NO. _____

Title:

Sorting Algorithms

Enjoy the most efficient handwriting experience! Taking notes on the go, whether for inspiration, ideas, knowledge learning, business insights, or even sketches...

NOTE IN

Creative notes

By reading we enrich the mind;
by writing we polish it.

Sorting Algorithms:

$$arr = \{3, 1, 5, 4, 2\} \rightarrow \text{unsorted array}$$

★ Bubble Sort:

Bubble sort is an comparision sort method
we need to sort this array step by step

- In bubble sort we compare adjacent element in each step

3 1 5 4 2

compare

3 > 1 ?

if true swap

1 3 5 4 2

compare

3 > 5 ?

False → move compare to next element

1 3 5 4 2

compare

5 > 4 ?

true → swap

1 3 4 5 2

compare

5 > 2

true

1 3 4 2 5

step 1: pass 1 finished

Why ?? with the first pass Largest element is at the end

1, 3, 4, 2, 5

F

1 3 4 2 5

F

1 3 4 2 5

T → swap

1 3 2 4 5

F

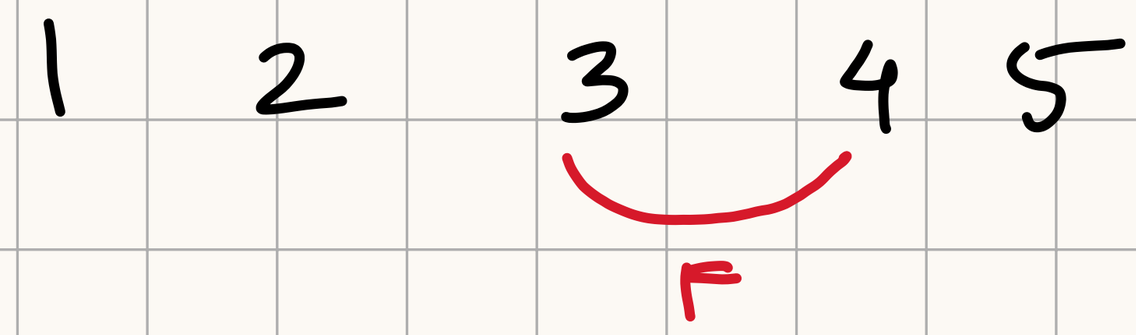
In pass 2 the second largest element comes into the place.

1 3 2 4 5

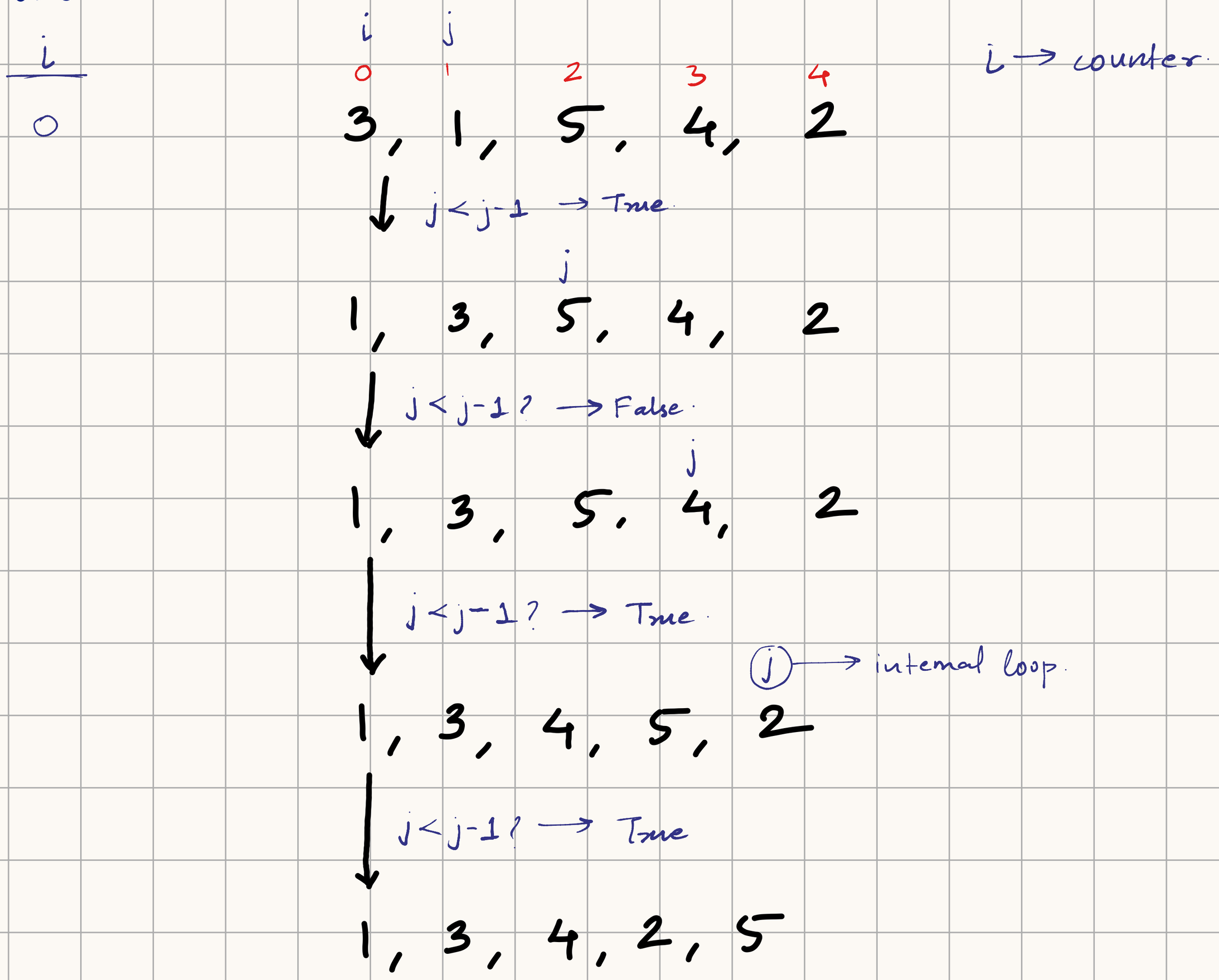
F

1 3 2 4 5

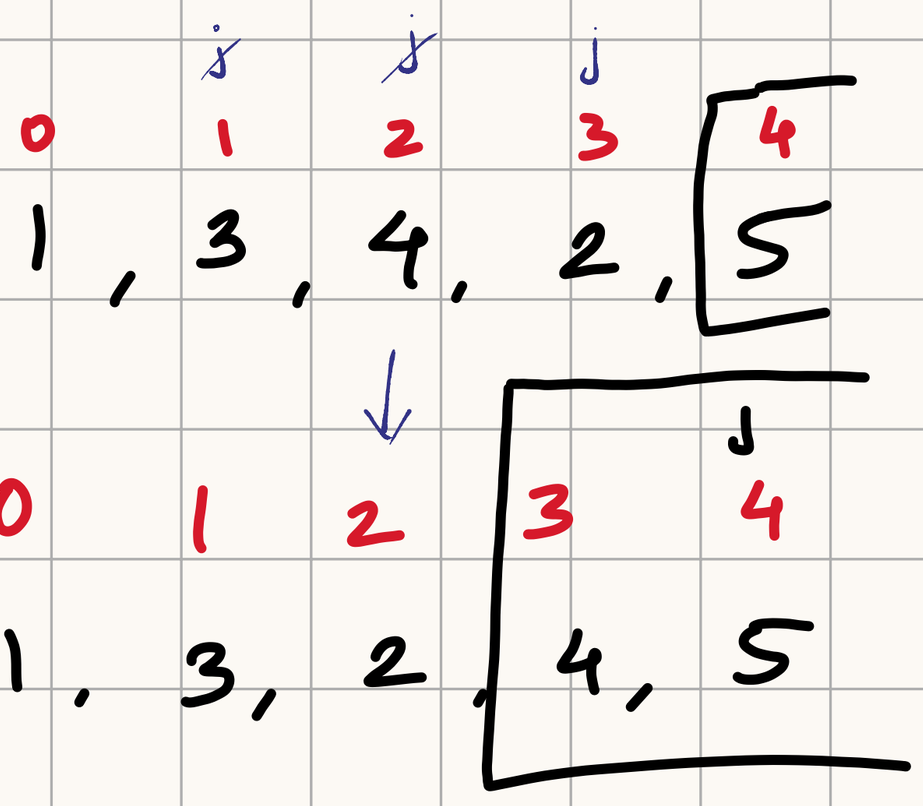
T



It is also known as sinking sort / exchange sort



for i = 1 // second pass

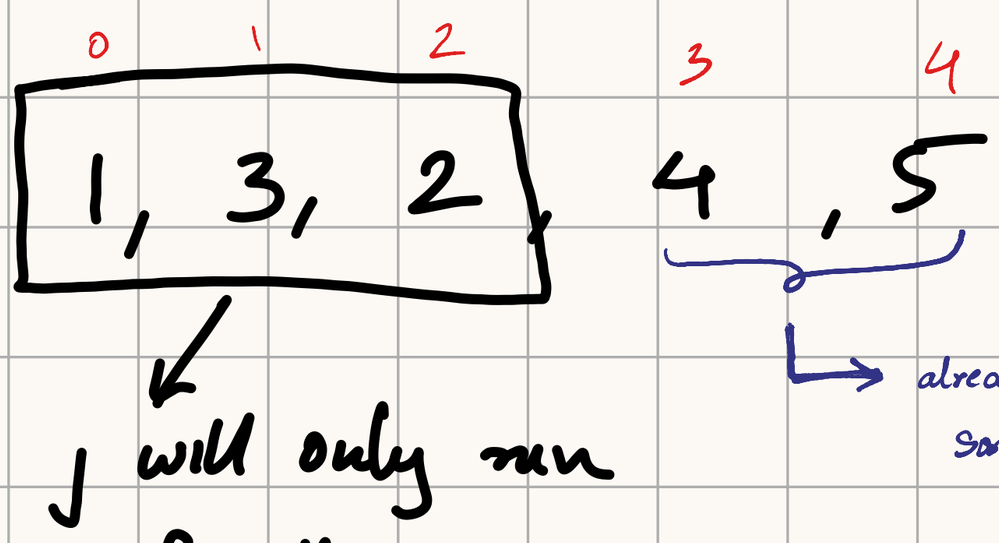


Length - i
= 5 - 1 = 4

or

<= len - i - 1

For i = 2 // third pass



Length - i
5 - 2
3

↓ will only run for these 3 element

* Complexity of bubble sort

Space complexity = O(1) // constant.

↳ No extra space required i.e copying the array etc. not required

aka inplace sorting algorithms

Time Complexity

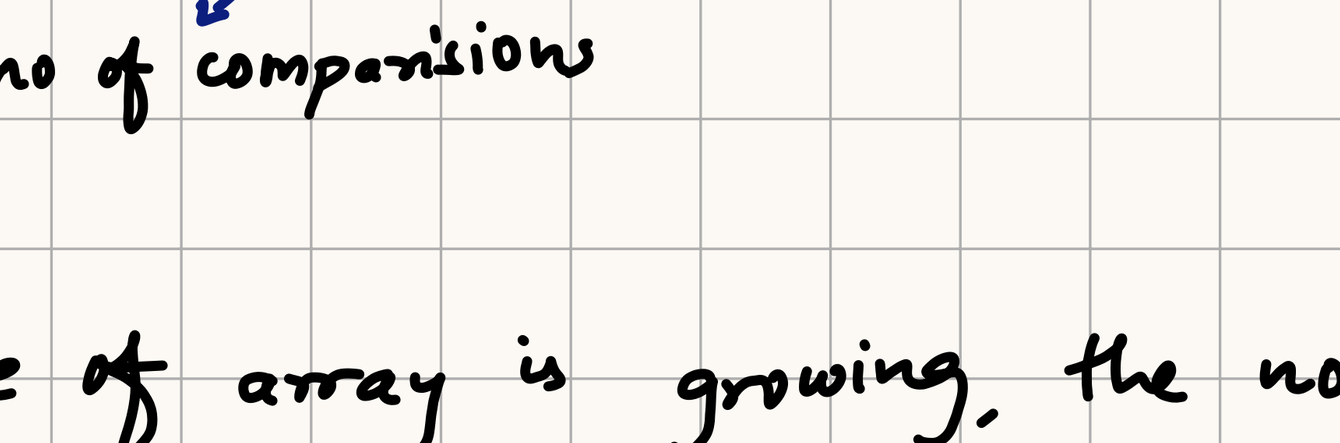
Best case : $O(N)$ ⇒ Sorted
Worst case : $O(N^2)$ ⇒ sorted in opposite

↙ no of comparisions

* As the size of array is growing, the no of comparisions is also growing

---

Best Case:



once

NOTE: when j never swaps for a value of i, it means array is sorted, hence you can end the program

Best case comparisions → N

Worst case.

0 1 2 3 4

5, 4, 3, 2, 1

4 5 3 2 1 — N-1

4 3 5 2 1

4 3 2 5 1

4 3 2 1 5 → pass 1

---

4 3 2 1 5 — N-2

3 4 2 1 5

3 2 4 1 5

3 2 1 4 5 → 2nd pass

---

3 2 1 4 5 — N-3

2 3 1 4 5

2 1 3 4 5 → 3rd pass

---

2 1 3 4 5 — N-

1 2 3 4 5 → 4th pass

Total Comparisions $\rightarrow (N-1)+(N-2)+(N-3)+(N-4)$

$= 4N - (1+2+3+4)$

$= 4N - \left(\frac{N*(N+1)}{2}\right)$

$= 4N - \left(\frac{N^2+N}{2}\right) \Rightarrow \frac{8N-N^2-N}{2}$

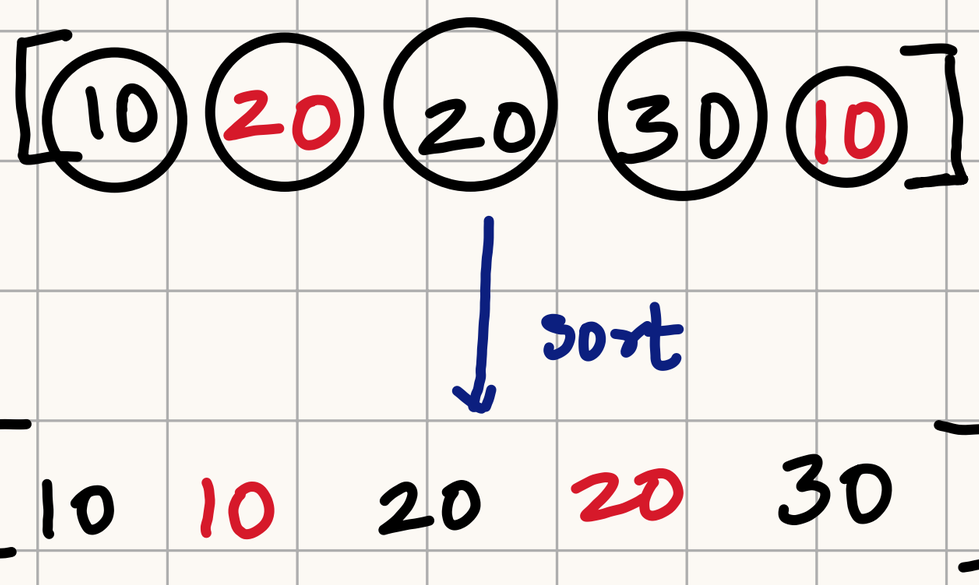
$= -\frac{N^2+7N}{2} \rightarrow \frac{7N-N^2}{2}$

$= O(N^2)$

* Stable sorting algorithm

(Order should be the same when value is same)

Example



In the original array black ball of 10 was before red ball of 10 and in the sorted one this order is maintained

[10, 10, 20, 20, 30] → This is unstable.

★ code for bubble sort.

```
public class Main {
    public static void main (String [] args) {
        int [] arr = {5, 4, 3, 2, 1};
        bubble (arr);
        System.out.println(Arrays.toString(arr));
    }
    static void bubble (int [] arr) {
        int n = arr.length;

        for (int i= 0; i < n-1; i++) {
            for(int j= 1; j < n-i-1; j++) {
                if (arr[j] > arr[j+1]) {
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }
    }
```

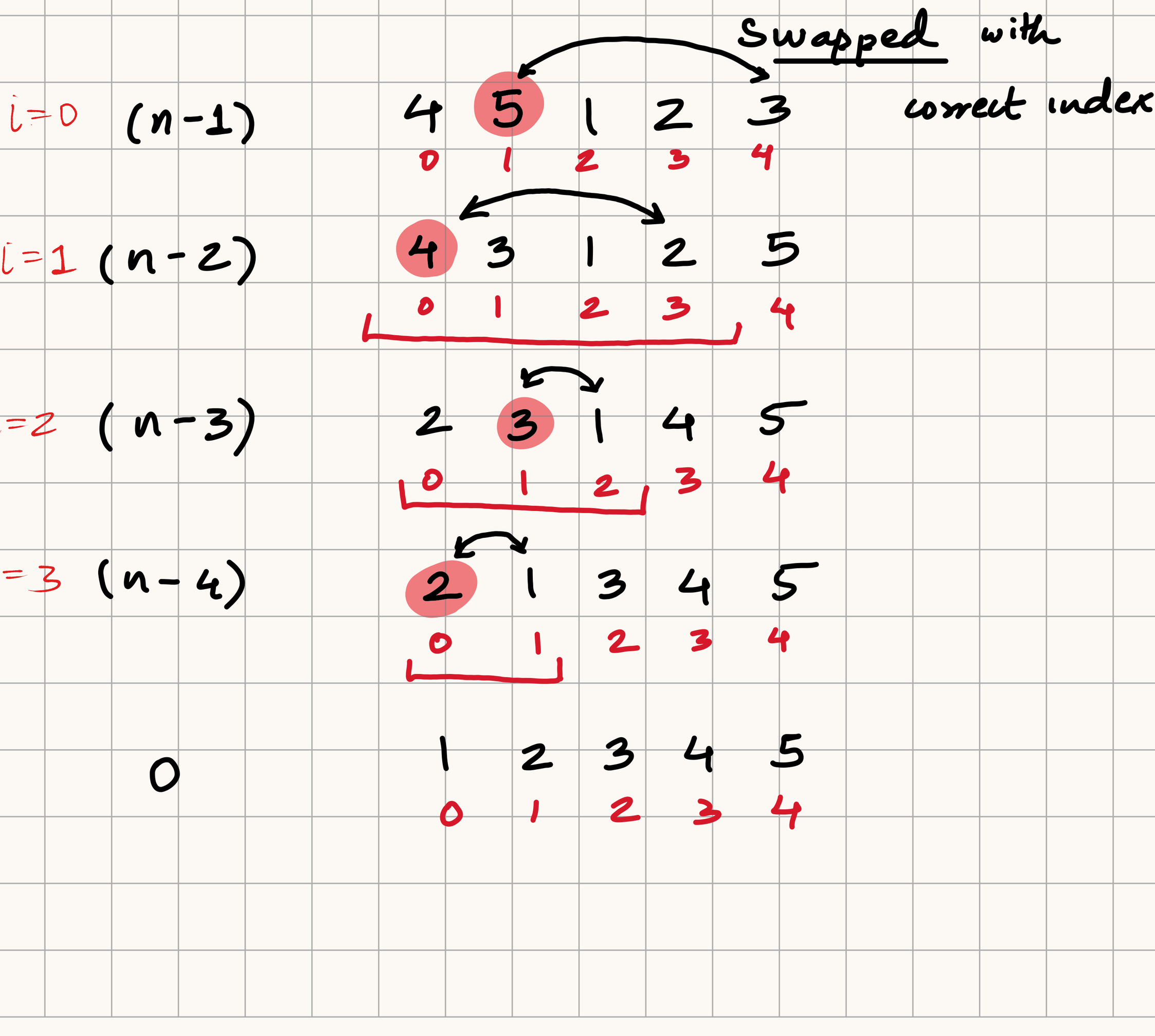
* Selection Sort

0 1 2 3 4
4, 5, 1, 2, 3

- Select an element and put it at its correct index

- Two ways: ↴

Way ① Select the largest element of the array and put it at its correct index Then select second largest element in an array and put it in correct index



Way ② Select the smallest element of the array and put it at its correct index Then select second smallest element in an array and put it in correct index

Time complexity

Total comparision · $0+1+2+3+\cdots+(n-1)$

$$\frac{(n-1)(n-1+1)}{2} = \frac{n(n-1)}{2}$$

$$= \frac{n^2-n}{2} = O(n^2)$$

Wont case = $O(N^2)$
Best case = $O(N^2)$

Stability = No

// It performs well on small lists

Algorithm:

① Find maximum item in whole array
② When the loop runs again find the maximum item in the remaining array
③ When you have sorted the two maximum items ignore them and find other maximum item from remaining array
④ Loop is running for → (n-i-1)

* Code for Selection sort algorithm.

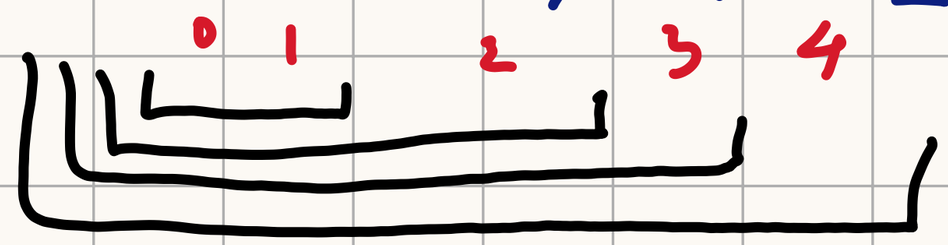
```
public class Main {
    public static void main (String[] args) {
    int[] arr = {3, 1, 5, 4, 2}
     selection (arr),
     System out println (Arrays toString (arr)),
}
```

```
Static void selection (int[] arr) {
        n = arr length,
    for (i=0, i< n ; i++) {
     int last = n- i-1;
 int maxIndex = getMaxindex (arr, 0, last),
     swap(arr, maxIndex, last)
    }
}


Static int getMaxindex (int[] arr, int start, int end) {
    int max = start,
    for (int i = start, i<= end, i++){
          if (arr[max] < arr[i]) {
              max = i;
          }
      }
      return max;
  }


Static void swap (int[] arr, int first , int second
     int temp = arr[first],
    arr [first] = arr [second],
    arr [second] = temp,
  }
```

Insertion Sort

- In insertion sort we are sorting the array partially

- Example [5, 3, 4, 1, 2]



For every index

- Put that index element at the correct index of LHS
- After 1st pass. i = 0 [3, 5, 4, 1, 2]

index 0 will be sorted

- After 2nd pass. i = 1 [3, 4, 5, 1, 2]

index 1 will be sorted

- After 3rd pass. i = 2 [1, 3, 4, 5, 2]

index 2 will be sorted

- After 4th pass. i = 3 [1, 2, 3, 4, 5]

index 3 will be sorted

0	1	2	3	4
5	3	4	1	2

Sort array till index 1 → pass no 1

Sort array till index 2 → pass no 2

Sort array till index 3 → pass no 3

Sort array till index 4 → pass no 4

i
0
1
2
3
4

i
1

i will run from 0 till (n-2)

n → length of array

5, 3, 4, 1, 2

i ≤ n-2	J > 0
0	1
1	2
2	3
3	4

J

3, 5, 4, 1, 2

j

3, 4, 5, 1, 2

When element J is not smaller than element J-1 break the loop

j

3, 4, 1, 5, 2

J

3, 1, 4, 5, 2

1, 3, 4, 5, 2

i J

1, 3, 4, 5, 2

for i = 2, till index 3 it is sorted

4 5

index out of bound

1, 3, 4, 2, 5

1, 3, 2, 4, 5

1, 2, 3, 4, 5

↓

1, 2, 3, 4, 5

* Complexity

Worst case · $O(N^2)$ → no of elements (desc sorted)

5 4 3 2 1

Sum of N nos $\frac{N(N+1)}{2}$ {Total comparisions}

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

$$= O\left(\frac{N^2-N}{2}\right) = O(N^2)$$

Best case: Array is already sorted

j(2<1) j(3<2) j(4<3) j(5<4)

1, 2, 3, 4, 5

No of comparisions = N-1 → linear

O(N).

* Why we use insertion sort?

1) Adaptive steps get reduced if array is sorted
   No of swaps reduced as compared to bubble sort

2) Stable.
3) Used for smaller values of N Works good when array is partially sorted It takes part in hybrid sorting algorithm.

* Code for insertion sort

```
public class InsertionSort {
  public static void main(String[] args){
     int[] arr = {5,4,3,2,1};
     insertion(arr);
     System.out.println(Arrays.toString(arr));
}

static void insertion(int[] arr){
  for(int i=0, i<=arr.length-2, i++){
       for(int j = i+1, j<0, j--){
           if(arr[j] < arr[j-1]){
              swap(arr, j.j-1),
         }else{
           break;
         }
      }
  }
}

static void swap(int[] arr, int first, int second){
   int temp = arr[first];
   arr[first] = arr[second],
   arr[second] = temp.
  }
}
```

Cyclic Sort.

0 1 2 3 4

3, 5, 2, 1, 4

★ When given numbers from range 1, N → use cyclic sort

★★★★ Very, Very Important.

★★

Amazon Questions would be like

Q) You are given a number from 1 to n find the missing number

Q) You are given an unsorted array. find the smallest positive missing number

Q) You are given numbers from 1 to N find the duplicate numbers

3, 5, 2, 1, 4

0 1 2 3 4 N = 5

- We know that array contains numbers from 1 to n
- When the array is sorted, all the numbers are going to be at their correct indices

- In the above Example.

After sorting: 1, 2, 3, 4, 5

0 1 2 3 4

After sorting, index = value - 1 because index starts from 0

- Sorting the array.

0	1	2	3	4
3,	5,	2,	1,	4

We are going to check whether 3 (index 0) is at the correct index by applying formula

$$\text{correct index} = \text{value} - 1$$
$$= 3 - 1$$
$$= 2$$

which means correct index is 2

Hence swap with correct index

0	1	2	3	4
2,	5,	3,	1,	4

We know that we have swapped 3 (index 0) to correct position to index 2. but we don't know whether we have swapped the other element to right place.

Hence,

correct index = value - 1

= 2 - 1

= 1

correct index = 1

5, 2, 3, 1, 4

0 1 2 3 4

correct index = value - 1

= 5 - 1

= 4

4, 2, 3, 1, 5

0 1 2 3 4

correct index = value - 1

= 4 - 1

= 3

1	2	3	4	5
0	1	2	3	4

Answer

We can see that every unique item is getting swapped once

No. of comparisions

- We are not incrementing i, when we are swapping the numbers

- SO while loop as we know to put it at correct index total (N-1) number of swap are need to be made

- The above example that we looked at was for the worst case
- It took 4 swaps + 5 swaps (for comparision)
- So total (N-1) + N swaps = 2N - 1 swaps

Time complexity of cycle sort:

O(N)

★ Code for above example (code for cycle sort)

```
public class CycleSort {
    public static void main (String [] args) {
        int [] arr = {3, 5, 2, 1, 4};
        cycle (arr);
        System out println (Arrays toString (arr));
}

 Static void cycle (int [] arr) {
        int i = 0;
      while (i < arr length) {
              int correct = arr [i] - 1;
```

```
            if (arr[i] != arr[correct]){
                swap(arr, i, correct);
            }else{
              i++,
            }
        }
    }

    Static void swap (int[] arr, int first, int second){
        int temp = arr[first],
        arr[first] = arr[second],
        arr[second] = temp,
    }
}
```

★ Merge Sort:

- Simple sorting technique in which we use recursion
  We do divide and conquer in Merge sort

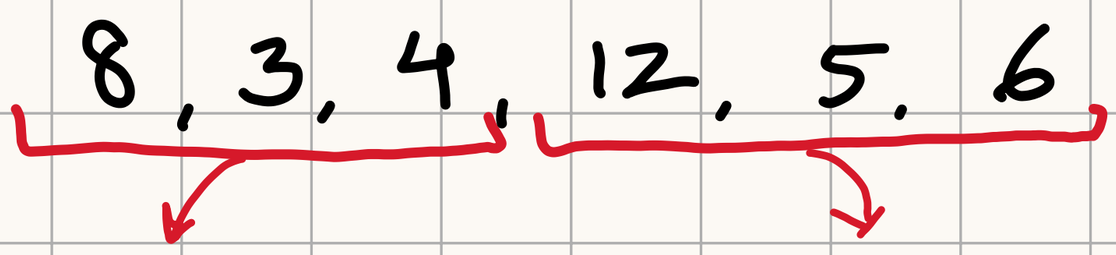
- We divide the bigger problems into smaller problems

Lets say we are given an unsorted array.

[8, 3, 4, 12, 5, 6]

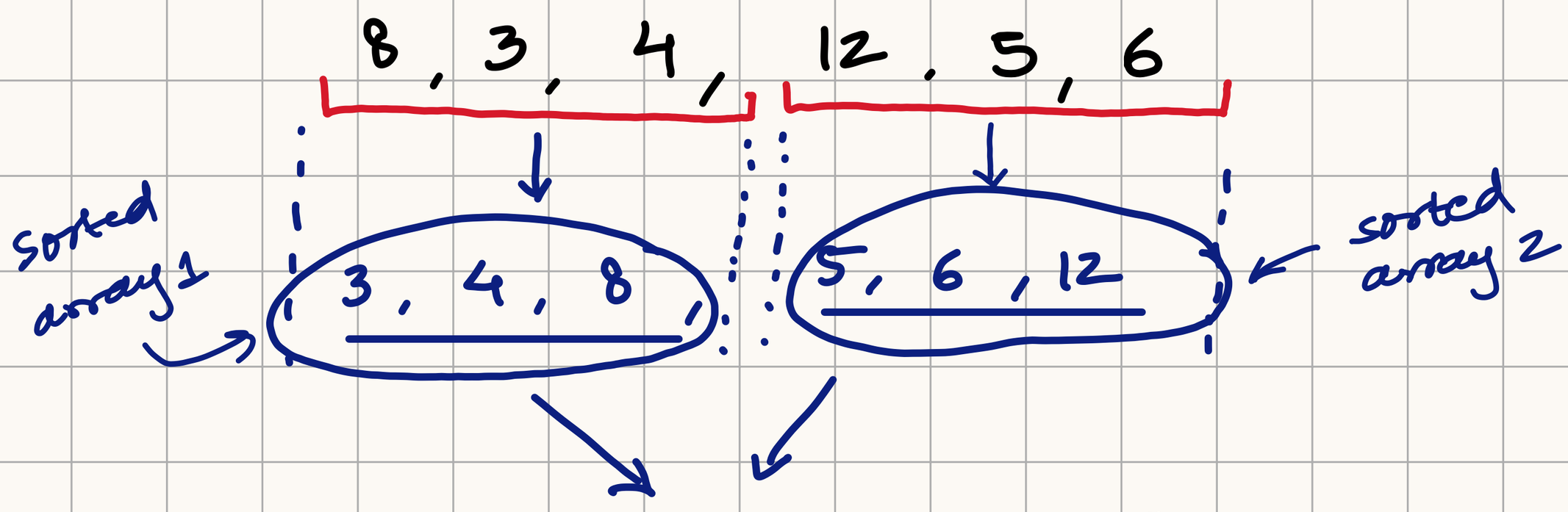
We need to sort it using merge sort

In recursion it is like divide the array into two parts



- Recursion will sort first half of the array, then it will sort second half of the array and merge the two sorted answers

After Recursion



- Merging happens simply by using 2 pointers.

P_1

↓

3, 4, 8

P_2

↓

5, 6, 12

compares P_1 and P_2

is $P_1 < P_2$

↓

True

arr = [3,]

P_1 gets shifted to n+1

P_1 ↓

~~3~~, 4, 8

P_2 ↓

5, 6, 12

compares P_1 and P_2

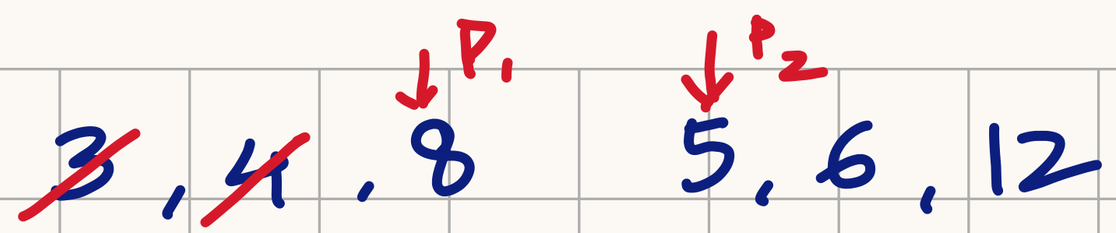
is $P_1 < P_2$

↓

True

P_1 shifted

arr = [3, 4]



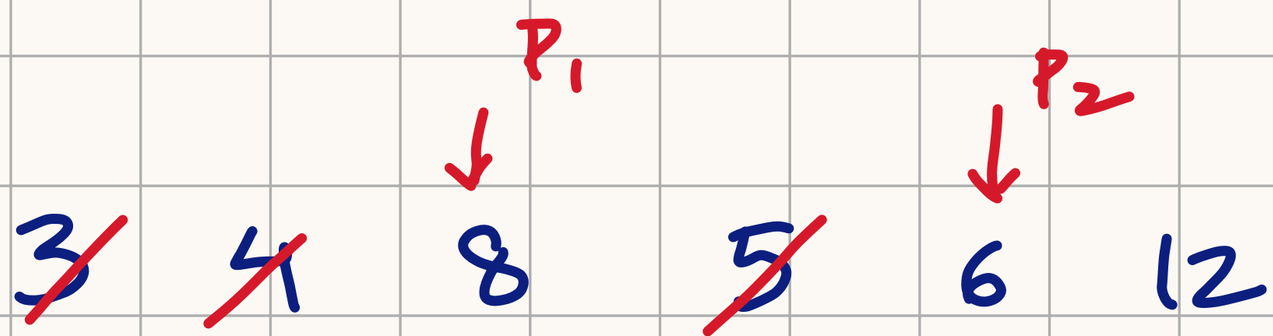
compares P_1 and P_2

is $P_1 < P_2$

↓

False arr = [3, 4, 5]

This time P_2 gets shifted to n+1.



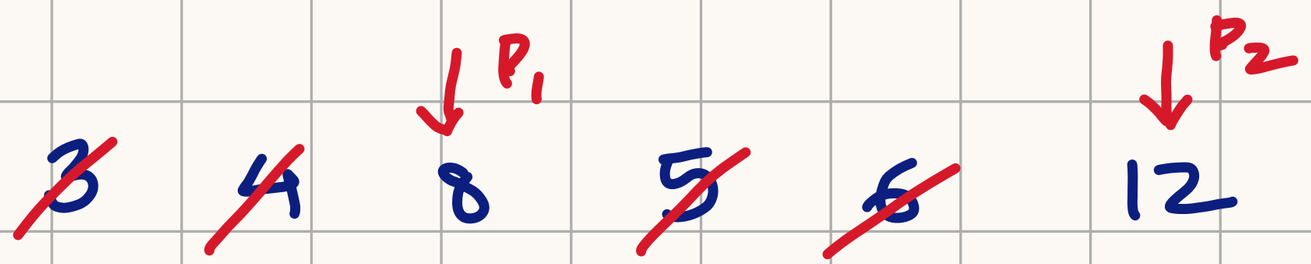
compares P_1 and P_2

is $P_1 < P_2$

↓

False arr = [3, 4, 5, 6]

P_2 shifted



compares P_1 and P_2

is $P_1 < P_2$

↓

True arr = [3, 4, 5, 6, 8,]

Last pointer element gets added at end

arr = [3, 4, 5, 6, 8, 12].

Steps to do merge sort:

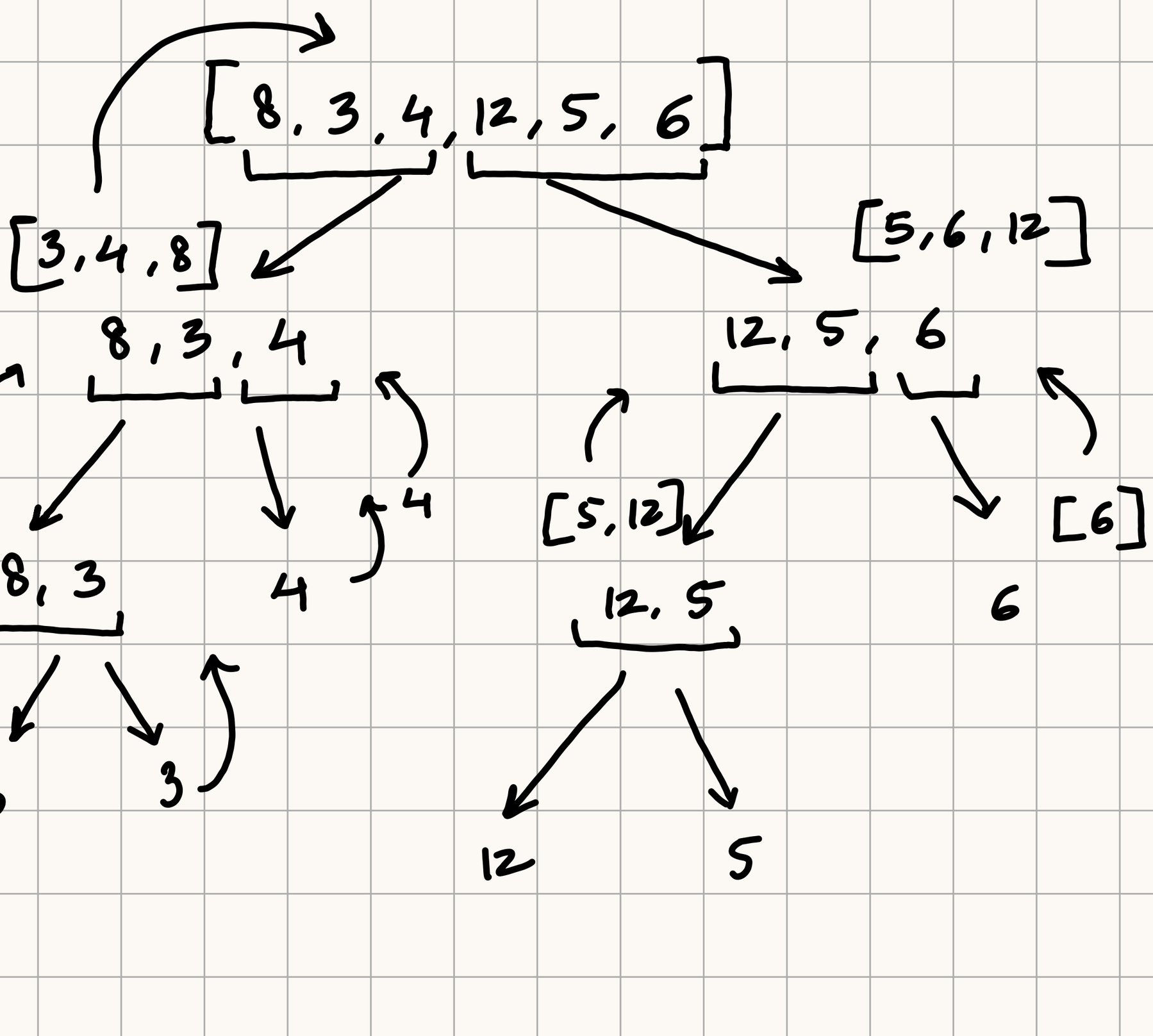
1. Divide array into 2 parts.
2. Get both parts sorted via recursion
3. Merge 2 sorted arrays

arr 1 = [3, 5, 9]

arr 2 = [4, 6, 8]

size arr

(arr 1 + arr 2) = [3, 4, 5, 6, 8, 9]

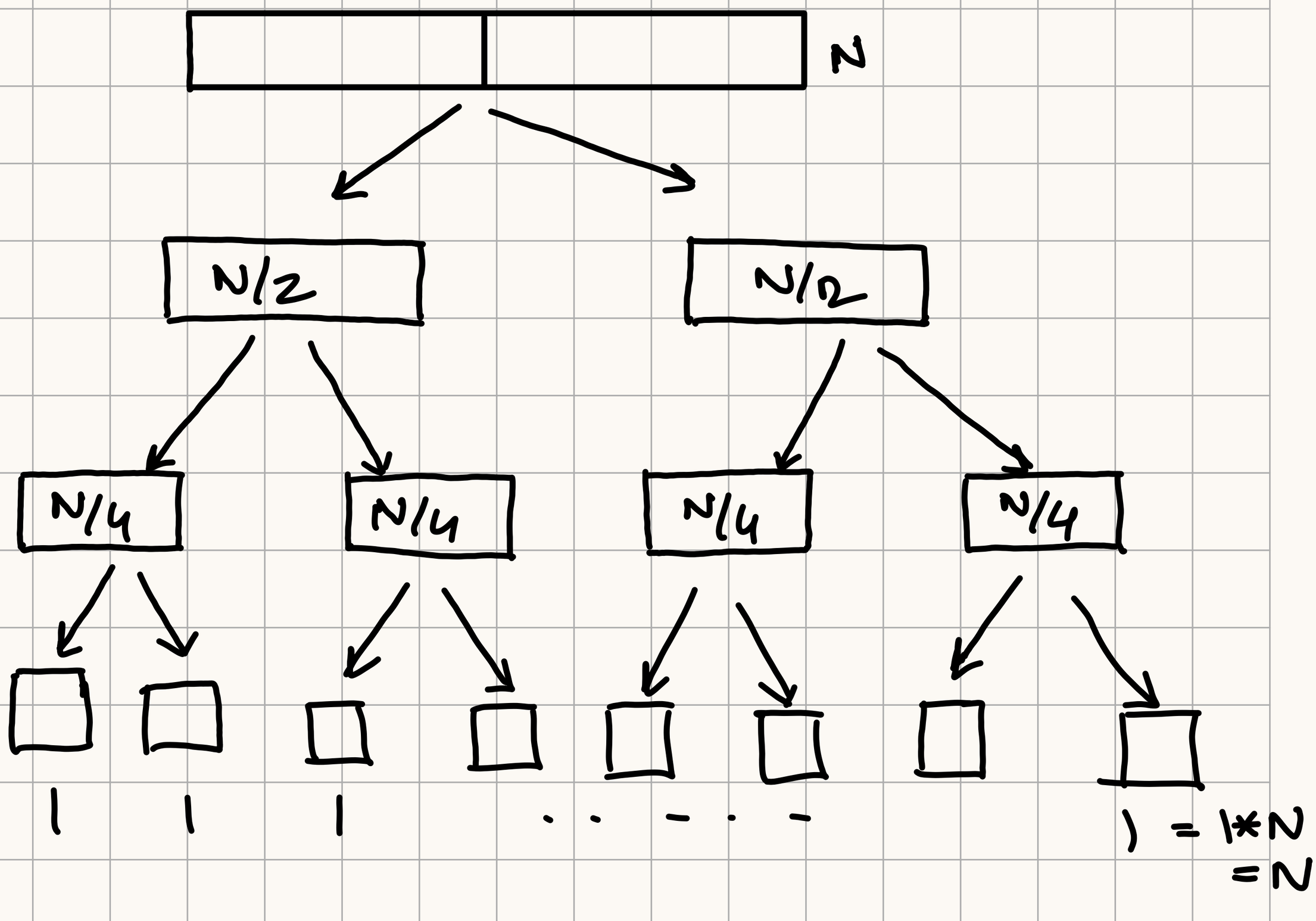


Code for Merge sort.

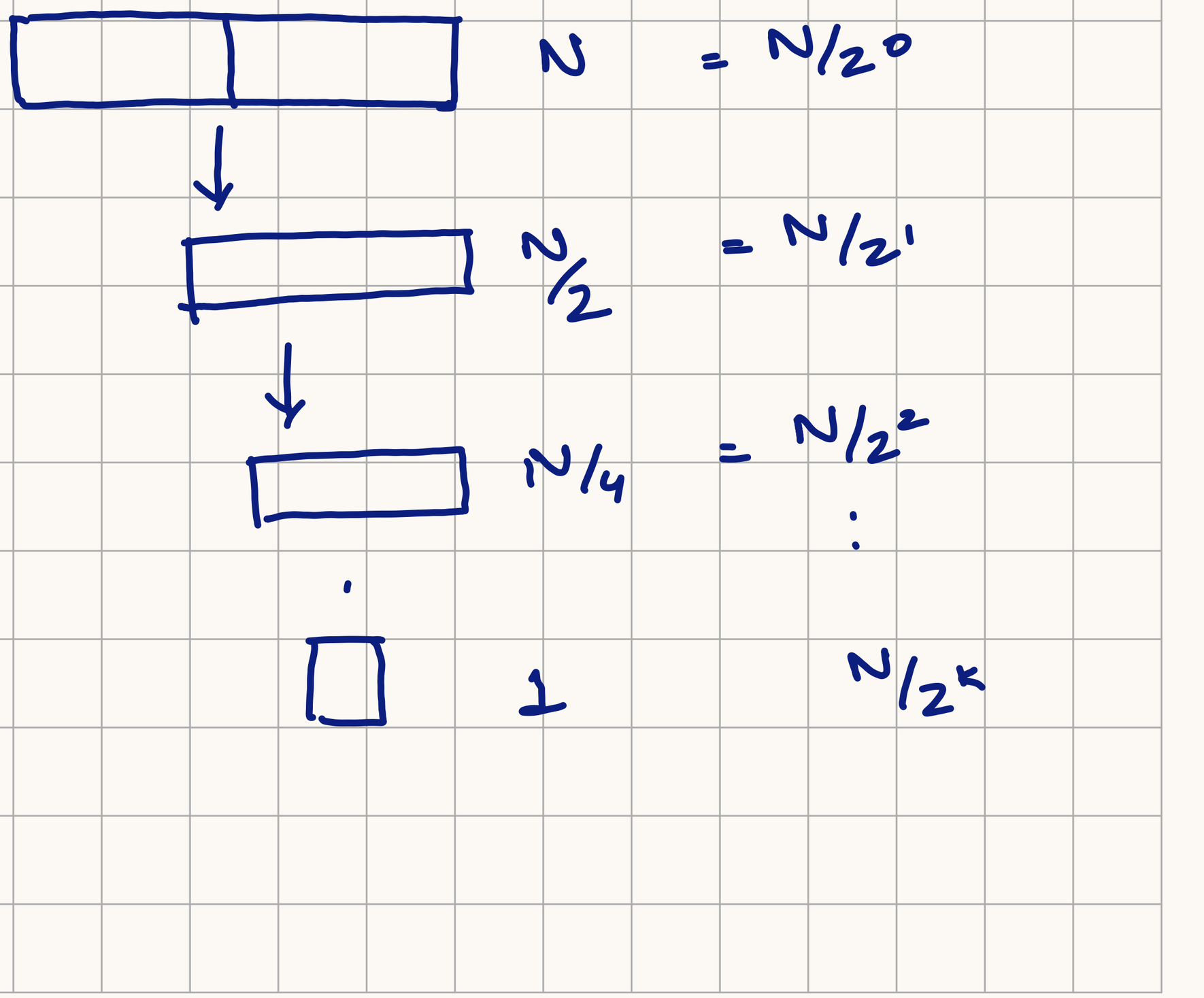
```
public class MergeSort {
    public static void main (String[] args) {
        int [] arr = {5,4,3,2,1},
         arr = mergeSort (arr),
       System.out.println (Arrays.toString (arr));
    }

Static int[] mergesort (int[] arr) {
      if (arr length == 1) {
                return arr,
          }
            int mid = arr Length /2,

int [] left = mergeSort (Array copyofRange (arr, 0, mid),
int [] right = mergeSort (Array copyofRange (arr, mid, arr length),

            return merge (left, right),
      }

private static int [] merge (int[] first, int[] second){
        int [] mix = new int [first length + second length ;
         int i=0,
        int j=0,
        int K=0,
```

```
        while (i < first.length && j < second.length) {
            if (first[i] < second[j]){
                mix[k] = first[i],
                    i++;
            }
            else {
                mix[k] = second[j],
                    j++;
            }
            k++,
        }
    // it may be possible that array is not complete
    // copy remaining elements
    while (i< first length) {
        mix[k] = first[i],
            i++,
            k++,
    }
while (j< second length) {
        mix[k] = second[j],
            j++,
            k++,
        }
        return mix,
    } }
}
```

Time Complexity.



At every level N elements are getting merged



$$1 = N/2^k$$

$$2^k = N$$

$$k \log 2 = \log N$$

$$k = (\log_2 N)$$

- Time complexity = $O(N^* \log N)$

Space complexity, Auxillary = $O(N)$

By Akra Bazzi formula.

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

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

$$2 \times \frac{1}{2^p} = 1 \quad \rightarrow p = 1$$

$$T(N) = x + x\int_1^x \frac{u-1}{u^2} \Rightarrow \left| \int_1^2 \frac{1}{u} - \frac{1}{u^2}\, du \right.$$

$$\int \frac{du}{u} - \frac{du}{u^2}$$

$$= \log u - \int u^2\, du$$

$$= \log u + u^{-1}$$

$$= \left[\log u + \frac{1}{u} \right]^{x}_{1}$$

$$= \log x + \frac{1}{x} - 1$$

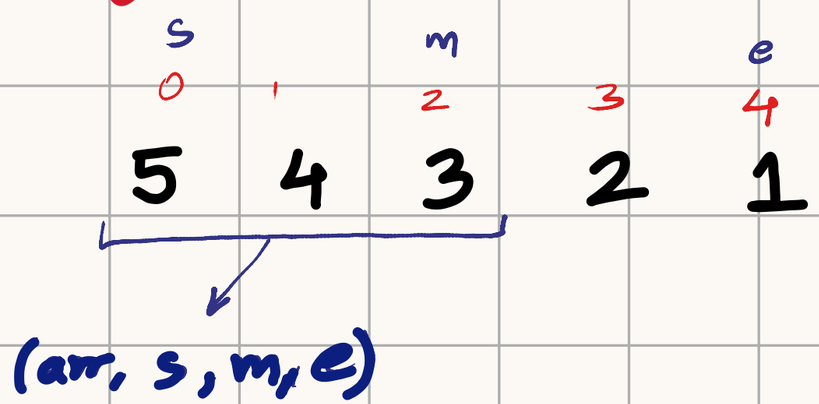
$$= x + x\left[\log x + \frac{1}{x} - 1 \right]$$

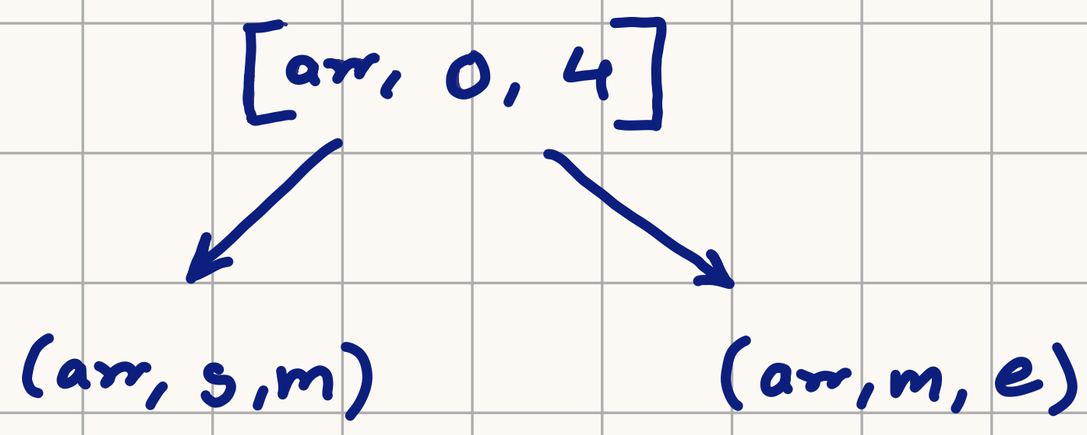
$$= \cancel{x} + x\log x + 1 - \cancel{x}$$

$$= x\log x + \cancel{1}$$

$$= O(x\log x) \longrightarrow O(n\log n)$$

Inplace Merge Sort -



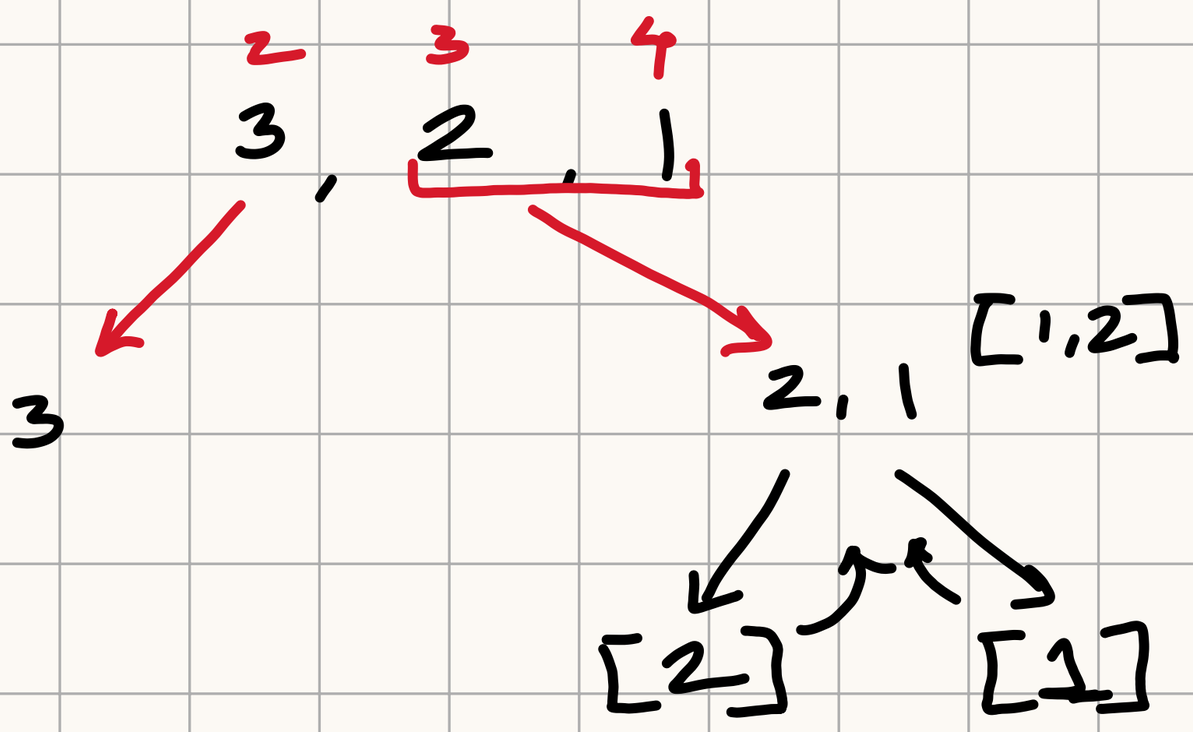


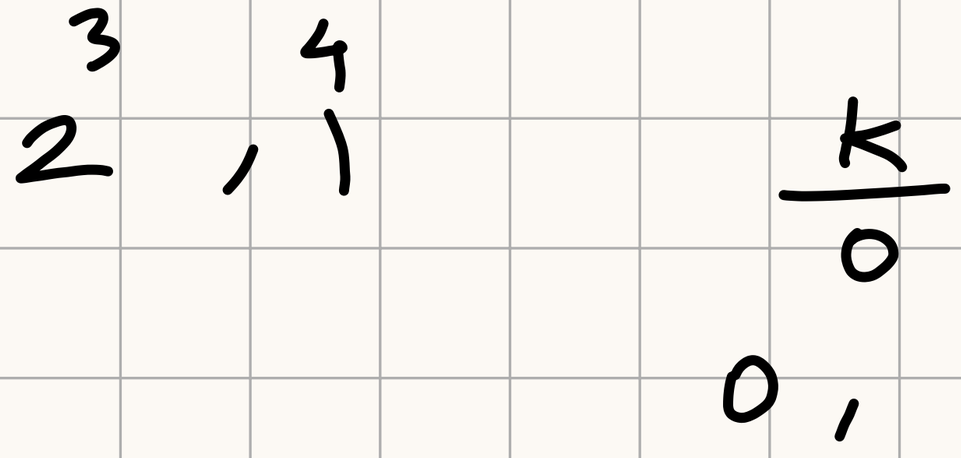
(arr, s, m, e)

mix = [1, 2, 3, 4, 5]

```
for (k = 0, k < len(mix); k++) {
    arr[s+k] = mix[k].
```

Lets say we are sorting.





arr [3+0] = 1

arr [3+1] = 2

* code for in-place Merge Sort.

```
public class main {
        psvm(S[] a) {
        int [] arr = {5,4,3,2,1};
        mergeSortInPlace(arr,0,arr Length),
        Sout (Arrays toString (arr))
    }
static void mergeSortInplace (int [] arr, int s, int e){
        if (e - s  ==1) {
            return;
        }
        int mid = (s+e) /2,
        mergesortInPlace (arr, s, mid),
        mergesortInPlace (arr, mid, e);

        return mergeInPlace (arr, s, mid, e),
}

pnvate static void mergeInPlace (int[] arr, int s, intm,
                                                int e){
        int [] mix = new int [e-s];
        int  i = s,
        int  j = m;
        int  k = 0;
```

```
while (i<m && j<e) {

        if (arr[i] < arr[j]) {
        mix[k] = arr[i],
          i++,
      } else {
          mix[k] = arr[j],
            j++,
      }
      k++,
  }
  while (i<m) {
      mix[k] = arr[i],
      i++,
      k++,
  }
  while (j<e) {
          mix[k] = arr[j],
          j++,
          k++,
      }
      for (int L=0, L< mix length; L++){
          arr[s+l] = mix[l];
        }
    }
}
```

☆ Quick Sort:

In Quick sort we use pivot A pivot is a reference point It can be any element in the array The idea is that whatever pivot you take after first pass, all the elements less than pivot will be on Left of pivot And elements greater than pivot will be on right side of pivot

5, 4, 3, 2, 1

① Lets take any pivot element randomly

P

5, (4), 3, 2, 1

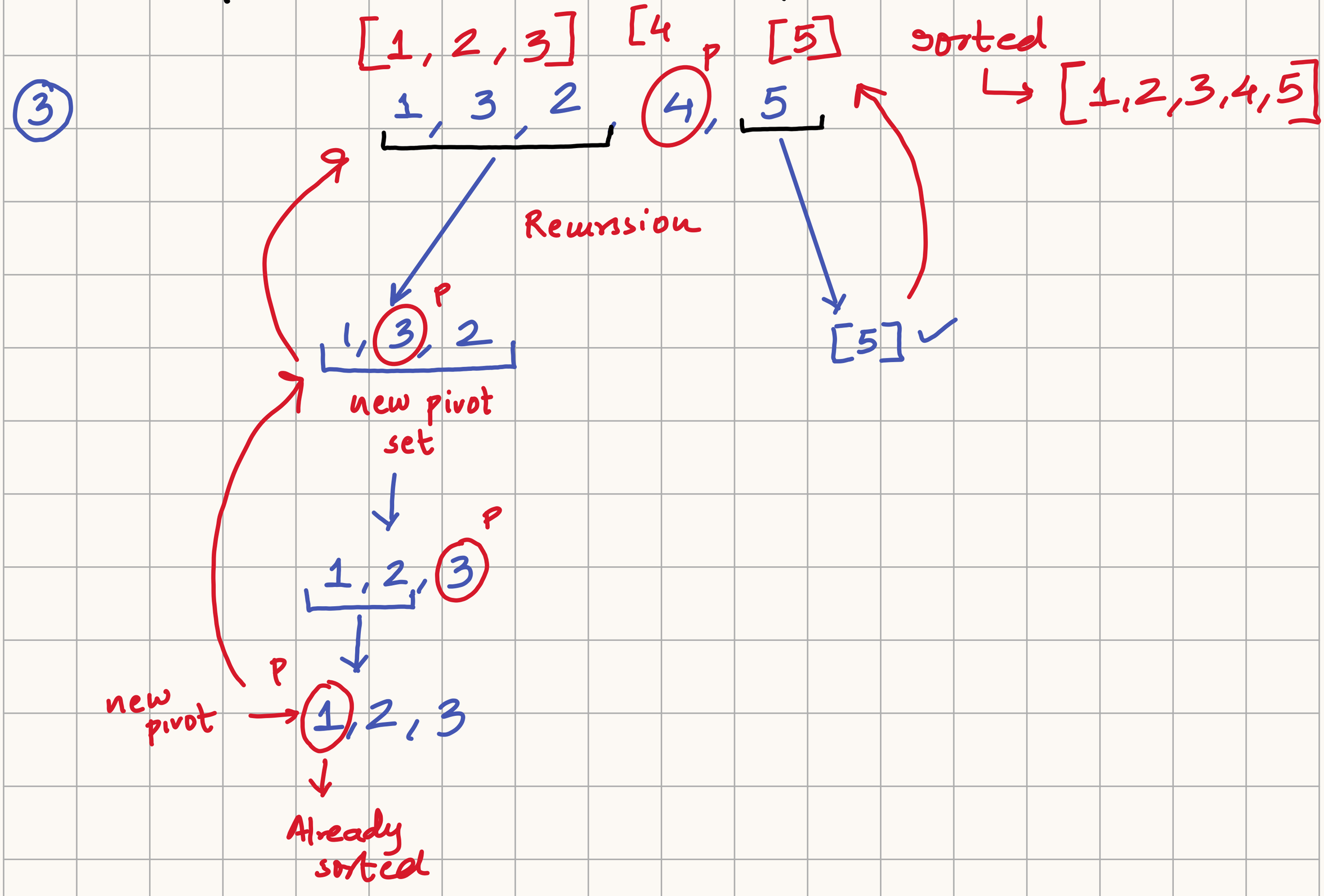
So the idea is that after every pass the elements which are smaller than 4 should be on left While the elements which are greater than 4 are on right side of 4

② P

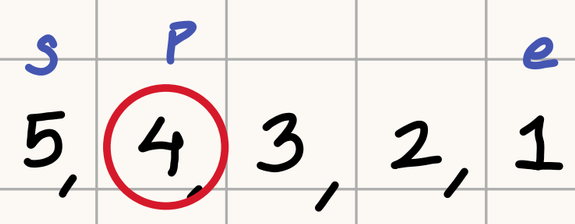
1, 3, 2, (4), 5

All the elements are now either on RHS or LHS of the Pivot They may or may not be sorted

Recursion will then try to sort the two part We know that how the pivot is at the correct position



★ How to put pivot at correct location -



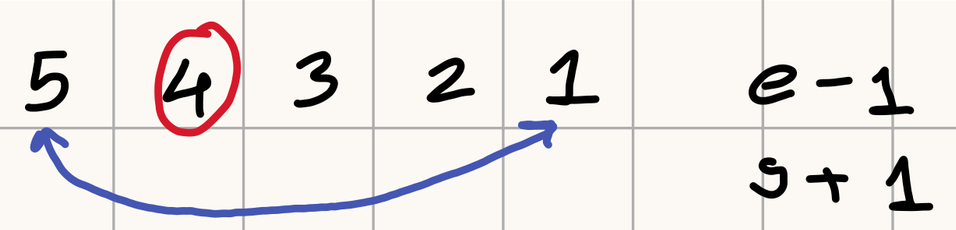
s

is 5 a voilation? → 4 > 5 X → It's voilation.

e

is 1 a voilation? → 4 < 1 X → It's voilation.

SWAP



s e

1 4 3 2 5

is 4 a voilation? ⟶ 4 > 4 X ⟶ It's violation

is 2 a voilation ? ⟶ 2 > 4 X ⟶ It's violation

SWAP

s,e

1 2 3 4 5

Now Pivot is at the correct position

1 2 3 4 5

This part of array may not be sorted.

e s e s

[1 , 2 , 3]

s p e

(low, end) (s, high)

while n[s] < p

start ++,

while n[e] > p;

end --;

swapping

s & end

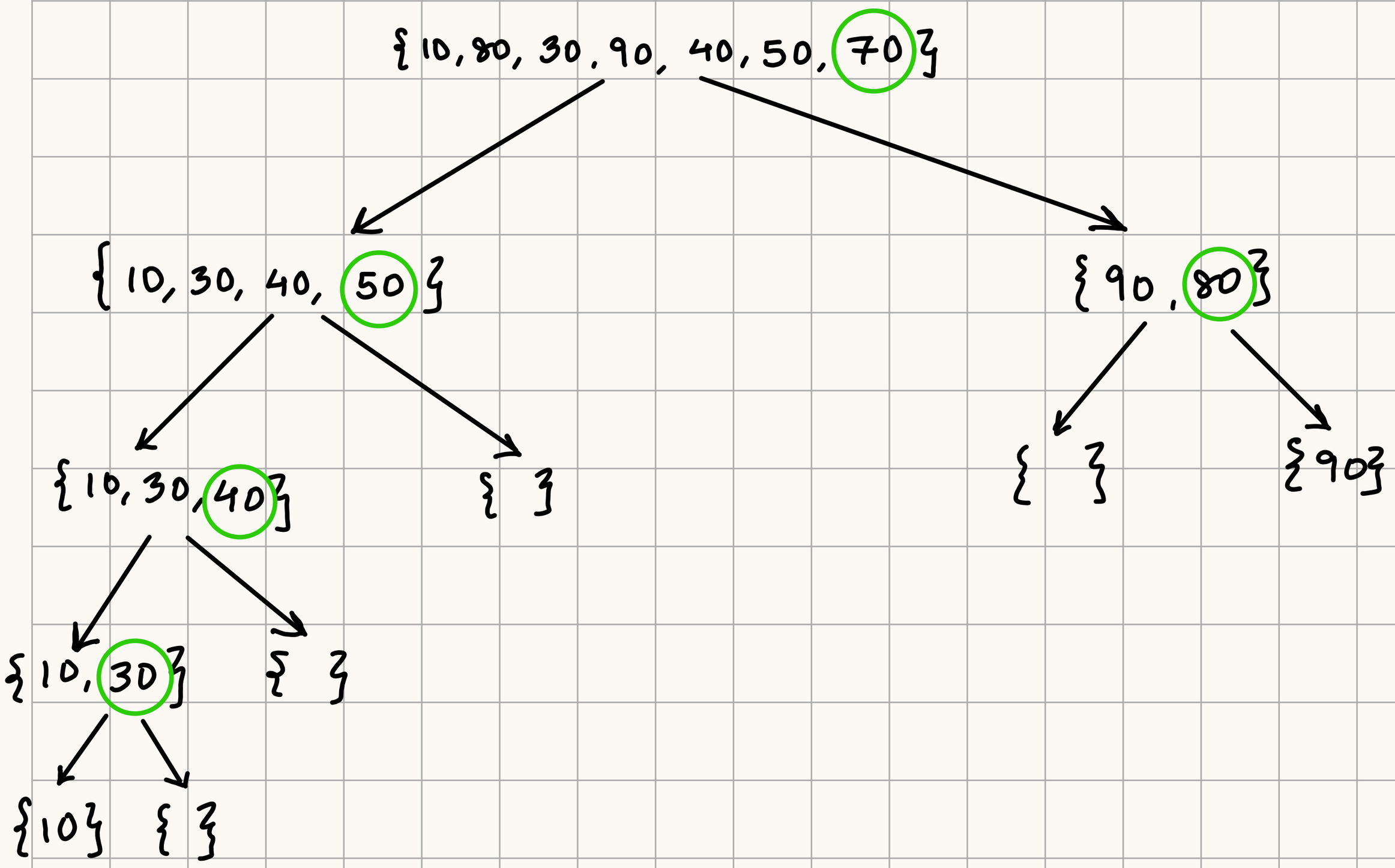
low & high ⟶ main

tells us which part of array you are working on

{10, 80, 30, 90, 40, 50, 70}

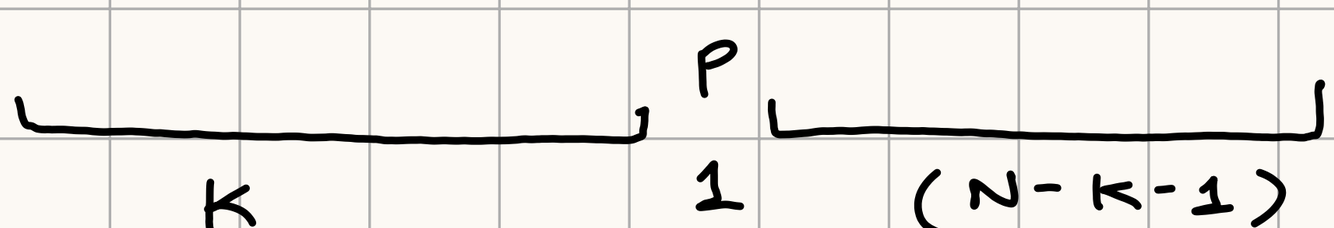
* How to actually pick the pivot

You can pick the last element as pivot



Pivot

* Random element
* corner element
* Middle element

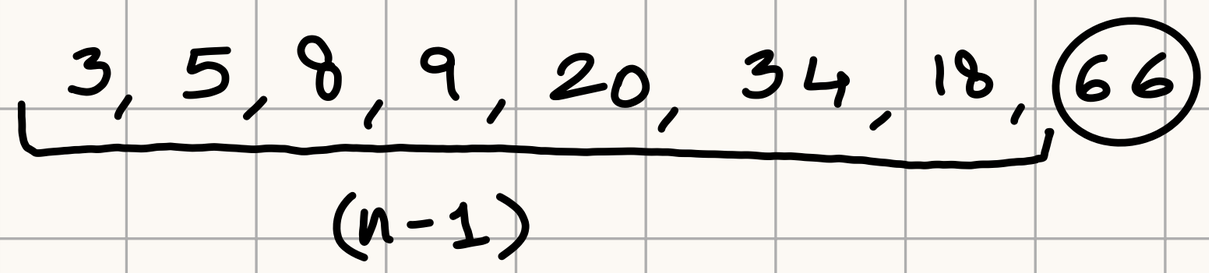


Important.

$$T(N) = T(K) + T(N-K-1) + O(N)$$

Recurrence relation of Quick sort

Worst Case. When K = 0 (one part will be empty)



$$T(N) = T(0) + T(N-1) + O(N)$$

$$\boxed{T(N) = T(N-1) + O(N)}$$

$$= O(N^2)$$

Best Case: When pivot is at middle position so that whenever it is at its correct position it gets divided into two halves

K = N/2

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

$$\boxed{T(N) = 2T\left(\frac{N}{2}\right) + O(N)}$$

$$= O(N \log N)$$

* Important Notes:

- Quick sort is not Stable
- In-place ⟶ we are not making new array, like in merge sort we are taking a mix array.
  Merge sort takes O(N) extra space

- Merge sort is better in Linked list due to memory allocation ⟶ not continous

* Hybrid Sorting Algorithms (Tim Sort)

Merge Sort + Insertion sort
↓
Works really well with partially sorted data.

★ Code for Quick Sort:

```
public class Quicksort {
    public static void main (String [] args) {
        int [] arr = { 5, 4, 3, 2, 1 };
        sort ( arr, 0, arr length-1);
        System out println ( Arrays toString (arr)
}

Static void sort (int [] nums, int low, int hi) {
        if (low > = hi) {
            return;
        }
    int s = low,
    int e = hi;
    int m = s + (e - s) /2;
    int pivot = nums [m],

    while (s <= e) { // also a reason why if already sorted it wont swap
        while (nums [s] < pivot) {
            s++,
        }
        while (nums [e] > pivot) {
            e--,
        }
        if (s <= e) {
            int temp = nums [s],
            num [s] = nums [e],
```

```
            num[e] = temp;
            s++;
            e--;
        }
    }

    // now my pivot is at correct index. please sort two halves now
        sort(nums, low, e),
        sort(nums, s, high),
    }
}
```

★ Count Sort

- It is a non comparision based sorting algorithm
- Good for smaller numbers

Lets take an example.

You need to sort an array

0	1	2	3	4	5	6	7
3	4	1	3	2	5	2	8

① Find the largest no

target = 8

0	1	2	3	4	5	6	7	8
0	1	2	2	1	1	0	0	1

② Create an array of size arr = largest + 1
= 8 + 1 = 9

→ frequency Array No of times the element occurs in the array where the index number is equal to the element number

③ traverse the original array and store the number of times the number occurs (frequency array)

0	1	2	3	4	5	6	7	8
0	1	2	2	1	1	0	0	1

0	1	2	3	4	5	6	7
1	2	2	3	3	4	5	8

④ Traverse the frequency array. If the value of the index is greater than zero that means its in the original array

Array sorted.

Time Complexity. Searching the original array takes O(N) time

+

Searching frequency array takes O(M) time

= O(N+M) ⟶ linear

Space Complexity O(N+M)

HashMap.

3 - 2
4 - 1
1 - 1
2 - 2
5 - 1
8 - 1

max = 8

i = 0 till 8

map get(i)

1	2	2	3	3	4	5	8

Code.

```
class countSort {

 public static void countSort(int[] arr){
    if(arr == null || arr length <= 1){
      return;
 }

 int max = arr[0];
for(int num · array){
   if(num > max){
```

```
            max = num;
        }
    }

    int[] countArray = new int[max + 1];
    for(int num : arr) {
        countArray[num]++;
    }

    int index = 0;
    for (int i=0; i<= max, i++) {
        while (countArray[i] >0) {
            arr[i] = i;
            index++;
            countArray[i]--;
        }
    }

    public static void main (String[] args){
        int[] arr = {6, 3, 10, 9, 2, 4, 9, 7};
        countSort (arr);
        System.out.println (Arrays.toString (arr));
    }
}
```

* CountSort using HashMaps.

```
public static void  countSortHas(int[] arr) {
    if (arr == null || arr length <= 1) {
        return;
    }
    int max = Arrays stream(arr) max() getAsInt(),
    int min = Arrays stream(arr) max() getAsInt(),
    Map<Integer, Integer> countMap = new HashMap<>(),
    for (int num arr) {
        countMap put(num, countMap.getOrDefault(num, 0)+1),
    }
    int index = 0,
    for(int i = min, i<= max, i++) {
        int count = countMap getOrDefault(i,0);
        for(int j=0, j< count, j++) {
            arr[index] = i,
            index++;
      }
    }
}
public static void main(String[] args) {
    int[] arr = {6, 3, 10. 9. 2, 4, 9, 7},
    countSortHas(arr);
    System.out println(Arrays.toString(arr));
  }
}
```

★ Radix Sort Algorithms

Divides numbers in buckets and sorts according to buckets.

29	83	471	36	91	8

Buckets Initially it is going to take the Largest element by checking number of digits

Largest number of digits · 3
run count sort 3-times

Sort the array using count sort digit by digit, starting from least

29 83 471 36 91 8

471 91 83 36 08 29

091 083 471 036 029 008

471	91	83	36	29	08

Answer

Code.

```
class Main {

  public static void radixSort(int[] arr) {
    int max = Arrays.stream(arr).max().getAsInt();
    // do count sort for every digit place
    for (int exp = 1; max/exp > 0; exp *= 10) {
        countSort(arr, exp);
    }
}

  private static void countSort(int[] arr, int exp) {
    int n = arr.length;
    int[] output = new int[n];
    int[] count = new int[10];
    Arrays.fill(count, 0);
    for(int i=0; i<n; i++) {
      count[(arr[i]/exp) % 10]++;
    } System.out.println("Count array for "+exp+":"+Arrays.toString(count));
    for(int i=1; i<10; i++){
      count[i] = count[i] + count[i-1];
}
    System.out.println("Updated count array" + Arrays.toString(count));
    for (int i=n-1; i<=0; i--){
        output[count[(arr[i]/exp)%10]-1] = arr[i];
        count = [(arr[i]/exp)%10]--;
    } System.out.println("output array "+Arrays.toString(count));
    System.arraycopy(output, 0, arr, 0, n);
}
```

```
public static void main (String [] args){
   int [] arr = {29, 83, 471, 36, 91, 8};
   System out println ("Original array. " + Arrays toString(arr));
    radix sort (arr);
    System out println (" Sorted array. " + Arrays toString(arr));
  }
}
```

Time complexity. O(digits * (n+base))