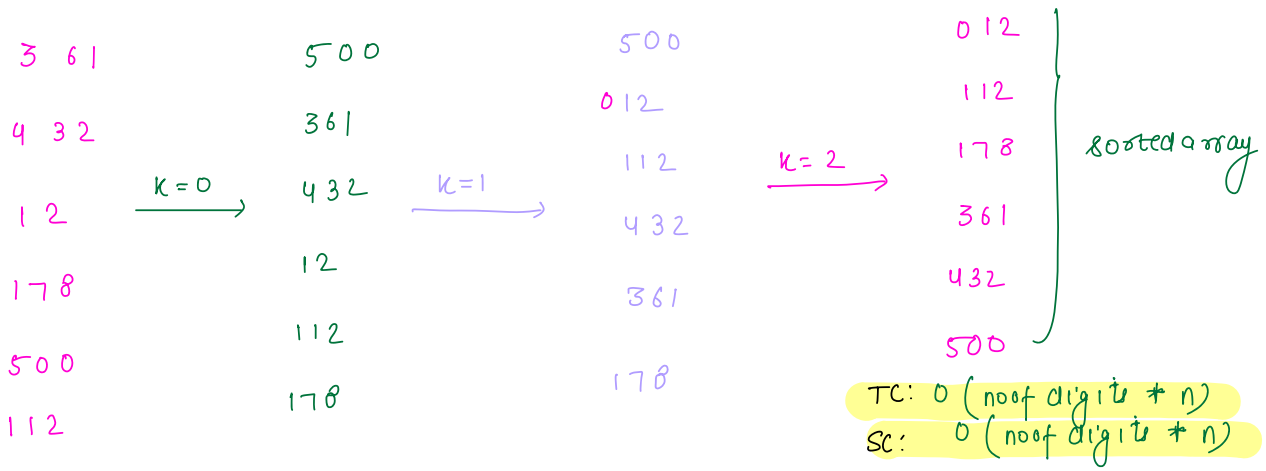
TC: O(n+10)

TC: O(n+10) ≃ O(n)

SC: O(n+10) ≃ O(n)

Radix sort: Sort the array on basis of all kth digits

arr[] = [361 , 432 , 12 , 178 , 500 , 112]

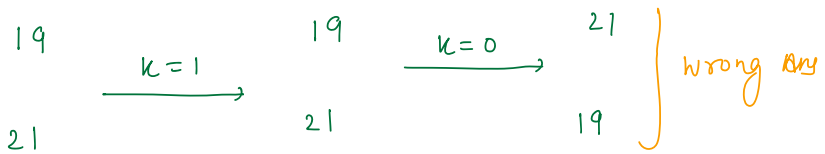


TC: O (no of digits * n)

SC: O (no of digits * n)

Q Can we sort on basis of 2nd digit, 1st digit, 0th digit

Eg: arr: [19 , 21] → final → [19 , 21]



Scenarios: $n = 10^5$

$1 <= A[i] <= 10^9$ → 10 digits

Sort → n log n. → $10^5 * \log_2 10^5$ → $10^5 * 16 \ldots$ → $1.6 * 10^6$

Radix → no of digits * n. → $10 * 10^5$ → 10^6

Problems with Radix sort:

1. Cannot work with decimal no.

arr[] = [3.14 ....., 0.12345 ....., 1.28 .....]

TC of radix = O(no of digits * O(n))

↑ very large

2. Radix sort does not work on sorting the objects. [H|W]

Eg:

	Movie1	Movie2
name:	Adhipurush	name: Pathaan
Est. coll:	150cr	est col: 500

↓

Sort movies on basis of est collection

Break: 8:40 AM

Qu: find max − min for all subsets of array.
sum

Eg: arr[] = [3 2 5]

subsets ⇒ 8 , 2^n subsets.

	max	−	min	
[ ]				→ 0
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

9 Ans

Brute force:

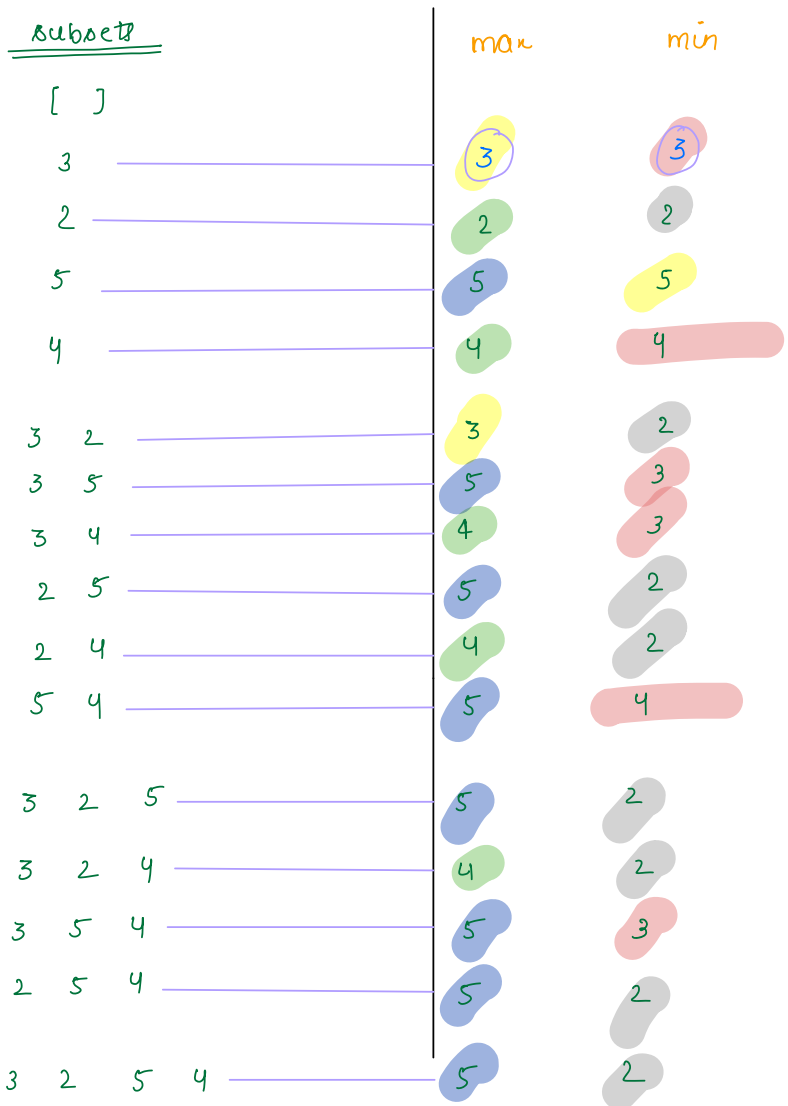
Go to all subsets of array − $O(2^n)$

$O(n)$ — for each subset, calculate max - min and update your ans.

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

Approach 2

arr[] = [3 2 5 4]



ans: 3 + 3 − 3 − 3 − 3 − 3 + 2 − 2 − 2 − 2 − 2 − 2 − 2 − 2 − 2

+ 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 − 5 + 4 + 4 + 4 + 4 − 4 − 4

ans: 3 (2 − 4) + 2 (1 − 8) + 5 (8 − 1) + 4 (4 − 2)

arr[0] arr[1] arr[2] arr[3]

2 is # times

(3) it is max in all subsets

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

ans: $3(x_1-y_1) + 2(x_2-y_2) + 8(x_3-y_3) + 7(x_4-y_4) + 4(x_5-y_5) + 6(x_6-y_6)$.

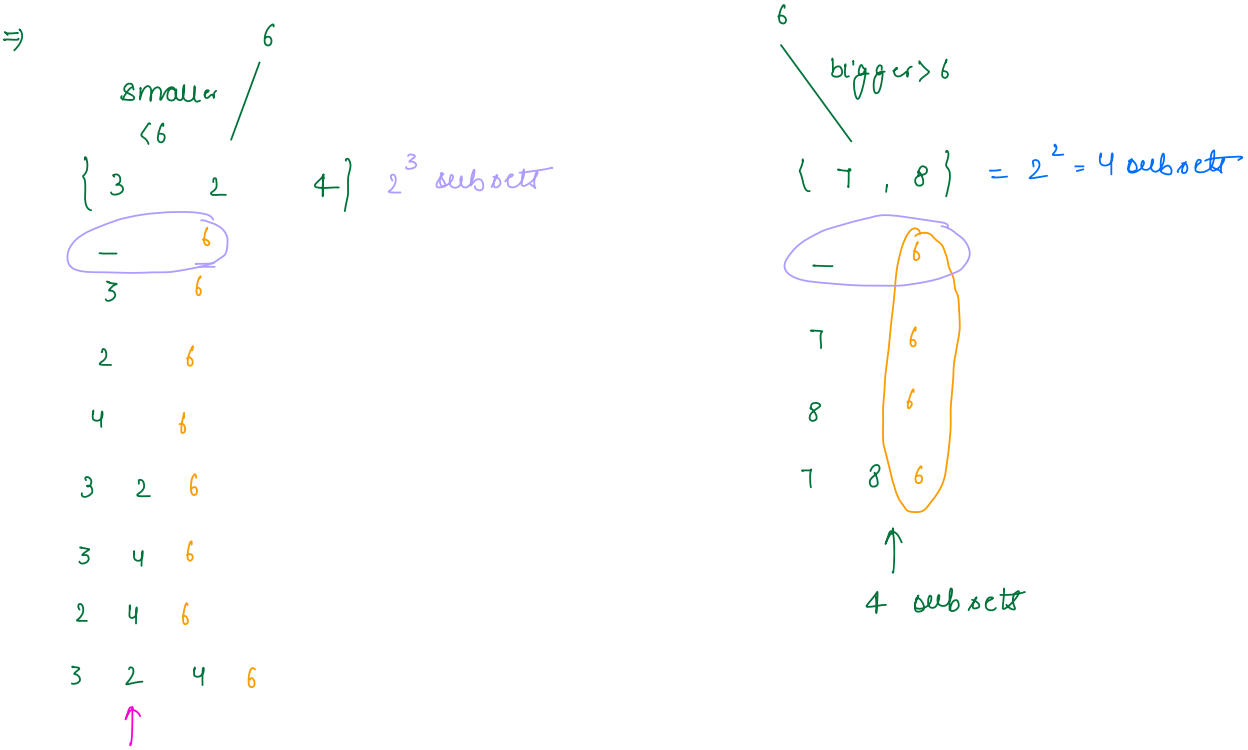
x_1 = no of times 3 is max in all subsets

y_1 = no of times 3 is min in all subsets

arr[] = [3 (0) 2 (1) 8 (2) 7 (3) 4 (4) 6 (5)]

Calculate how many times 6 is max & min in all subsets?

⇒



Does order matter in this problem?

arr[] = [3 2 8 7 4 6] (0 1 2 3 4 5) → 6 → max (8) subsets, 6 → min (4) subsets

↓ sort

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

Subset:
{ 7 8 6 }
{ 6 7 8 }
{ 8 7 6 }
{ 6 8 7 }
{ 8 6 7 }
{ 7 6 8 }

→ 6 will be min in my subset.

Logic

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

↓ sort

arr[] = [2 3 4 6 7 8] (0 1 2 3 4 5)

∃ el < 6 (2 3 4) | 2 els > 6 (7 8)

times 6 is max: $2^3 \simeq 2^i$

times 6 is min: $2^2 : 2^{n-1-i}$

0th idx:

times max: 2^0

min: $2^{6-1-0} = 2^5$

Generalise ⟹

ith idx:

↳ # times max: 2^i

min: 2^{n-1-i}

1st idx

times max: 2^1

min: $2^{6-1-1} = 2^4$

Approach 2.1 code:

```
int getMaxandMinDiffSum(int[] arr){
    Arrays.sort(arr);
    int ans = 0;

    O(n) ——— for(i=0; i<arr.length; i++){
    O(log2n) { —— int max = Math.pow(2,i);
             —— int min = Math.pow(2, n-1-i);

                ans += arr[i] * (max - min);
             }

    return ans;
}
```

TC: $O(n \log_2 n)$

SC: $O(1)$

Approach 2.2

	0th idx:	1st idx	2nd	3rd	4th	
# times max = 2^0		2^1	2^2	2^3	2^4	.....
# times min:	2^5	2^4	2^3	2^2	2^1	...

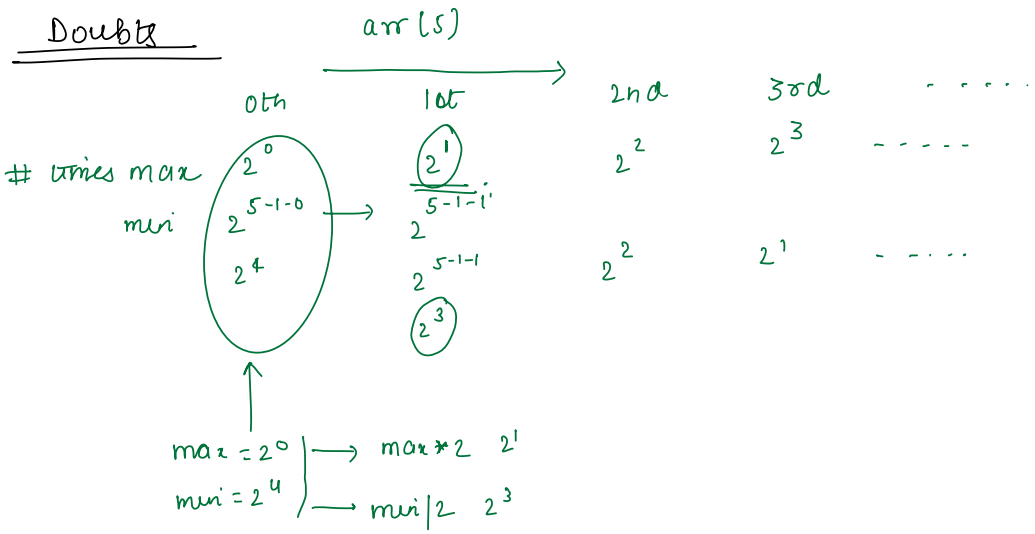
```
int getMaxandMinDiffSum(int[] arr){
    Arrays.sort(arr);
    int ans = 0;
    int max = Math.pow(2,0);
    int min = Math.pow(2,n-1-0);
O(n) ——— for(i=0; i<arr.length; i++){
        ans += arr[i] * (max - min);
        max = max*2;
        min = min/2;
    }
    return ans;
}
```

TC: O(n) ⟶ O(n log n)

SC: O(1)

Thankyou ☺

Doubts


arr[5]
0th
1st
2nd
3rd
times max
min
2^0
2^4
2^{5-1-0}
2^1
2^{5-1-1}
2^3
2^2
2^3
2^2
2^1
max = 2^0
min = 2^4
max*2 2^1
min/2 2^3