# Sorting 1 : CountSort & MergeSort

## Question

find the smallest no. that can be formed by rearranging the digits of a given no. in an array. $0 <= A[i] <= 9$

A = [ 3 2 4 1 ] → [ 1 2 3 4 ]

A = [ 6 3 4 2 7 2 0 ] → [ 0 2 2 3 4 6 7 ]

Soln : sorting in ascending order

Inbuilt sorting : TC = O(NlogN)

## Observation

Since A[i] is in [0, 9] range, can sort faster.

output will always look like :

000...0 11...1 22...2 33...3 . . . . . 88...8 99....9

freq(0) freq(1)

## Approach

1. Calculate freq. of each value from 0 to 9.
2. Use freq. array to calculate answer.
3. It is called **Count Sort**.

A = [ 9 2 9 7 1 1 9 0 0 2 9 1 3 7 ]

f = [ 0 1 2 3 4 5 6 7 8 9 ]

f = [ 2 3 1 1 0 0 0 2 1 4 ]

f(i) → frequency of i (0-9) in array

0 0 1 1 1 2 3 7 7 8 9 9 9 9

Code

```
f[10]
fi, fill =0

for(i=0 to n-1){
    f[A[i]) ++
}
```

→ calculating frequency — N iterations

```
for( d = 0 to 9){
    for( i=1 to f[d] ){
        print(d)
    }
}
```

sort array using freq. array

iteration → 10N or N ✓

| d | i | iteration |
|---|---|---|
| 0 | | f[0] |
| 1 | | f[1]++ |
| 2 | | ⋮ ++ |
| ⋮ | | ⋮ |
| ⋮ | | ⋮ |
| 9 | | f[9]++ |

f[0] + f[1] + . . . + f[9] = N

TC = O(N)

SC = O(10) = O(1)

size of freq. a

What if $0 <= A[i] < 10^9$ → Is count sort possible?

length of freq. array $= 10^9$

integer array

4B

$4B \times 10^3 \times 10^3 \times 10^3$ ($10^9$)

$\sim 4KB$

$\sim 4MB$

$\sim 4GB$ (Array space)

acceptable range:

$[10^6 - 10^7)$

Memory limit Exceeded (MLE) error

What if $-9 <= A[i] <= 9$ → Is count sort possible?

len $= [-9, 9) = 9 - (-9) + 1 = 19$ ✓ Yes

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| A = | -2 | 3 | 8 | -4 | -2 | 3 | 0 |

| | 0 | 1 | 2 | ... | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | ... | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| f = | 0 | 0 | 0 | | 1 | 0 | 2 | 0 | 1 | 0 | 0 | 2 | | 0 | 1 | 0 |
| | -9 | -8 | -7 | | -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | | 7 | 8 | 9 |

A = [-4 -2 -2 0 3 3 8]

code

```
f[9]
fi, f[i] = 0

for(i=0 to n-1) {
    f[A[i]]++  -->  f[A[i] + 9]++   // f[A[i] - minElement]
}

for(d=0 to 18) {
    for(i=1 to f[d]) {
        print(d)  -->  print(d - 9)   // print(d + minElement)
    }
}
```

in this case minElement = -9

TC = O(N)  SC = O(range of array)

What if $10^9 <= A[i] <= 10^9 + 100$

count sort possible.

$(max(A[i]) - min(A[i]) + 1) <= 10^6$

use count sort

Gmail All Inboxes

Account A = [ 1 5 6 9 ]

B = [ 2 4 8 ]

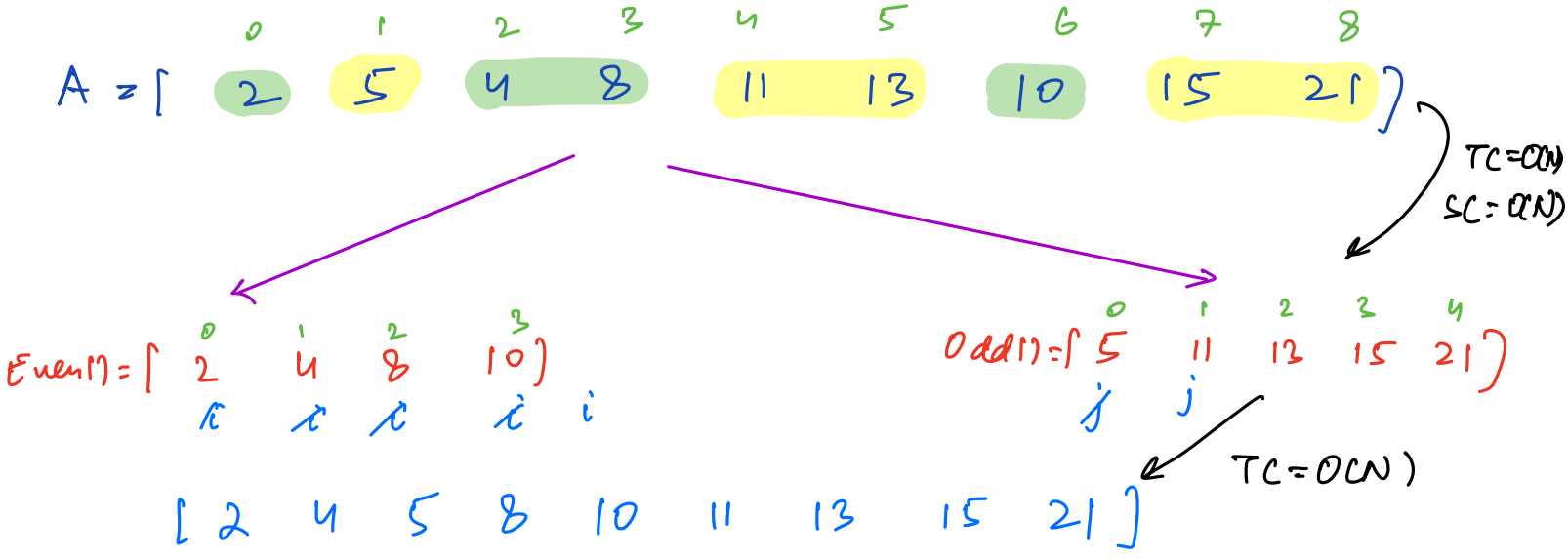
output = [ 1 2 4 5 6 8 9 ]

This is direct application of merge 2 sorted arrays.

## Question

Given an integer array where all odd elements are sorted & all even elements are sorted.

Sort the entire array.



Code

```
merge ( A[1] ) {
    n = A.length
    n1 = 0 , n2 = 0   // n1: count of even elements
                      // n2: count of odd elements

    for (i=0 to n-1) {
        if (A[i] % 2 == 0)
            n1++
        else
            n2++
    }

Even[n1] , Odd[n2]

    i=0 , j=0   // even & odd start index

    for ( K=0 to n-1) {
        if ( A[K] % 2 == 0) {
            Even[i] = A[K]
            i++
        }
        else {
```

count total even & odd elements

→ copy the data

N iterations

```
            Odd[j] = A[K]
            j++
        }
    }

i=0, j=0, K=0

while( i<n1 && j<n2) {
    if ( Even[i] <= Odd[j]) {
        A[K] = Even[i]
        K++, i++
    }
    else {
        A[K] = Odd[j]
        K++, j++
    }
}

while( i<n1) {
    A[K] = Even[i]
    K++, i++
}
while( j<n2) {
    A[K] = Odd[j]
    K++, j++
}
```

N iterations

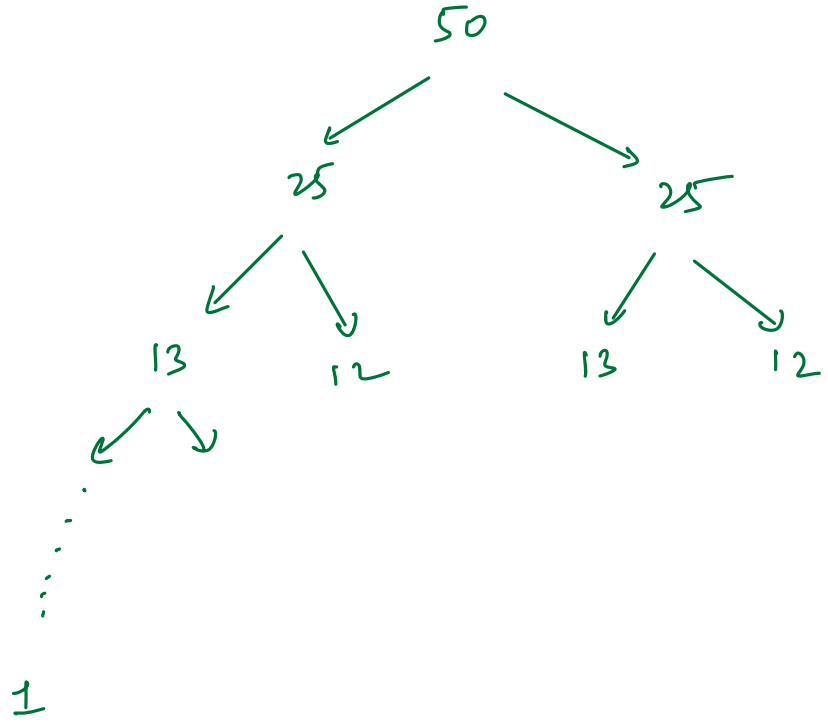
Even[] = [ 2 4 8 10 ]
(0 1 2 3)
i i i i i

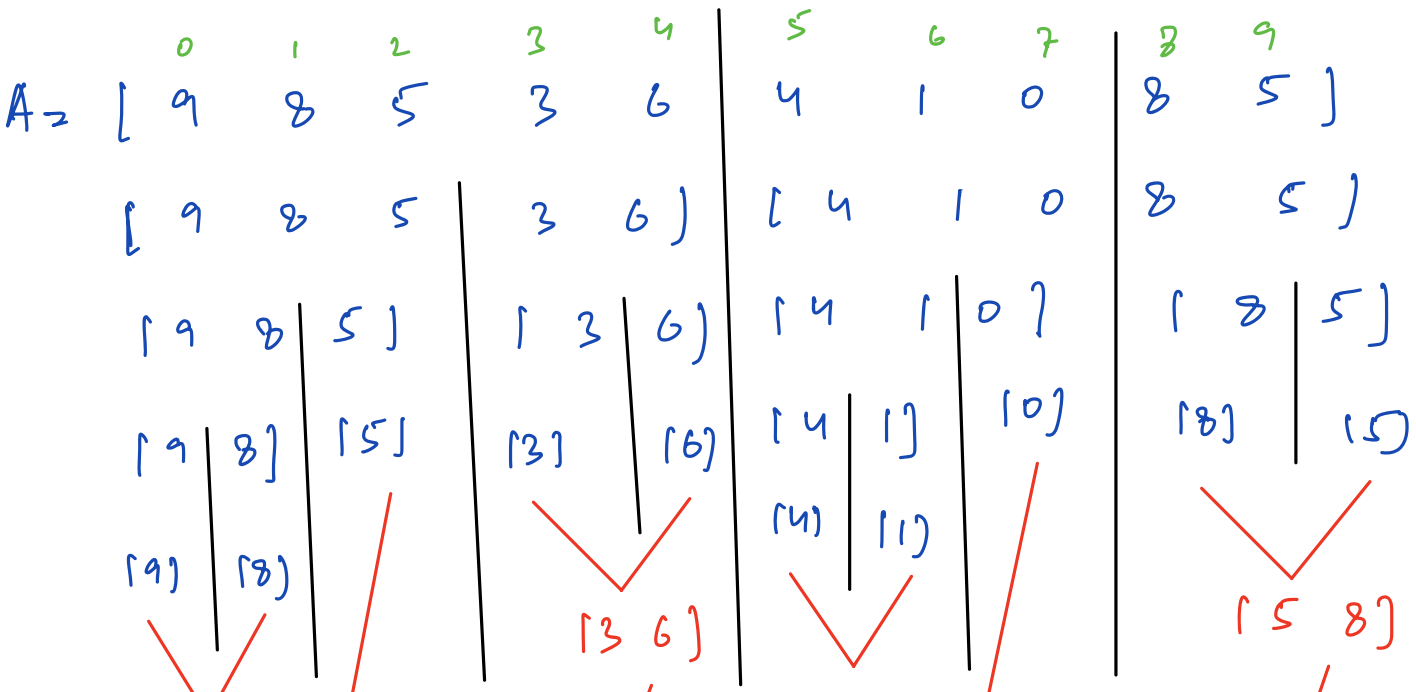
Odd[] = [ 5 11 13 15 21 ]
(0 1 2 3 4)
j j j j j j

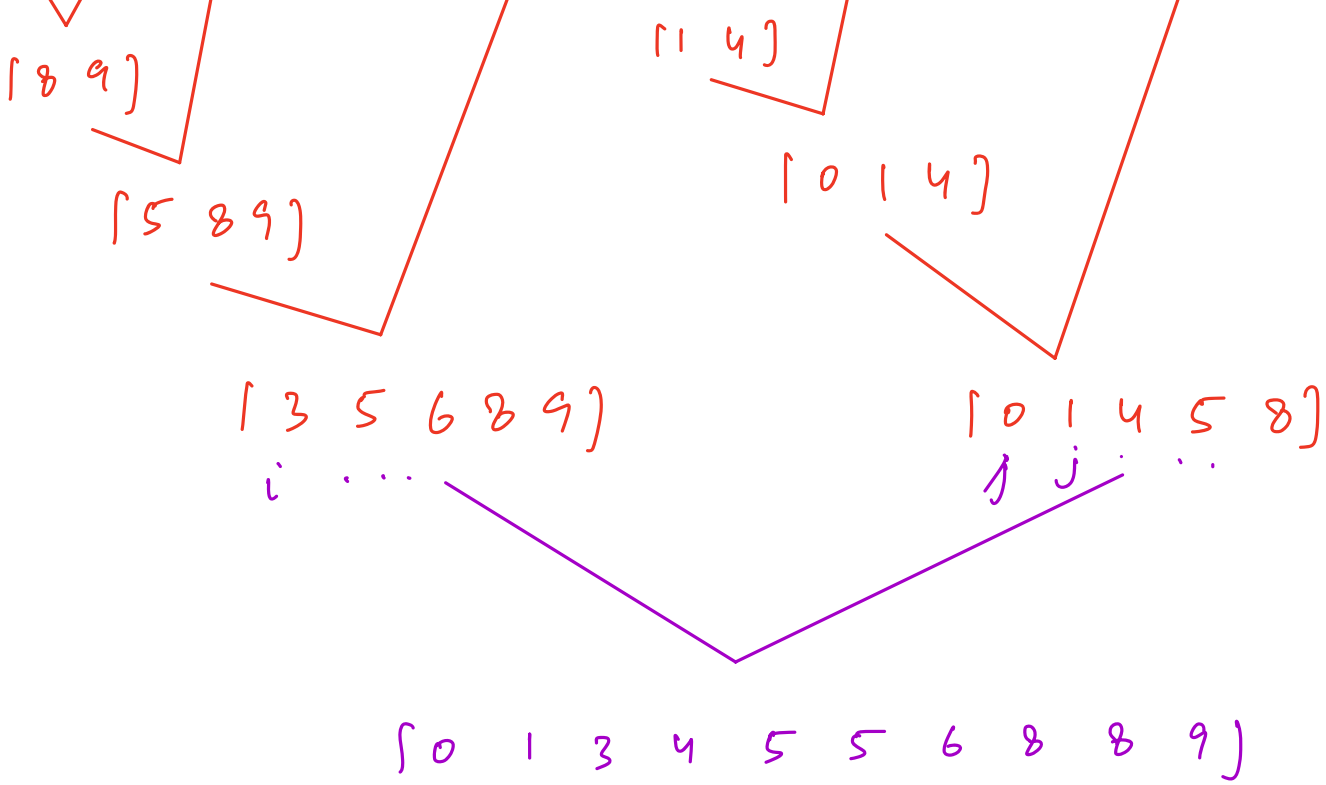
A[] = [ 2 4 5 8 10 11 13 15 21 ]
K K K K K K K K K

TC = O(N)  SC = O(N)

## Merge Sort





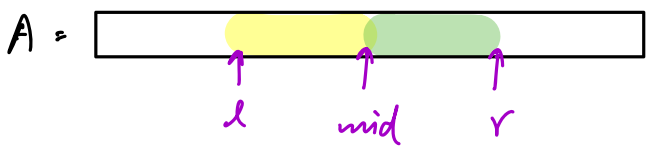

This algo. divides the array into multiple subproblems & merge them one by one →

Merge Sort.

Since we are breaking array recursively, use recursion.

Code

```
void mergeSort(A[], l, r) {        // l: 0, r: n-1
    if (l >= r) return
    mid = (l+r)/2
    mergeSort(A, l, mid)
    mergeSort(A, mid+1, r)
    merge(A, l, mid, r)
}
```




```
void merge(A[], l, mid, r) {
    n1 = mid - l + 1      // [l, mid]
    n2 = r - mid          // [mid+1, r]
    B[n1], C[n2]
    idx = 0
    for (i = l to mid) {
        B[idx] = A[i]
        idx++
    }
    idx = 0
```

```
for( i= mid+1 to r) {
    C[idx] = A[i]
    idx++
}

i=0, j=0
idx=l
while (i<n1 && j<n2) {
    if (B[i] <= C[j]) {
        A[idx] = B[i]
        i++, idx++
    }
    else {
        A[idx] = C[j]
        j++, idx++
    }
}
while ( i < n1 ) {
    A[idx] = B[i]
    i++, idx++
}
while ( j < n2) {
```

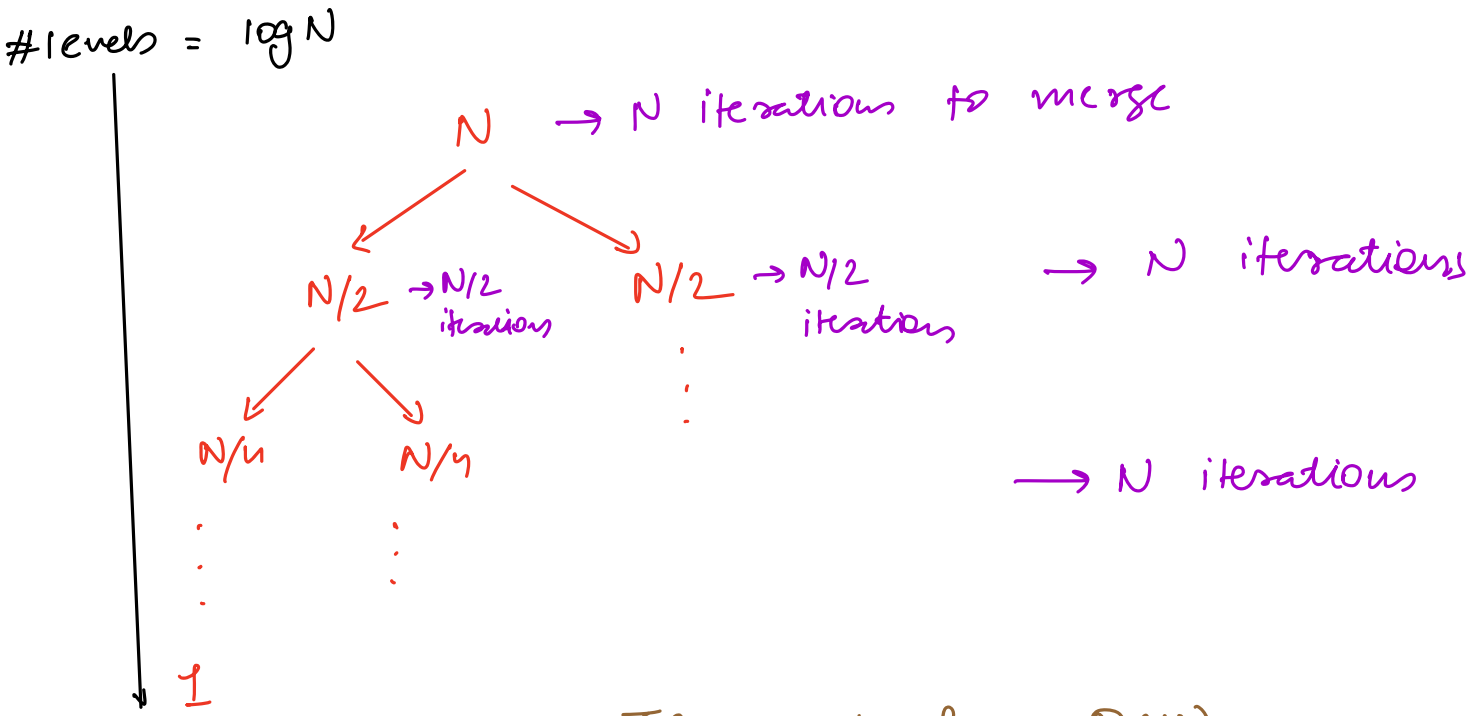
stable sort

TC = O(r-l+1)

= O(N)

```
        A[idx] = t[j]
        j++, idx++
    }
}
```

## Complexity Analysis of mergesort




TC per level = O(N)

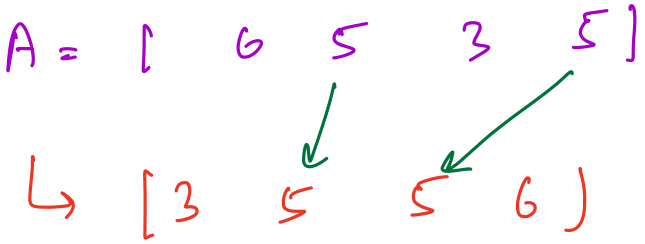
total TC = O(N log N)

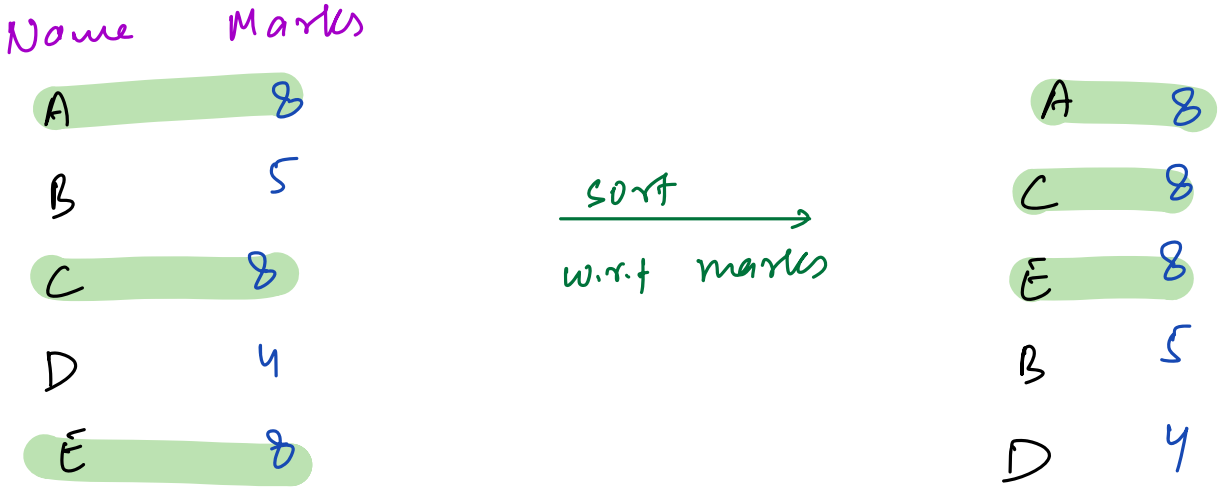
SC = O(N + log N) = O(N)

recursion Stack

## Stable Sorting

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




## Inplace Sorting

If no extra space is needed to sort, it is called Inplace Sorting.

if SC = O(1) ⇒ Inplace

Count sort → stable ✗
in place ✓

Merge sort → stable ✓
in place ✗

Doubt

[4 1 10 15]

1 0 1 1

```
am = 0
c = 0
for(i=0 to n-1){
    if(a[i] == 1){
        c++
    }
    else{
```

```
        c = 0
    }
    am = max(am, c)
}
```