Count Sort

Stable & In place Sorting

Merge 2 sorted arrays

Merge Sort

Inversion Count

Q. Find smallest no. that can be formed by rearranging digits given in an array.

A = < 6, 3, 4, 2, 7, 2, 1 >

A = < 1, 2, 2, 3, 4, 6, 7 >

< 1, 2, 3 >

↓

234

A = < 4, 2, 7, 3, 9, 0 >

A = < 0, 2, 3, 4, 7, 9 >

|A| = N

Obs → sort the digits in array

↓

smallest no.

Approach 1 : Arrays.sort( ) → TC: O(N log N)

↓

TC : O(N)

Approach 2 : Digit → 0 to 9

A = < 6, 3, 4, 2, 7, 2, 1 >

0 0 0 0 1 1 1 1 2 2 2 3 ... 4 ... 5 ....

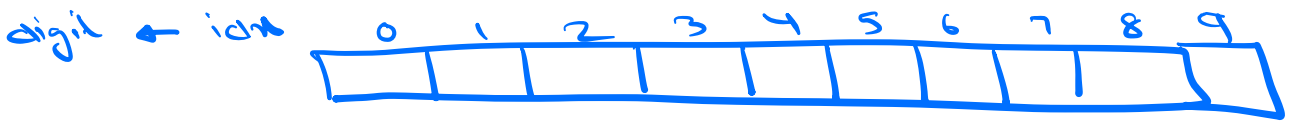
Maintain freq of digits

① hm < int, int > hm

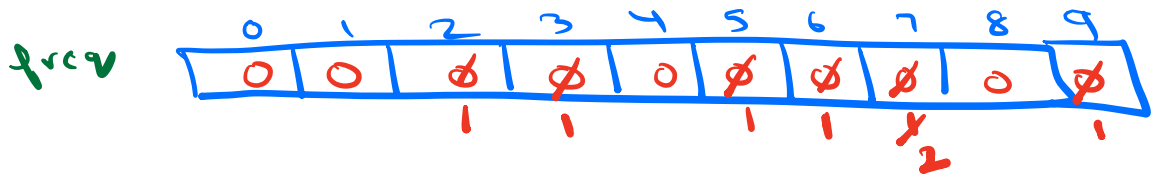
↓ digit ↓ freq

② Fixed sized arr digits 0→9

int freq[10]



A[ ] = < 7, 2, 3, 9, 5, 6, 7 >



A[ ] = < 2 3 5 6 7 7 9 >
0 1 2 3 4 5 6

Count sort → method of sort based on frequency counting

```
// code
void smallestno (int a[], int N){
  int freq[10] = {0};
  for (i=0 ; i<N ; i++) {        → N
    freq[a[i]] ++
  }
  // iterate the freq arr
  int idx=0                      → N
```

```
for (dig = 0 ; dig <= 9 ; dig++){
    for (cnt = 1 ; cnt <= freq[dig] ; cnt++) {
        A[idx] = dig
        idx++
    }
}
```

TC : O(N)

SC : O(1)

10

dig	freq[dig] → cnt
0	f_0
1	f_1
2	f_2
3	f_3
⋮	⋮
9	f_9
	N

$f_0 + f_1 + f_2 + f_3 + \ldots + f_9 = N$

Count Sort

A = [4, 6, 2, 11, 13, 20, 14, 15]

freq [21] idx → 0 to 20

0	1	2	3	4	5	6	. . .	20
0	0	1	0	1		1		1

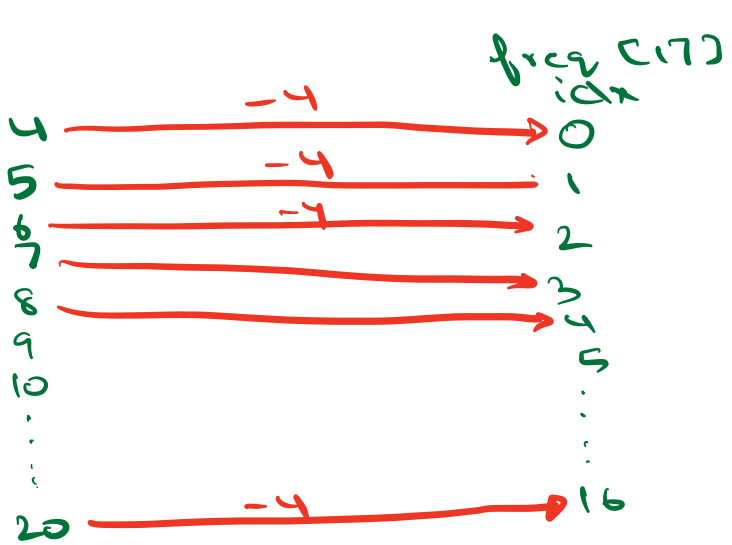
freq [maxele + 1] (if 0 or +ve ele in A[])

A = [100, 300, 600, 100001]

freq [100001]

min max

A = [4, 6, 8, 20, 15]



a b
4 → 20

b - a + 1

20 - 4 + 1

= 17

freq [17]

ele - min → idx

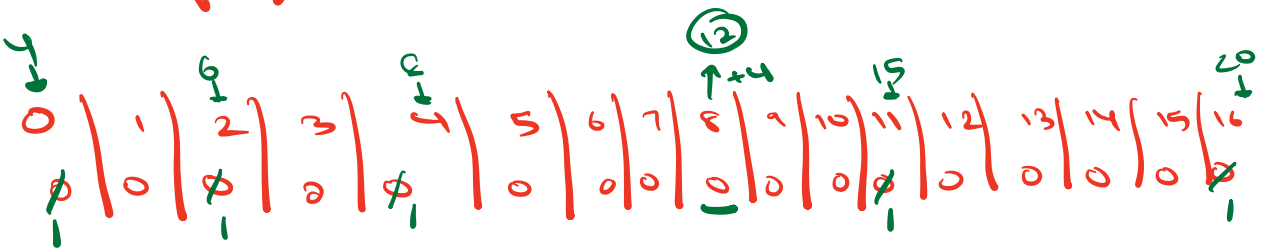
① int freq [max - min + 1]

② ele info → ele - min = idx

Min Max

A = [4, 6, 8, 20, 15]

freq [20 - 4 + 1] → freq [17]



dc - min → idx

dc ← idx + min

If range of A[i] is 0 - 10^5

freq [10^5] ✓

int freq [10^6] ✓ $10^6 \times 4B = 4MB$

If range of data is 0 - 10^9

int freq [10^9] X MLE occurs

4B

10^9 integers

$10^9 \times 4B = 4GB$ X

Count sort on -ve nos.

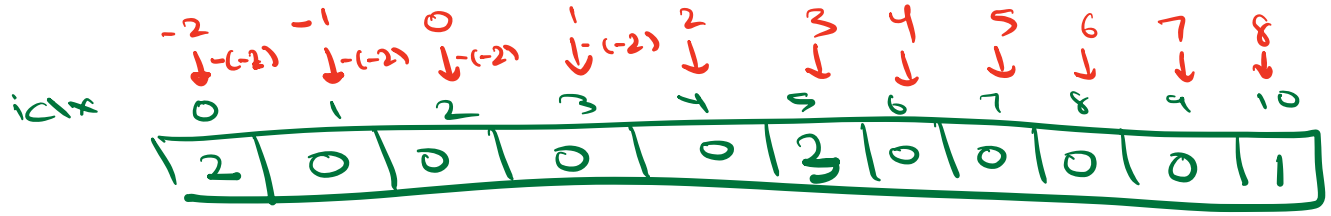
$A = [-2, 3, 8, 3, -2, 3]$ (Min = -2, Max = 8)

$-2 \rightarrow 8$

max - min + 1

$8 - (-2) + 1$

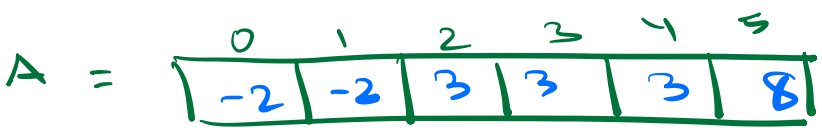
① int freq [11]



ele - min → idx

ele ← idx + min

②



```
void smallestno(int a[], int N){
    // get max & min  → N
    int freq[max-min+1] = {0};
    for (i=0; i<N; i++) {  → N
        freq[a[i]-min]++
    }

    // iterate the freq arr

    int idx=0

    for (i = 0; i < max-min+1; i++){   > N
        for(cnt = 1; cnt <= freq[i]; cnt++ ){
            A[idx] = i + min
            idx++
        }
    }
}
```

ele −min → idx

idx ← +min

TC : O(N)

SC : O(R)

↓

freq[] size → range of original a[]

Stable sort

If 2 data points have same parameter value, we maintain original order.

Name	Marks
Amrit	40
Chitra	25
Tushar	20
Veena	25
Kalu	16
Harshal	12
Saswata	30

Stable sort

Harshal	12
Kalu	16
Tushar	20
Chitra	25
Veena	25
Saswata	30
Amrit	40

Harshal	12
Kalu	16
Tushar	20
Veena	25
Chitra	25
Saswata	30
Amrit	40

Unstable sort

Ex 2 7, 2, 3, 4, 2, 8, 1

Sorting based on value

↓

Stable sort 1, 2, 2, 3, 4, 7, 8

Unstable sort 1, 2, 2, 3, 4, 7, 8

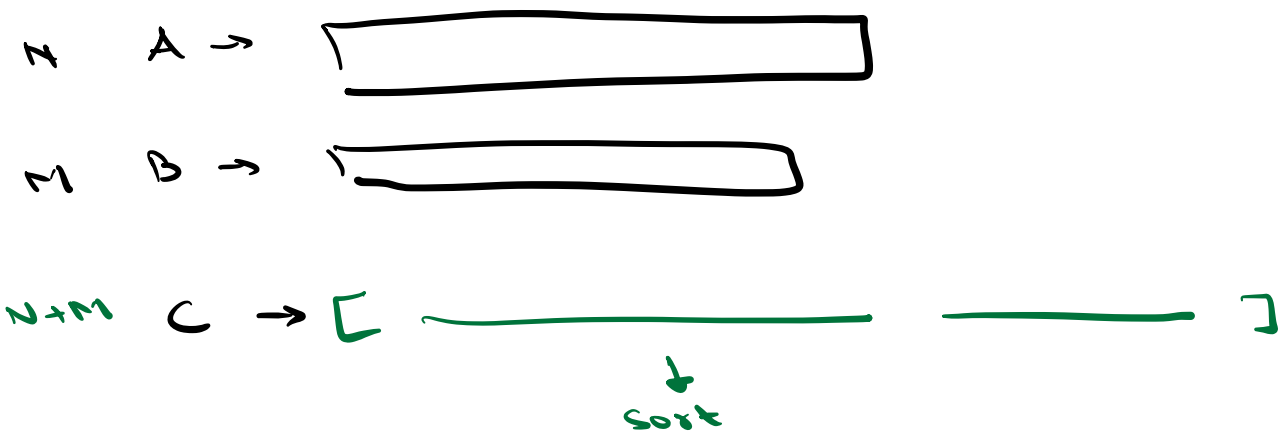
Inplace (No extra space / SC: O(1))

Q. Given 2 sorted arrays of size N & M, merge and return new sorted array.

(7) A = [2, 5, 7, 12, 20, 24, 29]

(6) B = [6, 9, 10, 14, 18, 19]

(13) C = [2, 5, 6, 7, 9, 10, 12, 14, 18, 19, 20, 24, 29]



Approach 1 : copy all el of A and B into C and then sort C.

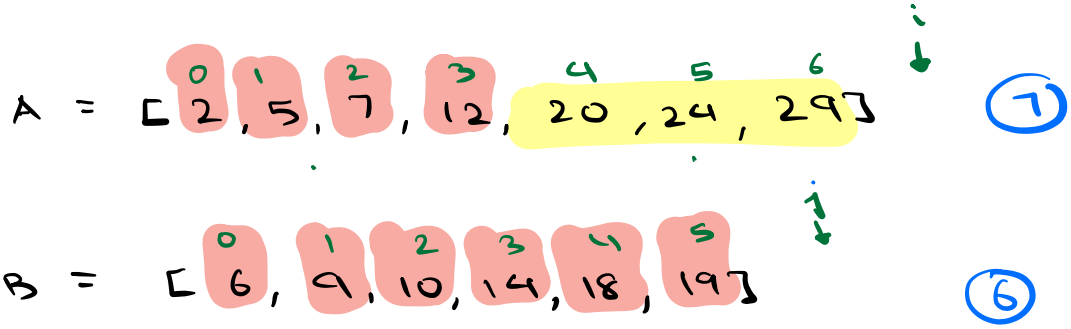
① Copy → N+M

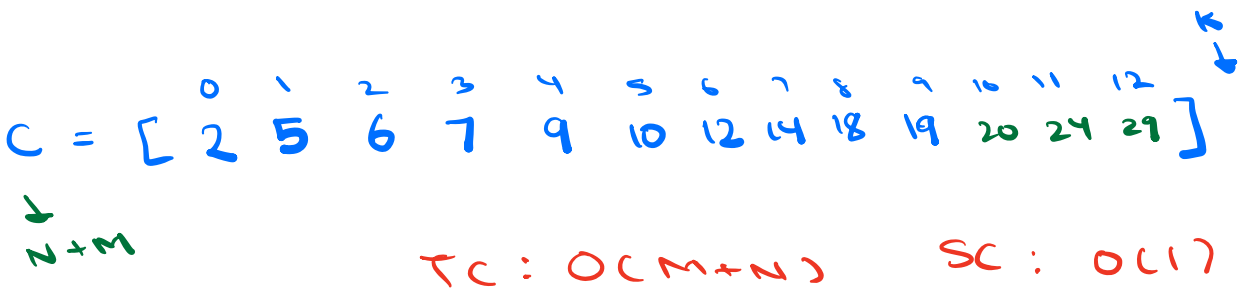
② Sort C → N+M log (N+M)

TC : O (N+M log (N+M))

SC : O (sorting algo)

Approach 2:





TC: O(M+N) SC: O(1)

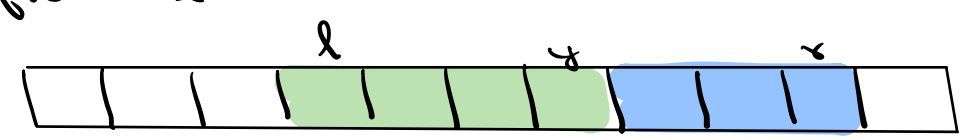
```
int [] merge (int A[], int B[], int N, int M){
    int C[N+M]
    int i=0, j=0, k=0

    while(i<N && j<M){
        if(A[i] <= B[j]){
            C[k] = A[i]
            k++   i++
        }
        else{   // A[i] > B[j]
            C[k] = B[j]
            k++   j++
        }
    }
```

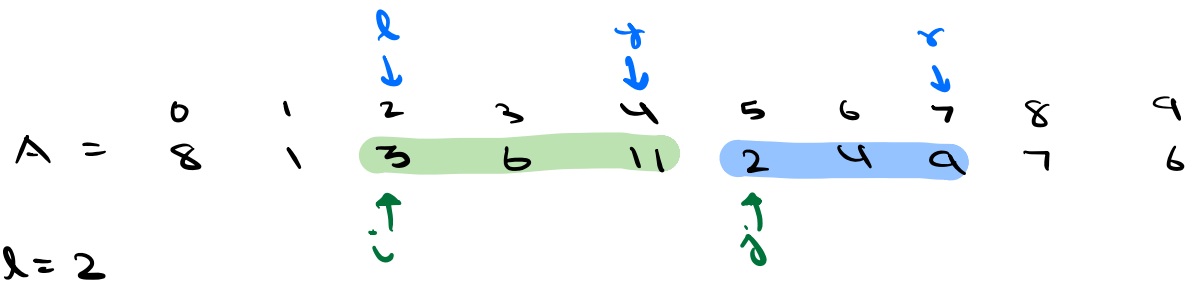
// loop will end if i reaches end of A or j reaches end of B

```
while (i < N) {          // check if A is remaining
    C[k] = A[i]
    k++   i++
}

while (j < M) {          // check if B is remaining
    C[k] = B[j]
    k++   j++
}
}
```

10:46

Q. Given an array of size N and 3 indices l, y, r. In the array, subarray from $l - y$ and $y+1 - r$ are sorted. Sort subarray from $l - r$

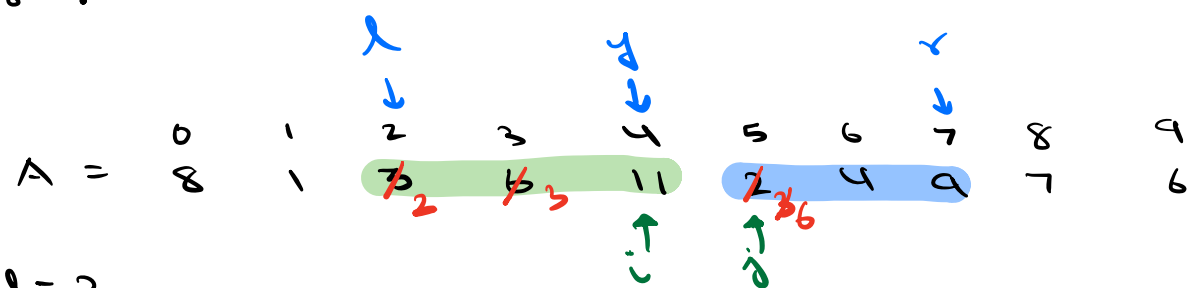




$l = 2$

$y = 4$

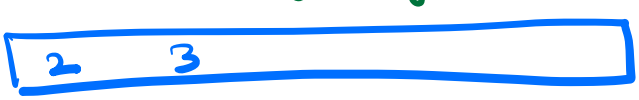
$r = 7$



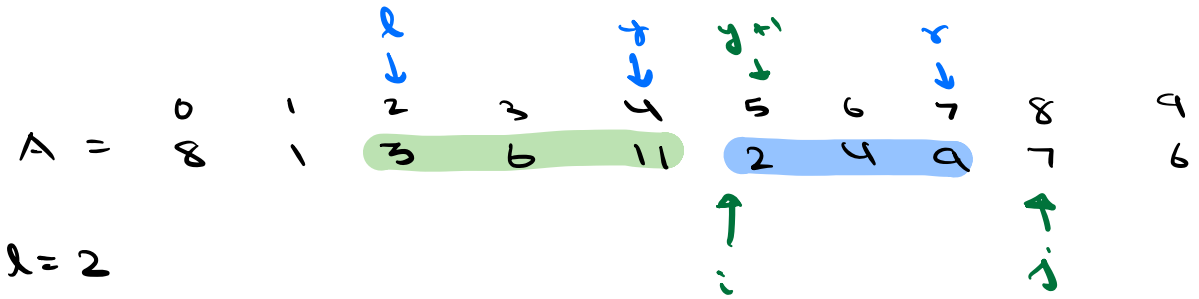
$l = 2$

$y = 4$

$r = 7$



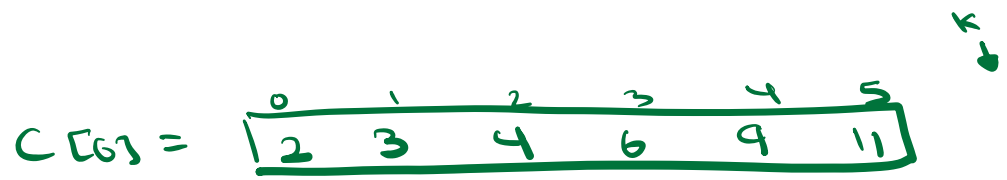
Swapping data → breaking sorted property of subarrays



l = 2

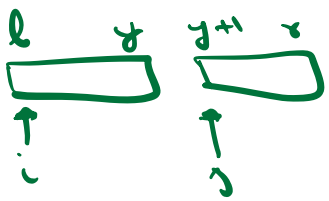
y = 4

r = 7



Total elements = r - l + 1 $\quad$ 7 - 2 + 1 = 6

```
int [] merge (int A[], int l, int y, int r ){

    int C[r-l+1]
    int i = l, j = y+1, k = 0

    while ( i <= y && j <= r ) {
        if ( A[i] <= A[j] ) {
            C[k] = A[i]
            k++   i++
        }
        else {          // A[i] > B[j]
            C[k] = A[j]
            k++   j++
        }
    }
```



// loop will end if i reaches end of A or j reaches end of B

```
while (i<=y) {   // check if A is remaining
    C[k] = A[i]
    k++   i++
}

while (j<=r) {   // check if B is remaining
    C[k] = A[j]
    k++   j++
}
```

10:46

// copy C arr

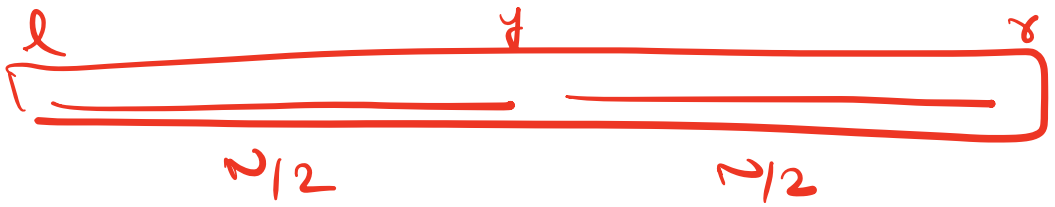
```
int i = l , k = 0

while (i <= r){
    A[i] = C[k]
    i++   k++
}
}
```

A[l] ← C[0]

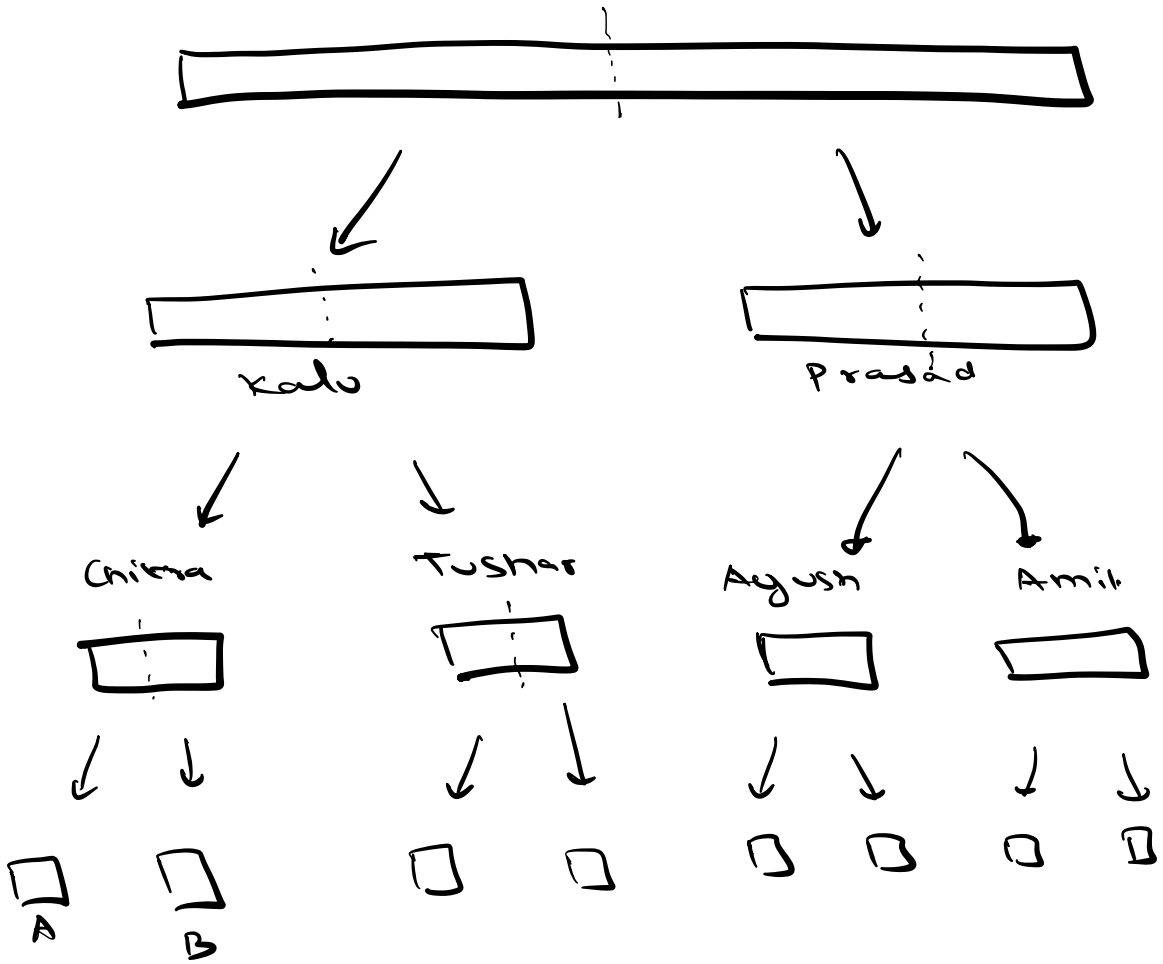
O(N+M)

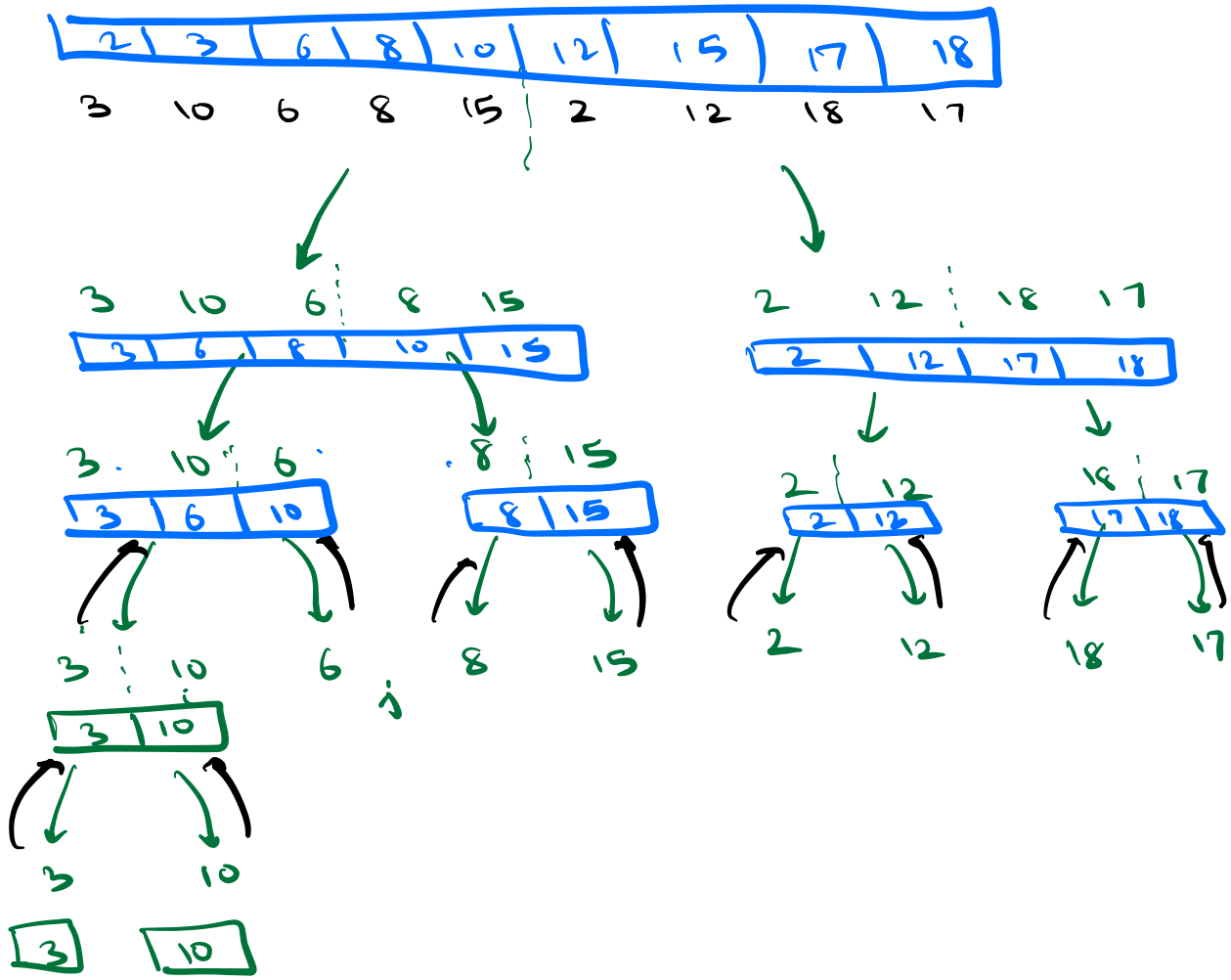
TC : O(N)

SC : O(N)



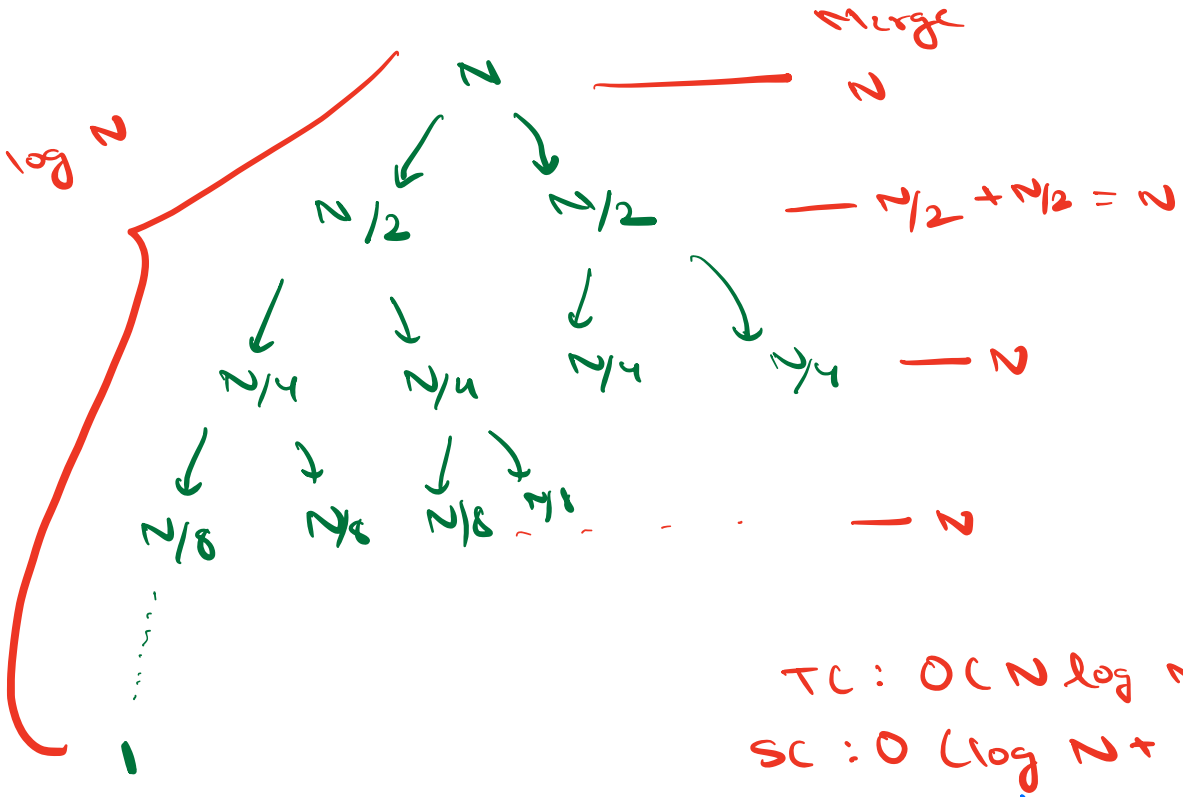
Merge sort






```
// Given l and r, sort A[] from l → r
void mergesort(int A[], l, r) {
    if(l == r) return
    int mid = (l+r)/2
    mergesort(A, l, mid)
    mergesort(A, mid+1, r)
    merge(A, l, mid, r) → C[]
}
```

Dry run of code



TC : O(N log N)

SC : O (log N + N)

Call stack + fn

Q. Given two array A[n] and B[m]. Calculate no. of pairs such that A[i] > B[j]

A[3] = <7 3 5>

B[3] = <2 0 6>

TC: O(N+M)

7,2 7,0 7,6

cnt = 7 pairs

3,2 3,0 5,2 5,0

Doubts

1 2 3 4 5 6 7 8 9 10

Nos. with odd factors →

24 1 to $\sqrt{24}$

i N/i

1	24
2	4
3	8
4	6

$i = N/i$

$i^2 = N$

1 to $\sqrt{100}$

100

i N/i

.

.

.

10 = 100/10=0

1 → A perfect square O(A)

1 → 10 → 1, 4, 9 (3)

1 → 20 1, 4, 9, 16 (4)

Factorial Array

B[x] → A[x]!

A = [2, 3, 2, 3] (0 1 2 3)

B = [2 6 2 6 4] (0 1 2 3)

<2, 2, 6> X

2 2 2,3

<2, 2> ✓

2 2

<2, 2, 7> ✓

2 2 2

Subset → subsequence

Subset → every elem should have same prime factors

$N = 10^5$

2^N

2^{10^5}