Pivot Partition

Quick Sort

Comparator Problems

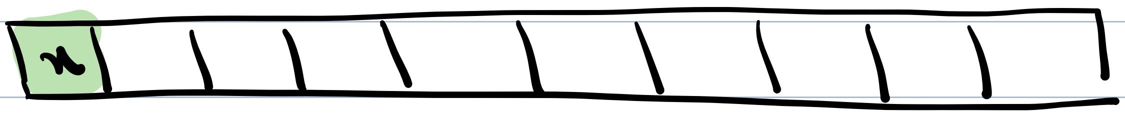
Contest 1 → Monday (25 Nov)

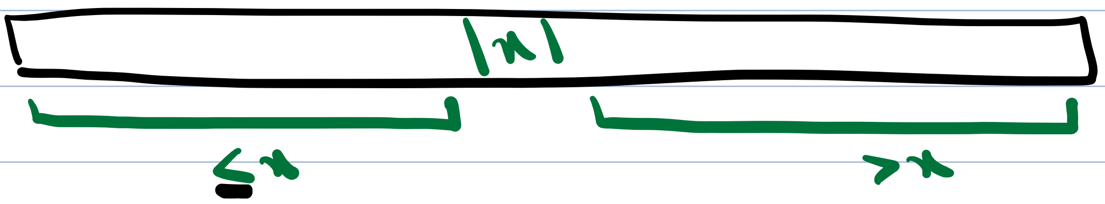
Arrays, Bit Manipulation, Recursion, Math, Hashing & Sorting

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





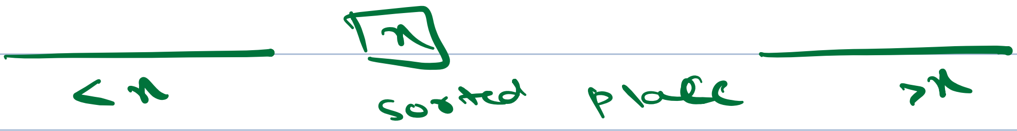
arr[] : 54, 26, 93, 17, 77, 31, 44, 55, 20

Ans : $\underbrace{26, 17, 31, 44, 20}_{<54}$, 54, $\underbrace{55, 77, 93}_{>54}$

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

Ans : $\underbrace{7, 8, 5}_{<10}$ 10 $\underbrace{13, 25, 20, 23}_{>10}$

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

↓

$\underbrace{17 \quad 20 \quad 26 \quad 31 \quad 44}_{<54} \quad 54 \quad 55 \quad \underbrace{77 \quad 93}_{>54}$

sol 2 :

$\overbrace{26 \quad \overset{20}{\cancel{93}} \quad 17 \quad \overset{44}{\cancel{77}} \quad \overset{r}{31}}^{<54}$ $\overbrace{\overset{l}{\overset{77}{\cancel{44}}} \quad 55 \quad \overset{93}{\cancel{20}}}^{>54}$

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

l → smaller
r → bigger

l crosses r
stop

$54 \quad \underbrace{26 \quad 20 \quad 17 \quad 44 \quad \overset{r}{\overset{\downarrow}{31}}}_{<54} \quad \underbrace{\overset{l}{\overset{\downarrow}{77}} \quad 55 \quad 93}_{>54}$

$\underbrace{\overset{31}{\cancel{54}} \quad 26 \quad 20 \quad 17 \quad 44}_{<54} \quad \overset{r}{\overset{\downarrow}{\overset{54}{\cancel{31}}}} \quad \underbrace{\overset{l}{\overset{\downarrow}{77}} \quad 55 \quad 93}_{>54}$

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

r l

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

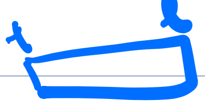
```
If l is happy
    l++
else if r is happy
    r--
else
    swap
    l++    r--
```

l r

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

↓

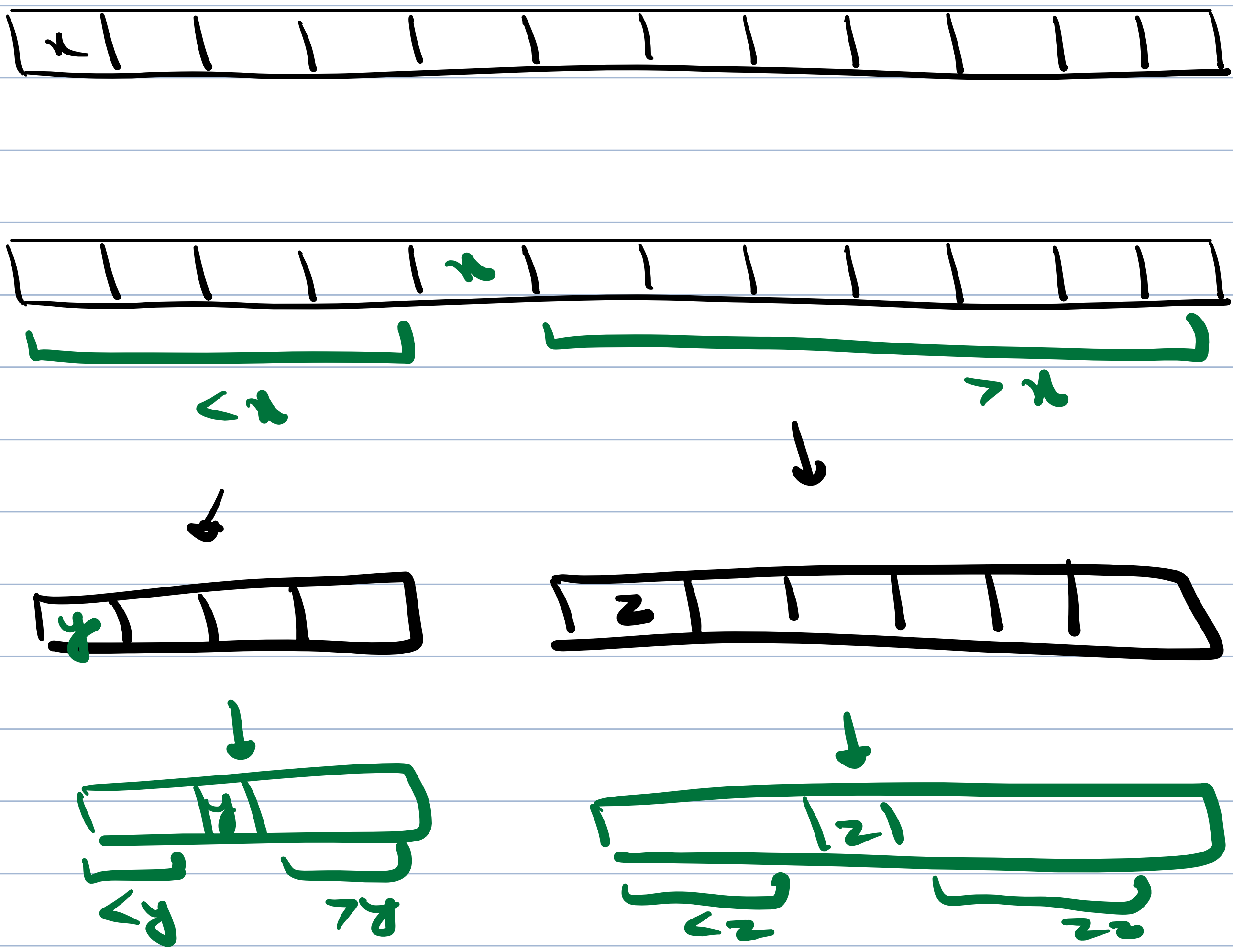
100 22 54 102

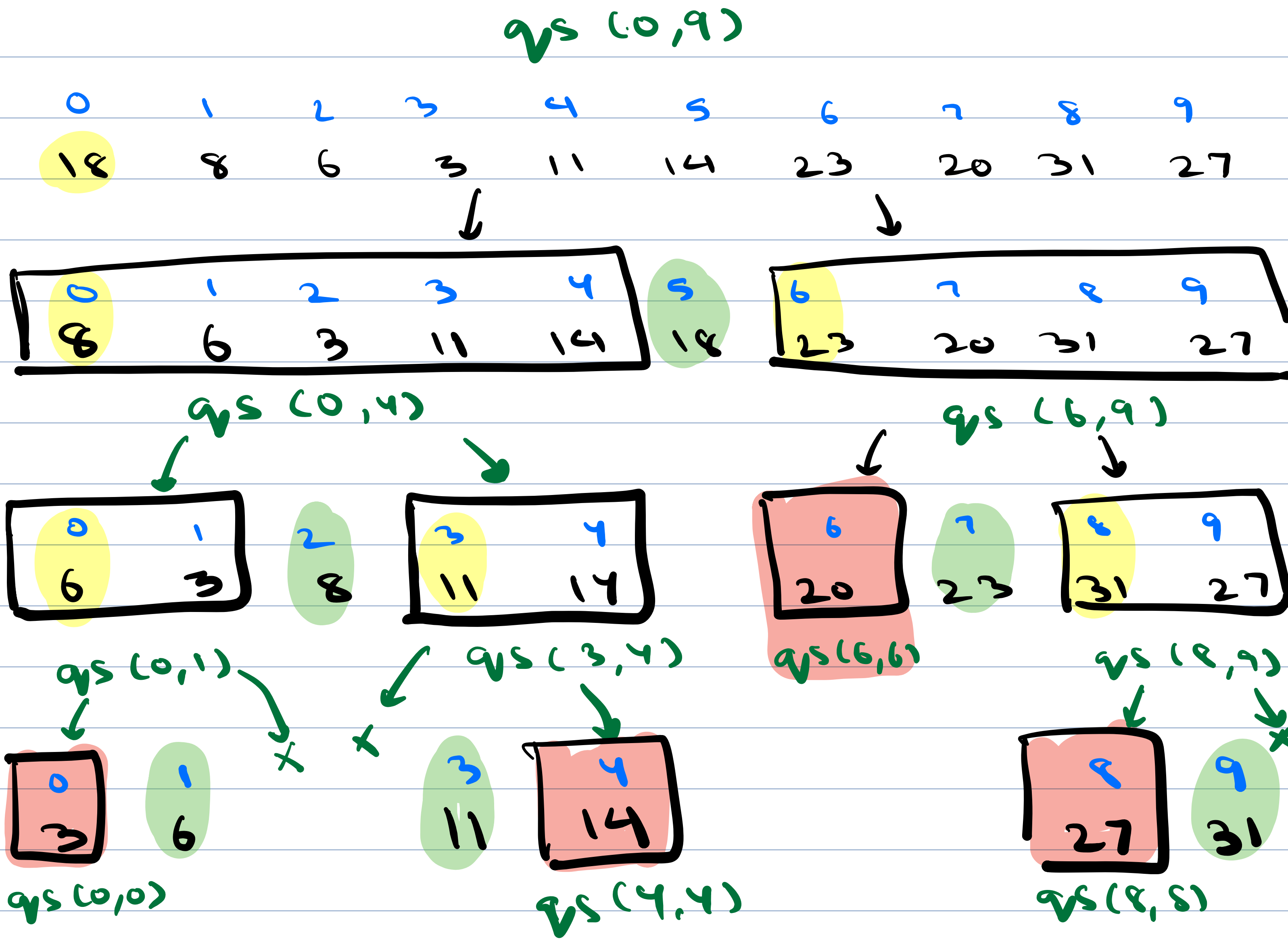


```
                                   0         N-1
int partition(A, first, last) {
        pivot = A[first]
            l = first + 1
            r = last

        while (l ≤ r) {                              TC : O(N)
                                                     SC : O(1)
            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
}
```

Quick sort → Divide and conquer strategy



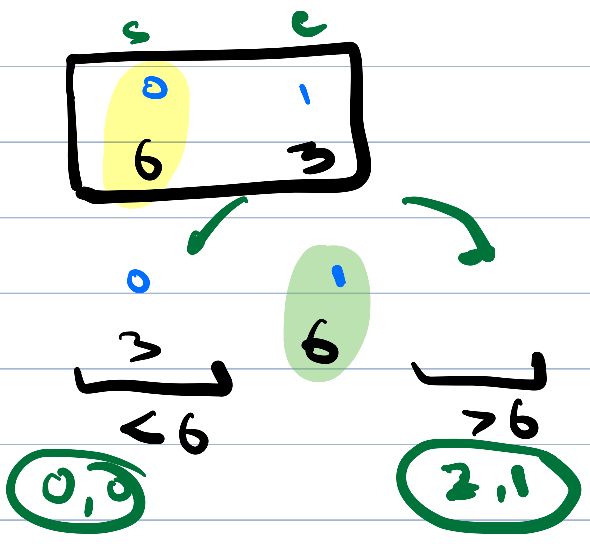


3 6 8 11 14 18 20 23 27 31

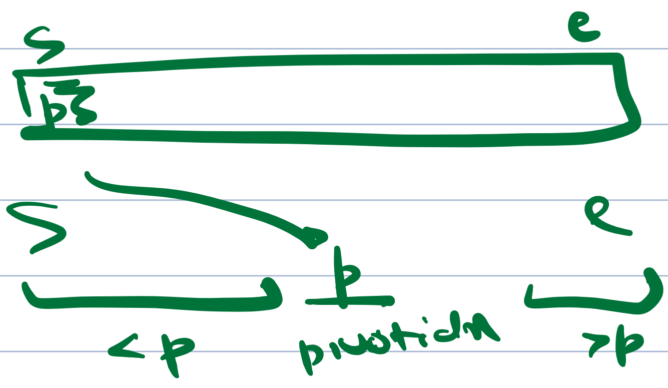
```
// Given A, this fn will sort A from s->e
void quicksort (int A[], int s, int e){
    if (s ≥ e)    return

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

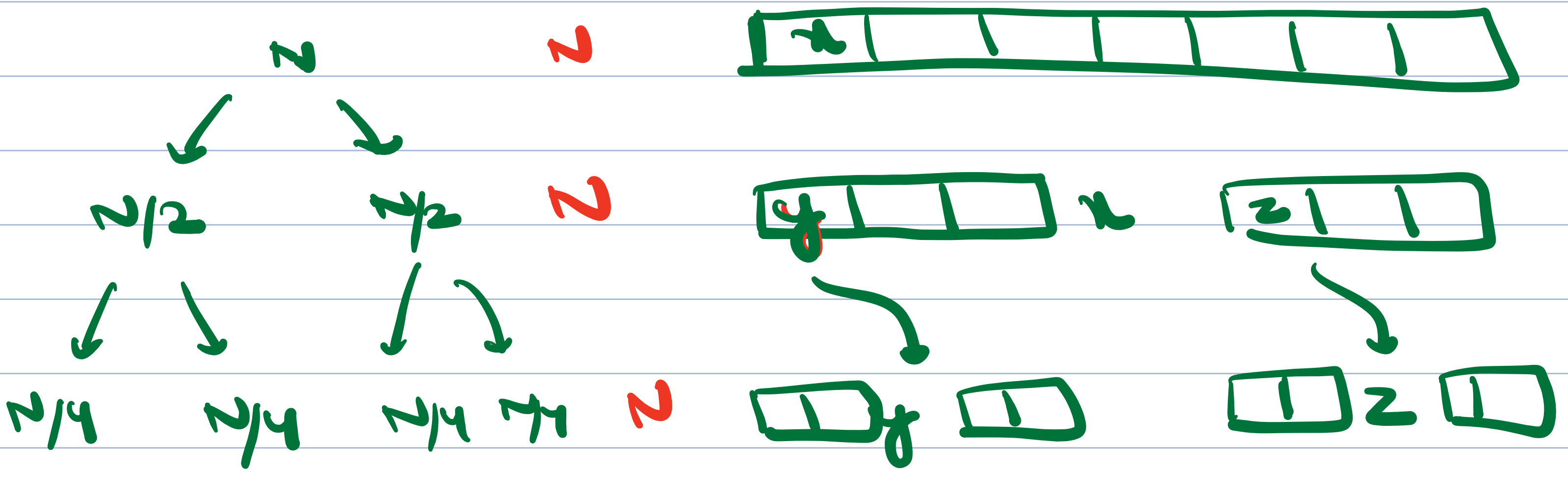
qs(0,1)



pivot-idx = 1



Best case TC

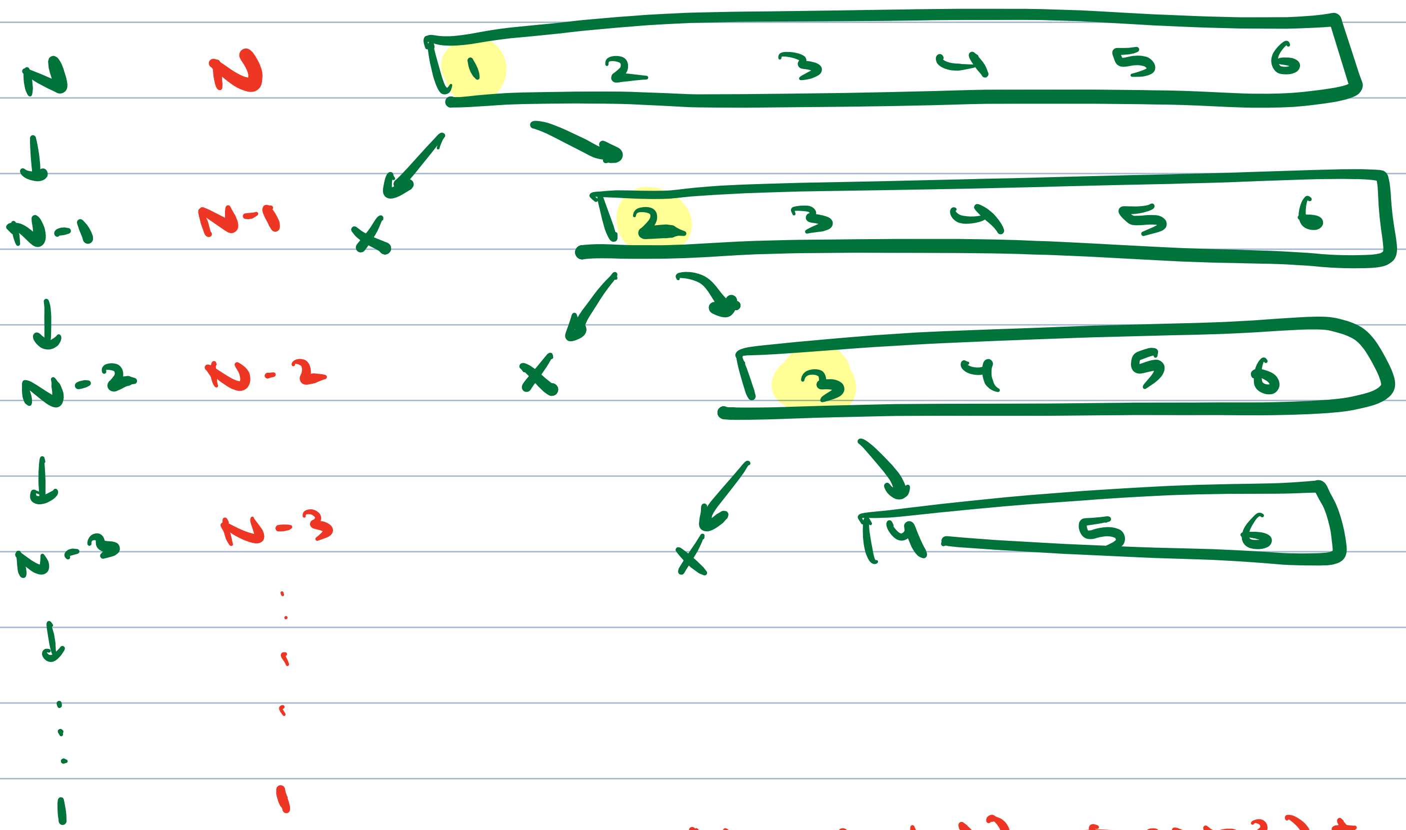


log N levels

TC : O(N log N)

SC : O(log N)

Worst case TC (sorted data ↑ or ↓)



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

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

$SC: O(N)$

N function calls

Pivot → min / Max (worst case)

	Quick sort TC	Quick sort SC	Merge sort TC	Merge sort SC
Best case:	N log N	log N	N log N	N+logN
Worst case:	N^2	N	N log N	N+logN

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

$$\overset{20}{\cancel{4}} \qquad 8 \qquad 10 \qquad \overset{P}{\underset{4}{\boxed{\cancel{20}}}} \qquad 60 \qquad 15$$

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

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

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

$\vdots$

Prob. of always picking min =

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

Avg TC $\rightarrow$ TC : O(N log N)

SC : O(log N)

$\boxed{10:40}$

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

A → 10, 4, 5, 13, 1

```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;
}
```

```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
```

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

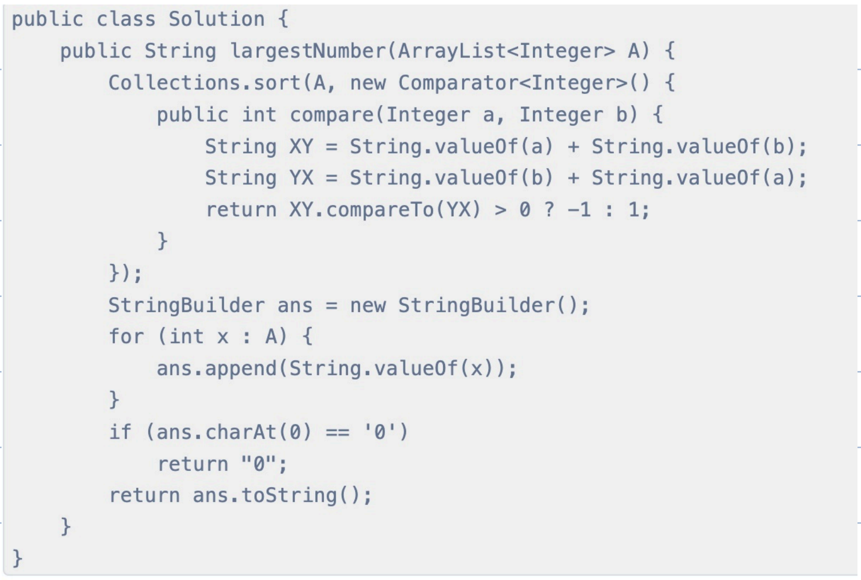
Since the result may be very large, so you need to return a string instead of an integer.

Ex. [10, 2]

Ex [3, 30]

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

TC : $O(n \log n) \times$ 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();
    }
}
```