Count Sort

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.

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

-> {1, 2, 2, 3, 4, 6, 7, 3}

arr[] → 4, 2, 7, 3, 9, 0

→ 0, 2, 3, 4, 7, 9

idea :-

→ sort the array → O(nlogn).

idea2 :-

freq →

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

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

freq →

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

ans → 1, 2, 2, 3, 4, 6, 7

```
int freq[10];
for i → 0 to n-1;          → n
    int val = arr[i];
    freq[val]++
```

3

T.C → O(n)
S.C → O(1).

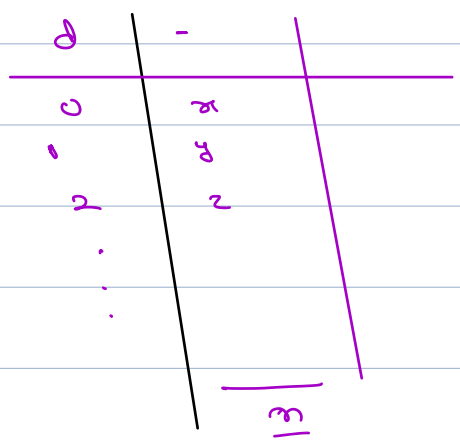
```
for d → 0 to 9
    for i → 1 to freq[d]
        print(d);          → O(n)
```

3

3

freq →

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



Count sort

arr[] → [10, 20, 80, 999]

0 . . . 999

Will Count Sort work if the range of A[i] is more than 10^9?

Integer → 4 Bytes.

arr[10^9] → space → $10^9 \times 4$ => 4 GB.

arr[10^6] → space → $10^6 \times 4B$ = 4 MB (manageable)

- Count Sort isn't suitable for a range of 10^9 because a frequency array of this size would demand too much memory.
- Count Sort works well when the range of A[i] is ~ 10^6.

Each integer typically occupies `4 Bytes`.

Storing 10^9 integers requires 4GB, which is often impractical. An array up to 10^6 in length is more manageable, needing 4MB.

Count sort on -ve numbers

int val = arr[i]

f[val - smallest]

A = [-2, 3, 8, 3, -2, 8]

Smallest = -2, largest = 8

Range = 8 - (-2) + 1 = 11

freq = [2 _ _ 3 _ _ _ _ _ 1]

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

-2, -2, 3, 3, 3, 8

freq[largest - smallest + 1];

for i → 0 to n-1 → O(n)

    f[A[i] - smallest]++;

3

for each idx in freq. Array

    while (f[i] > 0) :

        print (i + smallest elmt)

        O(n)

    3

2

T.C → O(n)

S.C → O(Range)

Merge Two Sorted Arrays

Given an integer array where all odd elements are sorted and all even elements are sorted. Sort the entire array.

A[] = { 2, 5, 4, 8, 11, 13, 10, 15, 21 }

0 1 2 3 4 5 6 7 8

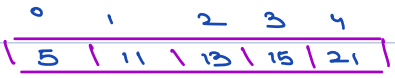
ideal :- Sort the whole array → nlogn.

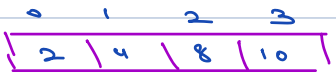
idea 2

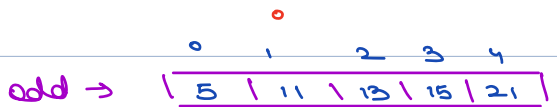
A[] = { 2, 5, 4, 8, 11, 13, 10, 15, 21 }

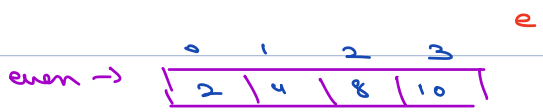
odd → 5

even → 4

odd → 

even → 





final sorted Array

2 4 5 8 10 11 13 15 21

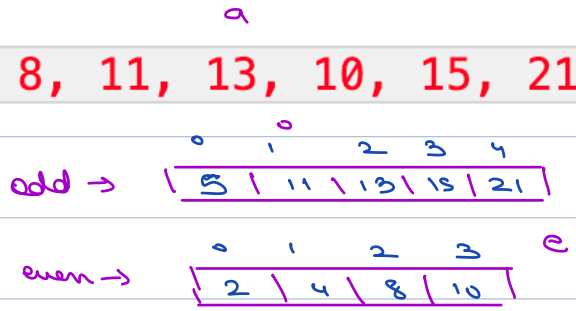
void merge (A[]) {

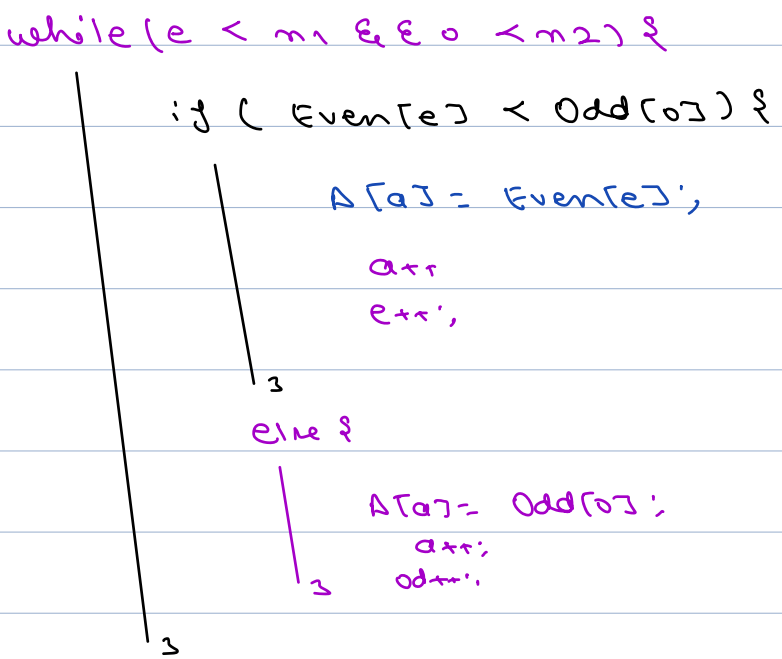
```
int N = A.length();
//n1: count of even elements
//n2: count of odd elements
int EVEN[n1], ODD[n2];
int a=0, e=0, o=0;
```

```
for(int i=0; i<N; i++) {
    if(A[a] % 2 == 0) {
        EVEN[e] = A[a];
        e++;
    }
    else {
        ODD[o] = A[a];
        o++;
    }
    a++;
}
```

a = 0;
o = 0;
e = 0;

A[] = { 2, 5, 4, 8, 11, 13, 10, 15, 21 }






```
while (e < n1) {
    A[a] = EVEN[e];
    e++;
    a++;
}

while (o < n2) {
    A[a] = ODD[o];
    a++;
    o++;
}
```

T.C → O(n)

S.C → O(n)

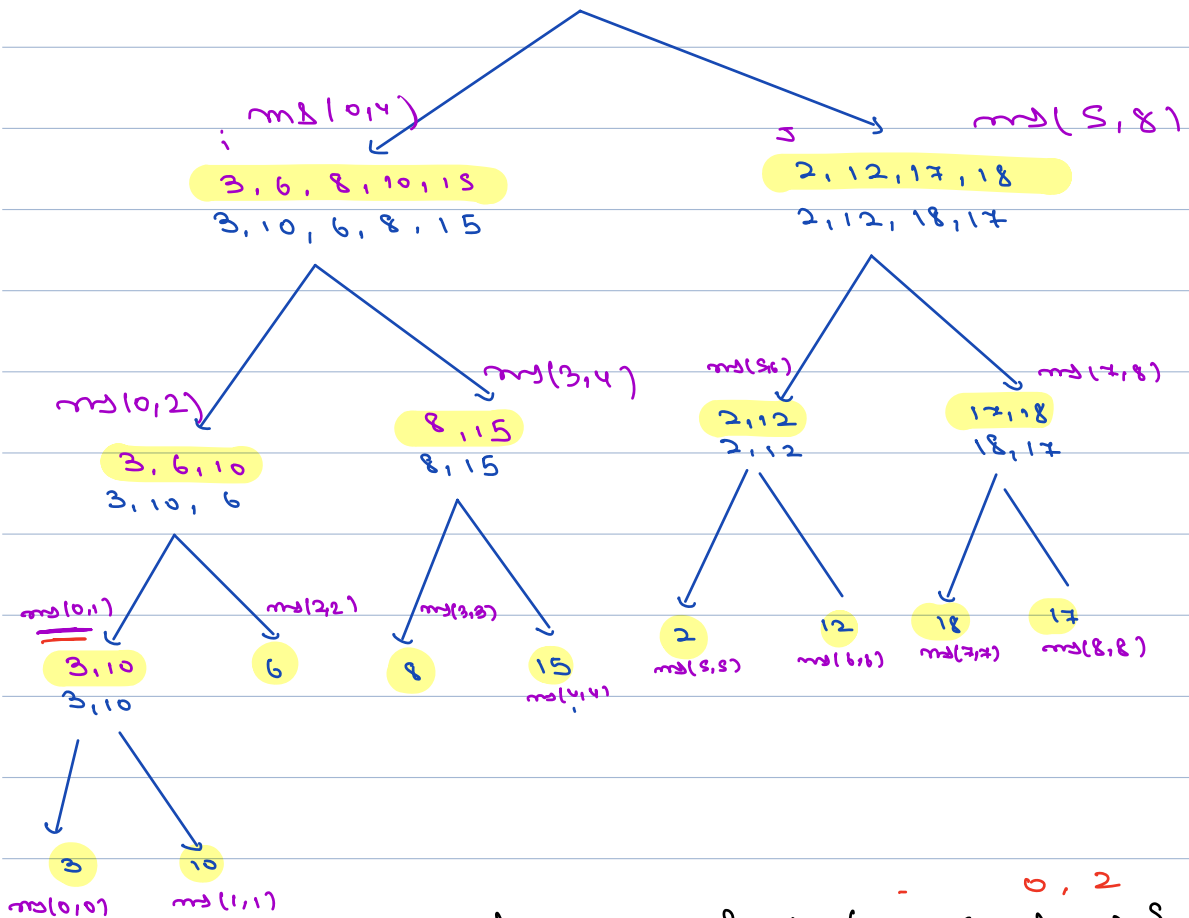
mergesort(0,8)

Merge Sort

0 1 2 3 4 5 6 7 8 ms(0,8)

2, 3, 6, 8, 10, 12, 15, 17, 18

3, 10, 6, 8, 15, 2, 12, 18, 17



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

```
void mergesort (arr, l, r) {        // 0, 2
    if (l == r) {return;}
    int mid = (l+r)/2
    mergesort (arr, l, mid);
    mergesort (arr, mid+1, r);
    merge(arr, l, mid, r);
}
```

3

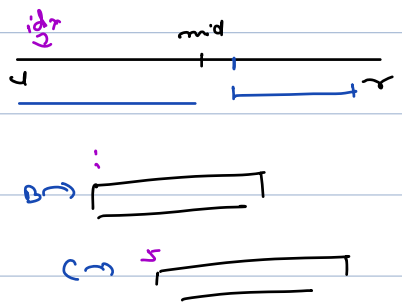
void merge (arr, l, mid, r) {

```
int N = A.length();
int n1 = mid-l+1;
int n2 = r-mid;
```

```
int B[n1], C[n2];

int idx=0;
for(int i=l; i<=mid; i++){
    B[idx] = A[i];
    idx++;
}

idx=0;
for(int i=mid+1; i<=r; i++){
    C[idx] = A[i];
    idx++;
}
```

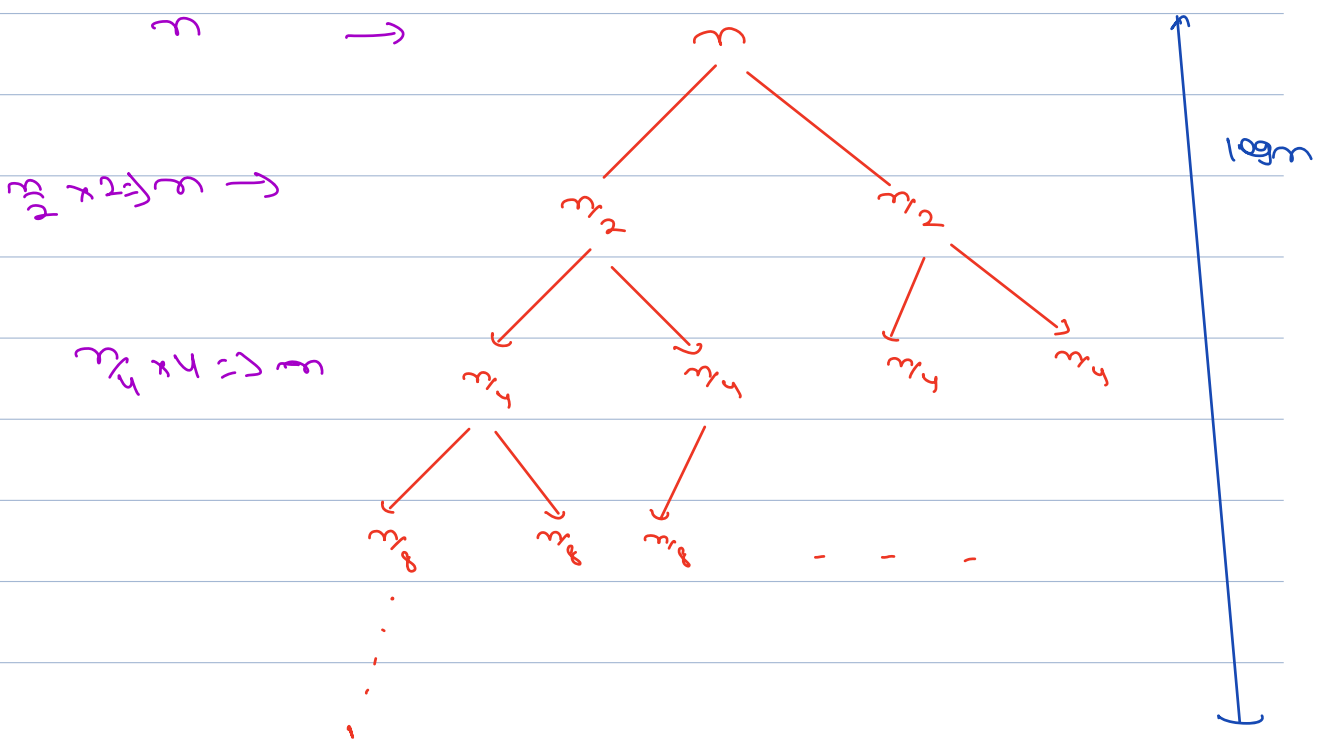



```
idx = l;
i = 0; // moves over A
j = 0; // moves over B

while (i < n1 && j < n2) {
    if (B[i] <= C[j]) {
        A[idx] = B[i];
        i++;
    } else {
        A[idx] = C[j];
        j++;
    }
    idx++;
}
```

```
while (i < n1) {
    A[idx] = B[i];
    idx++;
    i++;
}

while (j < n2) {
    A[idx] = C[j];
    idx++;
    j++;
}
```



T.C → O(logn) n ⇒ O(nlogn).

S.C → O(logn) + n

stack merge.

Stable Sort

Relative order of equal elements should not change while sorting w.r.t a parameter.

arr[] = {6, 5, 3, 5}

↓ sort

arr[] = {3, 5, 5, 6}