Pivot Partition

Quick Sort

Comparator Problems

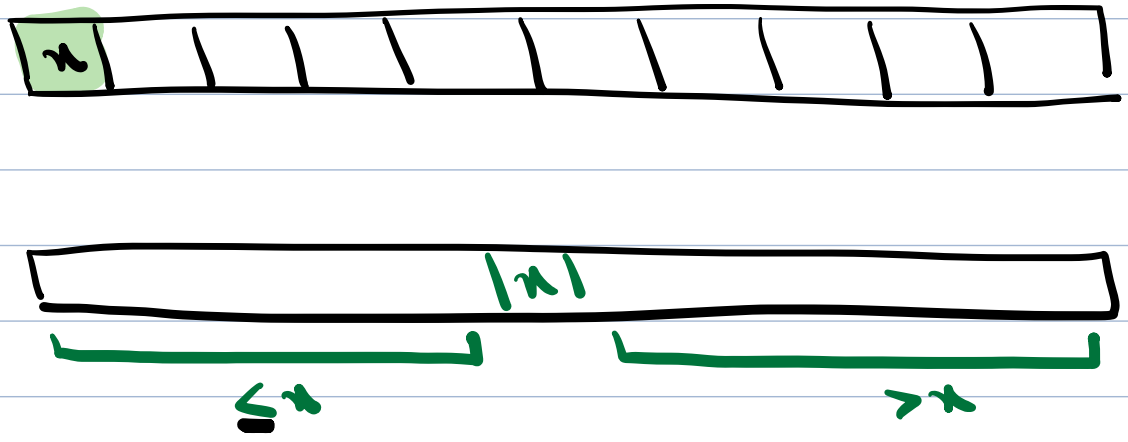
Contest 1 -> Monday (25 Nov)

Arrays, Bit Manipulation, Recursion, Math, Hashing & Sorting

4 Q -> 3/4 Passed

Given an integer array, consider 1st element as pivot rearrange the elements such that for all i:
if A[i] < p then it should be present on left side
if A[i] > p then it should be present on right side

Note: All elements are distinct



arr[] : 54, 26, 93, 17, 77, 31, 44, 55, 20

Ans : 26, 17, 31, 44, 20, 54, 55, 77, 93
(26, 17, 31, 44, 20 : <54; 55, 77, 93 : >54)

arr[] : 10, 13, 7, 8, 25, 20, 23, 5

Ans : 7, 8, 5, 10, 13, 25, 20, 23
(7, 8, 5 : <10; 13, 25, 20, 23 : >10)

① On partitioning the array based on pivot, pivot reaches its sorted place.

soln 1 : sort array TC : O(N log N)

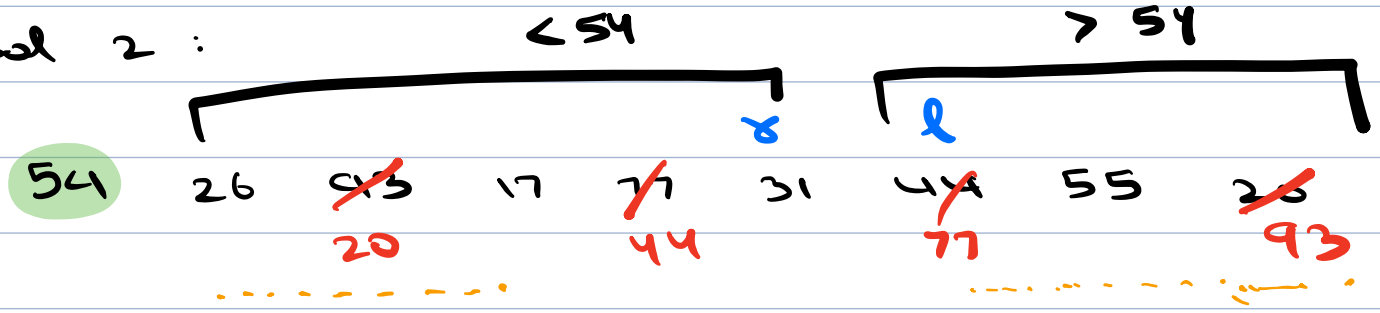
54 26 93 17 77 31 44 55 20

↓

17 20 26 31 44 54 55 77 93

< 54 | > 54

sol 2 :

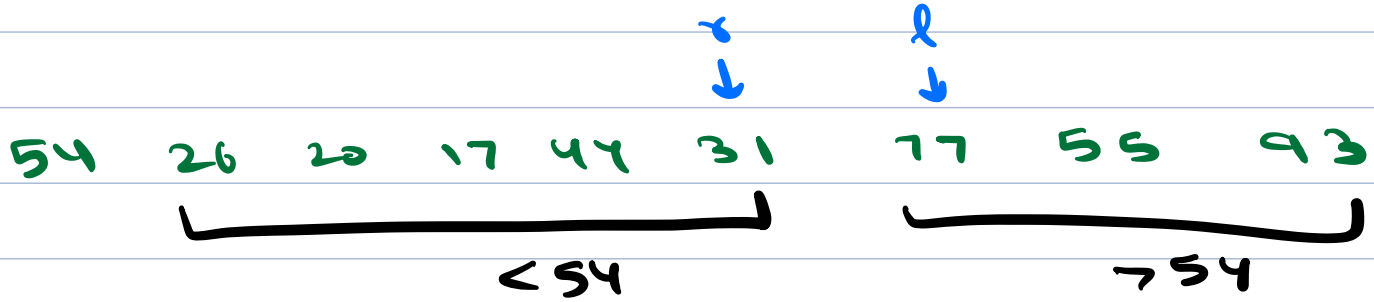


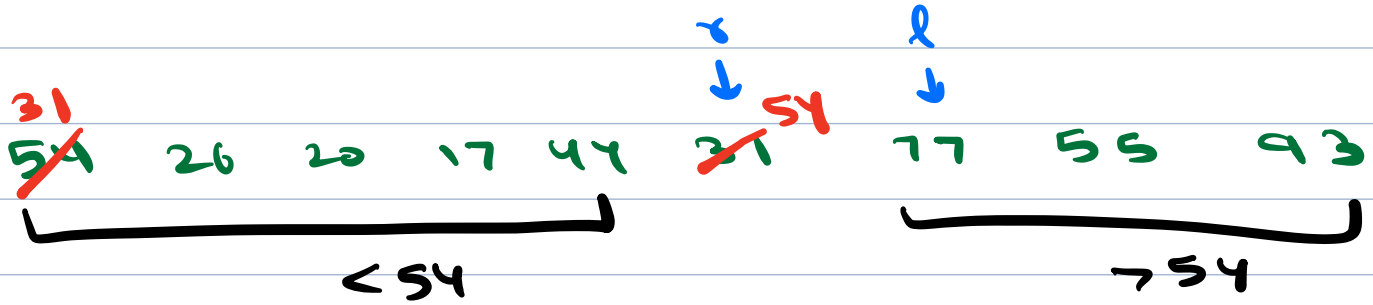
l → smaller

r → bigger

l crossed r

stop





swap (a[pivot], a[r])

r l

54 26 ~~93~~ 20 17 ~~77~~ 44 31 ~~44~~ 77 55 ~~20~~ 93

If l is happy
  l++
else if r is happy
  r--
else
  swap
  l++ r--

l r

54 ~~102~~ 22 100 ~~22~~ 102

↓

100 22 54 102

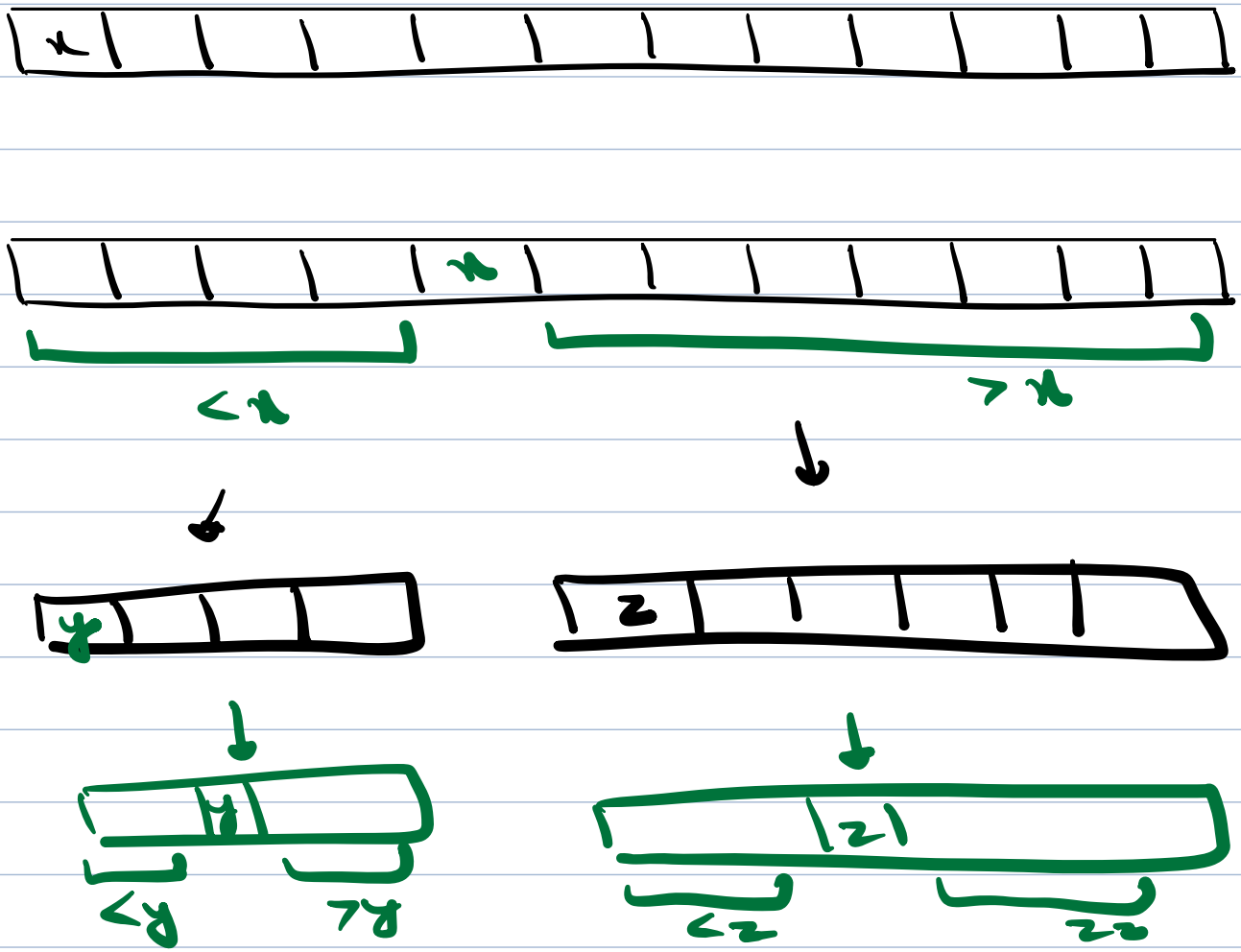
0 N-1

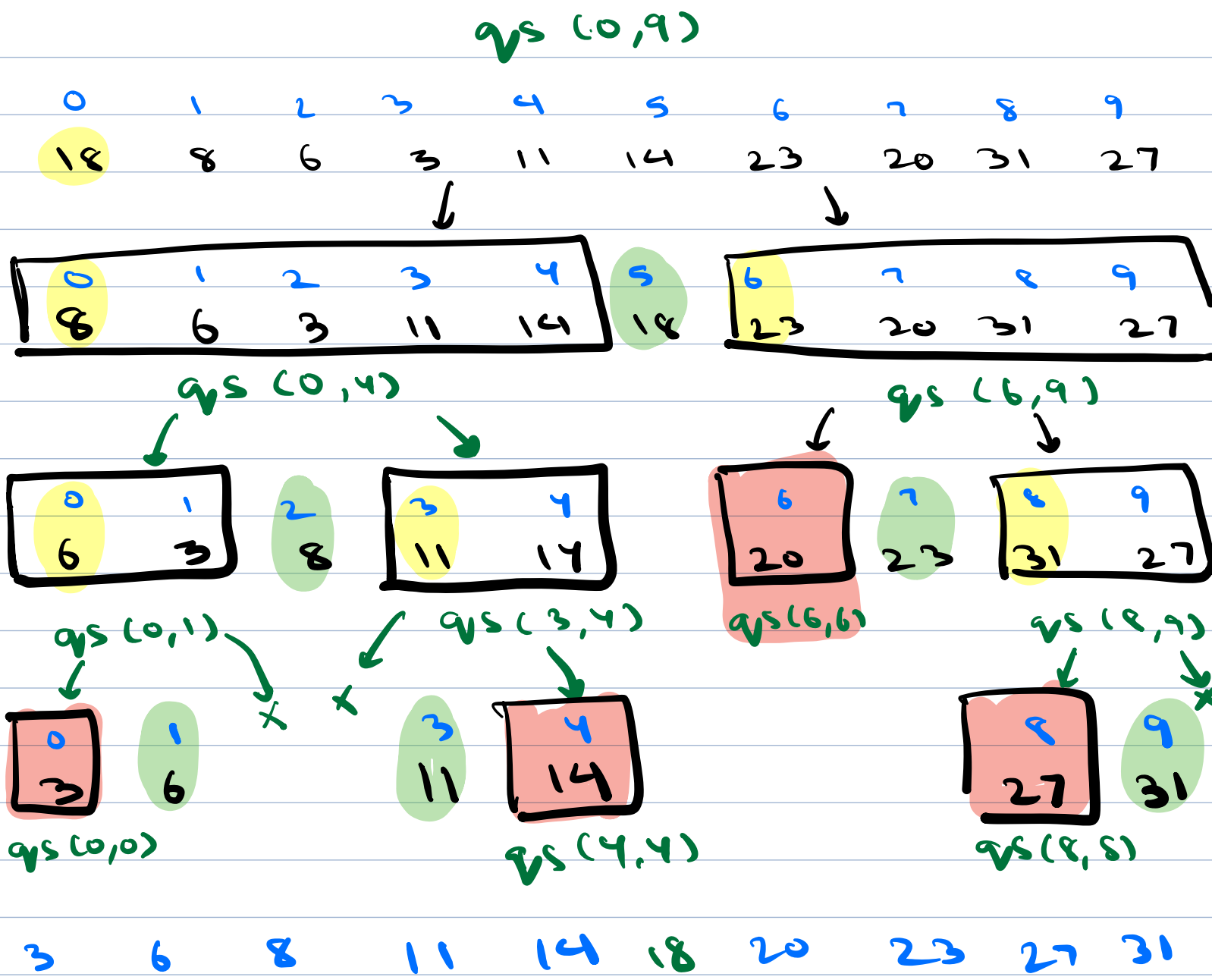
```
int partition(A, first, last) {
    int rnd = Math.random(first, last)
    swap(A[rnd], A[first])

    pivot = A[first]
    l = first + 1
    r = last

    while (l ≤ r) {
        if (A[l] ≤ pivot) {
            l++
        }
        else if (A[r] > pivot) {
            r--
        }
        else {
            swap(A[l], A[r])
            l++  r--
        }
    }

    swap(A[first], A[r])
    return r
}
```

TC : O(N)

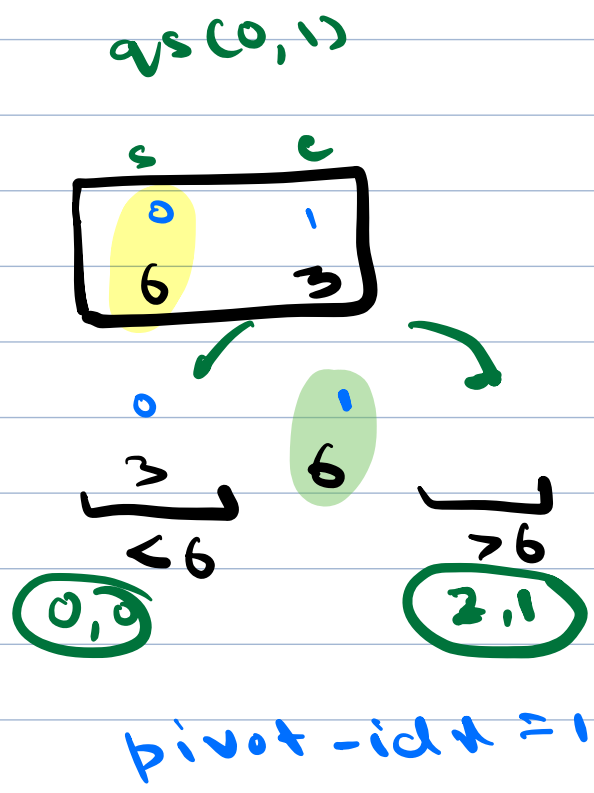
SC : O(1)

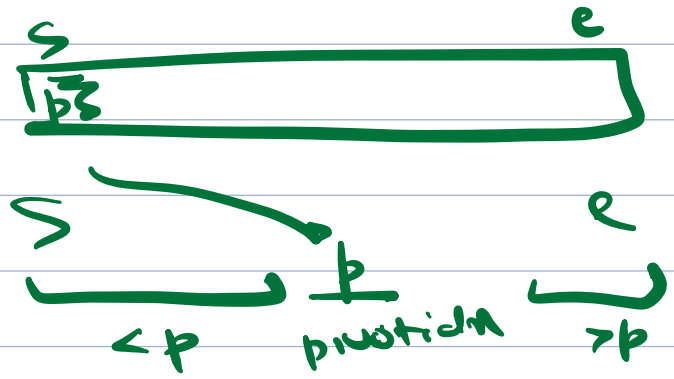
Quick Sort → Divide and conquer Strategy



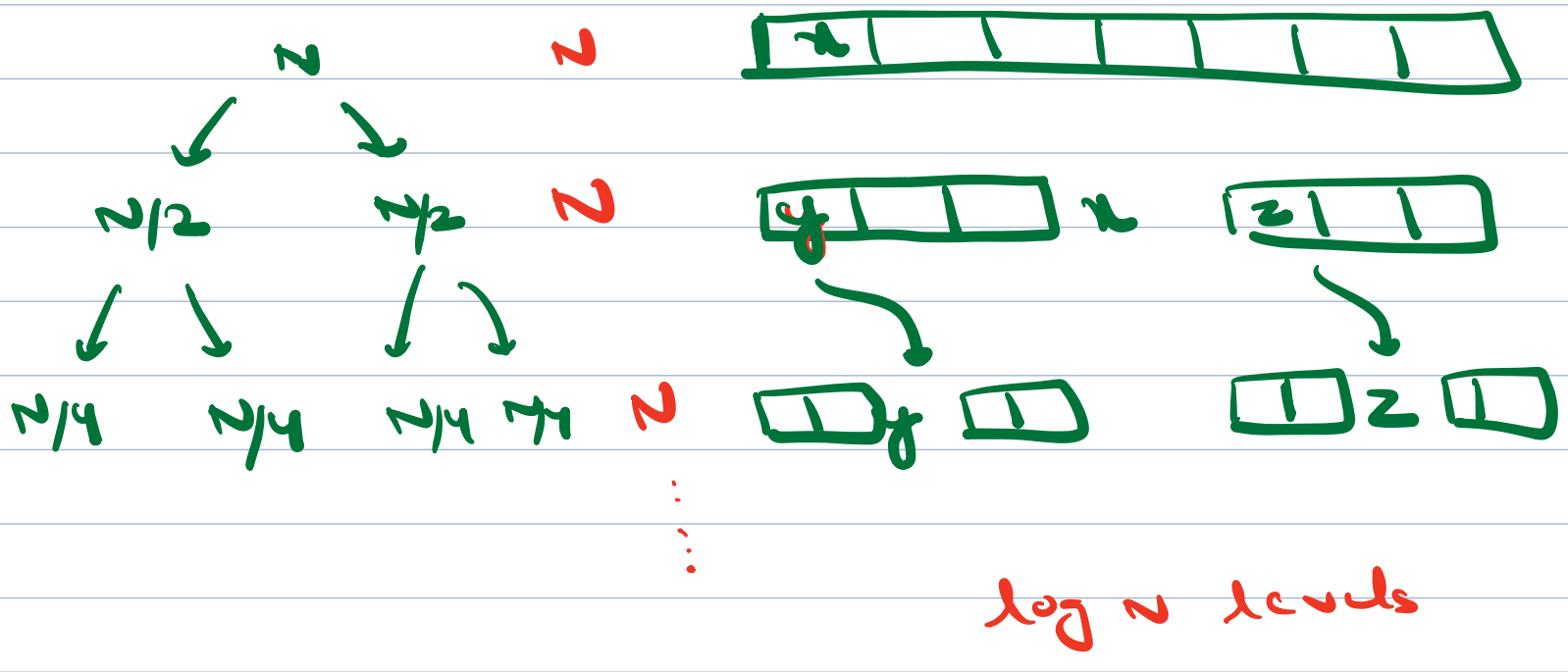



```
// Given A, this fn will sort A from s→e
void quicksort (int A[], int s, int e){
    if (s ≥ e) return

    int pivotidx = partition (A, s, e)
     quicksort (A, s, pivotidx - 1)
     quicksort (A, pivotidx + 1, e)
}
```





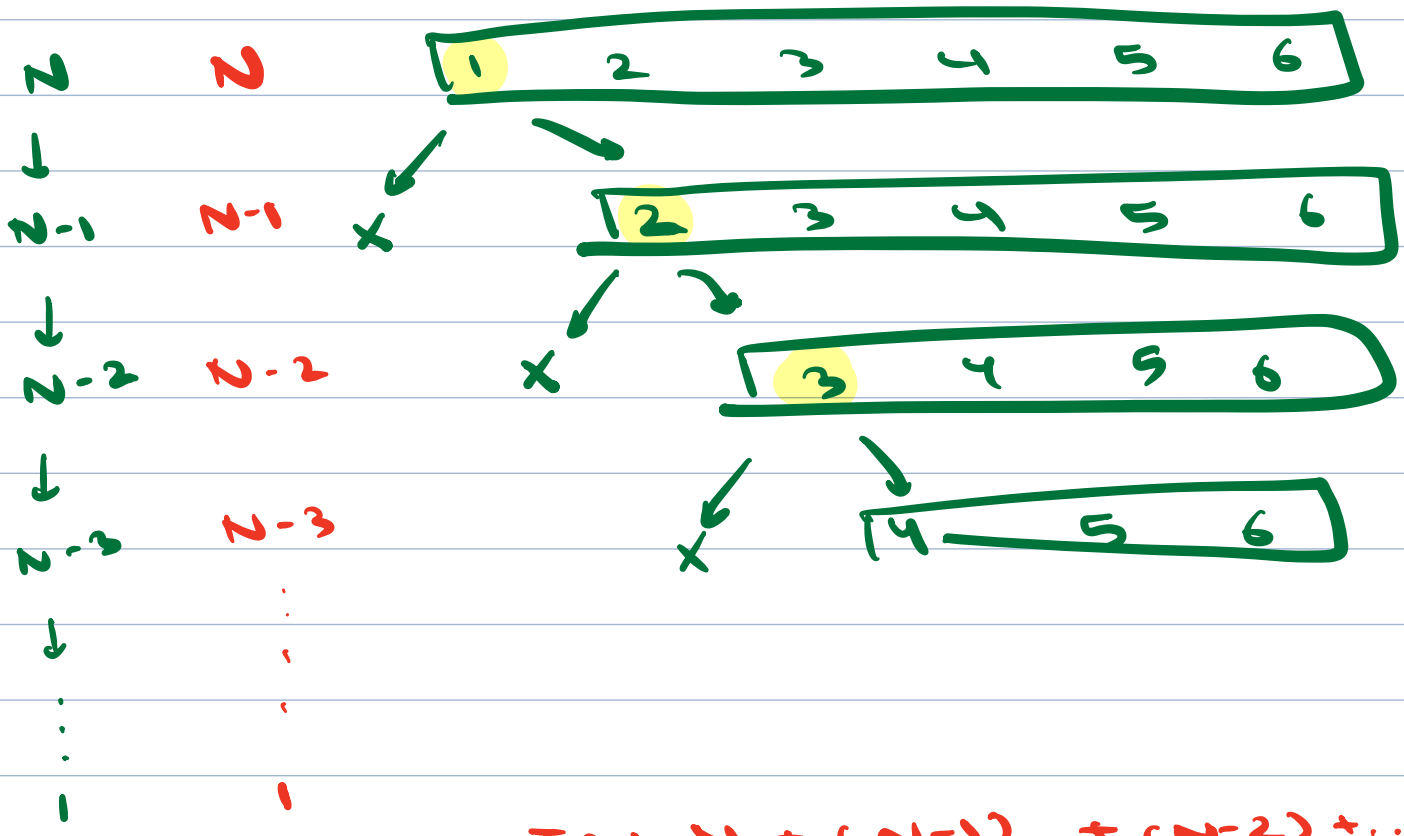
Best Case TC



TC: O(N log N)

SC: O(log N)

Worst case TC (sorted data ↑ or ↓)



TC: $N + (N-1) + (N-2) + \dots 1$

$= \frac{N(N+1)}{2} = O(N^2)$

SC: $O(N)$

N function calls

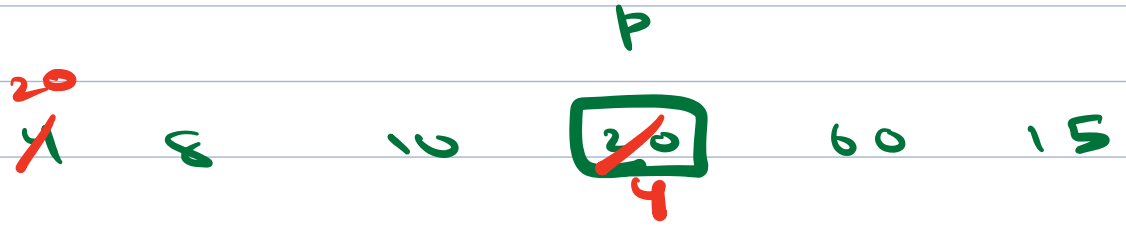
Pivot → Min / Max (worst case)

	Quick sort		Merge sort	
	TC	SC	TC	SC
Best case:	$N \log N$	$\log N$	$N \log N$	$N + \lg N$
Worst case:	N^2	N	$N \log N$	$N + \lg N$

Randomised quick sort →

Rather than always choosing first / last element as pivot, a random element as pivot.

→ First / Mid / Last
→ Median of first, mid, last
→ Randomly choose pivot



Prob of picking min de as pivot = $\frac{1}{N}$

Prob of picking 2nd min de as pivot = $\frac{1}{N-1}$

Prob of picking 3rd min de as pivot = $\frac{1}{N-2}$

⋮

Prob. of always picking min =

$$\frac{1}{N} \times \frac{1}{N-1} \times \frac{1}{N-2} \times \frac{1}{N-3} \times \cdots \frac{1}{1} = \frac{1}{N!}$$

Best / Avg TC → TC : O(N log N)

SC : O(log N)

10:45

C++ → Intro sort (Quick Sort + Heap Sort)

Java → Algo based on Quick Sort

Python → Tim sort (Merge + Insertion sort)

JS → Quick + Insertion sort

1, 3, 5, 7, ⑨, 2, 4, 6, 8

Sorting : arrangement in data in asc/dsc order based on some parameter

Arrays.sort(A) → asc order based on value

Collections.sort(A)

Collections.sort(A, new Comp(lass()))

```
compClass implements Comparator<int>{
    @Override
    int compare(int a, int b){

    }
}
```

Comparator

- In programming, a **comparator** is a function that compares two values and returns a result indicating whether the values are equal, less than, or greater than each other.
- The **comparator** is typically used in sorting algorithms to compare elements in a data structure and arrange them in a specified order.

Comparator is a function that takes **two arguments**.

For languages - **Java, Python, JS, C#, Ruby**, the following logic is followed.

```
1. In sorted form, if first argument should come before second, -ve value is returned.
2. In sorted form, if second argument should come before first, +ve value is returned.
3. If both are same, 0 is returned.
```

For **C++**, following logic is followed.

```
1. In sorted form, if first argument should come before second, true is returned.
2. Otherwise, false is returned.
```

Given an array of size N, sort data in ascending order of count of factors. If count of factors are equal, sort based on magnitude.

A → 9, 3, 10, 6, 4

3 4 9 6 10

2 3 4

(3) 9 → 1, 3, 9
(2) 3 → 1, 3
(4) 10 → 1, 2, 5, 10
(4) 6 → 1, 2, 3, 6
(3) 4 → 1, 2, 4

A → 10, 4, 5, 13, 1

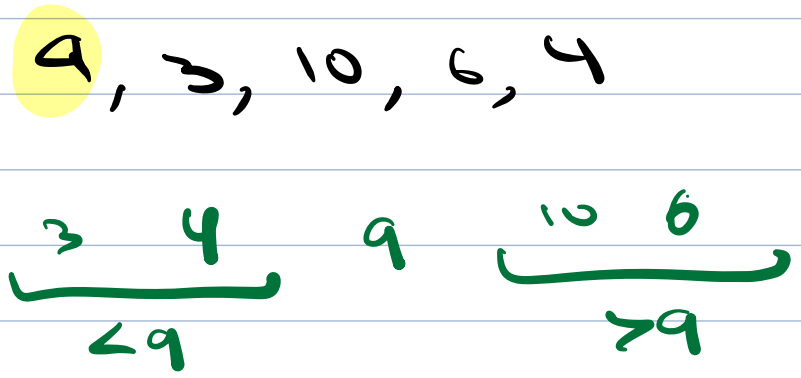
1, 5, 13, 4, 10

10 → 1, 2, 5, 10
4 → 1, 2, 4
5 → 1, 5
1 → 1
13 → 1, 13

```
compClass implements Comparator<int>{
    @Override
    int compare(int a, int b){
        int cnta = countFactors(a)
        int cntb = countFactors(b)
        if(cnta < cntb){
            return -1
        }
        else if(cntb < cnta){
            return 1
        }
        else{
            if(a<b) return -1
            else if(b<a) return 1
            else return 0
        }
    }
}
Solution{
    void main(){
        // A
        Collections.sort(A, new compClass())
    }
}
```

TC: $O(N \log N \times$ TC of compare function)

TC: $O(N \log N \times$ max num in array)




```python
import functools

//please write the code for finding factors by yourself

def compare(v1, v2):
    if(factors(v1) == factors(v2)):
        if(v1<v2):
            return -1;
        if(v2<v1):
            return 1;
        else
            return 0;
    elif (factors(v1)<factors(v2)):
        return -1;
    else
        return 1;


class Solution:
    def solve(self, A):
        A = sorted(A, key = functools.cmp_to_key(compare))
        return A
```

Python

```cpp
bool compare(int val1, int val2)
{
    int cnt_x = count_factors(x);
    int cnt_y = count_factors(y);

    if(factors(val1) == factors(val2))
    {
        if(val1<val2)
        {
            return true;
        }
        return false;
    }
    else if(factors(val1)<factors(val2))
    {
        return true;
    }
    return false;
}

vector<int> solve(vector<int> A) {
    sort(A.begin() , A.end() , compare);
    return A;
}
```

C++

Given a list of non-negative integers nums, arrange them such that they form the largest number and return it.

Since the result may be very large, so you need to return a string instead of an integer.

Ex. [10, 2] → "102"

[2, 10] → "210"

ans = "210"

Ex [3, 30] → "330"

[30, 3] → "303"

ans = "330"

Ex. [3, 30, 34, 5, 9]

ans → 9 5 34 3 30 ✓

[9, 5, 34, 3, 30]

Idea: Largest no. → sort arr in desc order

[34, 30, 9, 5, 3]

↓

"34309 53" ✗

Quiz: [10, 5, 2, 8, 200]

↓

[8, 5, 2, 200, 10] → "852200 10"

x, y

if (xy > yx)

x should come first

else

y should come first

x	y
2	200

xy = 2200 > yx = 2002

2 should come first

```
compclass implements Comparator<int>{
    @Override
    int compare(int a, int b){
        String a' = String.valueOf(a)
        String b' = String.valueOf(b)

        String ab = a' + b'
        String ba = b' + a'
        if (ab.compareTo(ba) > 0)
            return -1
        else if (ab.compareTo(ba) < 0)
            return 1
        else return 0
    }
}
```

TC : O(n log n) + TC of your comparator fn

```
public class Solution {
    public String largestNumber(ArrayList<Integer> A) {
        Collections.sort(A, new Comparator<Integer>() {
            public int compare(Integer a, Integer b) {
                String XY = String.valueOf(a) + String.valueOf(b);
                String YX = String.valueOf(b) + String.valueOf(a);
                return XY.compareTo(YX) > 0 ? -1 : 1;
            }
        });
        StringBuilder ans = new StringBuilder();
        for (int x : A) {
            ans.append(String.valueOf(x));
        }
        if (ans.charAt(0) == '0')
            return "0";
        return ans.toString();
    }
}
```

Java

s1.compareTo(s2)

s1 > s2 → +ve

s1 < s2 → -ve

s1 == s2 → 0

To Do

Recording : Inversion Count

Solve : Max Chunks

$A + B = (A \wedge B) + 2 * (A \& B)$

$x = x + 0$

$A \& B = 0$

$2^{s} *$ ——

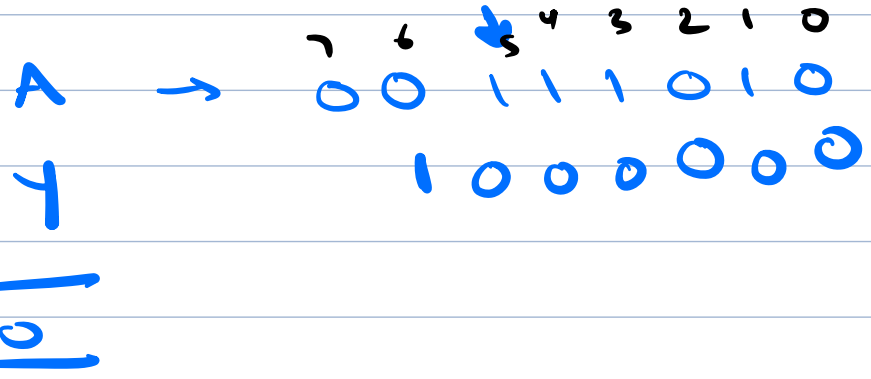
```
          7 6 5 4 3 2 1 0
A  →      0 0 1 1 1 0 1 0
& x       0 0 0 0 0 1 0 1
  ———————————————————————
  ———————————————————————
```

```
          →   ↓
A  →      0 0 0 1 0 1 0
& x  →    0 0 0 0 1 0 1
  ———————————————————————
  ———————————————————————
```

```
for (i = 31; i >= 0; i--) {
    if ( setBit (A, i)
        break
}
```

```
int x = 0
while (i >= 0) {
    if (unsetBit(A, i)
        x = x | (1<<i)
}
```

y > A



Last set bit → x

y = 1 << (x+1)