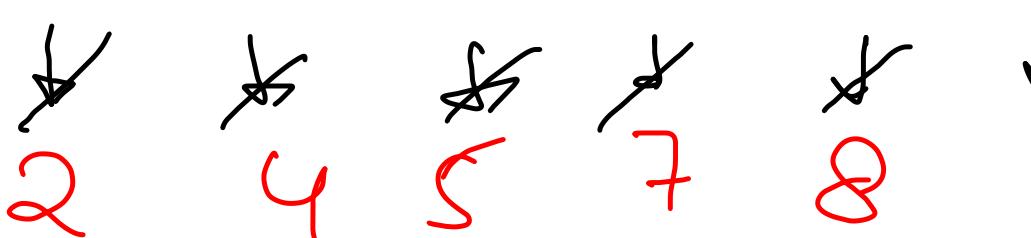
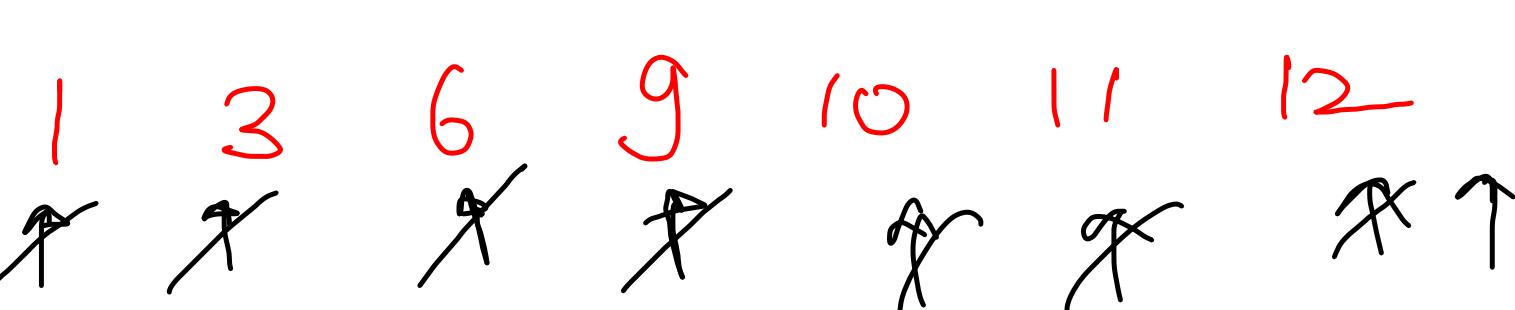


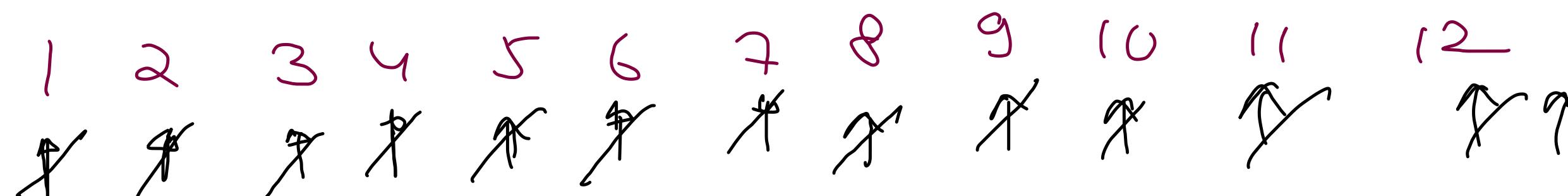
Sorting

- Merge 2 Sorted Arrays
- Merge Sort
- Partition Array (Sort O<sub>1</sub>, Sort O(2))
- Quick Sort

## Merge 2 Sorted Arrays

A1:   $O(N)$  Space

A2:   $O(M)$  space

res 

$O(N+M)$  time  $\rightarrow O(N+M)$  extra space

```

public static int[] mergeTwoSortedArrays(int[] a, int[] b){
    int i = 0, j = 0, k = 0;
    int[] ans = new int[a.length + b.length];
    while(i < a.length && j < b.length){
        if(a[i] <= b[j]){
            ans[k] = a[i];
            i++;
            k++;
        }else{
            ans[k] = b[j];
            j++;
            k++;
        }
    }
    while(i < a.length){
        ans[k] = a[i];
        k++;
        i++;
    }
    while(j < b.length){
        ans[k] = b[j];
        k++;
        j++;
    }
    return ans;
}

```

Trade off

	Time	Space
2 pointer	$O(N+M)$	$O(N+M)$
Insert Sort	$O(N \cdot M + M^2)$	$O(1)$
Shell Sort	$O((N+M) \log_2(N+M))$	$O(1)$

## With Out extra space

$$a_1[] = \boxed{1 \ 4 \ 7 \ 8 \ 10} \quad O(N)$$

$$a_2[] = \boxed{2 \ 3 \ 9} \quad O(M)$$

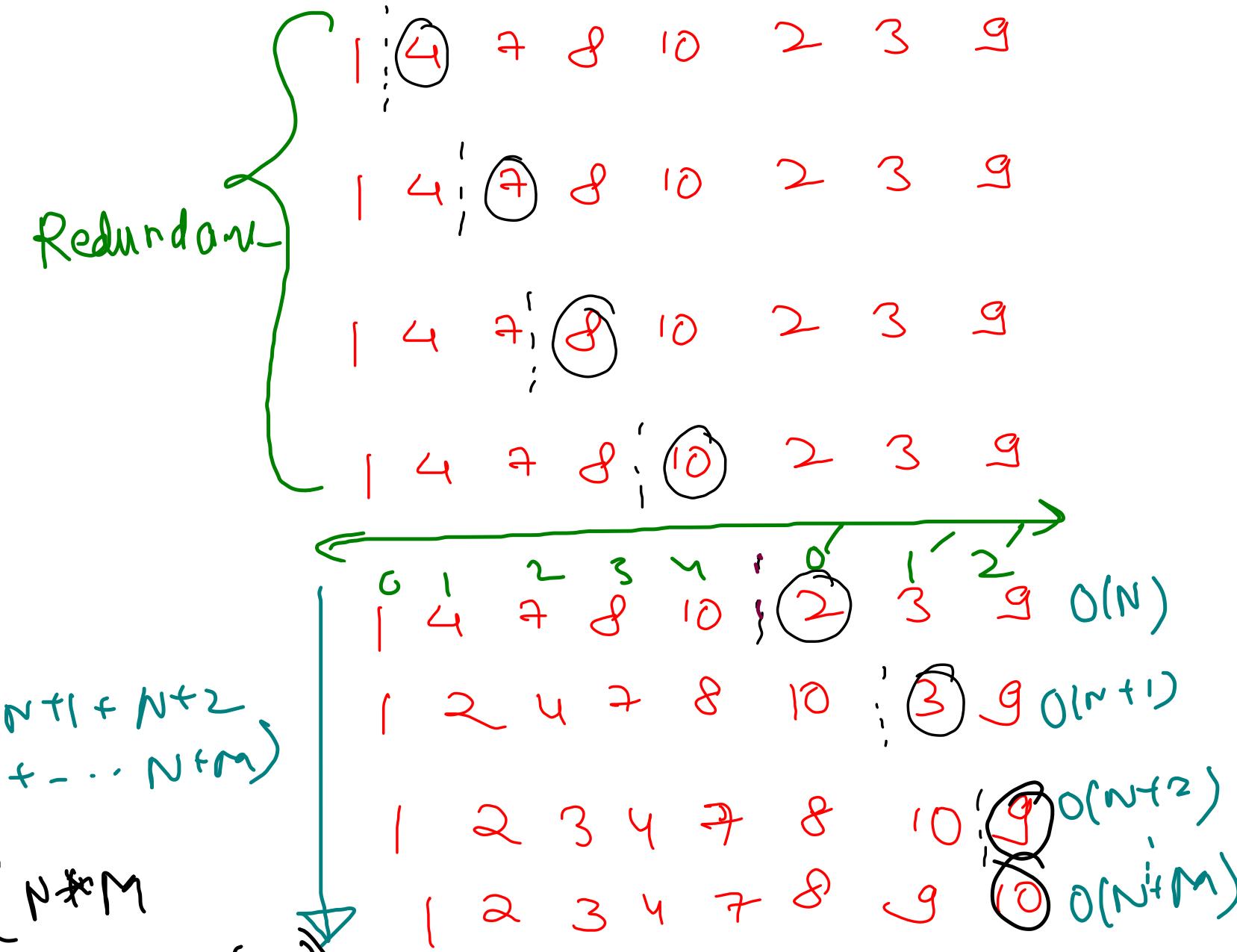
→ Using insertion Sort

$$O(n + n+1 + n+2 + \dots + n+m)$$

$$= O(N * M)$$

$$+ \frac{M * (M+1)}{2}$$

$$= O(N * M + M^2) \text{ time, } O(1) \text{ extra space}$$



```
public void swap(int[] nums1, int a, int b){  
    int temp = nums1[a];  
    nums1[a] = nums1[b];  
    nums1[b] = temp;  
}  
  
public void merge(int[] nums1, int m, int[] nums2, int n) {  
    for(int i = m; i < m+n; i++){  
        nums1[i] = nums2[i - m];  
    }  
  
    for(int i=m; i<m+n; i++){  
        for(int j=i-1; j>=0; j--){  
            if(nums1[j] > nums1[j + 1]){  
                swap(nums1, j, j + 1);  
            }  
            else break;  
        }  
    }  
}
```

## # Shell Sort

$a1[] = [1, 4, 7, 8, 10]$   
 $a2[] = [2, 3, 9]$

$$\text{gap sequence} = \left\lceil \frac{N+M}{2} \right\rceil, \left\lceil \frac{N+M}{4} \right\rceil, \left\lceil \frac{N+M}{8} \right\rceil, \dots, 1$$

$$= \frac{8}{2}, \frac{8}{4}, \frac{8}{8}$$

$$= 4, 2, 1$$

1 4 7 8 10 2 3 9

1 4 7 8 10 2 3 9

1 2 7 8 10 4 3 9

1 2 3 8 10 4 7 9

gap = 4  
 $n+m-4$  comparisons

1 2 3 8 10 4 7 9

1 2 3 8 10 4 7 9

1 2 3 8 10 4 7 9

1 2 3 4 8 10 7 9

gap = 2  
 $n+m-2$  comparisons

1 2 3 4 7 8 10 9

1 2 3 4 7 8 9 10

gap = 1

$n+m-1$   
 Comparisons

```

public void merge(int[] nums1, int m, int[] nums2, int n) {
    for(int i = m; i < m+n; i++){
        nums1[i] = nums2[i - m];
    }

    for(double gap = (m + n) / 2.0; gap > 0; gap = gap/2.0){
        int jStart = (int)Math.ceil(gap);
        for(int i=0, j=jStart; j<m+n; i++, j++){
            if(nums1[i] > nums1[j]){
                swap(nums1, i, j);
            }
        }
        if(jStart == 1) break;
    }
}

```

gaps

$$\frac{N}{\sum}$$

comparisons

$$N - \frac{N}{\sum}$$

$$\frac{N}{4}$$

$$N - \frac{N}{4}$$

$$\frac{N}{8}$$

$$N - \frac{N}{8}$$

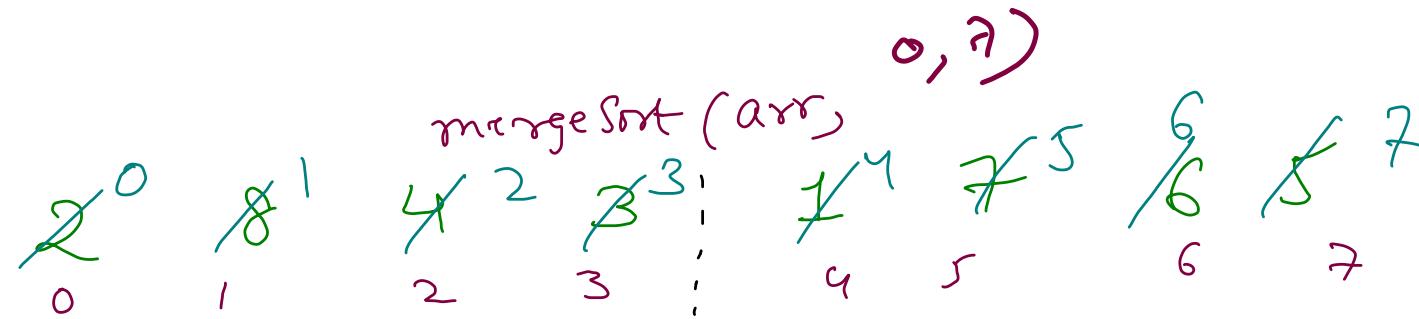
$$TC = O\left( N \log_2 N - N \left\{ \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots + \frac{1}{2^{k-1}} \right\} \right)$$

$$N - 1$$

$$= O\left( N \log_2 N \right) = O\left( \underbrace{(N+M)}_{\text{underbrace}} \log_2 (N+M) \right)$$

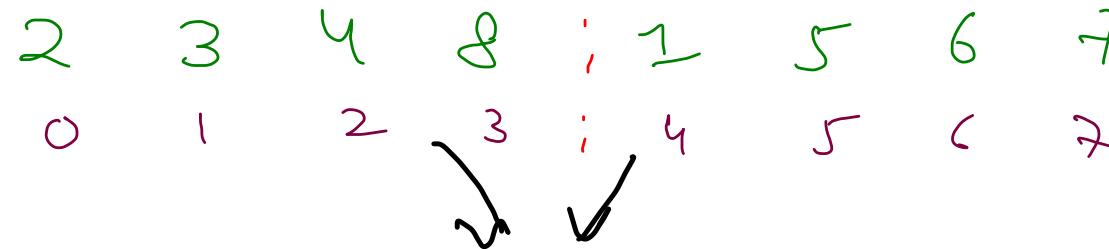
## Merge Sort

(with extra space) { in place }



mergeSort(arr, 0, 3)

mergeSort(arr, 4, 7)



```
public void mergeSort(int[] nums, int left, int right){
    if(left >= right){
        return; // 0 or 1 elements in array
    }

    int mid = left + (right - left) / 2;
    mergeSort(nums, left, mid); // left Sorted
    mergeSort(nums, mid + 1, right); // right Sorted
    merge(nums, left, mid, mid + 1, right);
}
```

{ Stable }

```
public void merge(int[] nums, int all, int a1r, int a2l, int a2r){
    int totalRes = (a1r - all + 1) + (a2r - a2l + 1);
    int[] res = new int[totalRes];

    int ptr1 = all, ptr2 = a2l, ptr3 = 0;
    while(ptr1 <= a1r && ptr2 <= a2r){
        if(nums[ptr1] <= nums[ptr2]){
            res[ptr3] = nums[ptr1];
            ptr3++; ptr1++;
        } else {
            res[ptr3] = nums[ptr2];
            ptr3++; ptr2++;
        }
    }

    while(ptr1 <= a1r){
        res[ptr3] = nums[ptr1];
        ptr3++; ptr1++;
    }

    while(ptr2 <= a2r){
        res[ptr3] = nums[ptr2];
        ptr3++; ptr2++;
    }

    for(int i=all; i<=a2r; i++){
        nums[i] = res[i - all];
    }
}
```

Two pointer

$$2^0 T(N) = 2T(N/2) + O\left(\frac{N}{2} + \frac{N}{2}\right)$$

$$2^1 T(N/2) = 2T(N/4) + 2O\left(\frac{N}{4} + \frac{N}{4}\right)$$

$$2^2 T(N/4) = 2T(N/8) + 2^2 O\left(\frac{N}{8} + \frac{N}{8}\right)$$

$$2^3 T(N/8) = 2^4 T(N/16) + 2^3 O\left(\frac{N}{16} + \frac{N}{16}\right)$$

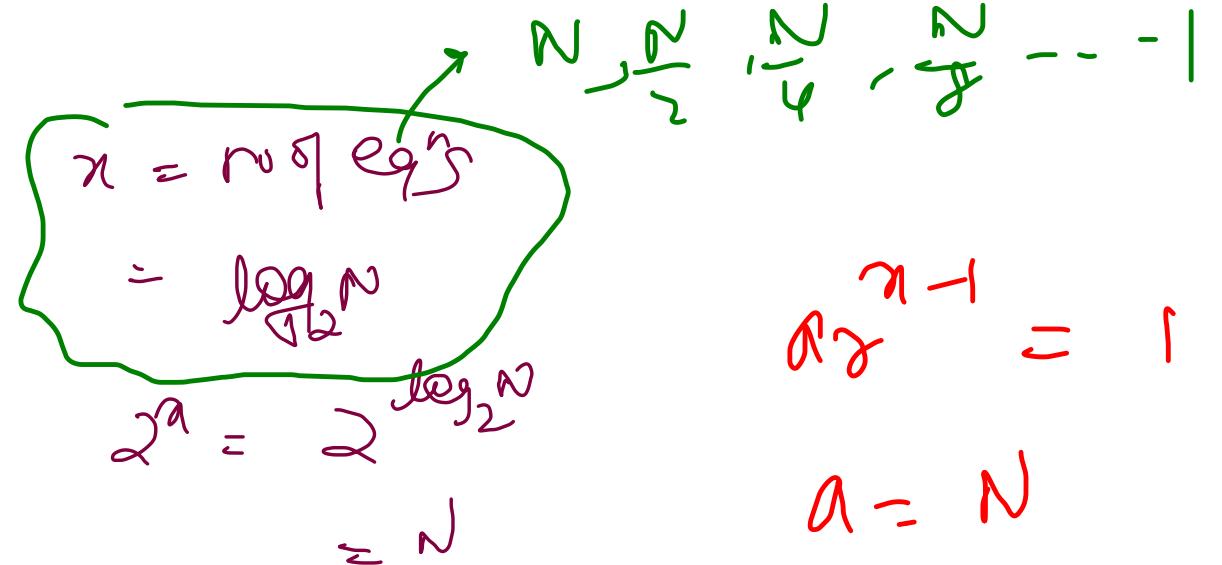
$$\vdots \quad \vdots \quad \vdots$$

$$2^x T(1) = 2T(1) + O(1)$$

$$T(N) = O(N) + 2 * O\left(\frac{N}{2}\right) + 2^2 * O\left(\frac{N}{4}\right) + 2^3 * O\left(\frac{N}{8}\right) + \dots$$

$\approx N + N + N + \dots$   $x$  times

$$= N * x = O(N * \log_2 N)$$



$$2^{n-1} = 1$$

$$a = N$$

$$r = 1/2$$

$$N\left(\frac{1}{2}\right) = 1$$

$$2^x = N$$

$$\log_2(2^x) = \log_2 N$$

$n = \log_2 N$

$S^-$      $3^x 4^p |^a 2$      $4^q 3^y |^b$

Stage

$\{ |^a 3^x 4^p |^b \}$   $\{ 2 \}$   $\{ 3^y 4^q \}$

$\{ [ |^a |^b ] 2 [ 3^x 3^y ] [ 4^p 4^q ] S^- \}$

→ preserving the original order of elements are equal

(without extra space)

~~Shell sort~~

$$2^0(T(N)) = 2T(N/2) + O(N \log N)$$

$$2^1(T(N/2)) = 2T(N/4) + O\left(\frac{N}{2} \log \frac{N}{2}\right)$$

$$2^2(T(N/4)) = 2T(N/8) + O\left(\frac{N}{4} \log \frac{N}{4}\right)$$

:

:

$$2^x(T(1)) = 2T(0) + O(1)$$

$$T(N) = N \log N + 2^0 \cancel{N \log \frac{N}{2}} + 2^1 \cancel{\frac{N}{2} \log \frac{N}{2^1}}$$

$$+ \dots 2^x \cancel{\frac{N}{2^x} \log \frac{N}{2^x}}$$

$$= N \log N + N \log \frac{N}{2} + N \log \frac{N}{2^2} + N \log \frac{N}{2^3} + \dots N \log \frac{N}{2^x}$$

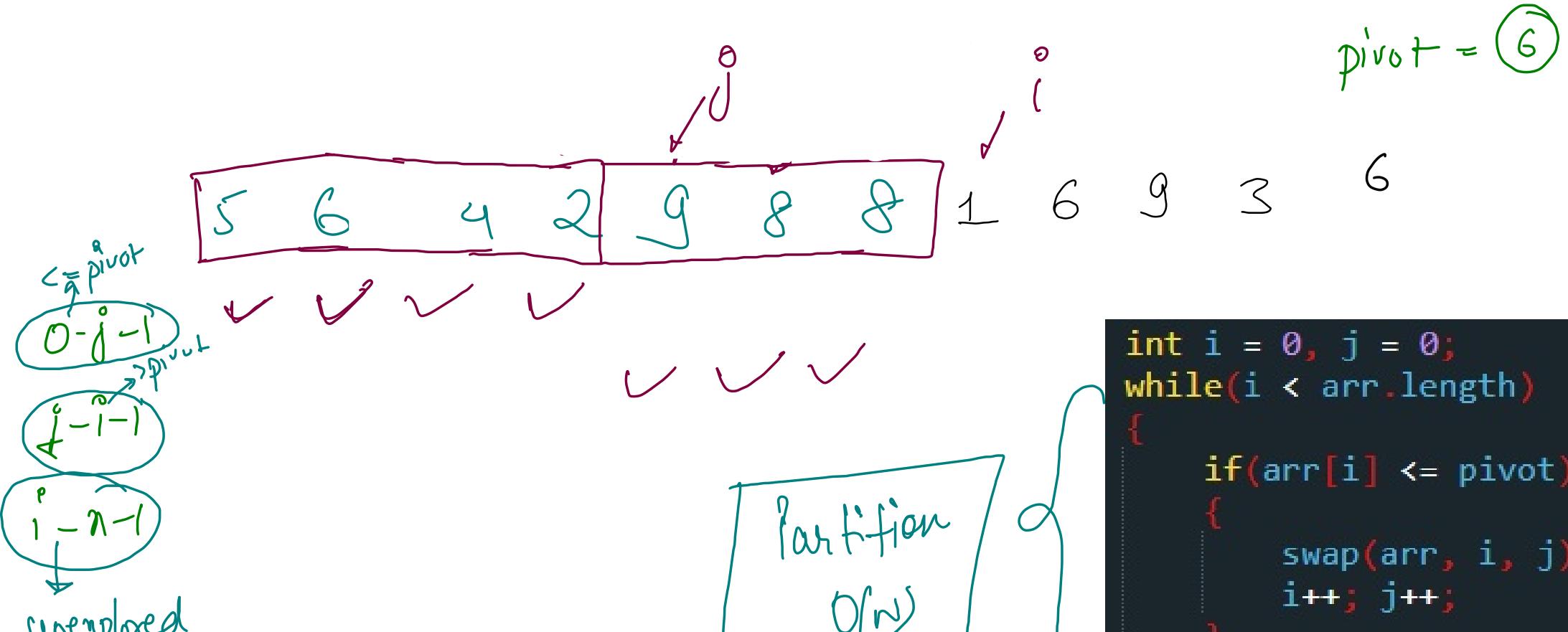
$$= N \left\{ \log N + \log \frac{N}{2} + \log \frac{N}{2^2} + \log \frac{N}{2^3} + \dots \log \frac{N}{2^x} \right\}$$

$$= N \left\{ \log \left\{ \frac{N \cdot N \cdot N \cdots n^{\text{final}}}{2^0 \cdot 2^1 \cdot 2^2 \cdots 2^x} \right\} \right\} = N \left( \log \frac{N^x}{2^{\log_2 N}} \right)$$

$$= N \log_2^2 N$$

## Partition Array

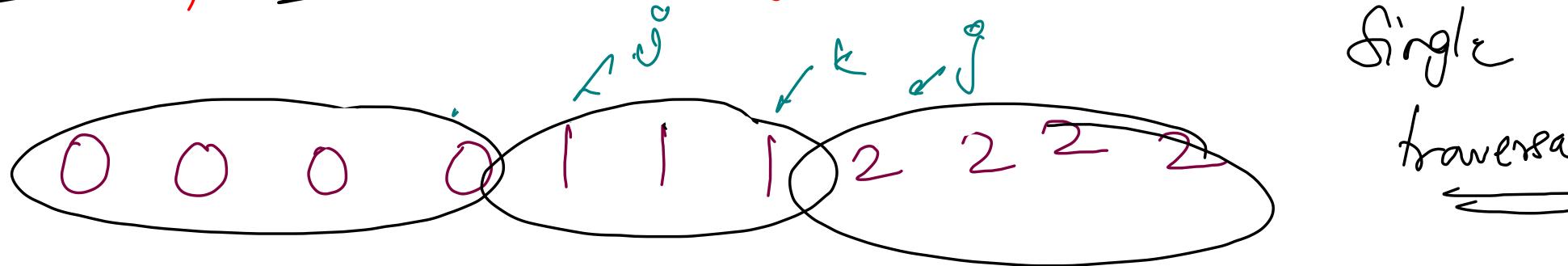
~~quicksort~~  $\Rightarrow$  ~~quicksort~~  
~~↓~~ ~~↓~~  $\Rightarrow$  quickSort



```
int i = 0, j = 0;
while(i < arr.length)
{
    if(arr[i] <= pivot)
    {
        swap(arr, i, j);
        i++; j++;
    }
    else i++;
}
```

- -ve, +ve
- 0, 1 {Sort 01}
- 3<sup>em</sup>, non- $\geq$  zero
- odd, even
- $\leq$  pivot,  $>$  pivot

# DNF Sort / Sort 0/1/ Three way Partitioning



= 0's  
(0-i-1)

= 1's  
(i-j-1)

= 2's  
(k+l)(n-1)

unexplored  
(j-k)

0 , 1 , 2  
%3 = 0 , %3 = 1 , %3 = 2

< pivot , = pivot , > pivot

< a , [a,b) , > b { Dual pivot partitioning }

```

int i = 0, j = 0, k = arr.length - 1;
while(j <= k)
{
    if(arr[j] == 0)
    {
        swap(arr, j, i);
        i++;
        j++;
    }
    else if(arr[j] == 2)
    {
        swap(arr, j, k);
        k--;
    }
    else j++;
}
  
```

# Sorting-Lecture → 2  
(IMPORTANT)

- Inversion Count → using Merge Sort
- Reverse Pairs
- Count of Smaller Elements After Self
- Count subarrays with  $1s > 0s$
- Minimum swaps to Sort - I & II

## Count Inversions

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

$(i, j)$        $i < j$        $a[i] > a[j]$

$(8, 5)$     $(8, 3)$     $(8, 4)$     $(8, 1)$     $(8, 6)$     $(8, 2)$

$(5, 3)$     $(5, 4)$     $(5, 1)$     $(5, 2)$

$(3, 1)$     $(3, 2)$     $(4, 1)$     $(4, 2)$     $(6, 2)$

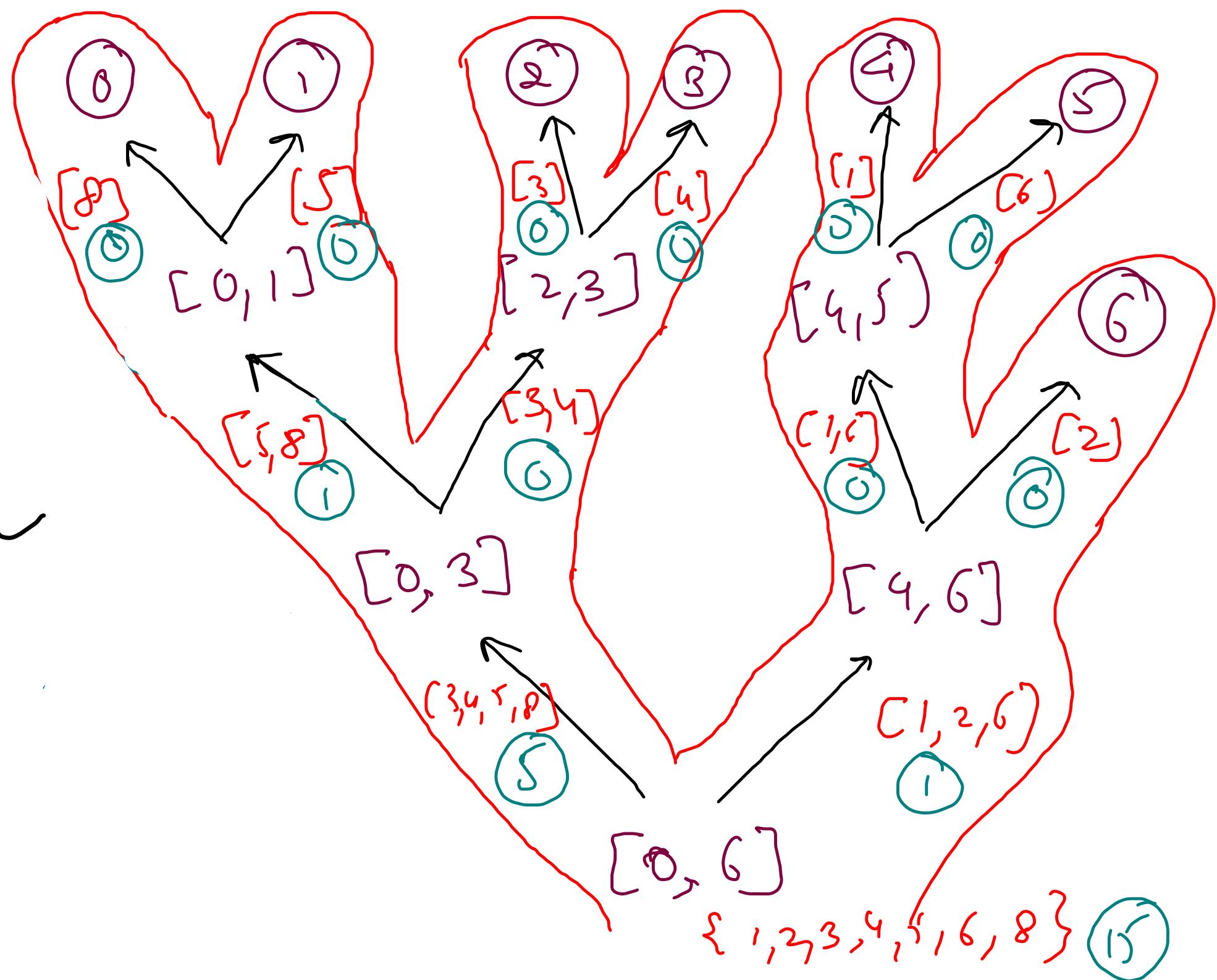
15 pairs

Brute force :  $\rightarrow O(N^2)$

- optimization
  - Merge Sort ( $N \log N$ )
  - Fenwick Tree
  - Policy Based Data Structure
  - Self Balancing BST

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

① { (6,2) }  
② { (8,4) }  
③ { (6,2) }  
④ { (8,3) }  
⑤ { (5,3) }  
⑥ { (3,1) }  
⑦ { (4,1) }  
⑧ { (5,1) }  
⑨ { (8,1) }  
⑩ { (3,2) }  
⑪ { (4,2) }  
⑫ { (5,2) }  
⑬ { (8,2) }  
⑭ { (3,1) }  
⑮ { (4,1) }  
⑯ { (5,1) }  
⑰ { (8,1) }



```

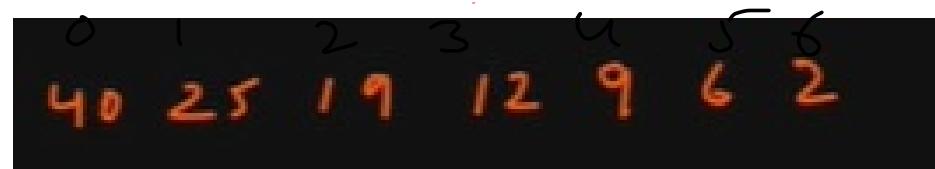
public void mergeSort(int[] nums, int left, int right){
    if(left >= right){
        return; // 0 or 1 elements in array
    }

    int mid = left + (right - left) / 2;
    mergeSort(nums, left, mid); // left Sorted
    mergeSort(nums, mid + 1, right); // right Sorted
    merge(nums, left, mid, mid + 1, right);
}

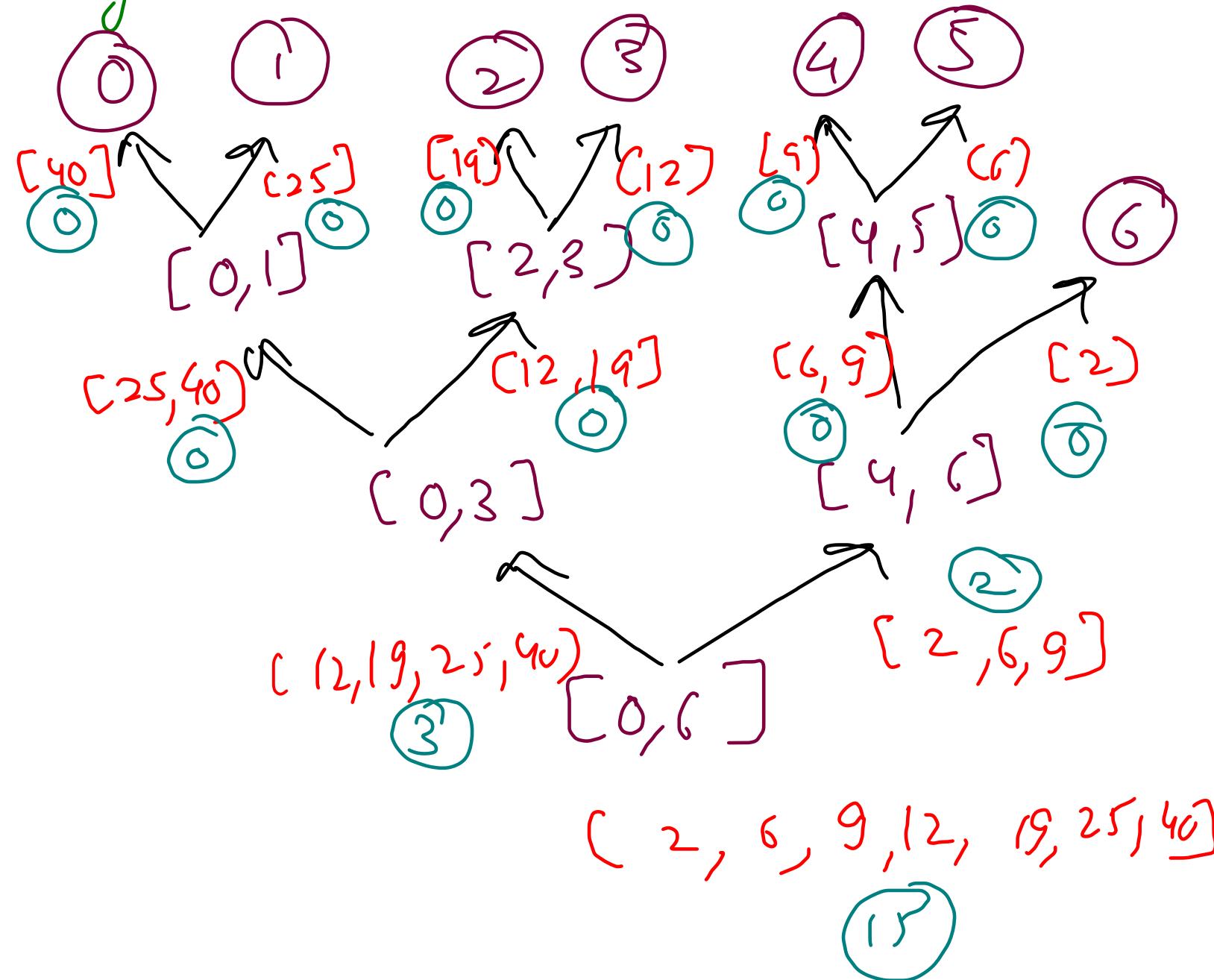
```

## Reverse Pairs

$i < j$  but  $a[i] > 2 \neq a[j]$



$(2^5, 12)$     $(40, 12)$     $(40, 13)$   
 $(6, 2)$     $(9, 2)$     $(12, 2)$     $(13, 2)$     $(25, 2)$   
 $(13, 6)$     $(25, 6)$     $(40, 6)$     $(13, 9)$     $(25, 9)$     $(40, 9)$



```
public int findInversionCount(int[] nums, int a1l, int a1r, int a2l, int a2r){  
    int invCount = 0;  
    int ptr1 = a1l, ptr2 = a2l, ptr3 = 0;  
    while(ptr1 <= a1r && ptr2 <= a2r){  
        if((long)nums[ptr1] <= 21 * (long)nums[ptr2]){  
            ptr3++; ptr1++;  
        }  
        else {  
            invCount += (a1r - ptr1 + 1);  
            ptr3++; ptr2++;  
        }  
    }  
    return invCount;  
}
```

GOOGLE Count Winning Streaks (NADAS)

[Premium] Leetcode 2031

Count subarrays with more 1s than 0s

0 1 2 3 4 5 6 7  
[0, 1, 1, 0, 1, 1, 1, 0]

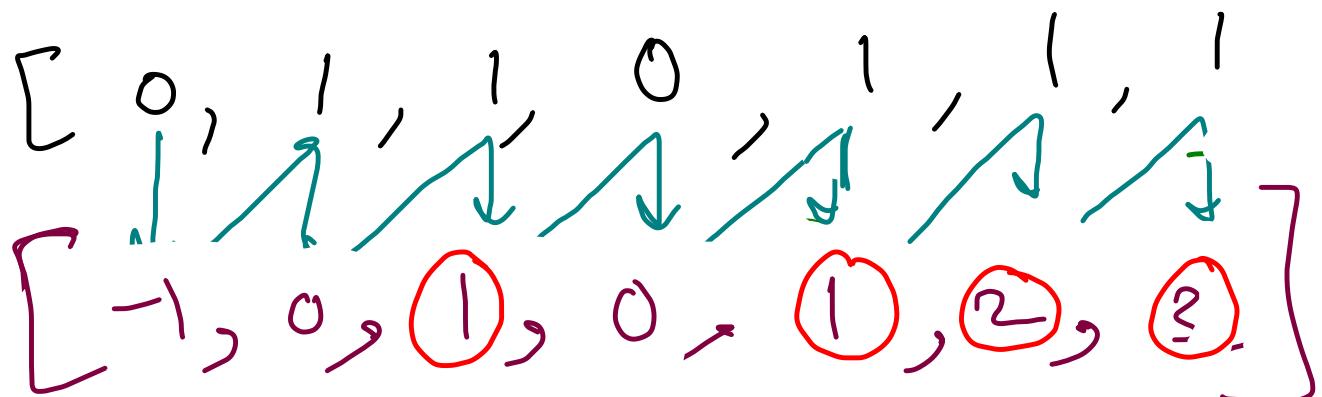
$N^3 \rightarrow N^2 \rightarrow N \log N$   
↑  
3 loops      Optimised Subarray (2 loops)  
for count {beginning} {permutation}

(0, 1, 1)      (1, 1, 0, 1)  
(1, 1, 0)      (1, 0, 1)  
- - - - -  
- - - - -

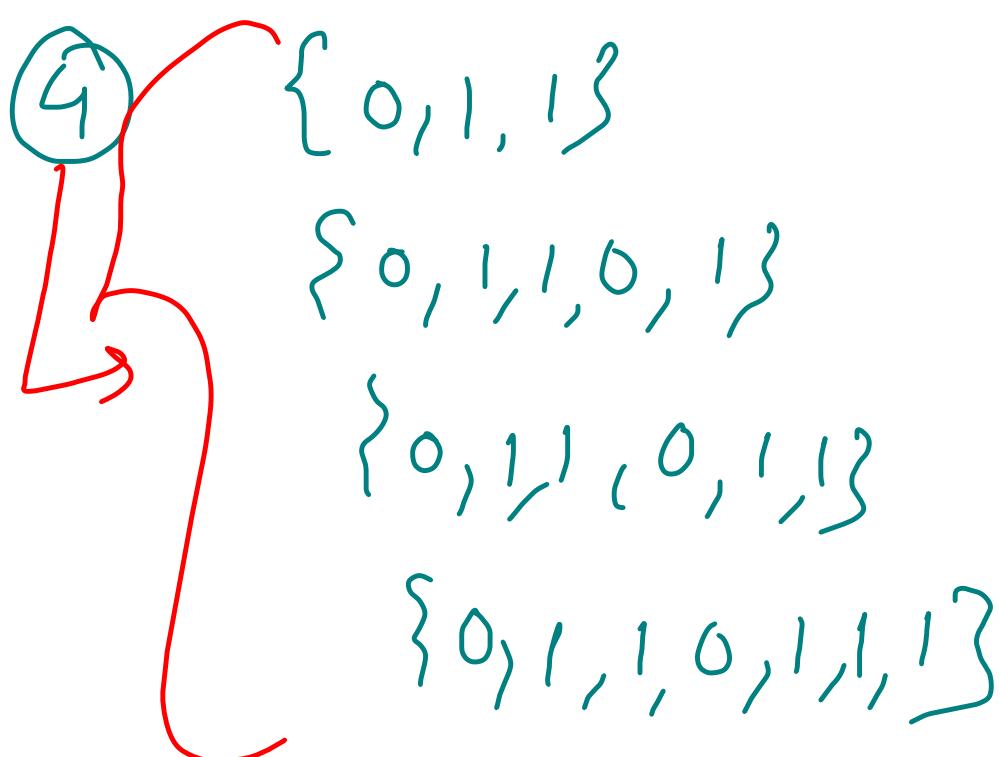
```
public static int winningStreak(int[] arr){  
    int ans = 0;  
  
    for(int st=0; st<arr.length; st++){  
        int countof0s = 0, countof1s = 0;  
        for(int end=st; end<arr.length; end++){  
            if(arr[end] == 0){  
                countof0s++;  
            } else {  
                countof1s++;  
            }  
  
            if(countof1s > countof0s){  
                ans++;  
            }  
        }  
    }  
    return ans;  
}
```

$O(N^2)$   $(T(E))$

“consider all  
subarrays”



IS > OS  
↓  
+1      -1



$[1]$   
 $(1, 1)$   
 $(1, 1, 1)$

$(1, 1, 1, 0, 1)$

$(1)$   
 $(1, 1)$   
 $(0, 1, 1)$   
 $\vdots$   
 $)$

$(1, 1, 0)$   
 $\vdots$

```
int subarraysWithMoreZerosThanOnes(vector<int> &nums)
{
    int n = nums.size();
    vector<int> pref(n);
    int sum = 0;

    for (int i = 0; i < n; i++) {
        if (nums[i] == 0)
            nums[i] = -1;
        sum += nums[i];
        pref[i] = sum;
    }

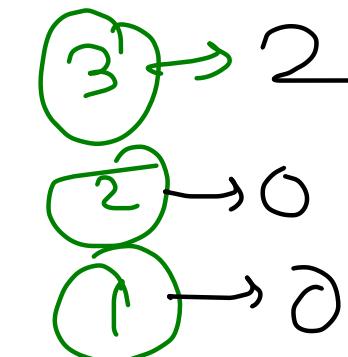
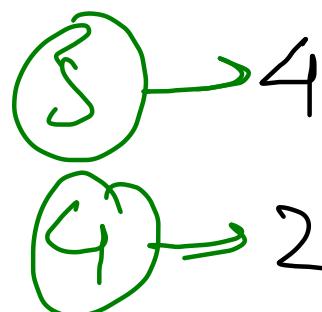
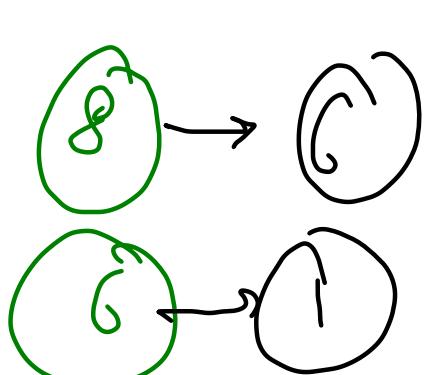
    int inversions = 0;
    for (int i = 0; i < n; i++) {
        if (pref[i] > 0)
            inversions = (inversions + 1) % mod;
    }

    reverse(pref.begin(), pref.end());
    mergeSort(pref, 0, n - 1, inversions);
    return inversions;
}
```

Minimum

Swaps to Sort (II)

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



bubble sort-

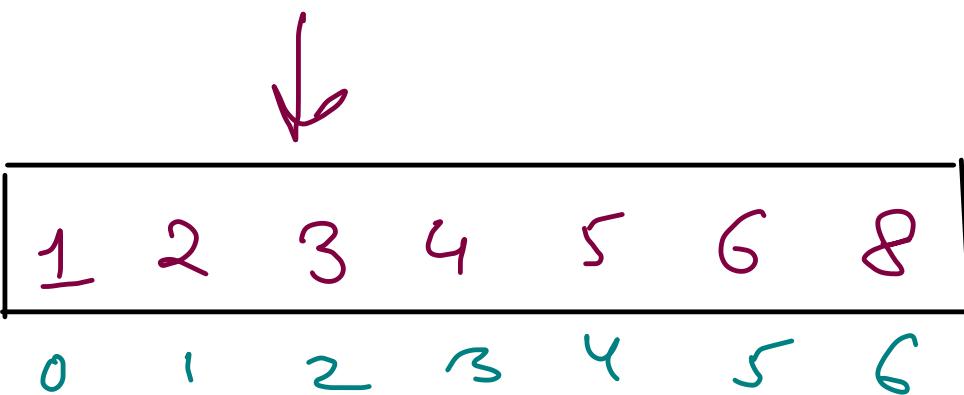
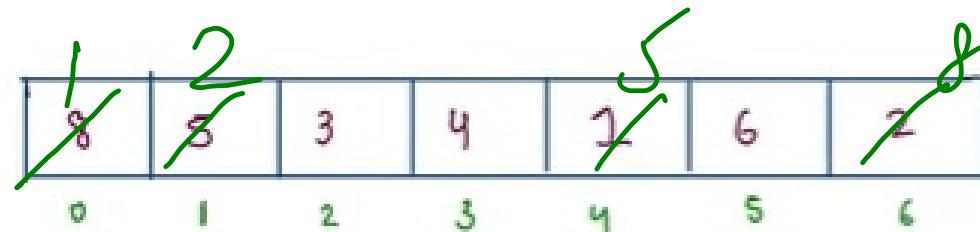
Min adjacent swaps to sort

Inversion Count

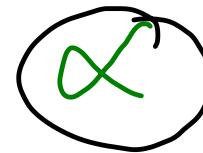
$O(N \log N)$

Merge Sort

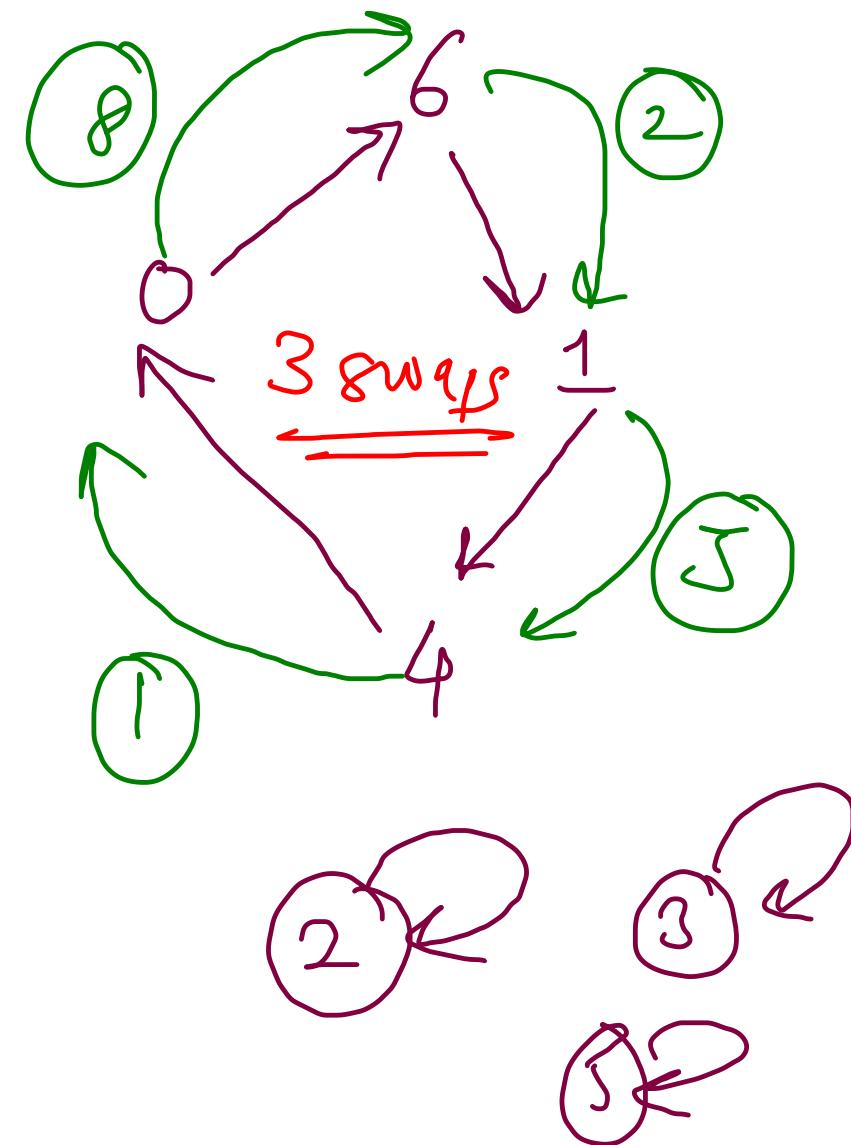
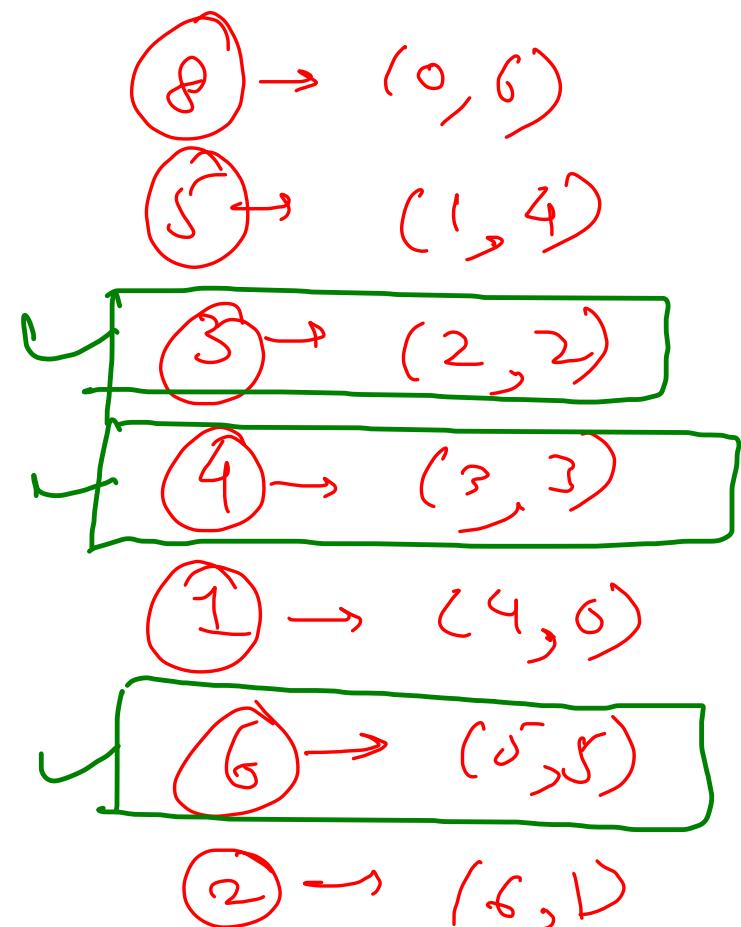
# Minimum swaps to Sort (I)

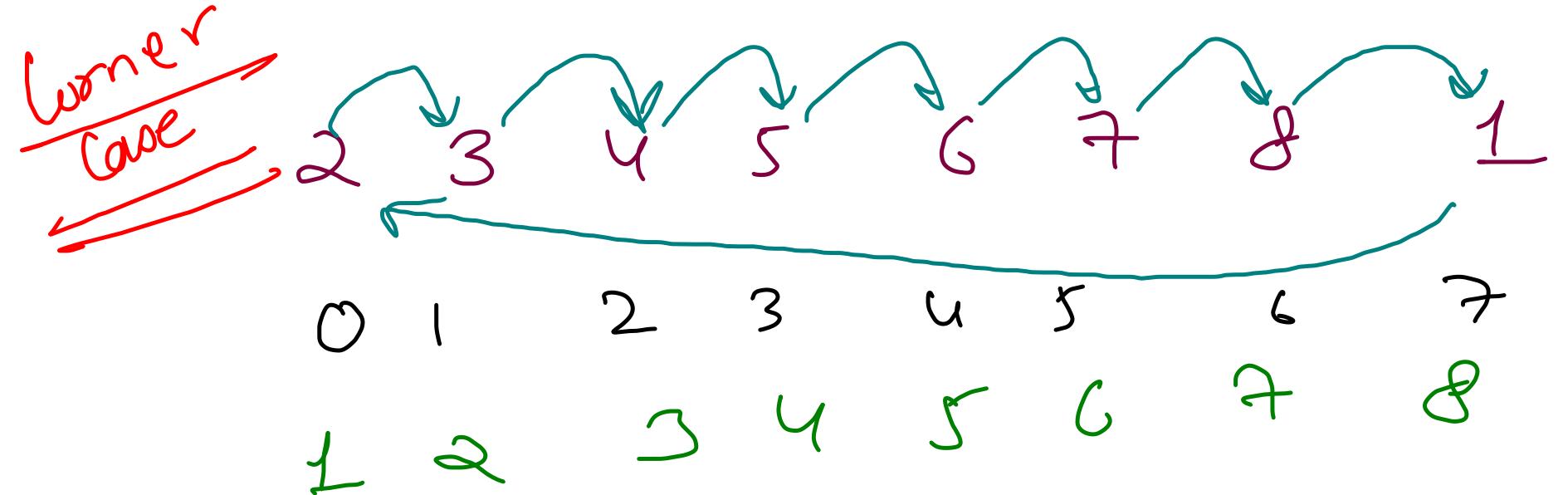


→ Adjacent



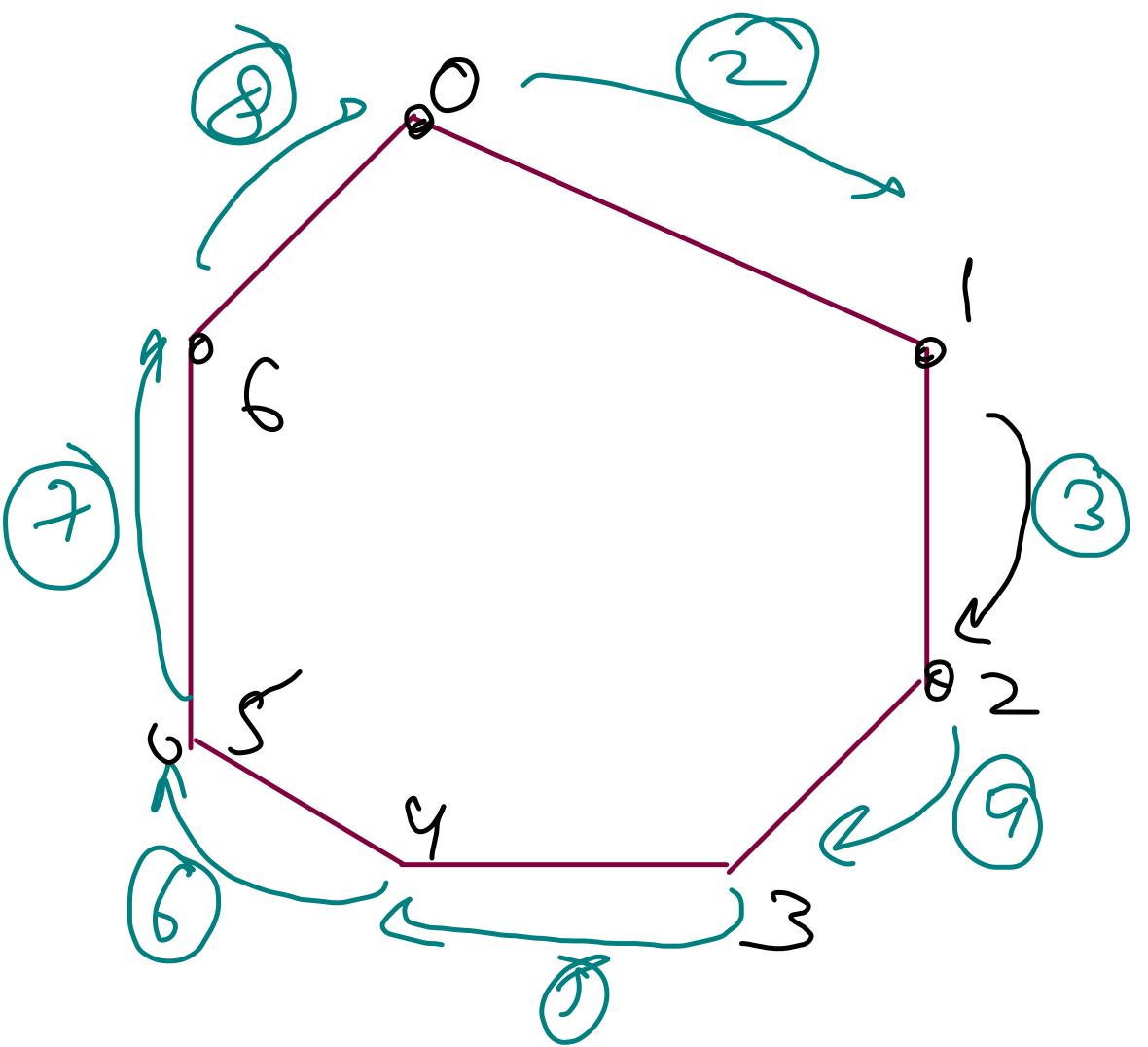
val → (initial, final)





$0 \rightarrow 1$   
 $1 \rightarrow 2$   
 $2 \rightarrow 3$   
 $3 \rightarrow 4$   
 $4 \rightarrow 5$   
 $5 \rightarrow 6$   
 $6 \rightarrow 7$   
 $7 \rightarrow 0$

*6 swaps*  
*cycle length = 1*



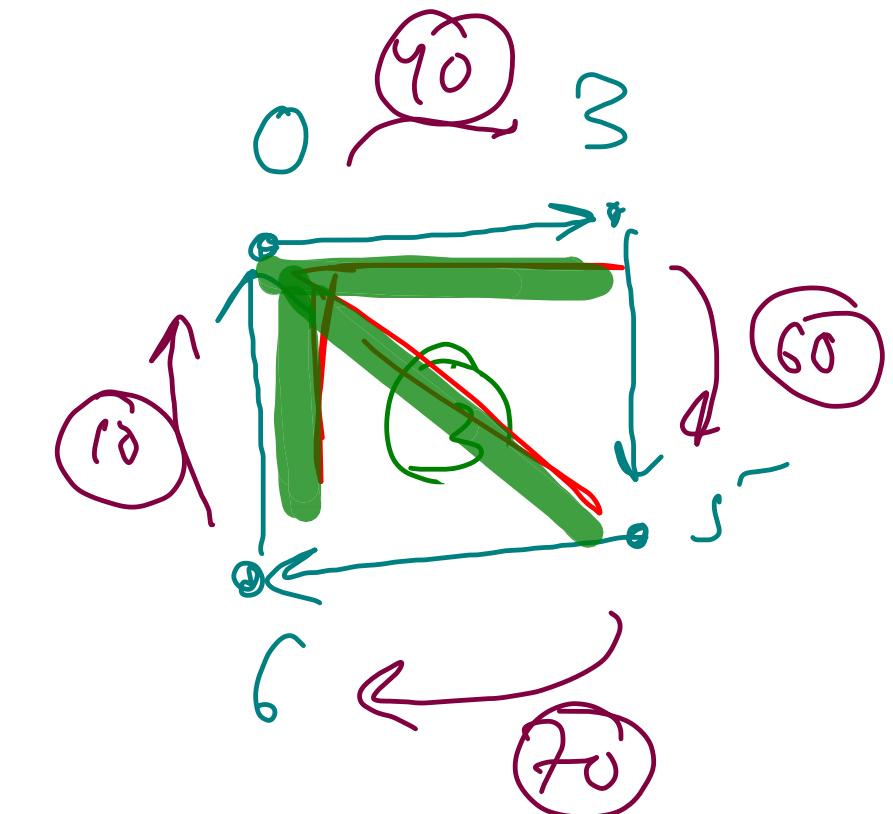
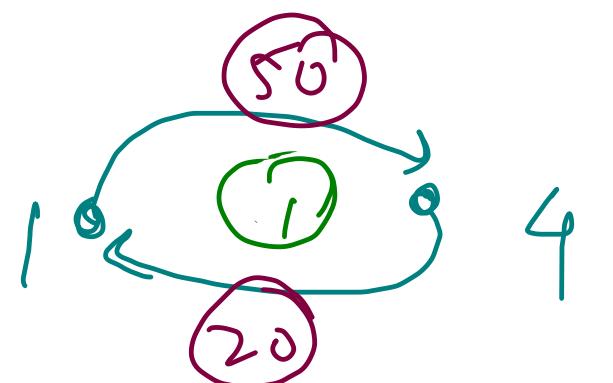
40	50	30	60	20	70	10	80
0	1	2	3	4	5	6	7
10	20	30	40	50	60	70	80

distinct

0 → 3      60 → 60      3 → 5      10 → 0      6 → 0

1 → 4      20 → 20      4 → 1      80 → 80      7 → 7

2 → 2      70 → 70      5 → 6



{ Total nodes - no of components }

```

int cycle_size(vector<pair<int,int>> &arr, int start, vector<bool> &vis)
{
    int res = 0;
    int j = start;
    while (!vis[j])
    {
        vis[j] = true;
        j = arr[j].second;
        res++;
    }
    return res - 1;
}
int minSwaps(int a[]){
    vector<pair<int, int>> arr(n);
    for(int i=0;i<n;i++)
        arr[i] = make_pair(a[i], i);
    sort(arr.begin(), arr.end());
    vector<bool> vis(n, false);
    int res = 0;
    for(int i=0;i<n;i++)
    {
        if(vis[i] == false)
            res += cycle_size(arr, i, vis);
    }
    return res;
}

```

$$\begin{aligned}
 & O(N \log N) \\
 \xrightarrow{\quad} & O(N)
 \end{aligned}$$

# Sorting - lecture ③

Revision



Count Sort, Radix Sort

# Bucket Sort

- (~~O(n)~~) Top K frequent Elements
- (~~O(n)~~) Frequency Sort

# Coordinate Compression

Extra Question

- (Q) Maximum Gap
- (Q) Relative Sort Array ( $O(n \log n)$ )

Radix Sort

```

// Karenge to Sorting hi, but tareeka hoga 'Radix Sort'
// Time Complexity - O(N * digits) = O(N * 5) -> Linear
// Space Complexity - O(N + Digits) = O(N) -> Linear

void countSort(vector<int> &arr, int n, int exp)
{
    int output[n];
    vector<int> count(10, 0);

    for (int i = 0; i < n; i++)
        count[(arr[i] / exp) % 10]++;
}

for (int i = 1; i < 10; i++)
    count[i] += count[i - 1];

for (int i = n - 1; i >= 0; i--)
{
    output[count[(arr[i] / exp) % 10] - 1] = arr[i];
    count[(arr[i] / exp) % 10]--;
}

for (int i = 0; i < n; i++)
    arr[i] = output[i];
}

int maximumGap(vector<int>& nums)
{
    if(nums.size() <= 1) return 0;

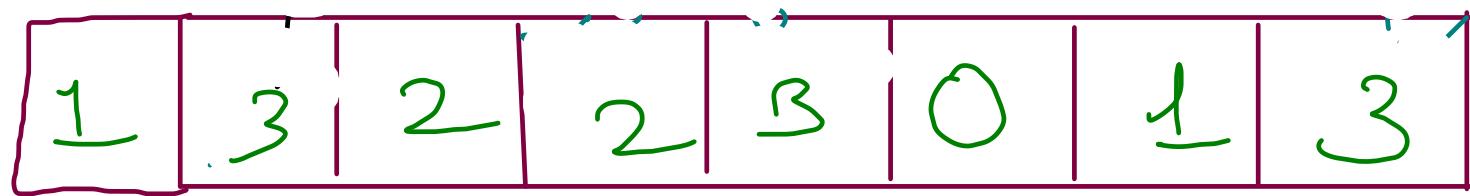
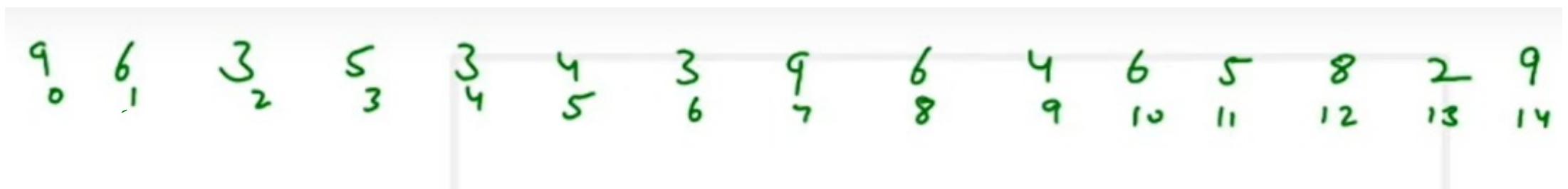
    // Radix Sort
    int n = nums.size();
    int m = *max_element(nums.begin(), nums.end());
    for(int exp = 1; m/exp > 0; exp *= 10)
        countSort(nums, n, exp);

    int ans = 0;
    for(int i=0; i<n-1; i++)
        ans = max(ans, nums[i+1] - nums[i]);
    return ans;
}

```

Maximum Gap  
↓  
Radix Sort

## Count Sort



↓  
min

$$O(m \cdot n - \min + 1)$$

extra space

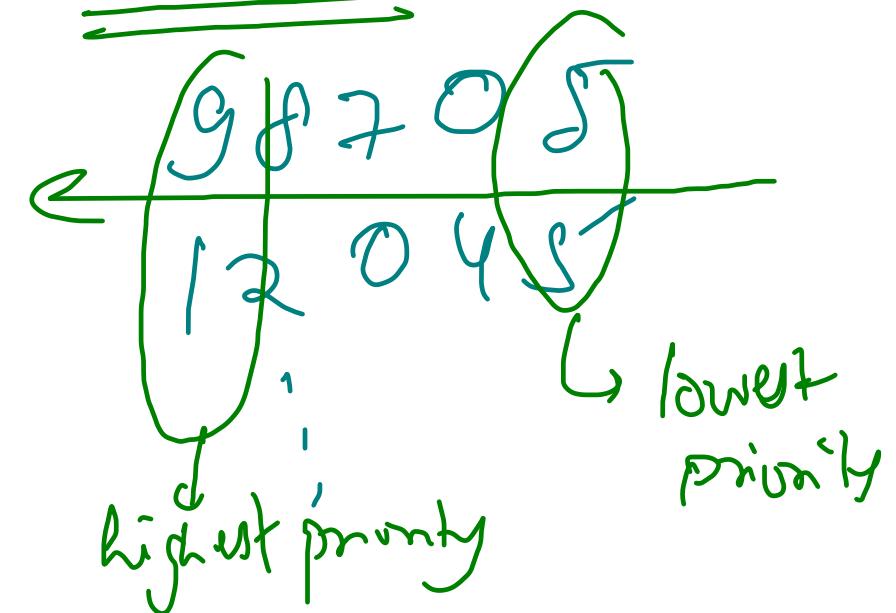
Time  $\Rightarrow O(N)$

Elements in small range

2 3 3 3 4 4 5 5 6 6 6 8 9 5 3

TC  $\Rightarrow O(10 \cdot n)$   
SC  $\Rightarrow O(n)$

# Radix Sort



# Bucket List

① Characters → String Sort

The diagram shows the word "Characters" in blue at the top left. A curved arrow points from it to the text "String Sort" in black at the top right. Below this, three green circles contain the numbers 26, 52, and 256. The number 26 is labeled "lowercase" below it and "uppercase" above it. The number 52 is labeled "uppercase" above it and "lowercase" below it. The number 256 is labeled "maximum characters" below it.

2 Marks  $\rightarrow (0, 100)$

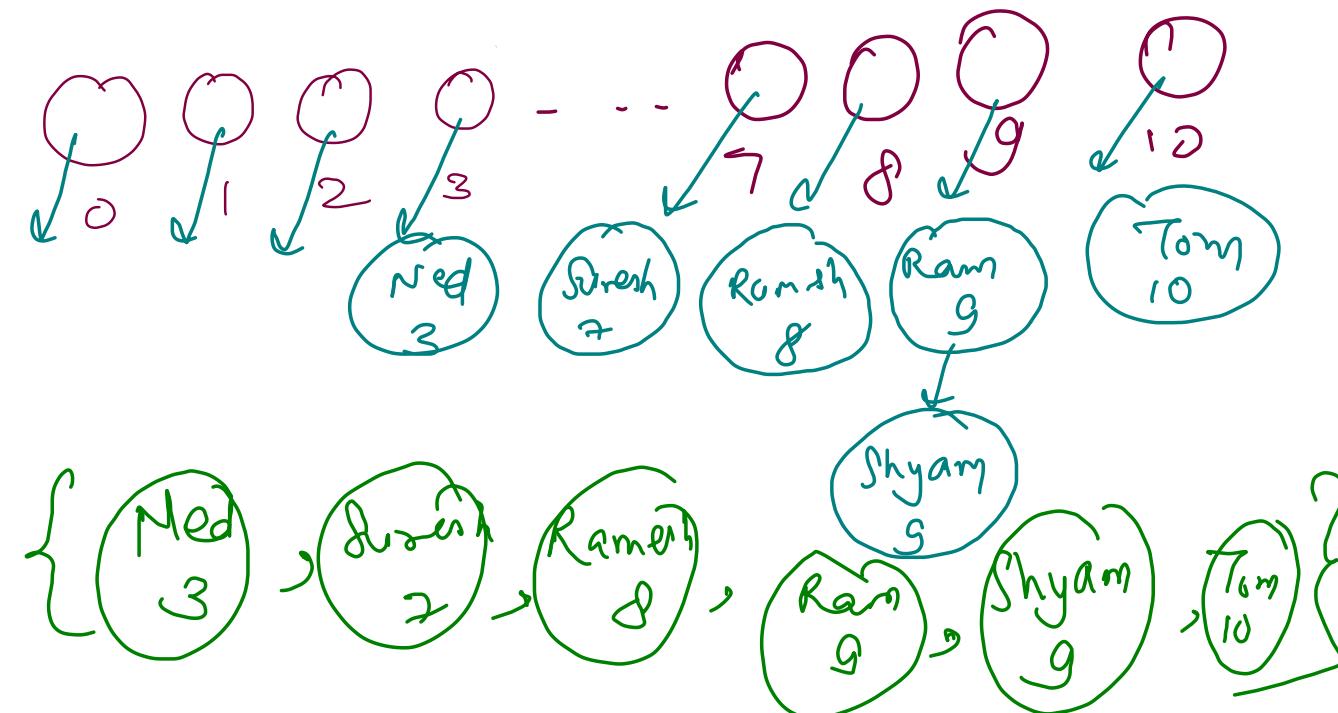
Ram → ♂  
 Ramon → ♀  
 Shyam → ♂  
 Suresh → ♂

Tom → 10  
Ned → 3

No of buckets  
= no of distinct values  
= 11

"before coding"

$\begin{array}{ccccccc} d & \downarrow & c & - & - & - & z \\ \downarrow & \downarrow & \downarrow & & & & \\ o & o & 1 & - & - & - & \end{array}$



$$O(\text{no of buckets} + \text{no of elements}) = O(11 + N) \Rightarrow O(N)$$

Time  $\Rightarrow$  O(N)

③ Values  $\Rightarrow$   $n$  elements  $\Rightarrow$  values  $[0, n-1]$  or  $[1, \underline{\underline{n}}]$   
 $\hookrightarrow n$  buckets

# 1636

## Frequency Sort

```
public int[] frequencySort(int[] s) {
    HashMap<Integer, Integer> freq = new HashMap<>();
    for(int i=0; i<s.length; i++){
        freq.put(s[i], freq.getOrDefault(s[i], 0) + 1);
    }

    TreeSet<Integer>[] buckets = new TreeSet[s.length + 1];
    for(int i=0; i<buckets.length; i++){
        buckets[i] = new TreeSet<>();
    }

    for(Integer ch: freq.keySet()){
        int val = freq.get(ch);
        buckets[val].add(ch);
    }

    int[] res = new int[s.length];
    int idx = 0;

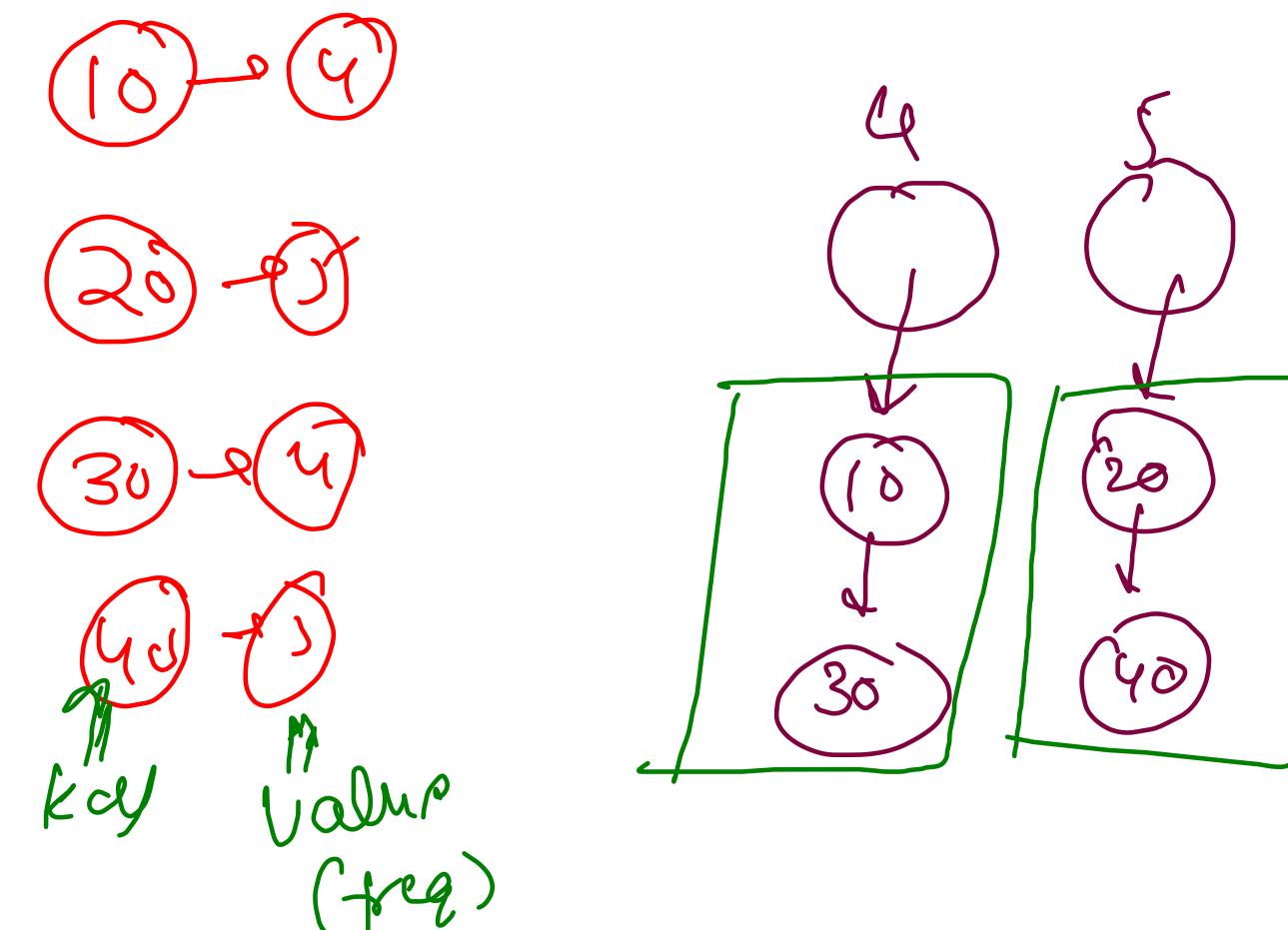
    for(int b=0; b<buckets.length; b++){
        for(Integer ch: buckets[b].descendingSet()){
            for(int f=0; f<b; f++){
                res[idx++] = ch;
            }
        }
    }
    return res;
}
```

{ 1, 2, 3, 4, 5, 6, 7, 8, 9 }  $\{ \overset{1}{\textcircled{1}} \rightarrow \{ \overset{1, 2, 3, 4, 5}{\textcircled{1, 2, 3, 4, 5}} \}$

$O(N \log N)$  } all distinct keys  
Worst case

Best Case  
 $O(N^2)$

Tree Set (Java)  $O(\log N)$  ↑ Removal  
Ordered set (C++) ↑  
↳ Self Balancing BST (AVL)



```
vector<int> frequencySort(vector<int>& nums)
{
    vector<int> res;
    int n = nums.size();
    if (n == 0) return res;

    unordered_map<int, int> freq;
    for (auto i : nums) freq[i]++;

    vector<set<int, greater<int>>> bucket(n + 1);
    for (auto &p: freq)
        bucket[p.second].insert(p.first);
    |

    for (int i=1; i<=n; i++)
        for (auto &e: bucket[i])
            for(int j=0; j<i; j++)
                res.push_back(e);

    return res;
}
```

C++ Code

# 451 Sort Characters by Frequency

```

public String frequencySort(String s) {
    HashMap<Character, Integer> freq = new HashMap<>();
    for(int i=0; i<s.length(); i++){
        char ch = s.charAt(i);
        freq.put(ch, freq.getOrDefault(ch, 0) + 1);
    }

    ArrayList<Character>[] buckets = new ArrayList[s.length() + 1];
    for(int i=0; i<buckets.length; i++){
        buckets[i] = new ArrayList<>();
    }

    for(Character ch: freq.keySet()){
        int val = freq.get(ch);
        buckets[val].add(ch);
    }

    StringBuilder sb = new StringBuilder("");
    for(int b=buckets.length-1; b>=0; b--){
        for(Character ch: buckets[b]){
            for(int f=0; f<b; f++){
                sb.append(ch);
            }
        }
    }

    return sb.toString();
}

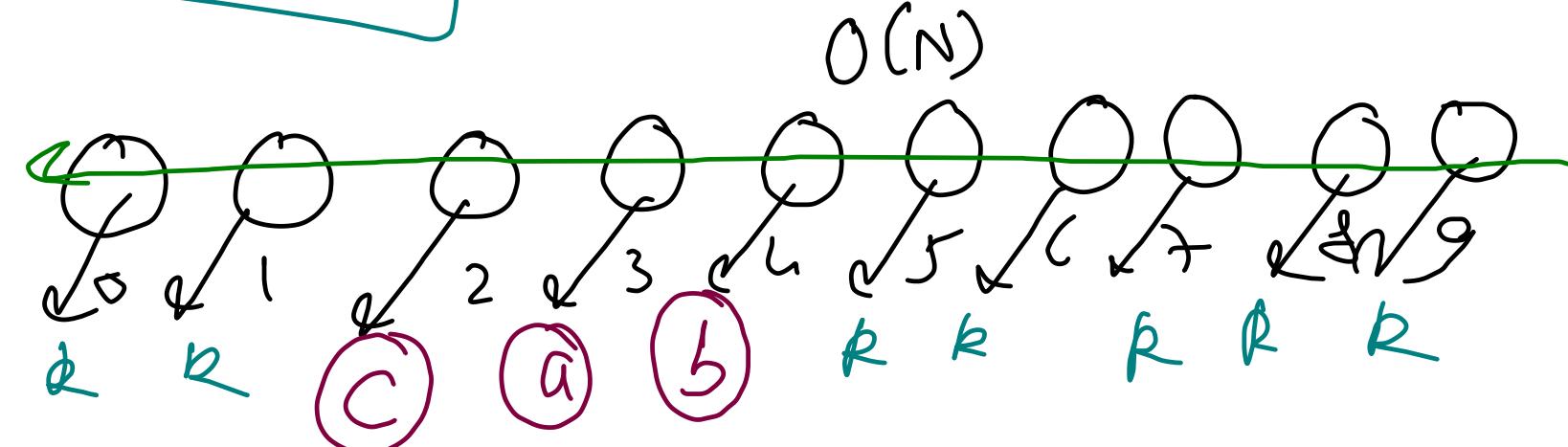
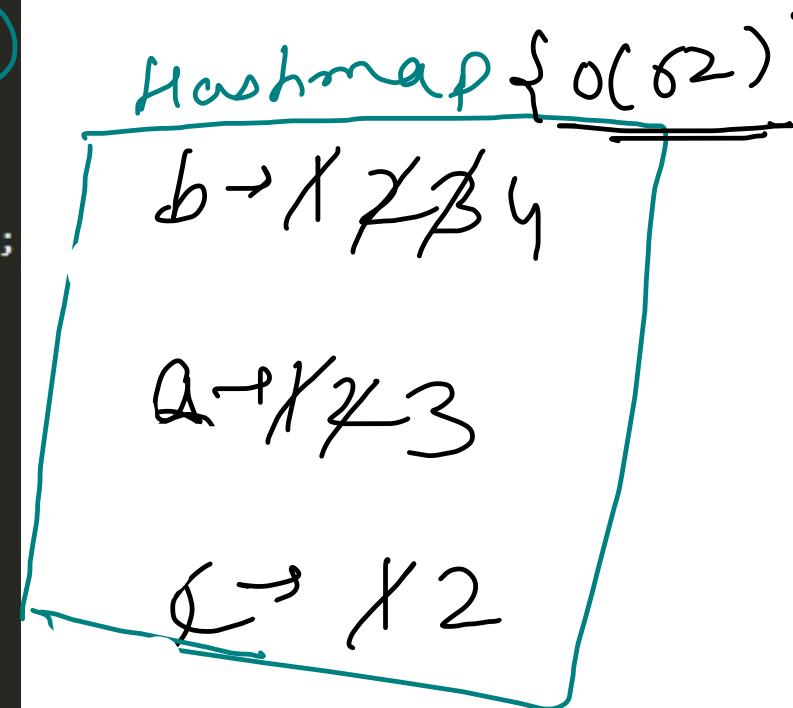
```

*Annotations:*

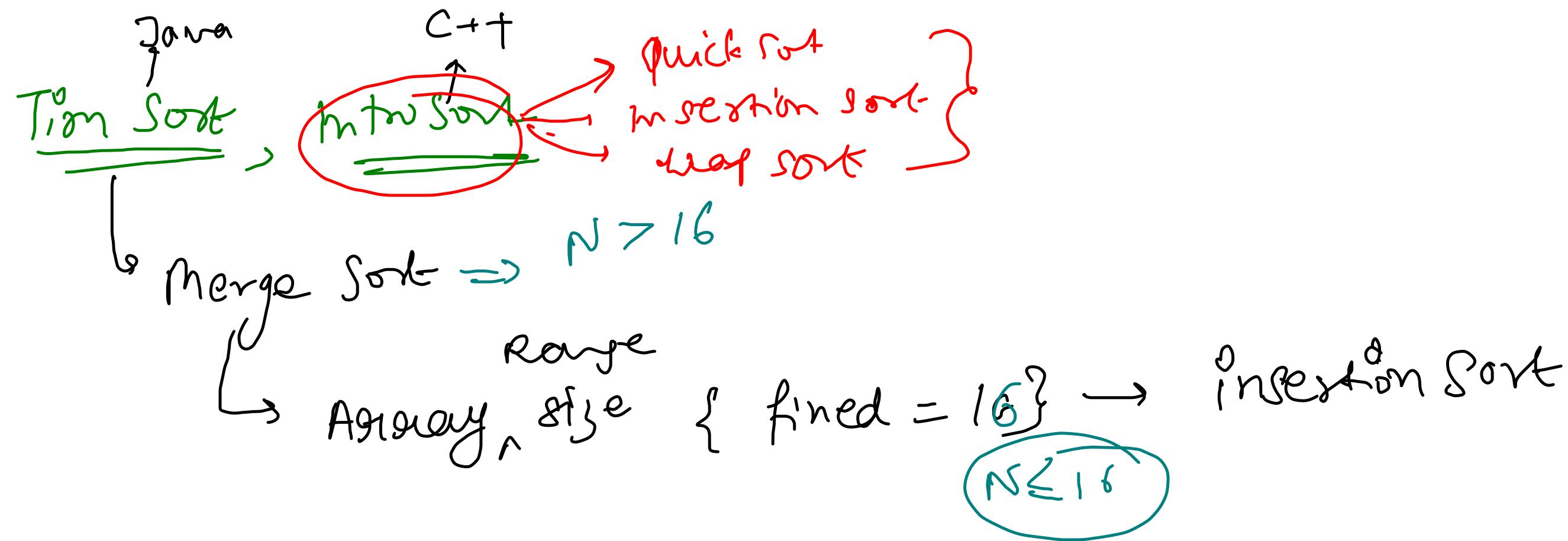
- ①  $O(N)$  - Hashmap fill
- ②  $O(N)$  - Bucket init
- ③  $O(62)$  - Bucket fill
- ④  $O(N)$  - Sort on basis of freq (DEC)

"bbbbaadgccc"

$[0, 9]$   
 $\downarrow$   
no of buckets  
= String length + 1



"bbbbaadgccc"



# Top K Frequent Elements

(347, 692)

Given an integer array `nums` and an integer `k`, return the `k` most frequent elements. You may return the answer in **any order**.

Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $k$  is in the range  $[1, \text{the number of unique elements in the array}]$ .
- It is **guaranteed** that the answer is **unique**.

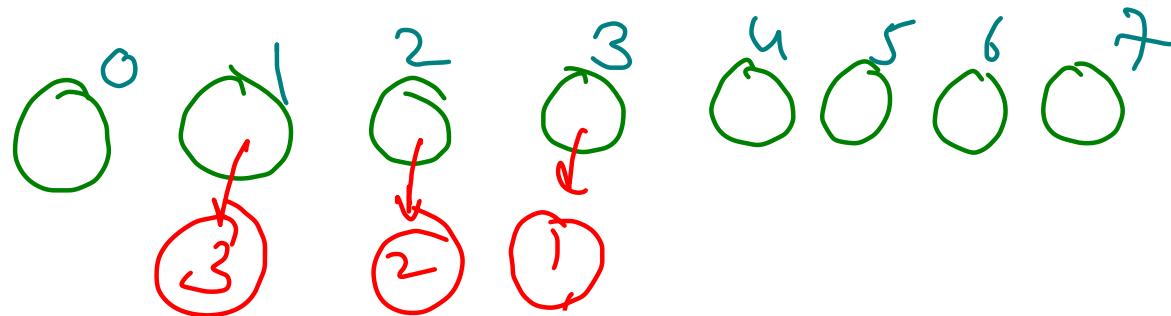
Follow up: Your algorithm's time complexity must be better than  $O(n \log n)$ , where  $n$  is the array's size.

① Priority Queue  $\Rightarrow O(n \log n)$

② Bucket Sort  $\Rightarrow O(N)$

$\{1, 1, 1, 2, 2, 3\}$

$1 \rightarrow 3$        $2 \rightarrow 2$        $3 \rightarrow 1$



Java Code

```
public int[] topKFrequent(int[] s, int k) {
    HashMap<Integer, Integer> freq = new HashMap<>();
    for(int i=0; i<s.length; i++){
        freq.put(s[i], freq.getOrDefault(s[i], 0) + 1);
    }

    ArrayList<Integer>[] buckets = new ArrayList[s.length + 1];
    for(int i=0; i<buckets.length; i++){
        buckets[i] = new ArrayList<>();
    }

    for(Integer ch: freq.keySet()){
        int val = freq.get(ch);
        buckets[val].add(ch);
    }

    int[] res = new int[k];
    int idx = 0;

    for(int b=buckets.length-1; b>=0; b--){
        for(Integer ch: buckets[b]){
            res[idx++] = ch;
            if(idx == k) return res;
        }
    }
    return res;
}
```

C++ Code

```
// Bucket Sort
// First Find Frequency of All Elements using Hashmap O(N) Extra Space.
// Now Create N + 1 buckets for where ith bucket will contain elements with freq = i
// Since Frequency can vary from 0 to N, hence N + 1 buckets.
// Hence, another O(N) Extra Space

// Now Starting with last bucket, keep inserting elements, until size becomes k.
// Hence Time Complexity = O(N) for Filling hashmap + O(N) for filling buckets
// + O(K) for taking out elements in decreasing frequency.
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
        vector<int> res;
        int n = nums.size();
        if (n == 0) return res;

        unordered_map<int, int> freq;
        for (auto i : nums) freq[i]++;
        vector<vector<int>> bucket(n + 1);
        for (auto &p: freq)
            bucket[p.second].push_back(p.first);

        for (int i=n; i>=0; i--)
        {
            for (auto &e: bucket[i])
            {
                res.push_back(e);
                if (res.size() == k) return res;
            }
        }
        return res;
    }
}.
```

692  
Java code

```
public List<String> topKFrequent(String[] s, int k) {
    HashMap<String, Integer> freq = new HashMap<>();
    for(int i=0; i<s.length; i++){
        freq.put(s[i], freq.getOrDefault(s[i], 0) + 1);
    }

    TreeSet<String>[] buckets = new TreeSet[s.length + 1];
    for(int i=0; i<buckets.length; i++){
        buckets[i] = new TreeSet<>();
    }

    for(String word: freq.keySet()){
        int val = freq.get(word);
        buckets[val].add(word);
    }

    List<String> res = new ArrayList<>();

    for(int b=buckets.length-1; b>=0; b--){
        for(String word: buckets[b]){
            res.add(word);
            if(res.size() == k) return res;
        }
    }
    return res;
}
```

C++ code

```
static bool Compare(pair<string,int> p1, pair<string,int> p2)
{
    if(p1.second>p2.second || (p1.second==p2.second && p1.first<p2.first))
        return true;
    return false;
}

vector<string> topKFrequent(vector<string>& words, int k) {
    unordered_map<string,int> m;

    for(int i=0; i<words.size(); i++)
        m[words[i]]++;

    vector<pair<string,int>> arr;
    for(auto it=m.begin(); it!=m.end(); it++)
        arr.push_back({it->first,it->second});

    sort(arr.begin(),arr.end(),Compare);

    vector<string> ans;
    for(int i=0; i<k; i++)
        ans.push_back(arr[i].first);

    return ans;
}
```

# Coordinate Compression

Given an array with  $N$  distinct elements, convert the given array to a reduced form where all elements are in range from 0 to  $N-1$ . The order of elements is same, i.e., 0 is placed in place of smallest element, 1 is placed for second smallest element, ...  $N-1$  is placed for largest element.

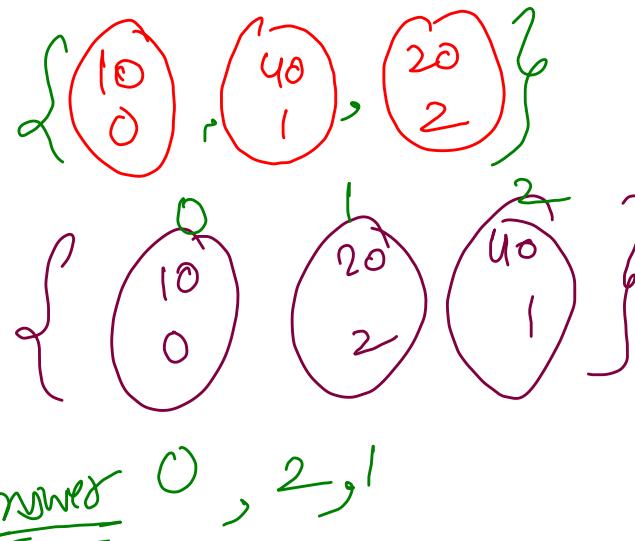
Expected Time Complexity:  $O(N \log N)$

Expected Auxiliary Space:  $O(N)$

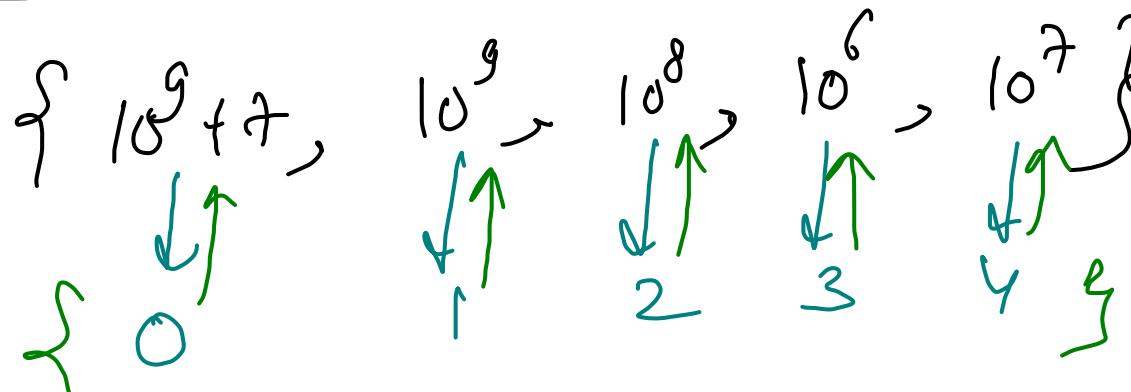
Constraints:

$1 \leq N \leq 10^5$

$1 \leq \text{Arr}[i] \leq 10^6$



## Ptime factors

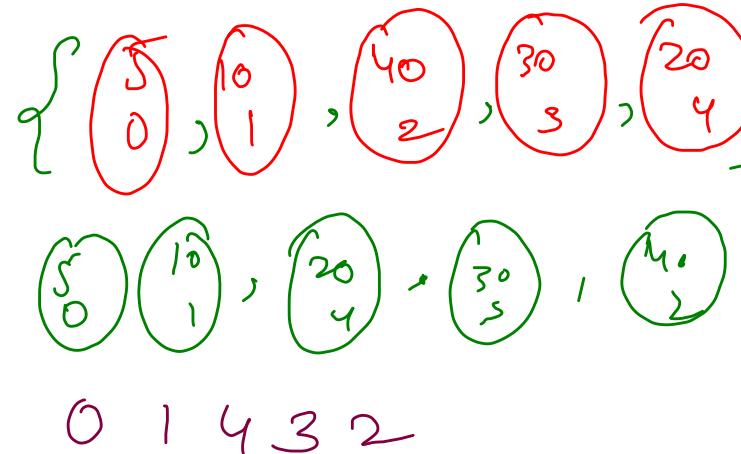


## Example ①

**Input:**  
 $N = 3$   
 $\text{Arr[]} = \{10, 40, 20\}$   
**Output:** 0 2 1  
**Explanation:** 10 is the least element so it is replaced by 0. 40 is the largest element so it is replaced by  $3-1 = 2$ . And 20 is the 2nd smallest element so it is replaced by 1.

## Example ②

**Input:**  
 $N = 5$   
 $\text{Arr[]} = \{5, 10, 40, 30, 20\}$   
**Output:** 0 1 4 3 2  
**Explanation:** As 5 is smallest element, it's replaced by 0. Then 10 is 2nd smallest element, so it's replaced by 1. Then 20 is 3rd smallest element, so it's replaced by 2. And so on.



```
static class Pair implements Comparable<Pair>{
    int val;
    int idx;

    Pair(int val, int idx){
        this.val = val;
        this.idx = idx;
    }

    public int compareTo(Pair other){
        return this.val - other.val;
    }
}
```

```
void convert(int[] arr, int n) {
    Pair[] sorted = new Pair[n];
    for(int i=0; i<n; i++){
        sorted[i] = new Pair(arr[i], i);
    }
    Arrays.sort(sorted);

    for(int i=0; i<n; i++){
        arr[sorted[i].idx] = i;
    }
}
```

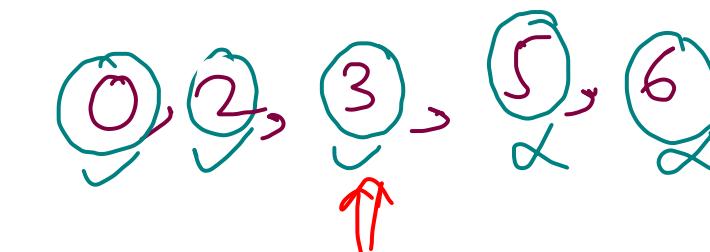
# H Index (274)

Given an array of integers `citations` where `citations[i]` is the number of citations a researcher received for their  $i^{\text{th}}$  paper, return compute the researcher's **h-index**.

According to the definition of h-index on Wikipedia: A scientist has an index  $h$  if  $h$  of their  $n$  papers have at least  $h$  citations each, and the other  $n - h$  papers have no more than  $h$  citations each.

If there are several possible values for  $h$ , the maximum one is taken as the **h-index**.

$\{3, 0, 6, 2, 5\}$



h index  $\Rightarrow$  {at least  $h$  papers having value  $\geq h$ }  
maximum

example ②  
 $\{1, 3, 4, 5, 7, 5, 10, 11\}$  8 elements  
T T T T F F F F

0	1	2	3	4	5	6	7	8	$> 8$
8	8	7	7	6	5	4	4	3	3
✓	✓	✓	✓	✓	✗	✗	✗	✗	✗

h index

- ①  $>n$  extra bucket.
- ② suffix array

TC  $\Rightarrow O(n)$   
SC  $\Rightarrow O(n)$

```
public int hIndex(int[] citations) {  
    int[] buckets = new int[citations.length + 2];  
    for(int val: citations){  
        if(val > citations.length){  
            buckets[buckets.length - 1]++;  
        } else {  
            buckets[val]++;  
        }  
    }  
  
    for(int i=buckets.length-2; i>=0; i--){  
        buckets[i] += buckets[i + 1];  
        if(buckets[i] >= i){  
            return i;  
        }  
    }  
    return 0;  
}
```

$O(N)$

$\rightarrow O(N)$