Pivot Partition

Quick Sort

Comparator Problems

Contest 3

① Reattempt 1

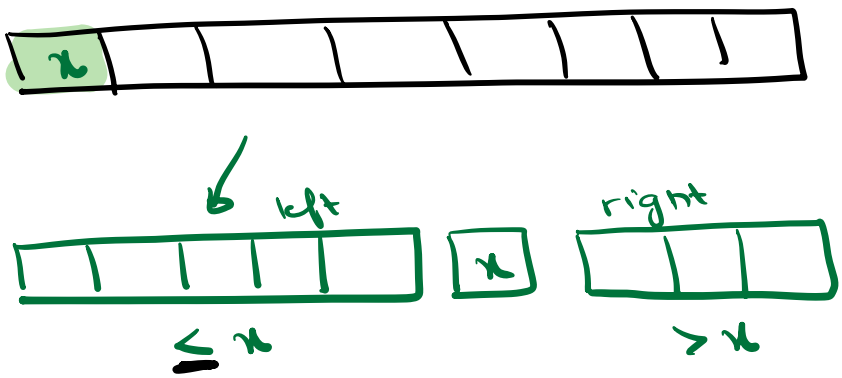
Tue to Thurs

② Advanced Language Concept → wed

Assignment & HW

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

44 31 17 26 20 54 93 77 55
< 54 > 54

17 20 26 31 44 54 55 77 93

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

< x | x sorted place | > x

Soln: sort array

54 26 93 17 77 31 44 55 20

17 20 26 31 44 | 54 | 55 77 93

< 54 ≥ 54

↳ O(n log n)

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

20 44 77 93

r l

l → smaller

r ← bigger

if (l > r) break

swap with 3'

swap with r

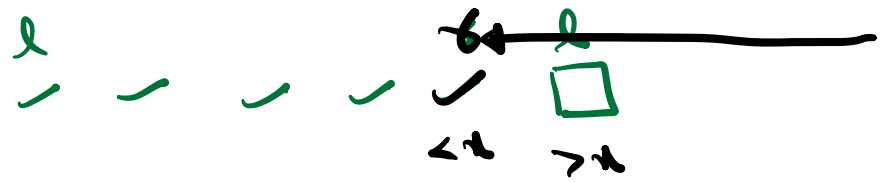
31 26 20 17 44 | 54 | 77 55 93

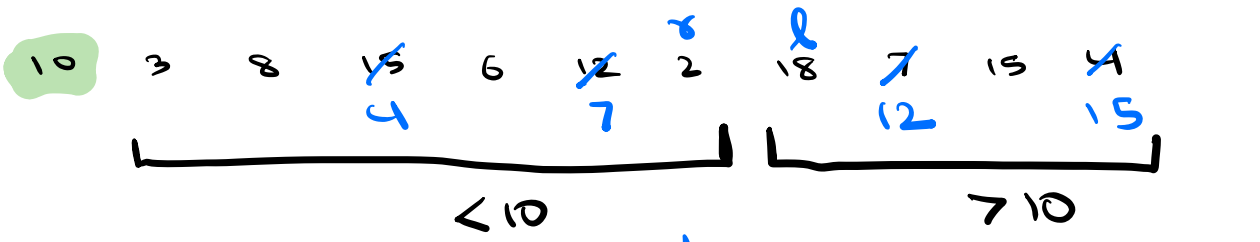
< 54 > 54

r l

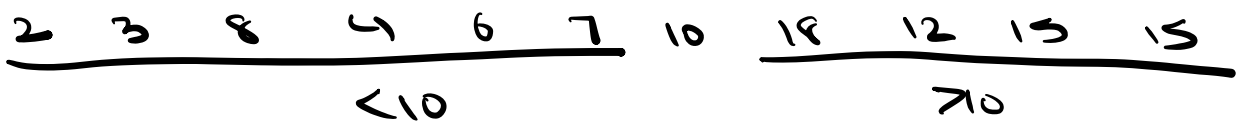
10 | 8 6 5 2 20 30 40

< 10 > 10





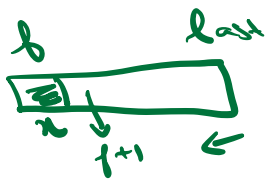
Swap pivot with r
10 with 2



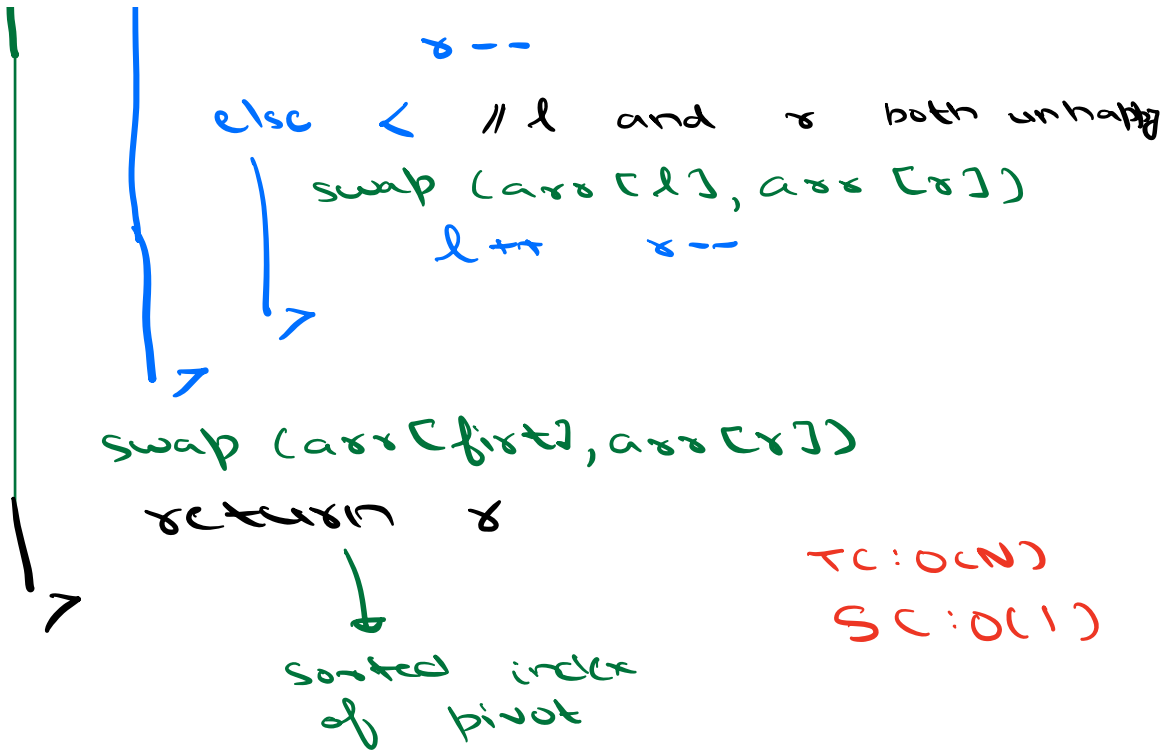
- We begin by incrementing leftmark until we locate a value that is greater than the pivot value.
- We then decrement rightmark until we find a value that is less than the pivot value.
- At this point we have discovered two items that are out of place with respect to the eventual split point.

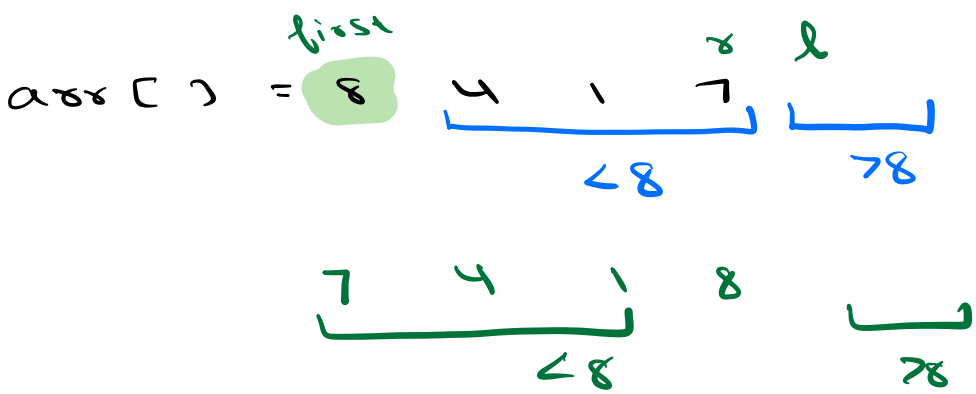
swap l and r

```
                                  0        n-1
partition (arr, first, last) <
    pivot = arr[first]
    l = first+1
    r = last
    while (l <= r) <
        if (arr[l] <= pivot)
                l++
        else if (arr[r] > pivot)
```

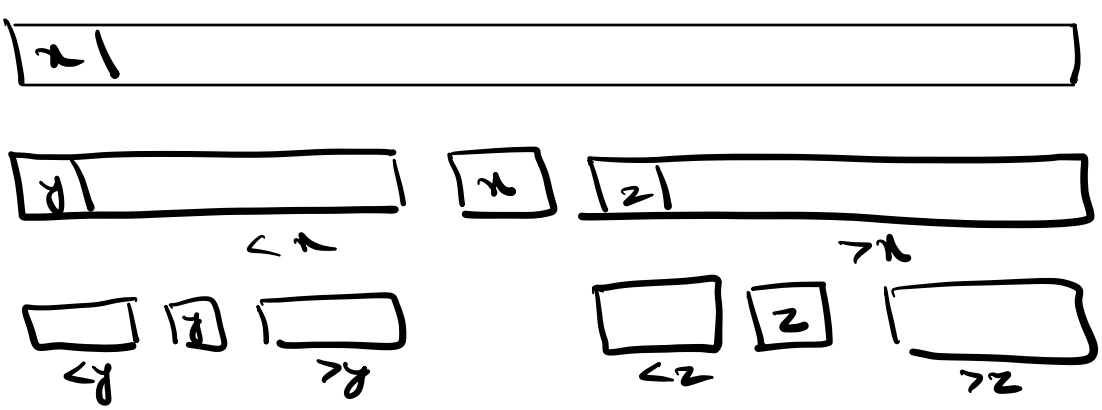


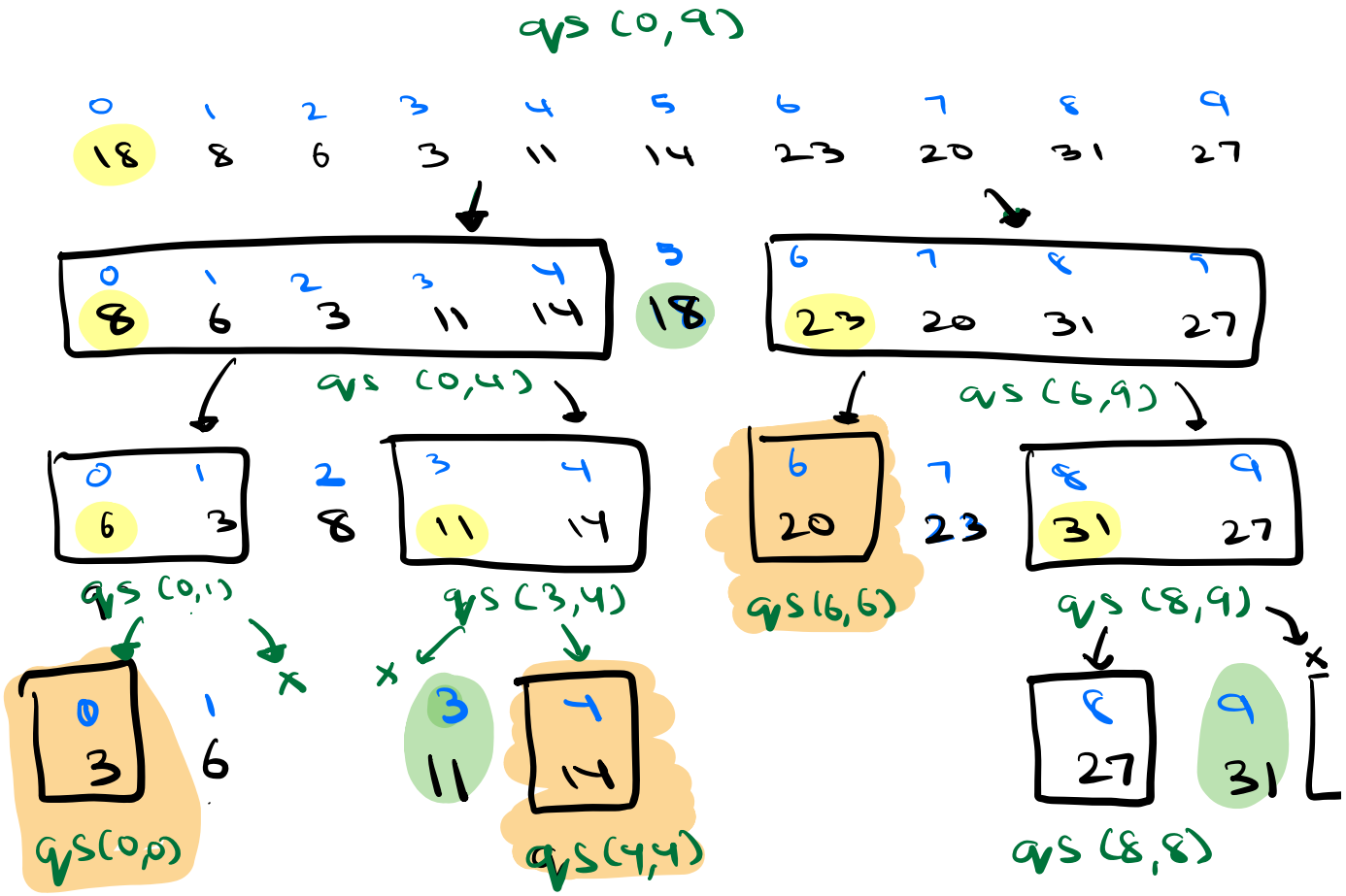
l is happy



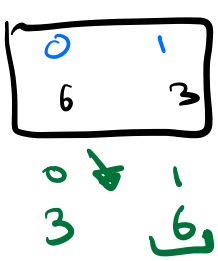


Quick sort → Divide and conquer strategy

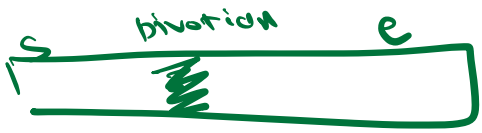


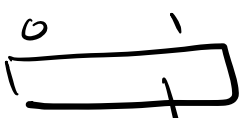


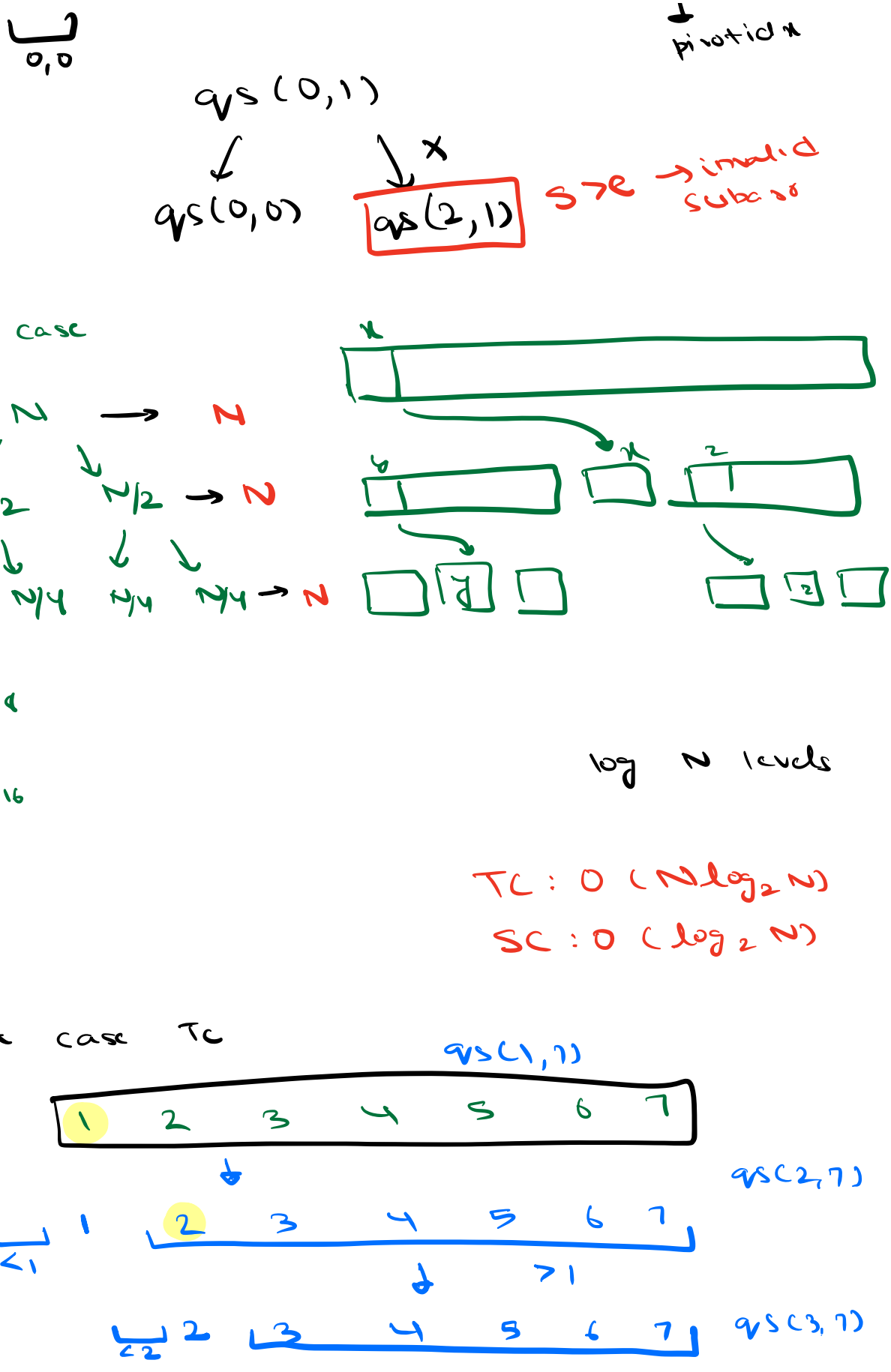

```
// given s and e, it will sort ar[s] → ar[e]
quicksort (arr, start, end) {
    if (start >= end)    return
    int pivotidx = partition (arr, start, end)
            quicksort (arr, start, pivotidx-1)
            quicksort (arr, pivoticlx+1, end)
}
```

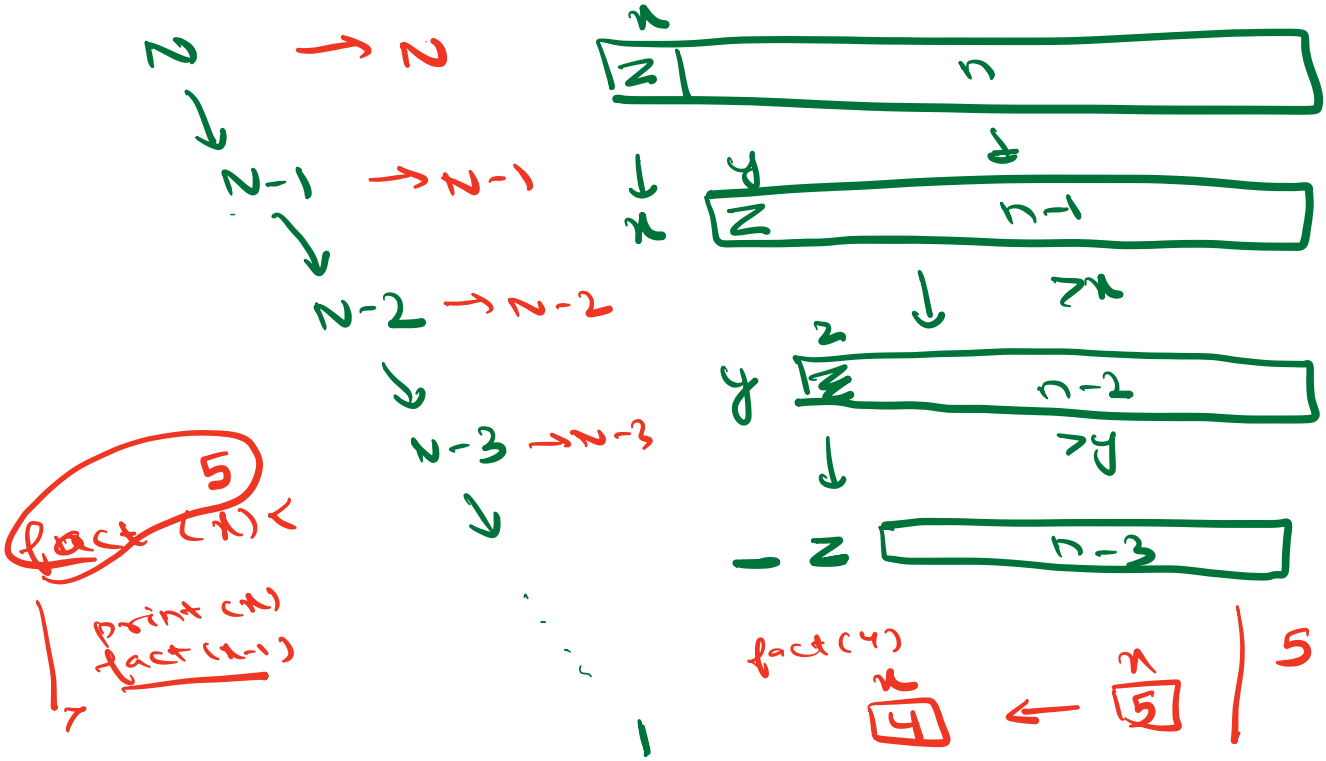


s 1
0 e






0,0
pivotidx
qs(0,1)
qs(0,0)
x
qs(2,1)
size → invalid subarr
Best case
N → N
N/2 N/2 → N
N/4 N/4 N/4 N/4 → N
N/8 N/8
N/16 N/16
1
log N levels
TC: O(Nlog2N)
SC: O(log2N)
Worst case TC
qs(1,7)
1 2 3 4 5 6 7
qs(2,7)
<1
>1
qs(3,7)



sum of first N natural nos.

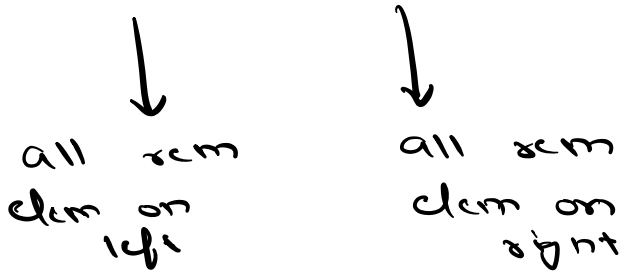
$$= \frac{N(N+1)}{2}$$

TC : $O(N^2)$

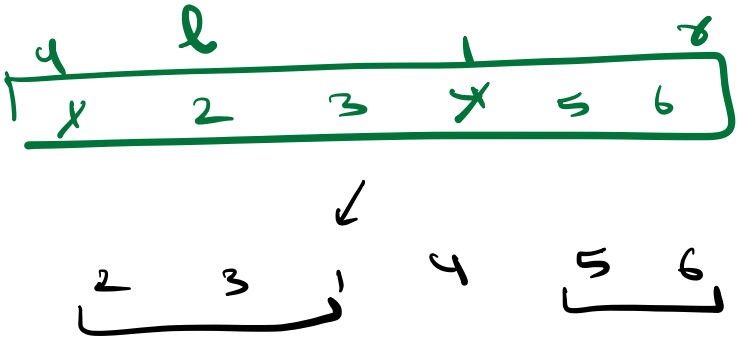
SC : $O(N)$

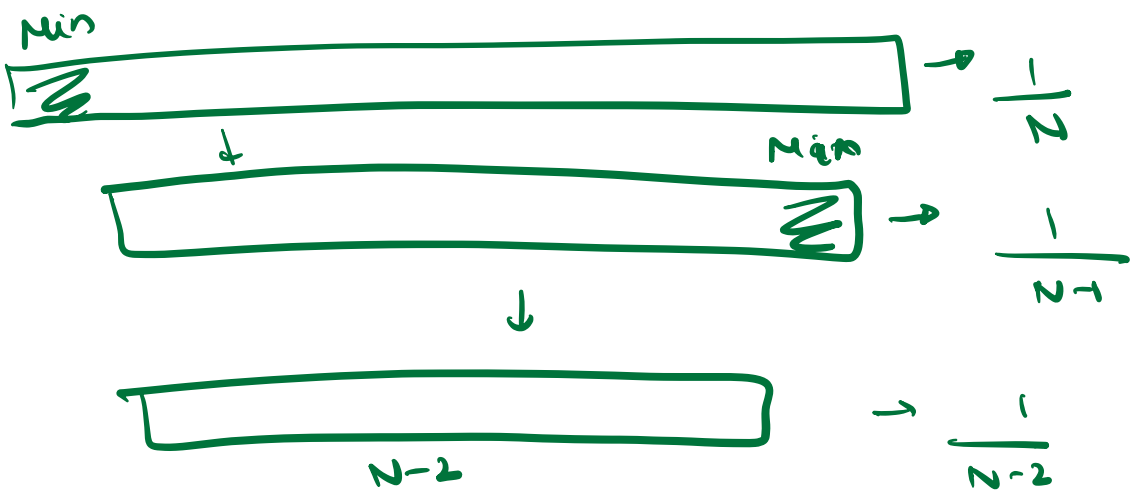
Best case TC	Worst case TC
TC : $O(n \log n)$	TC : $O(n^2)$
SC : $O(\log n)$	SC : $O(n)$

If we pick max or min element as pivot



	Merge sort	Quick sort
TC :	N log N	N log N → N^2
SC :	log N + N	log N → N





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

Randomized quick sort →

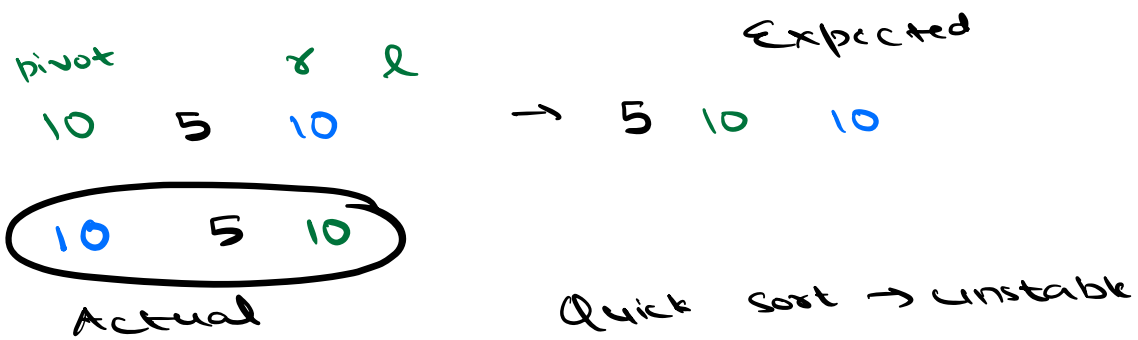
Avg TC : O(n log n)

SC : O(log n)

The randomised quicksort is a technique where we randomly pick the pivot element, not necessarily the first and last.

There is a random function available in all the languages, to which we can pass Array and get random index. Now, we can swap random index element with first element and execute our algorithm as it is.

11:03



Comparator

6 , 2 , 4

Arrays. sort (a, b)

fn where you provide custom sorting logic

- 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**.

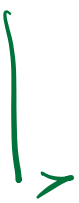
For languages - **Java, Python, JS, C#, Ruby**, the following logic is followed.

```
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.
```

fn (x, y) <



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

↓ ↓ ↓ ↓ ↓

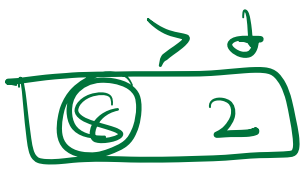
3 2 4 4 3

Sorted A → 3, 4, 9, 6, 10

Input → ArrayList<Integer> A

Collections.sort(A, new Comparator<Integer>(){

@Override
public int comp(Integer a, Integer b){
if(factors(a) < factors(b))
return -1
else if(factors(a) > factors(b))
return 1
}else{
if(a<b)
return -1
else if(a>b)
return 1
else
return 0
}
}



2 8

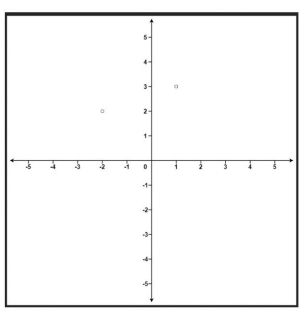
```
import functools

//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
```

```
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;
}
```

TC : O(n log n) x √max dc

Given an array of points where points[i] = [xi, yi] represents a point on the X-Y plane and an integer k, return the B closest points to the origin (0, 0).

The distance between two points on the X-Y plane is the Euclidean distance (i.e., $\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$).



(x_1, y_1) (x_2, y_2)

$$dist = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$$ ✓

$(0,0)$ $(2,3)$

$$dist = \sqrt{(2-0)^2 + (3-0)^2} = \sqrt{13}$$

$(0,0) \longrightarrow (a,b)$

$$dist = \sqrt{a^2 + b^2}$$

of (a,b) from origin

points = [[1,3], [-2,2]] B = 1

$\sqrt{1^2+3^2}$ → $\sqrt{10}$

$\sqrt{(-2)^2+2^2}$ → $\sqrt{8}$

O/P → [-2,2]

points = [[3,3], [5,-1], [-2,4]] B = 2

$\sqrt{3^2+3^2}$ → $\sqrt{18}$

$\sqrt{5^2+(-1)^2}$ → $\sqrt{26}$

$\sqrt{(-2)^2+4^2}$ → $\sqrt{20}$

O/P → [3,3] [-2,4]

Sort points based on distance from origin

```
(Point p1, Point p2) {
    int d1 = sqrt((p1.x)^2 + (p1.y)^2)
    int d2 = sqrt((p2.x)^2 + (p2.y)^2)
    if (d1 < d2)
        return -1
    else if (d1 > d2)
        return 1
    else
        return 0
}
```

```
class Point {
    int x
    int y
}
```

Arr of points

[<2,2> <3,4>]

→ return d1-d2

TC : O(n log n) x O(1)

```
if (d1 < d2)
    return true
else
    return false
```

C++

0 / +ve

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.

e.g. → [10, 2] → 102

[2, 10] → 210 ✓

ans = 210

e.g. [3, 30, 34, 5, 9]

[9, 5, 34, 3, 30] → 9534330 ✓

34, 30, 9, 5, 3 → 3430953 ✗

[3, 30] → 330 ✓

[30, 3] → 303

3 come first

a, b → ab
→ ba

if ab > ba
a should come 1st

else
b should come 1st

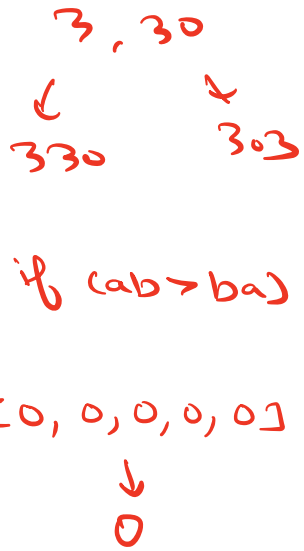
[3, 30, 34, 5, 9]

↓

[9, 5, 34, 3, 30]

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

```java
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();
    }
}
```



Doubts

[2, 4, 3, 2]

2, 4, 3 → 9
2, 4, 2 → 8
2, 3, 2 → 7
4, 3, 2 → 9

Triplets = $\frac{N(N-1)(N-2)}{6}$

$N = 5 \times 10^2$

Triplets = $10^2 \times 10^2 \times 10^2 = 10^6$

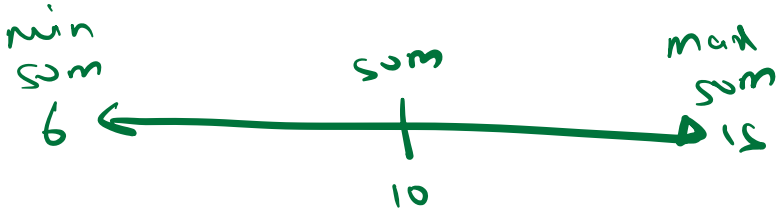
int 3×10^6

$3 \times 10^6 \times 4$

$10^5 - 10^6$

A = 1, 5, 7, 3, 2 ⇌ B = 9th

B^th smallest triple



3rd smallest sum

a → 4
b → 5
c → 6
d → 9
e → 10