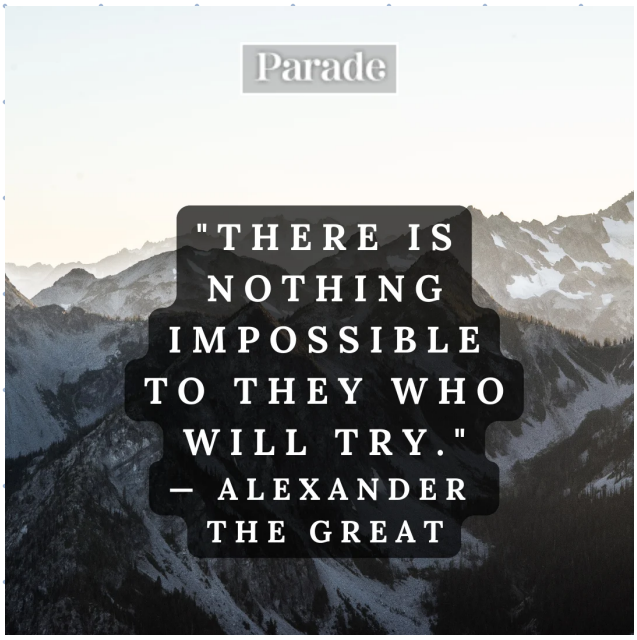
Sorting : Count Sort & Merge Sort



Good Evening

Content

01. Smallest no. from digits
02. Merge Two sorted Arrays
03. MergeSort Idea
04. Merge sort
05. Inversion count

OnePlus is a phone manufacturing company, where phones of various models are being made. After manufacturing, these phones end up in a mixed line, not organized by model.

OnePlus wants to pack the same models of phones together to make packaging easier. So, they've decided the best way to do this is to arrange the phone models in ascending order by their model numbers. They have 3 models being manufactured presently.

Problem :

You are given a list of numbers **A** where each number represents a model of phone coming off the manufacturing line. Your job is to sort these numbers so that the same models are grouped together, making it ready for packaging.

Manufacturing line = {1, 2, 3, 2, 2, 1, 3, 3, 3}

Sorted packaging line = {1, 1, 2, 2, 2, 3, 3, 3, 3}

Arrays.sort(A)

→ Sort the entire data in ascending order

TC: O(nlogn)

Q Rearrange the given digits & find the smallest no. that can be formed.

Note → An array contains all these digits

A[] = {1, 3, 5, 2, 3}

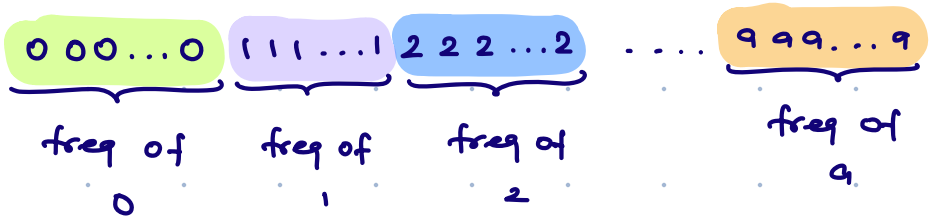
O/P[] = {1, 2, 3, 3, 5}

A[] = {3 4 0 11 4 9}

O/P[] = {0 1 1 3 4 4 9}

Observation → All the digits lie in between 0 to 9

Idea 2 -



size of freq Arr = 10

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

A[] = {1 3 8 3 2 6 5 3 8}

freq[10] =

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

O/P = {1 2 3 3 3 5 6 8 8}

```
int [] freq = new int [10];

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

TC: O(n)

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

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

TC: O(n)

Issues of countsort

01

Range of $A[i] > 10^9$

To Create a freq arr = 10^9

= 4 GB

Range of $A[i] < 10^6$

We can create a freq array

02

$A[] = \{3 \quad 100\}$

Note → Count sort should only be used for small Range

* Count sort on Negative numbers

A[] = {-3 2 2 1 -4 5 -3}

OP[] = {-4 -3 -3 1 2 2 5}

Range ⇒ max - min + 1

⇒ 5 - (-4) + 1 ⇒ 10

freq[10] =

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

Mapping array element with freq[] indexes

↓

A[i] + |min|

→ Calculate max & min of A[]

```
int [] freq = new int [max - min + 1];

for ( i = 0 ; i < n ; i++ ) {
    int val = A[i];
    freq [val - min ] ++;
}

for ( i = min ; i ≤ max ; i++ ) {
    int cnt = freq [i - min ]

    for ( j = 1 ; j ≤ cnt ; j++ ) {
        print (i)
    }
}
```

Scenerio

Google's Gmail offers an "***All Inboxes***" feature that allows users to view emails from **multiple email accounts** in one seamless interface. This is particularly useful for users managing personal and professional communications through separate accounts.

The feature ensures that emails from all accounts are merged into a single feed sorted by date and time, facilitating better email management and access.

Problem

Develop a function to emulate the "**All Inboxes**" feature of **Gmail**.

- You are given **two sorted arrays** that represent **timestamps** of emails from two different email accounts. Each element in the array is an email object.
- Your task is to merge these two arrays into a single list, ensuring that the resulting list is sorted by the **timestamp**, allowing the user to view emails in a **chronological order** from both accounts combined.

Acc A = { 1 5 6 9 }

Acc B = { 2 4 8 }

O/P = { 1 2 4 5 6 8 9 }

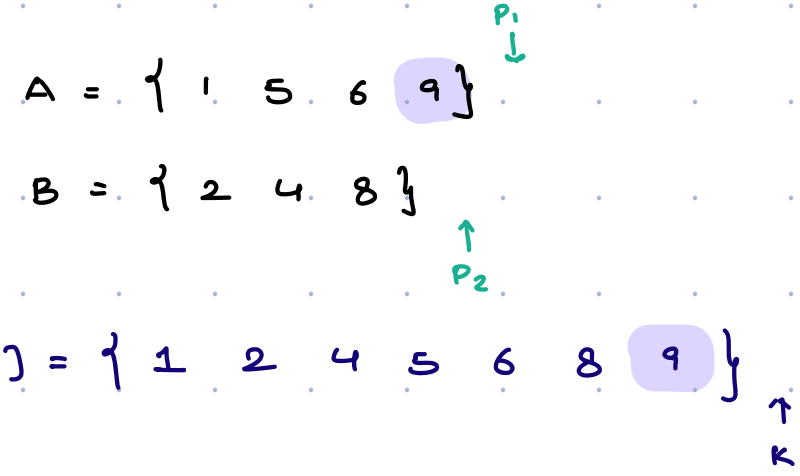
Idea 1 ⇒ Form a C array of size N+M & fill it with all the elements.

C[] = { 1 5 6 9 2 4 8 }

Arrays.sort(c)

TC: O(n+m log(n+m))

Idea 2 = Use two pointers Approach




```
int [] merge (int [] A, int [] B) {
    int [] C = new int [N+M];
    P1 = 0 , P2 = 0 , K = 0
    while (P1 < n && P2 < m)
        if (A[P1] < B[P2]) {
            C[K] = A[P1];
```

```
        P1 ++;
        K ++;
    }
    else {
        C[K] = B[P2];
        P2 ++;
        K ++;
    }
}
```

TC : O(n+m)

SC : O(1)

```
while (P1 < n) {
    C[K] = A[P1];
    P1 ++;
    K ++;
}

while (P2 < m) {
    C[K] = B[P2];
    P2 ++;
    K ++;
}

return C;
}
```

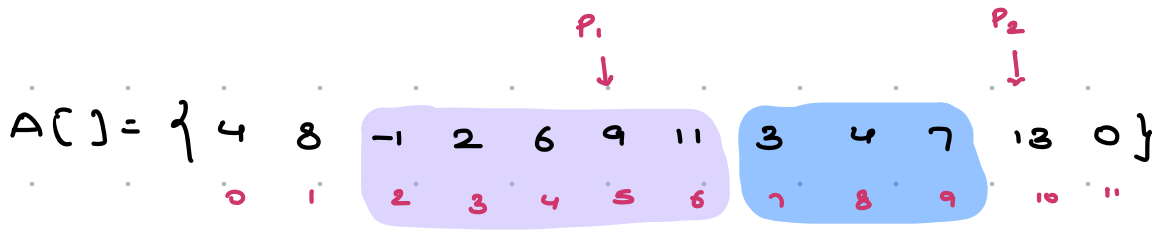
10:26 pm → 10:36 pm

Q4 Given A[N] elements, 3 indices → s, m & e

subarr [s m] is sorted

subarr [m+1 e] is sorted

Sort the entire A[] from s to e



s = 2

m = 6

e = 9

c[] = { -1 2 3 4 6 7 9 11 }

```
int [] merge ( int [] A, s, m, e ){

    int [] C = new int [e-s+1];

    P1 = s , P2 = m+1  k = 0

    while ( P1 ≤ m && P2 ≤ e )
    |
    |   if ( A[P1] < A[P2] ){
    |   |
    |   |   c[k] = A[P1];
    |   |
    |   |   P1 ++;
    |   |   k ++;
    |   }
    |   else {
    |   |
    |   |   c[k] = A[P2];
    |   |
    |   |   P2 ++;
    |   |   k ++;
    |   }
    }

    while ( P1 ≤ m ){
    |
    |   c[k] = A[P1];
    |   P1 ++;
    }   k ++;
```

TC : O(n+m)

SC : O(n+m)

```
while (p2 <= e){
    c[k] = A[p2];
    p2++;
    k++;
}

for (i = 0; i < c.length; i++){
    A[p+s] = c[i];
}
}
```

* Merge Sort Idea

Given A[N] elements, sort it without using inbuild function. $N \rightarrow 100$

Case - I

TC

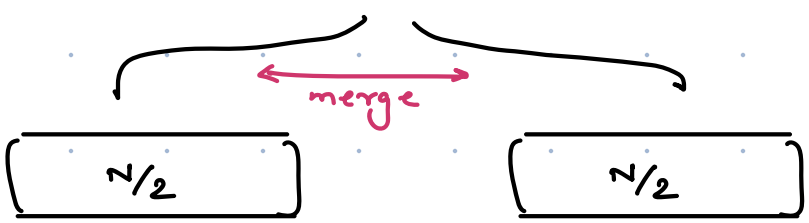
$O(N^2)$

No. of iterations

$(100)^2$

$\Rightarrow 10^4$ iterations

Case - II



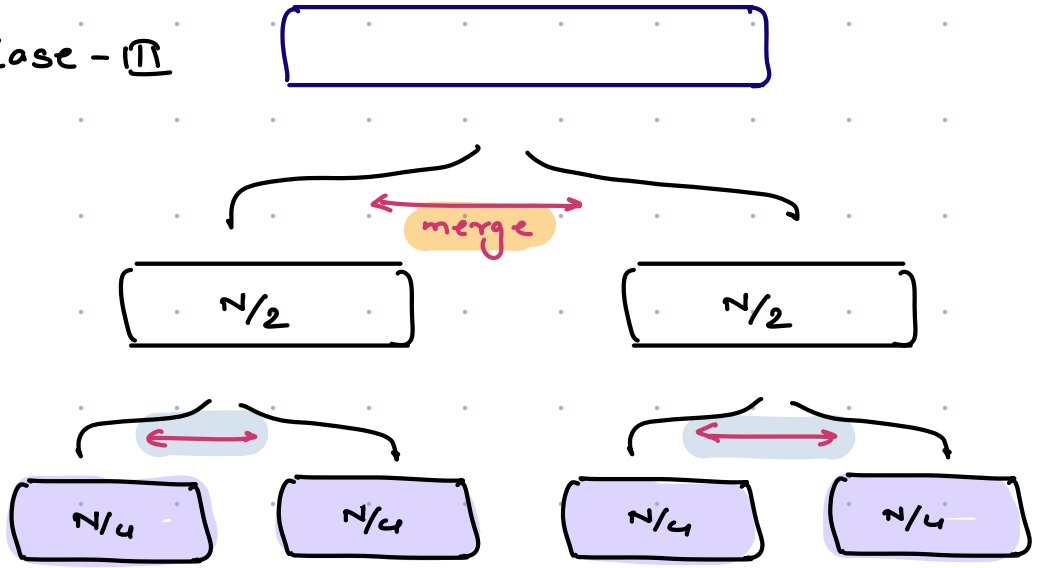
$TC = \left(\frac{N}{2}\right)^2 + \left(\frac{N}{2}\right)^2 + N$

$= \left(\frac{N}{2}\right)^2 * 2 + N$

$= \left(\frac{100}{2}\right)^2 * 2 + 100$

$\Rightarrow$ 5100 iterations

Case - III



$$TC = \left(\frac{N}{4}\right)^2 * 4 + \left(\frac{N}{2}\right) * 2 + N$$

$$= \left(\frac{N}{4}\right)^2 * 4 + 2N$$

$$= \left(\frac{100}{4}\right)^2 * 4 + 2 * 100$$

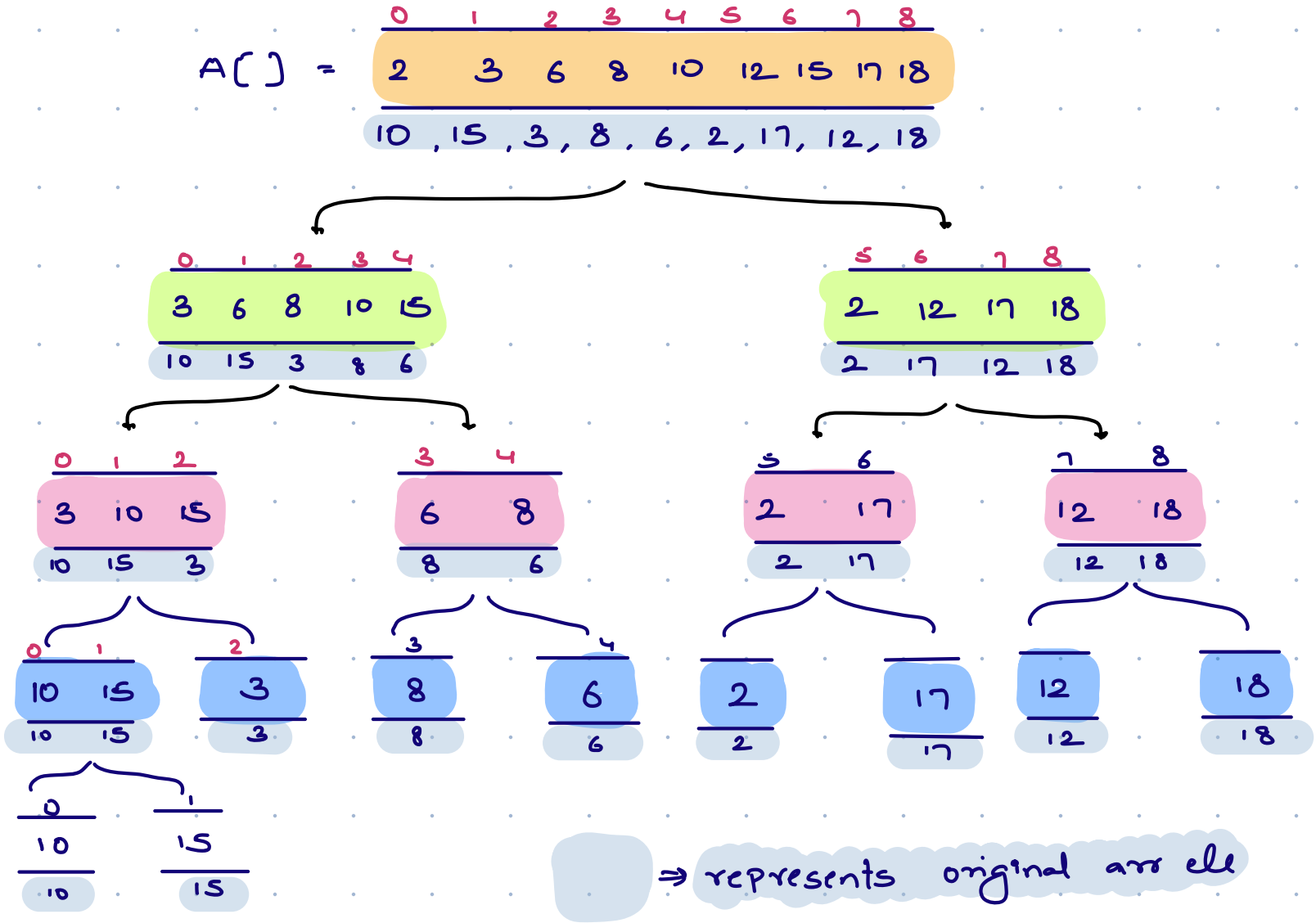
→ 2700 iterations

Idea for Merge Sort

→ Split the A[] again & again until it reaches to a point where a single ele is left.

→ Start merging them back

A[] = 10, 15, 3, 8, 6, 2, 17, 12, 18




```
void mergeSort ( int [] A, int s, int e)
    if (s == e) return;
    int m = (s+e)/2;
    mergesort (A, s, m);
    mergesort (A, m+1, e);
    merge ( A, s, m, e);
3
```

TC : O(nlogn)
SC : O(n)

Recurrence relation

$f(n) = f(n/2) * 2 + n$ $= 2^1 f\left(\frac{n}{2^1}\right) + n$

$f(n/2) = f(n/4) * 2 + n/2$

$f(n) = \left(f(n/4) * 2 + n/2\right) * 2 + n$

$= \left(2f(n/4) + n/2\right) * 2 + n$

$f(n) = 4f(n/4) + 2n$ $= 2^2 f\left(\frac{n}{2^2}\right) + 2n$

$f(n/4) = f(n/8) * 2 + n/4$

$f(n) = 4 * \left(f(n/8) * 2 + n/4\right) + 2n$

$= 4\left(2f(n/8) + n/4\right) + 2n$

$= 8f(n/8) + n + 2n$

$f(n) = 8f(n/8) + 3n$ $= 2^3 f\left(\frac{n}{2^3}\right) + 3n$

After k^{th} substitution

$f(n) = 2^k f\left(\frac{n}{2^k}\right) + k * n$ $\qquad f(1) = 1$

$$\frac{n}{2^k} = 1 \qquad n = 2^k \qquad k = \log n$$

$$f(n) = n * 1 + \log n * n$$

$$f(n) = n + n\log n$$

$$TC = O(n \log n)$$

* Inplace Sorting Algo

→ Sorting Algo where we are not taking any extra space.

* Stable sort Algorithm

→ If 2 data points have same parameter value, then the one appearing first in input should appear first in the output as well.

People	Marks
Yogi	20
Akshay	100
Saswath	70
Shantanu	70

Sort it using marks →

STABLE SORTING

O/P 1

Yogi	20
Saswath	70
Shantanu	70
Akshay	100

O/P 2

Yogi	20
Shantanu	70
Saswath	70
Akshay	100

HW

Next class

Which all sorting Algos are stable sort? ⇒ Examples