Ques

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

A[] = { 6, 3, 4, 2, 7, 2, 1 }

ans → { 1, 2, 2, 3, 4, 6, 7 }

1) Sort → nlogn.

2) Observation / Hint :-

↳ Every idx. → 0 to 9

Frequency Count → arr[10]

↓

freq. of digit.

→ reconstruct array in asc order.

→ Count Sort.

0 1 2 3 4 5 6

A[] = { ~~6~~, ~~3~~, 4, 2, 7, 2, 1 }

1 2 2

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

1 2 2 3 4 6 7

r = ~~0~~,

~~1~~

~~2~~

3

// freqArr[10];

for i → 0 to n-1 { O(n).

    f[A[i]] ++;
}

k = 0;

for d → 0 to 9 { // for each digit.

    for (i = 0; i < f[d]; i++) {

        A[k] = d
        k++
    } O(n)
}

d	i
0	—
1	—
2	—
⋮	
9	—
10	n

T.C → O(n)

S.C → O(1).

O(10+n)

↓

O(n).

↑

Count Sort

a c c c a b d e f g

→

a → 2
b → 2
c → 3

a a b b c c c

→

Sort the array → { 10, 10, 5, 5, 5, 15, 15, 2, 2, 2 }, 10^9.

$1 <= A[i] <= 10^9$

2 →	3
5 →	3
10 →	2
15 →	2
10^9 →	1

count sort
↓
creating an array of 10^9 → 4 GB of space.

if elements are upto 10^6
↓
4 MB.

Ques How to use count sort for -ve numbers?

A = [-2, 3, 8, 3, -2, 3]
↓
smallest − 2
largest → 8

Range 8 − (−2) + 1
⇒ 11.

-2	-1	0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8	9	10
2					3					1

for i → 0 to (n−1)

f[A[i] − smallest element] + 1;

3

for each idx in freq Array

while f[i] > 0 ;

Append (i + smallest_element) to A.

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

← Stable Sort →

↳ Relative Order of Equal elements should not change while sorting w.r.t a parameter

A[] = {6, 5, 3, 5 3

↓ sorting

{3, 5, 5, 6}

why?

Airport

checkin line

A 3
B 1
C 4
D 2
E 5

B
D
A
C
E

Name	Marks
A	8
B	5
C	8
D	4
E	8

→ Sort

Name	Marks
D	4
B	5
A	8
C	8
E	8

Inplace → No Extra Space
↳ S.C → O(1).

Merge Two Sorted Arrays

Ques Given 2 Sorted Arrays, A[N] & B[M], Merge, create & return a new Sorted Array.

a[3] = {-1, 4, 8}

b[2] = {2, 9}

c[5] = {-1, 2, 4, 8, 9}

idea (Brute Force) :-

A[N] B[M], C[N+M]

1) Copy A[] → C[] → O(n)
2) Copy B[] → C[] → O(M)
3) Sort C[] → (n+m) log (n+m)

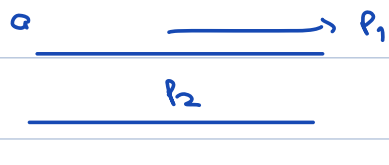
idea 2 :-

p_1

$a[7] = \{ -5, -1, 3, 7, 10, 12, 15 \}$ (indices 0 1 2 3 4 5 6)

p_2

$b[5] = \{ -4, 0, 2, 8, 9 \}$ (indices 0 1 2 3 4)



p_3

$c[12] =$ 0 1 2 3 4 5 6 7 8 9 10 11

-5

```
int [] merge (int A[], int B[], int n, int m) {
    int p1 = 0, p2 = 0, p3 = 0;
    int C[N+M];
    while (p1 < N && p2 < M) {
        if (A[p1] < B[p2]) {
            C[p3] = A[p1];
            p1 = p1+1;
            p3 = p3+1;
        }
        else {
            C[p3] = B[p2];
            p2 = p2+1;
            p3 = p3+1;
        }
    }
```

```
while (p1 < N) {
    C[p3] = A[p1];
    p1 = p1+1;
    p3 = p3+1;
}

while (p2 < M) {
    C[p3] = B[p2];
    p2 = p2+1;
    p3 = p3+1;
}
return C;
}
```

T.C → O(N+M)

S.C → O(1).

Ques) Given N elements, 3 idxes, s, m, e.

Subarray [s m] is sorted

Subarray [m+1 e] is sorted.

you have to sort subarray [s e].

		s			p1	m			e	p2	
	0	1	2	3	4	5	6	7	8	9	10	11
arr[12] = {	4	8	-1	2	6	9	11	3	4	7	13	0 }

s = 2
m = 6
e = 9

temp

0	1	2	3	4	5	6	7
-1	2	3	4	6	7	9	11

temp

	0	1	2	3	4	5	6	7	8	9	10	11
arr[12] = {	4	8	-1	2	3	4	6	7	9	11	13	0 }

```
void merge (int A[], int s, int m, int e ) {
    int p1 = s , p2 = m+1 , p3 = 0;
    int temp [e-s+1];
    while (p1 <= m && p2 <= e) {
        if (A[p1] < A[p2]) {
            temp [p3] = A [p1];
            p1 = p1+1;
            p3 = p3+1;
        }
        else {
            temp [p3] = A [p2];
            p2 = p2+1;
            p3 = p3+1;
        }
    }

    while (p1 <= m) {
        temp[p3] = A [p1];
        p1 = p1+1;
        p3 = p3+1;
    }
```

T.C → O(N)

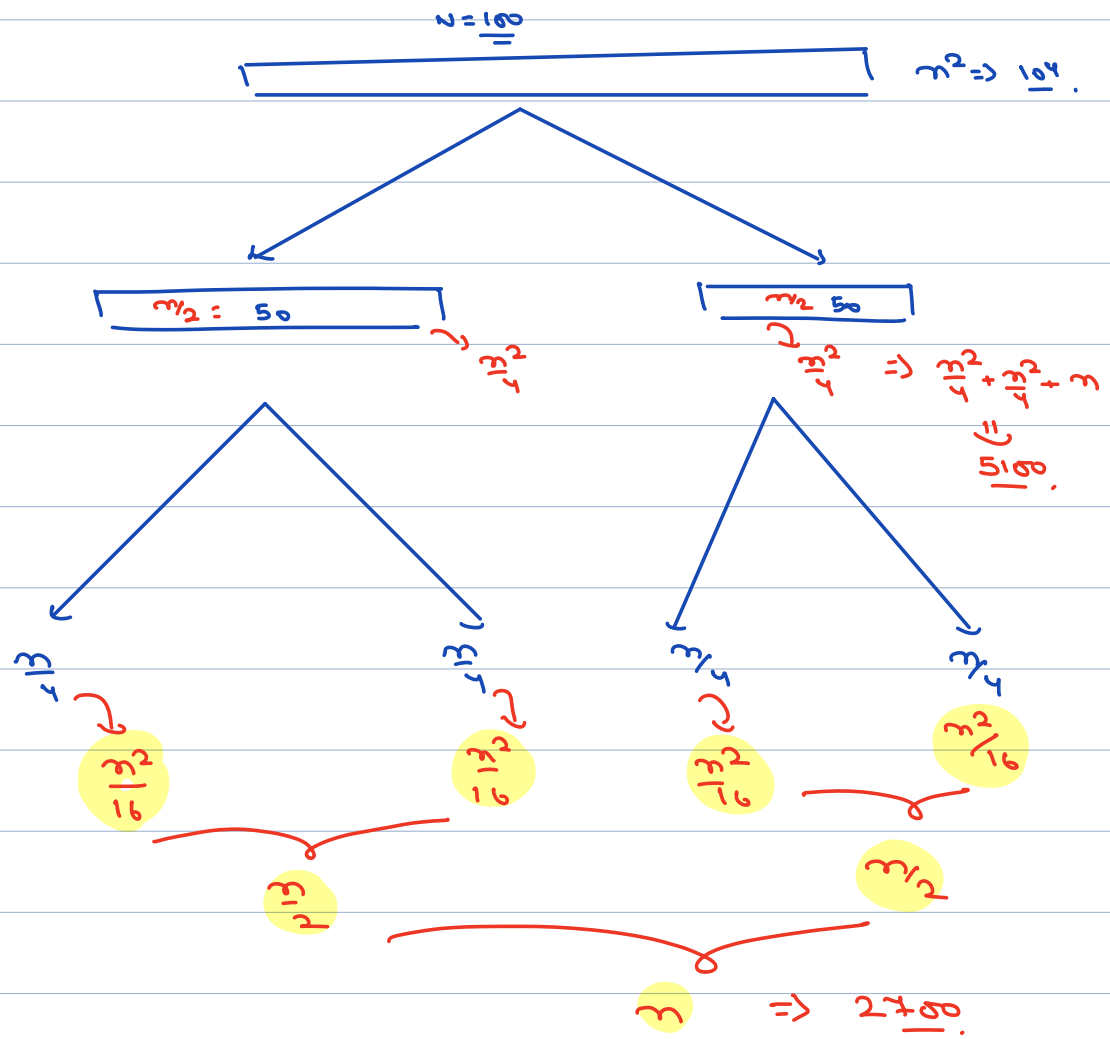
S.C → O(N)

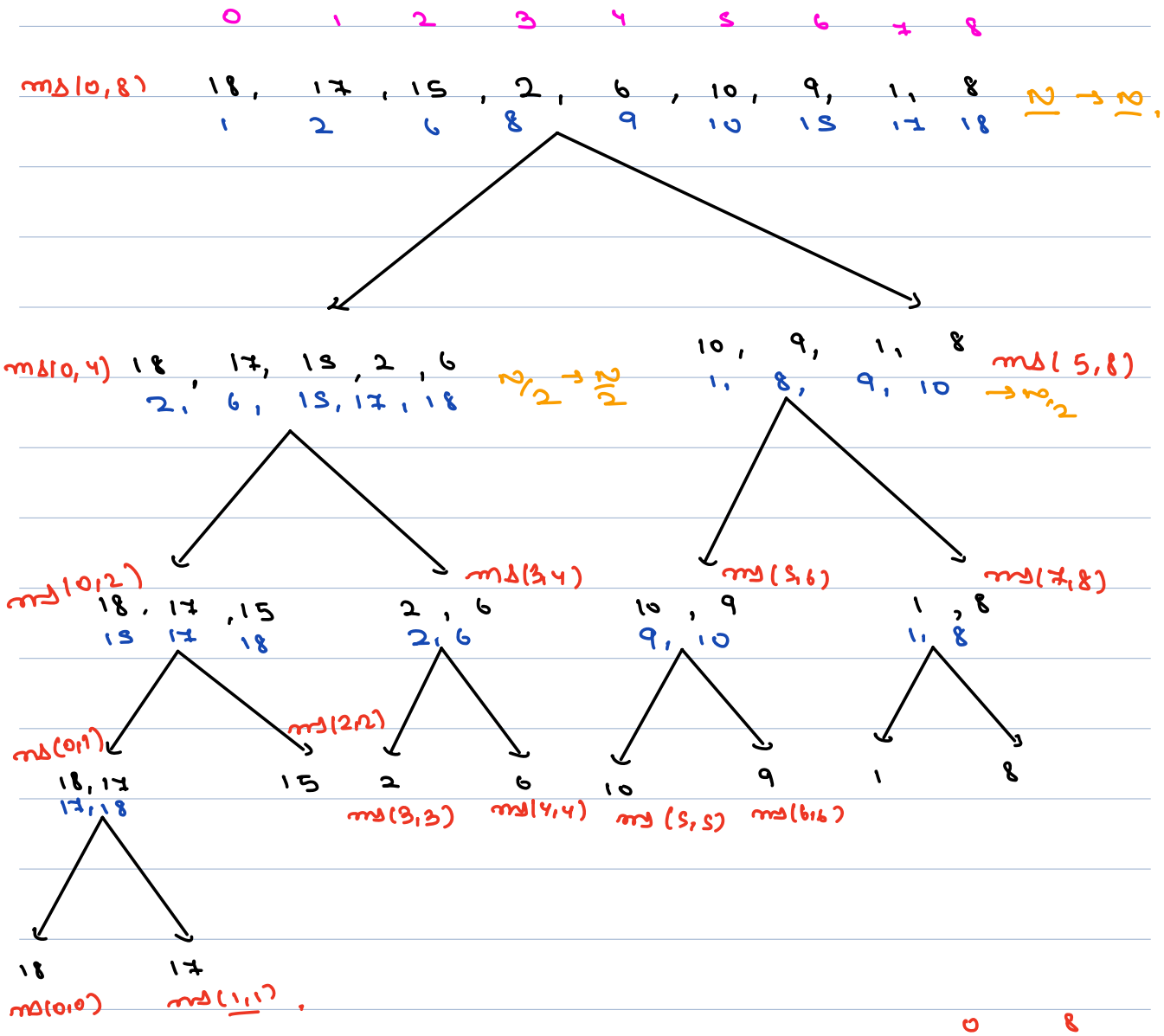
```
while ( p2<=E ) {
    temp [p3] = A [p2];
    p2 = p2+1;
    p3 = p3+1;
}

for (i=0 ; i<= e-s; i++) {
    A[s+i] = temp[i];
}
}
```

Break 8:40 Am - 8:50 Am

-: Merge Sort :-

Bubble Sort → Today
Selection Sort → $O(n^2)$

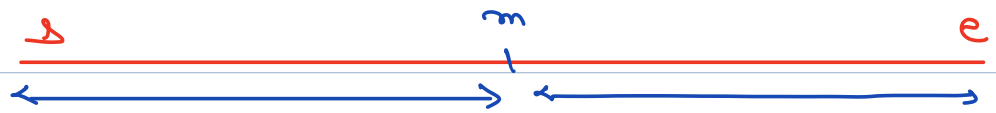


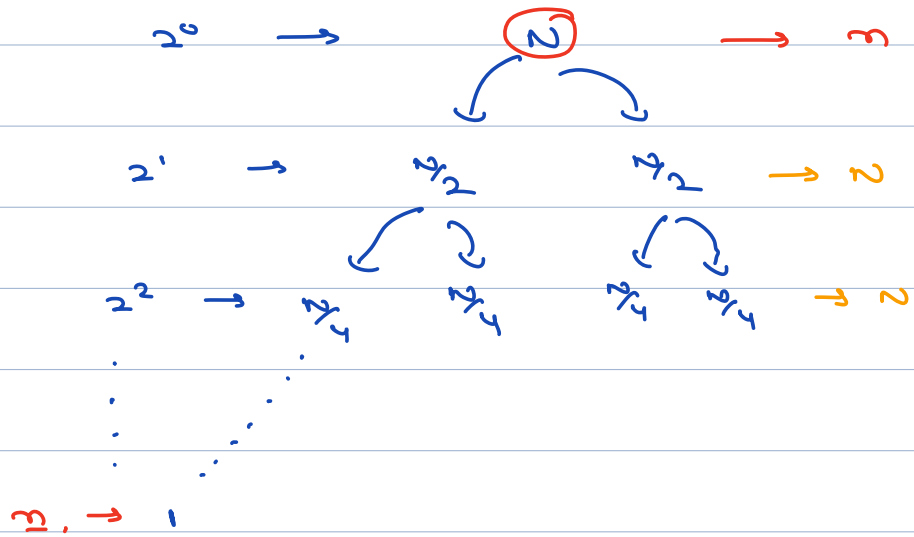


0 ____ 8

```
void mergeSort (int A[], int s, int e) {
    if (s==e) { return }

    m = (s+e)/2;
    mergeSort (A, s, m);      ms(0,4)
    mergeSort (A, m+1, e);    ms(5,8)
    merge (A, s, m, e);
}
```

ms (0, 8) m = 4





T.C → n log n

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

T.C → $O(n \log n)$.

S.C → $O(\log n) + O(n)$

$O(\log n)$ → stack space

$O(n)$ → merge algorithm's temporary array.