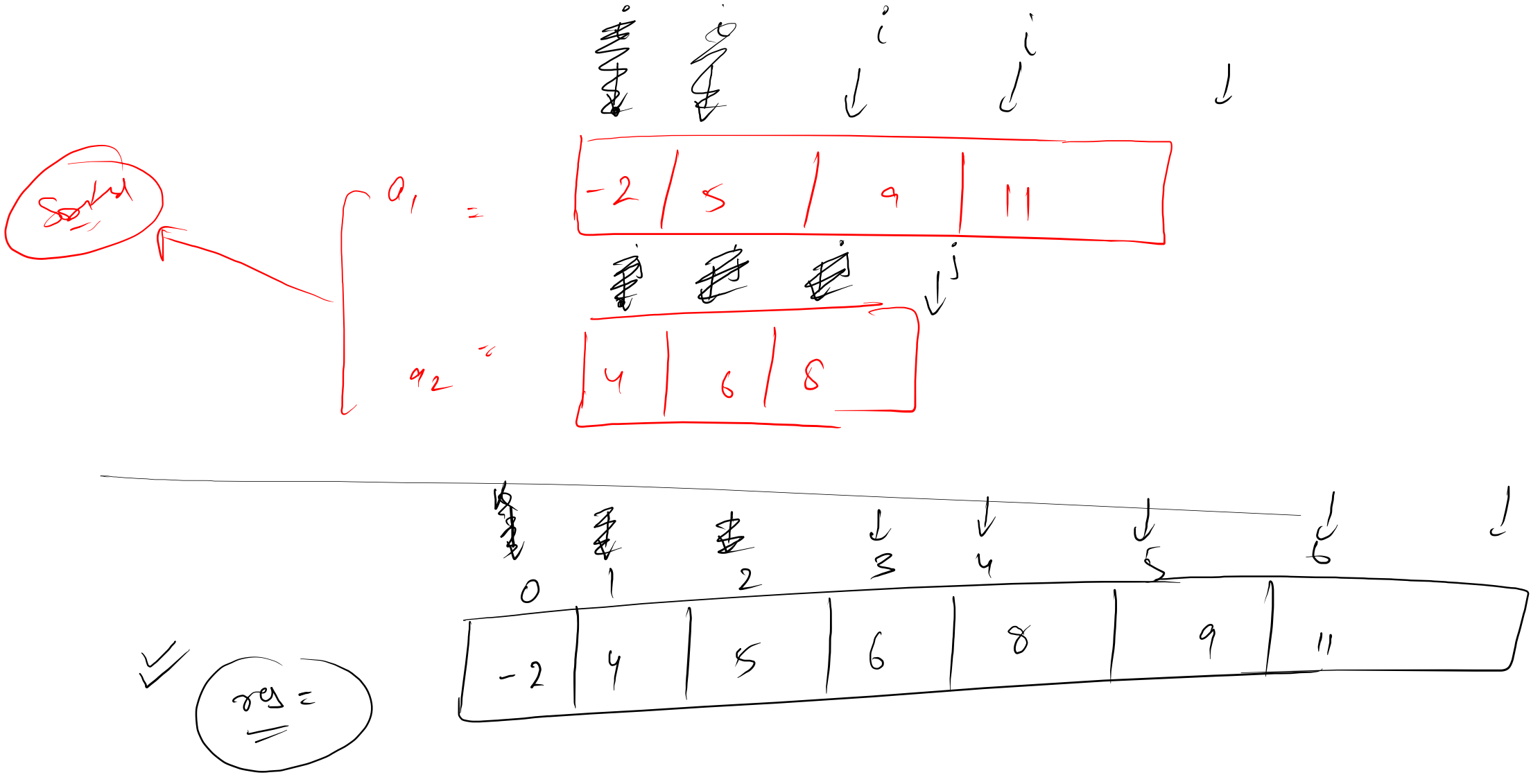
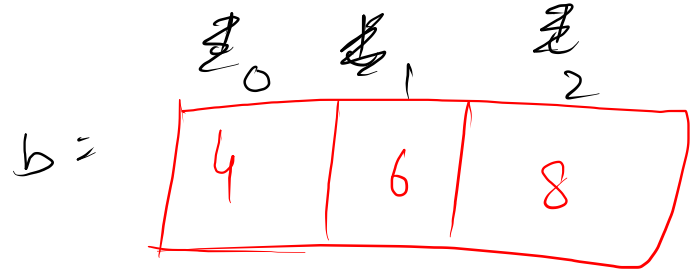

T.S
Recursive Analysis
Question

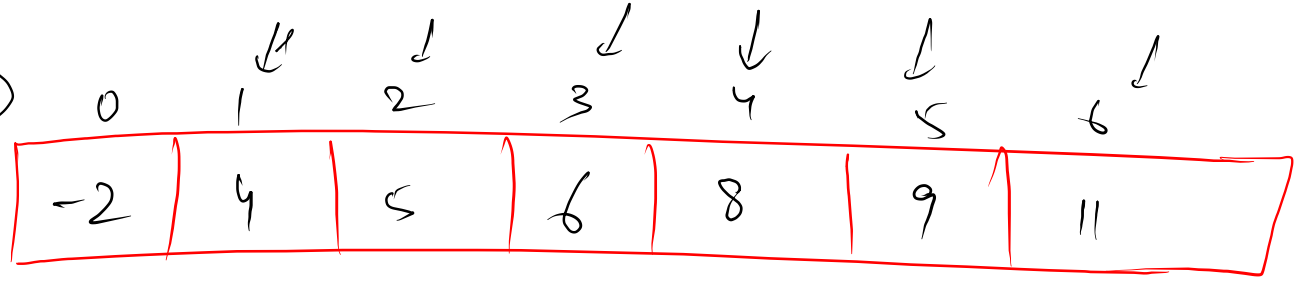

Solve
a_1 =
-2 | 5 | 9 | 11
a_2 =
4 | 6 | 8
0 1 2 3 4 5 6
-2 | 4 | 5 | 6 | 8 | 9 | 11
ans =

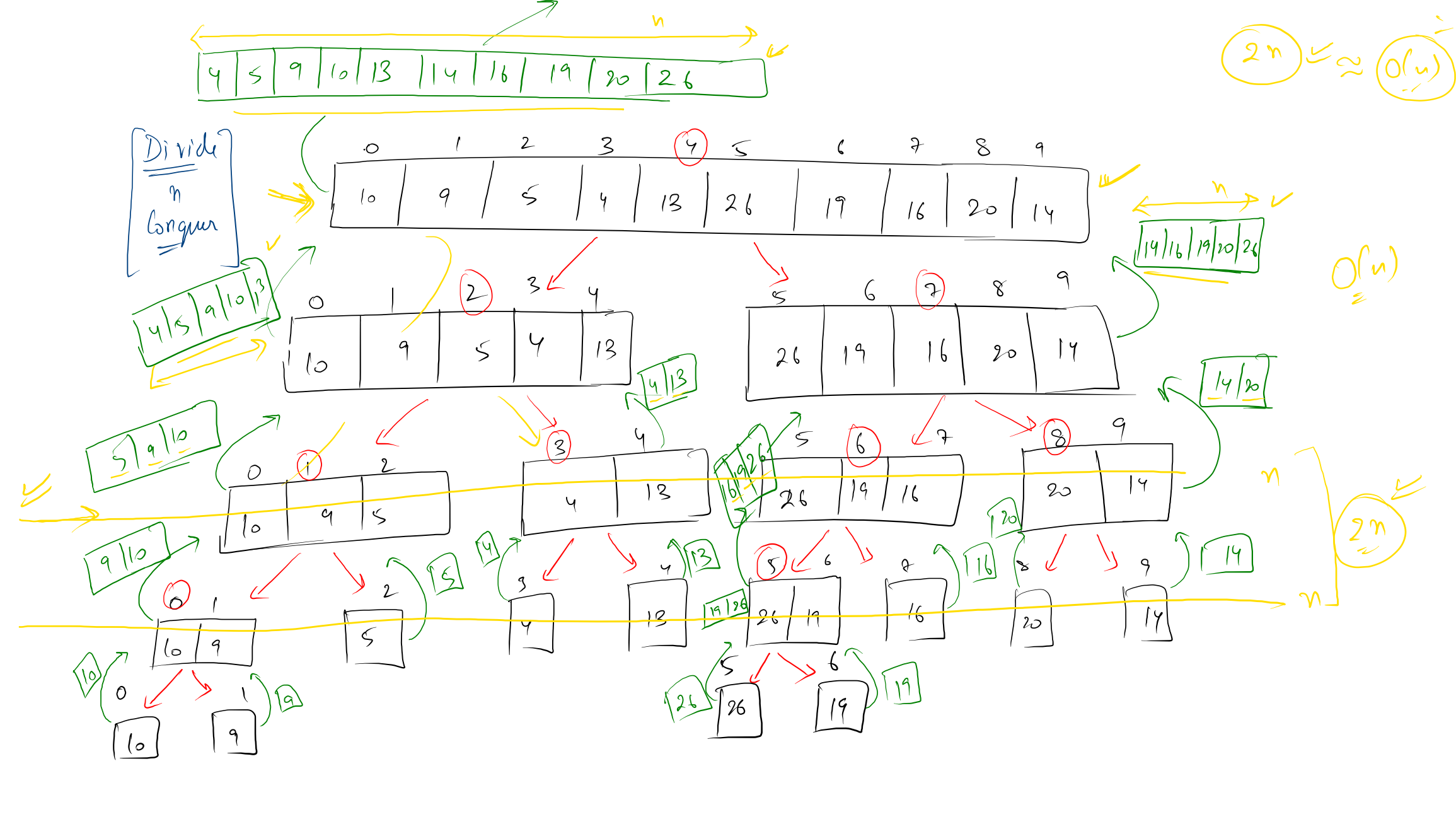
```java
int n1 = a.length , n2 = b.length;
int res[] = new int[n1+n2];

int i = 0 , j = 0, k = 0;

while(i < n1 && j < n2){
    if(a[i] <= b[j]){
        res[k] = a[i];
        i++;
        k++;
    }else{
        res[k] = b[j];
        j++;
        k++;
    }
}

while(i < n1){
    res[k] = a[i];
    i++;
    k++;
}

while(j < n2){
    res[k] = b[j];
    j++;
    k++;
}

return res;
```

$n_1 \Rightarrow 4$

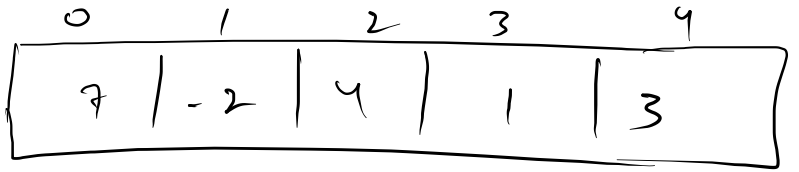


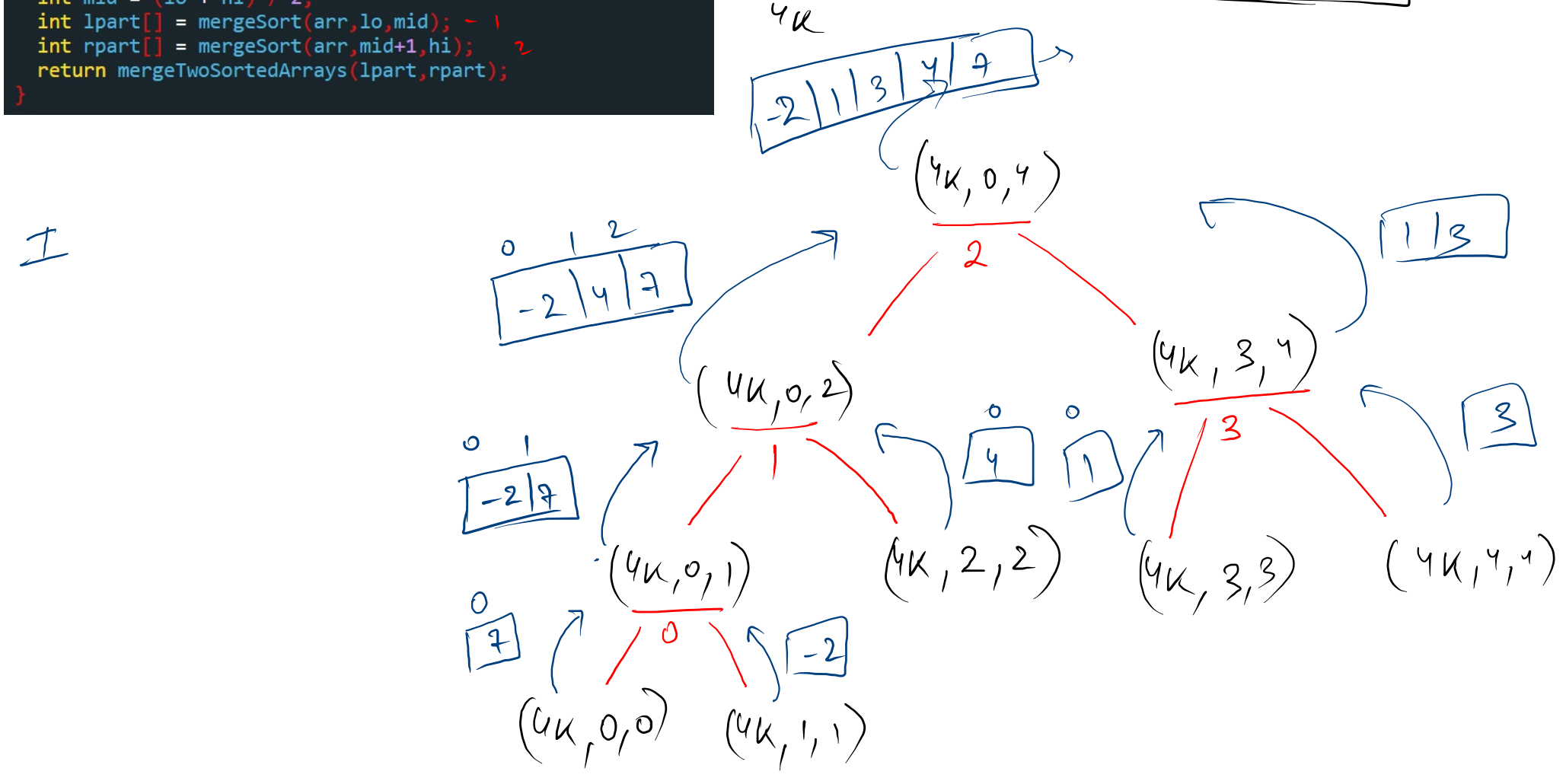
$n_2 = 3$

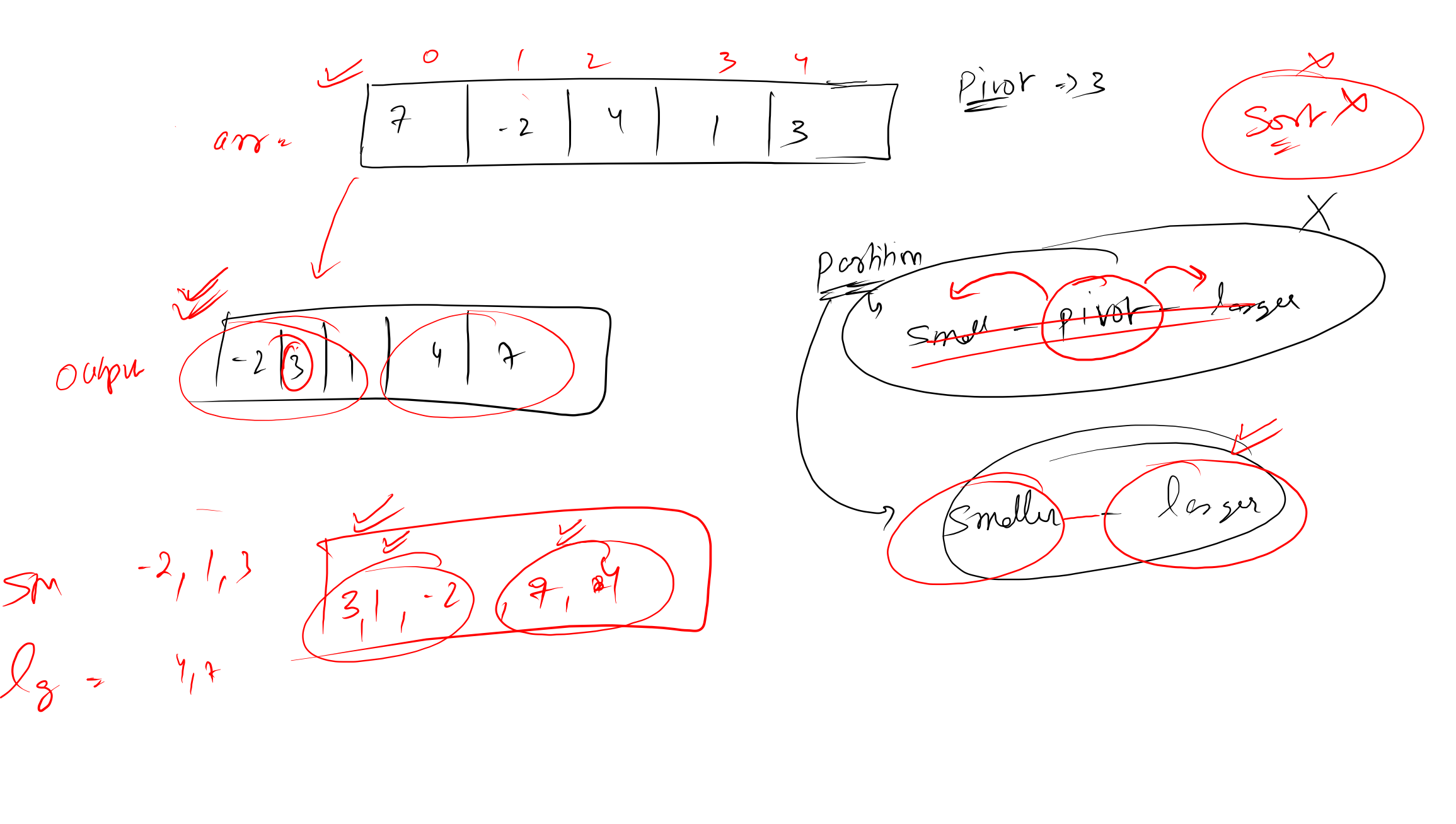



Divide
n
Conquer
2n ≈ O(n)
O(n)
2n

```java
public static int[] mergeSort(int[] arr, int lo, int hi) {
  if(lo == hi){
      return new int[]{arr[lo]};
  }
  int mid = (lo + hi) / 2;
  int lpart[] = mergeSort(arr,lo,mid);
  int rpart[] = mergeSort(arr,mid+1,hi);
  return mergeTwoSortedArrays(lpart,rpart);
}
```

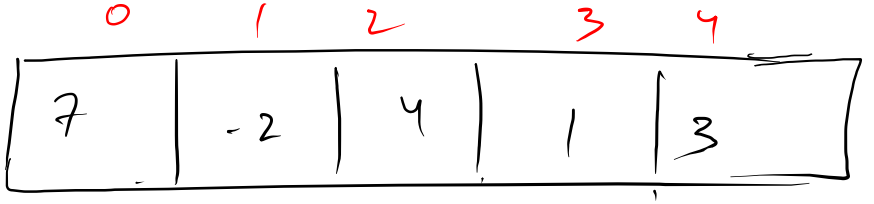





0 1 2 3 4
arr =
7 -2 4 1 3
Pivot => 3
Sort
partition
Smaller pivot larger
output
-2 3 1 4 7
Smaller larger
sm -2, 1, 3
lg = 4, 7
3, 1, -2
7, 4

What

arr =



pivot => 7

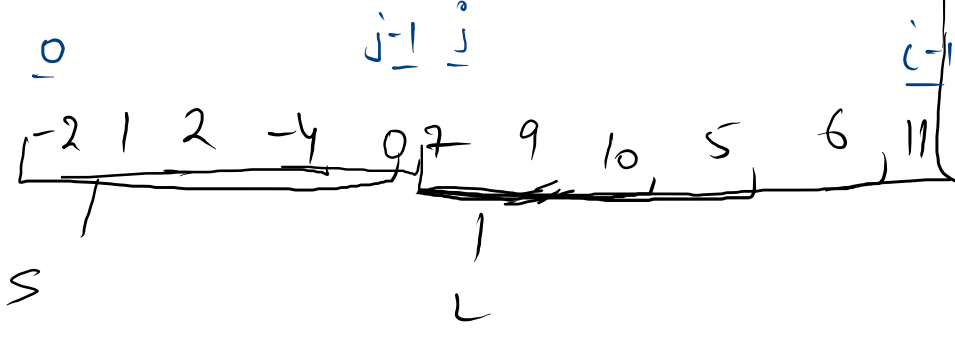
i j

0 — j-1 ⇒ smaller

j → i-1 ⇒ larger

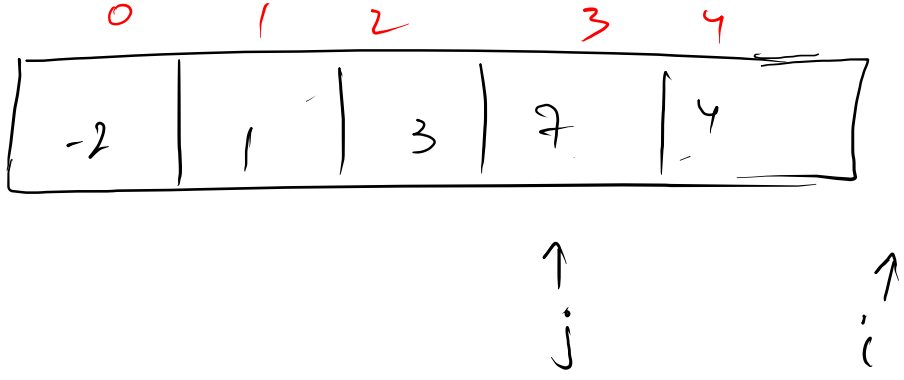
i → len-1 ⇒ unknown

```
if(a[i] <= pivot){
    swap(i,j)
    i++, j++
}else{

    i++
}
```

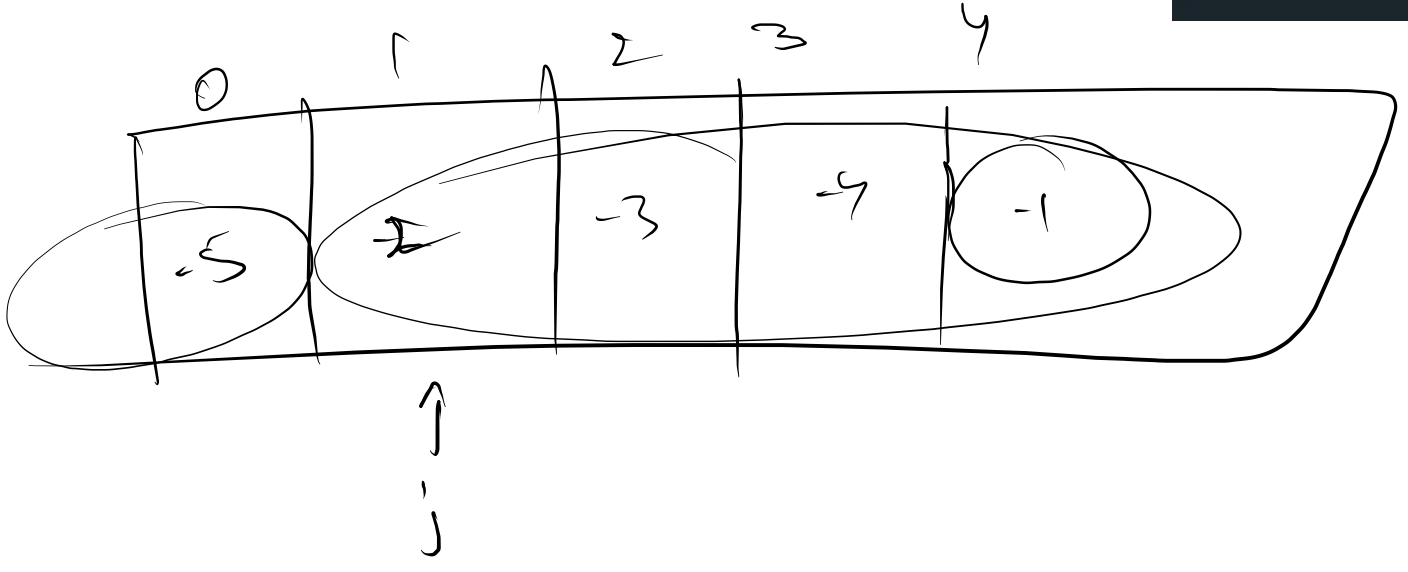


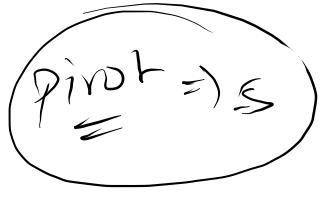
Pivot -> 3

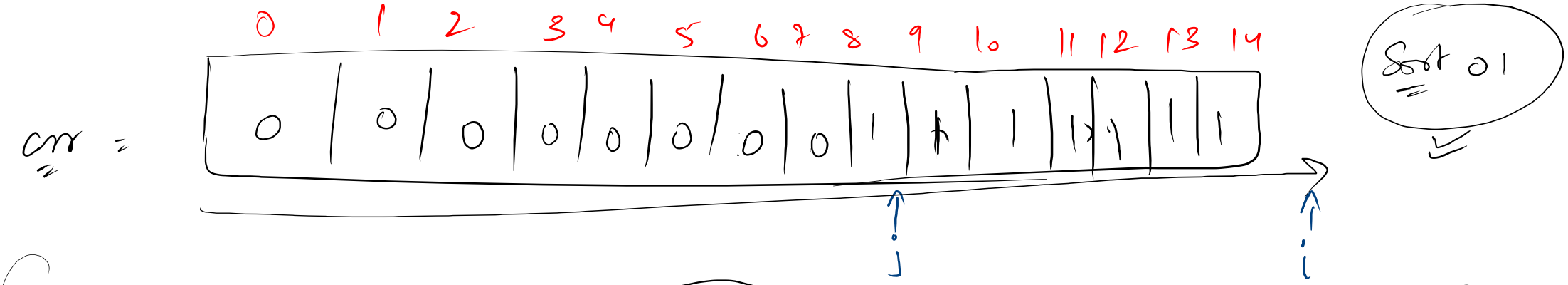
arr =

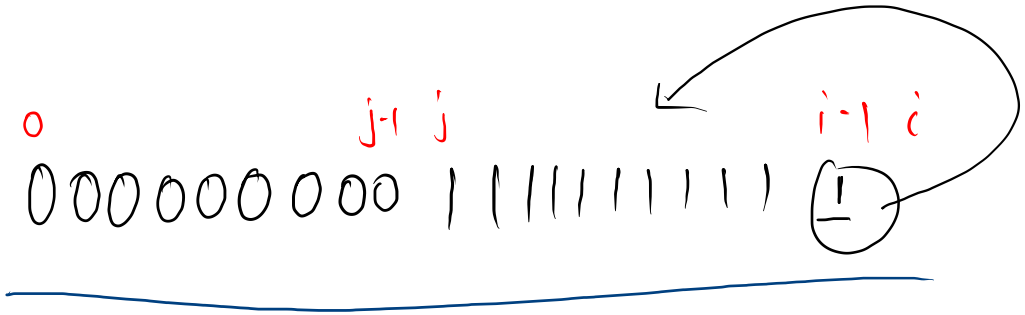


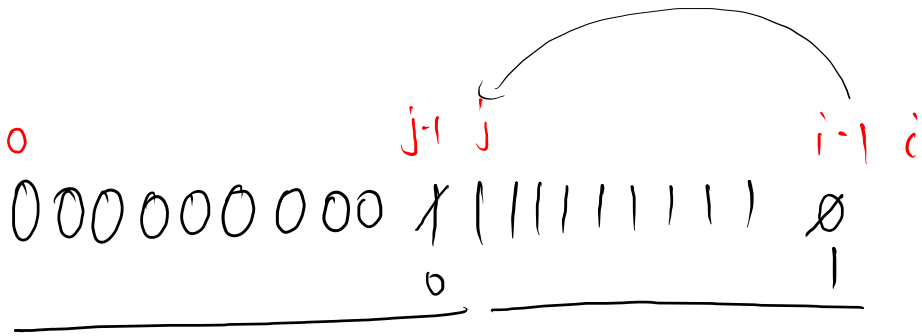

```
public static void partition(int[] arr, int pivot){
  int i = 0  , j = 0;

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












```
if(a[i] == 1) {
    i++;
} else {
    swap(i,j)
    i++
    j++
}
```

H.W. → Discuss

arr =

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2

j k i

0 → (j-1) ⇒ 0's

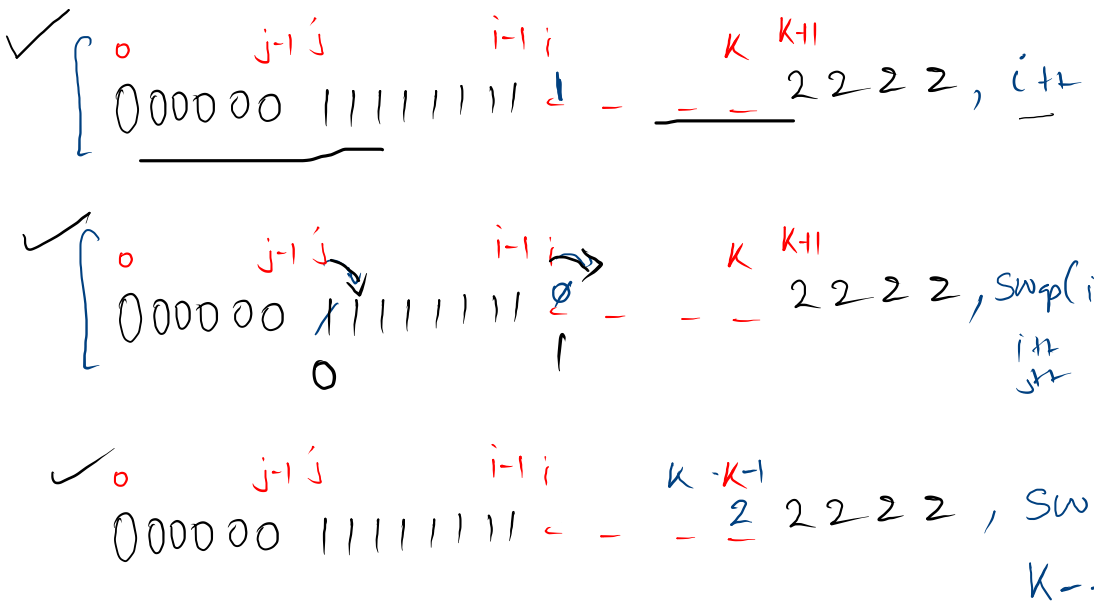
j → (i-1) ⇒ 1's

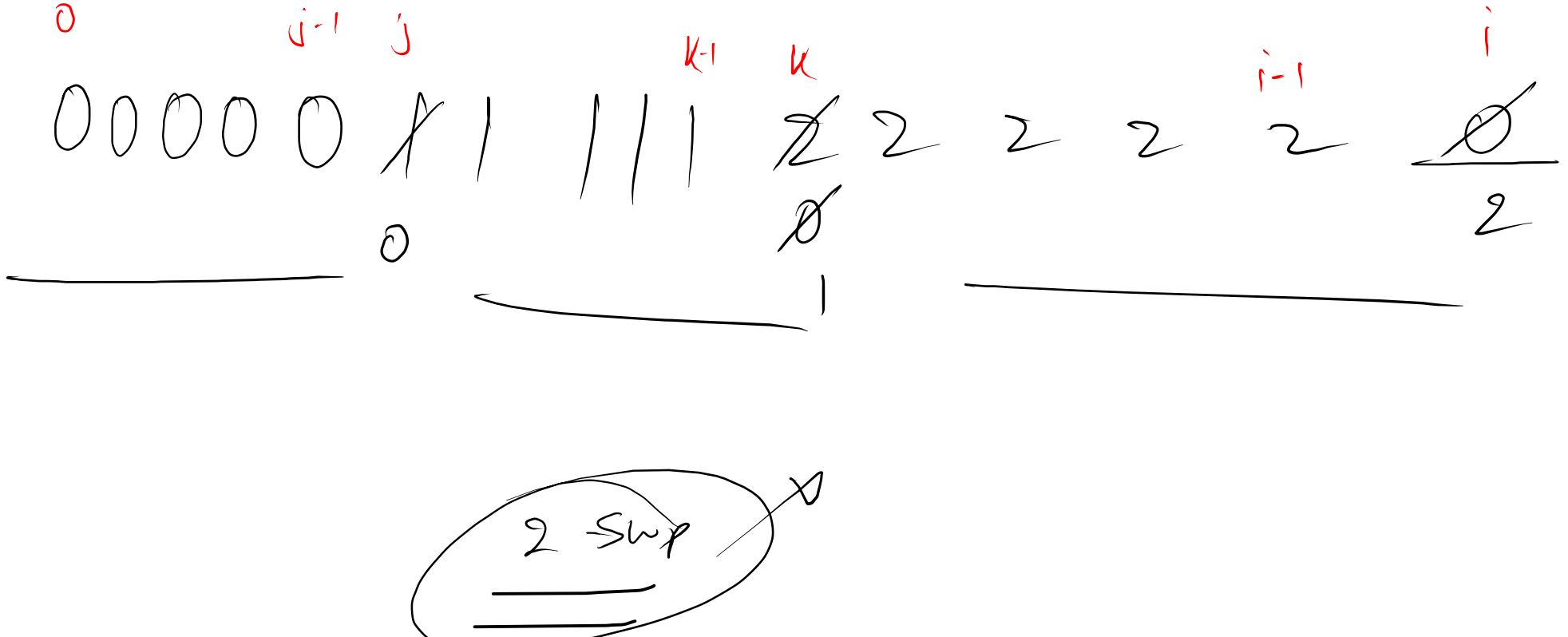
(i → k) ⇒ (unknown)

k+1 → len-1 ⇒ 2's

```
while (i<=k){
    if(a[i]==0){
        swap(i,j)
        i++
        j++
    }else if(a[i]==2){
        swap(i,k);
        k--;
    }else{
        i++
    }
}
```

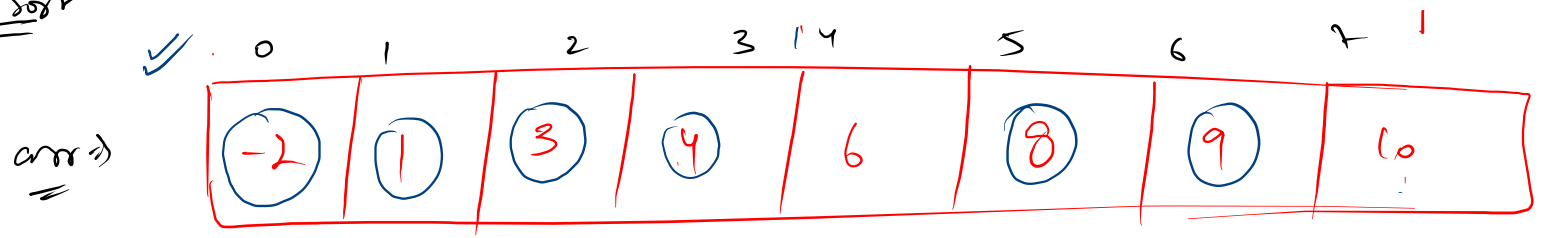
i = 0
j = 0
k = len-1




0
j-1
j
k-1
k
i-1
i
0 0 0 0 0 1 1 1 1 2 2 2 2 2 0
0
1
2

2 swp

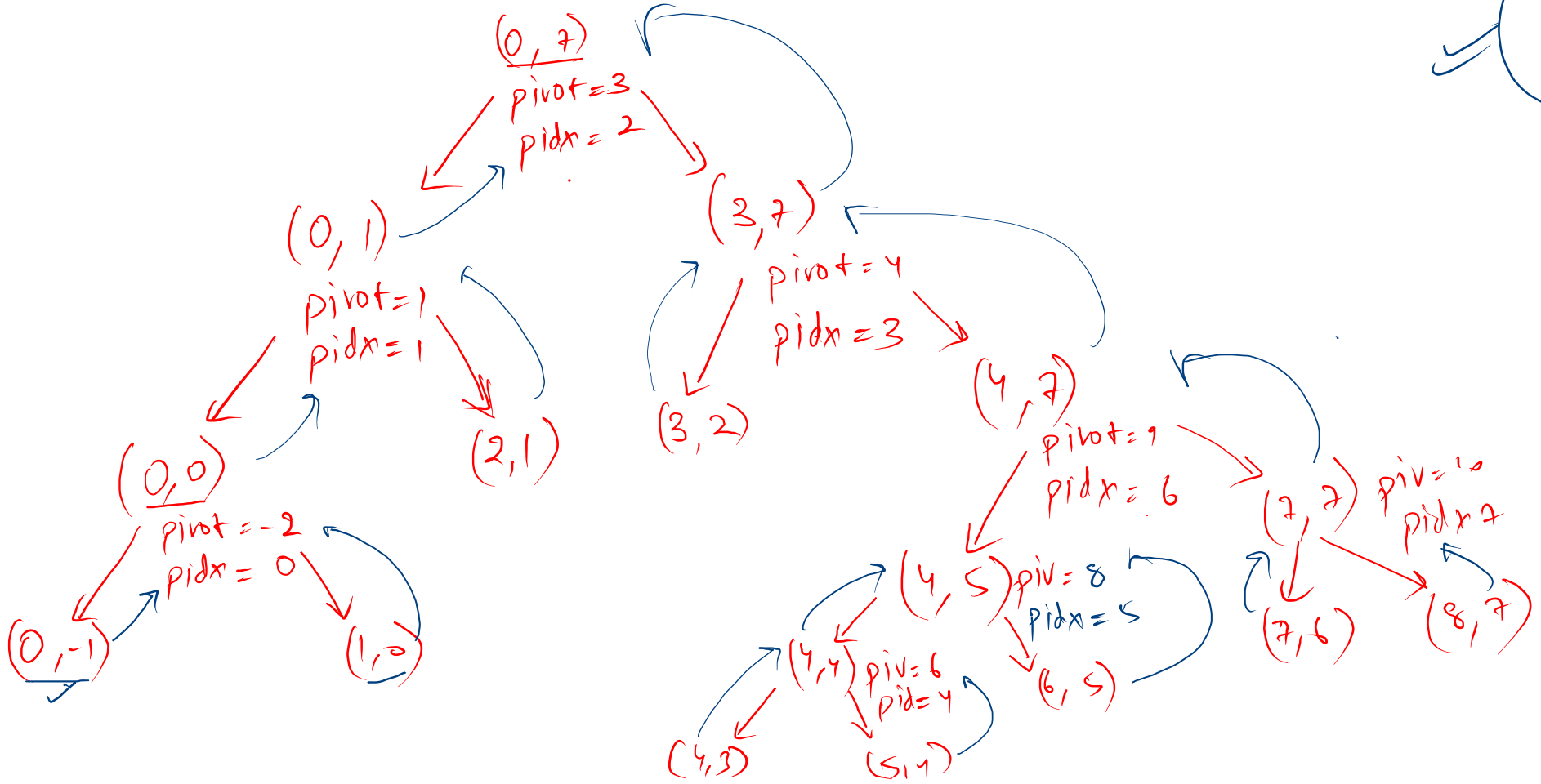
Quick Sort

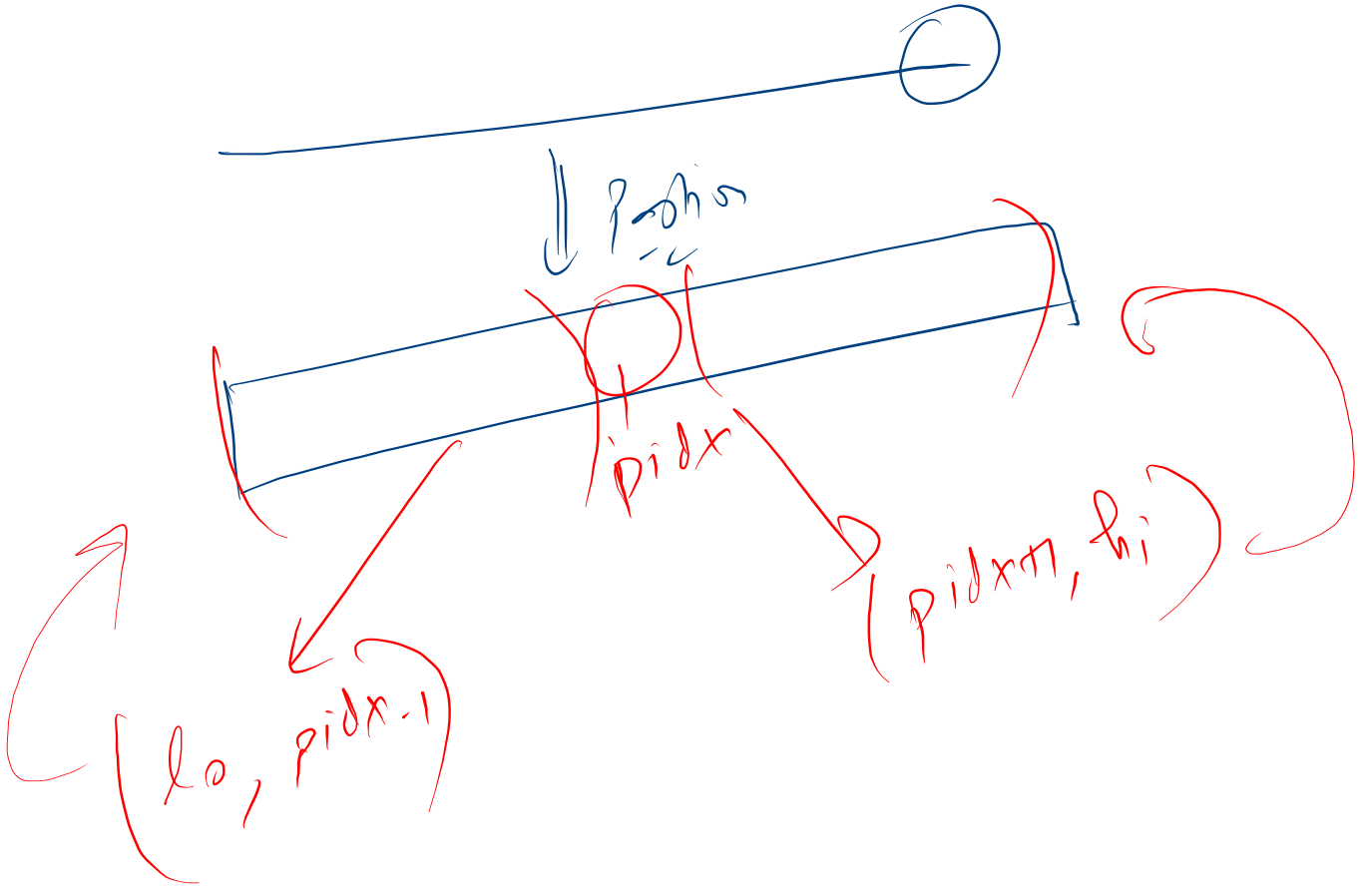


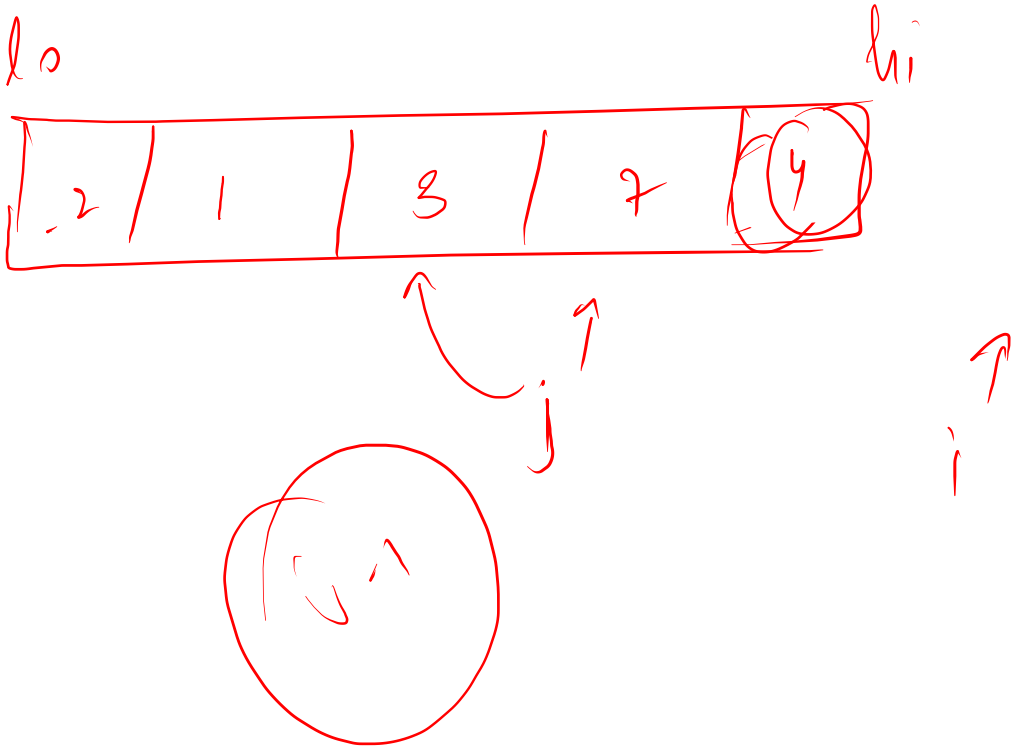
pivot = 3

pidx = 2

Inplace




pidx
(lo, pidx-1)
(pidx+1, hi)


lo
hi
2
1
3
7
4
j
i
j-1

```java
public static int[] mergeSort(int[] arr, int lo, int hi) {
    if(lo == hi){
        return new int[]{arr[lo]};
    }
    int mid = (lo + hi) / 2;
    int lpart[] = mergeSort(arr,lo,mid);
    int rpart[] = mergeSort(arr,mid+1,hi);
    return mergeTwoSortedArrays(lpart,rpart);
}
```

$O(1)$

$n/2$ $T(n/2)$

$n/2$

$T(n/2)$

Recurrence Equation

$$T(n) = T(n/2) + T(n/2) + n$$

$$T(n) = 2T(n/2) + n$$

Let's assume total n elements takes $T(n)$ time to be sorted using merge sort

So according to this assumption $n/2$ elements should take $T(n/2)$

K times

$T(n) = 2 \cdot T(n/2) + n$

$2 \cdot T(n/2) = 4 \cdot T(n/4) + n$

$4 \cdot T(n/4) = 8 \cdot T(n/8) + n$

$T(1) \Rightarrow$

$T(n) \Rightarrow n + n + n + \ldots\ldots + n$ (K terms)

$T(n) = n(k)$ — ①

$n \to n/2^0$

$n/2 \to n/2^1$

$n/4 \to n/2^2$

$1 \to n/2^{k-1}$

$1 = \frac{n}{2^{k-1}}$

$2^{k-1} = n$

taking log both sides

$k-1 = \log_2 n$

$k \Rightarrow \log_2 n + 1$ — ②

from ① & ②

$T(n) = n(\log_2 n + 1)$

$T(n) = n \log_2 n + n$

T.C. $= O(n \log_2 n) = \Omega(n \log_2 n)$

S.C. $\Rightarrow O(n)$

```java
public static void printDecreasing(int n) {
    if (n == 0) {
        return;
    }
    System.out.println(n); // my work
    printDecreasing(n - 1); //  faith
}
```

$$T(n) = T(n-1) + 1$$

$T(n) = T(n-1) + 1$

$T(n-1) = T(n-2) + 1$

$T(n-2) = T(n-3) + 1$

$\vdots$

$T(0) \Rightarrow 1$

nth terms

$T(n) \Rightarrow \underbrace{1+1+1+1+1+\ldots+1}_{n+1}$

$T(n) \Rightarrow n+1$

$T.C. \Rightarrow O(n)$

```java
public static void pzz(int n) {
    if (n == 0) {
        return;
    }
    System.out.print(n + " ");
    pzz(n - 1);
    System.out.print(n + " ");
    pzz(n - 1);
    System.out.print(n + " ");

}
```

$$T(n) = 2T(n-1) + 1$$

n+1 levels

$T(n) = 2 \cdot T(n-1) + 1$

$2 \cdot T(n-1) = 4 \cdot T(n-2) + 2$

$4 \cdot T(n-2) = 8 \cdot T(n-3) + 4$

.
.
.

$T(0) =$

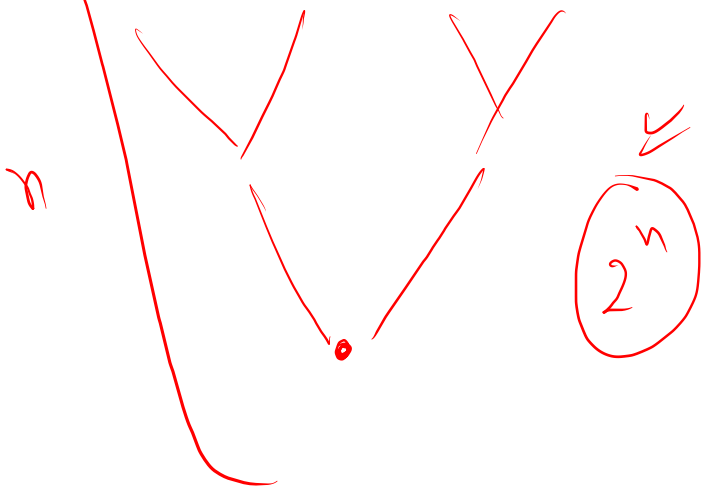
$T(n) = 1 + 2 + 4 + \cdots$

$T(n) = 1\left[\frac{2^{n+1} - 1}{2 - 1}\right]$

$T(n) = 2^{n+1} - 1$

$T.C. \Rightarrow O(2^n)$

```
public static int fibo(int n){
    if(n ==1 || n==2){
        return n-1;
    }

    int  fibNm1 = fibo(n-1);
    int fibNm2 = fibo(n-2);
    int fiboN = fibo(n-1) + fibo(n-2);
    return fiboN;
}
```



$T(n) = T(n-1) + T(n-2) + 1$

$T(n-2) < T(n-1)$

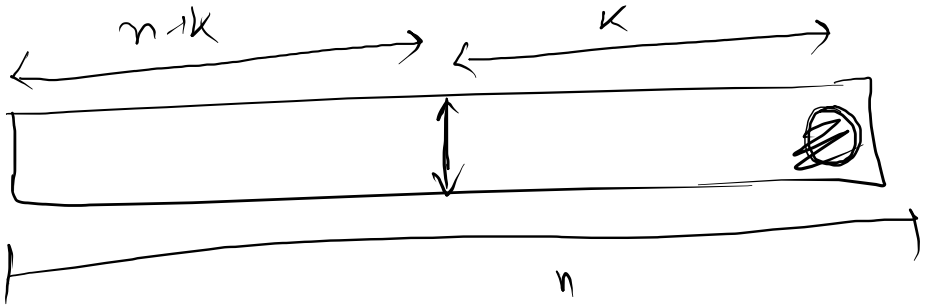
$T(n-2) + T(n-1) < T(n-1) + T(n-1)$

$T(n) = T(n-1) + T(n-2) + 1 < T(n-1) + T(n-1) + 1$

$T(n) < 2 \cdot T(n-1) + 1$

$T.C. = 2^n$

```java
public static void quickSort(int[] arr, int lo, int hi) {
  if(lo > hi){
      return;
  }
  int pivot = arr[hi];
  int pidx = partition(arr,pivot,lo,hi);
  quickSort(arr,lo,pidx-1);
  quickSort(arr,pidx+1,hi);
}
```



$T(n) = n + T(n-k) + T(k)$

$K \Rightarrow n/2$

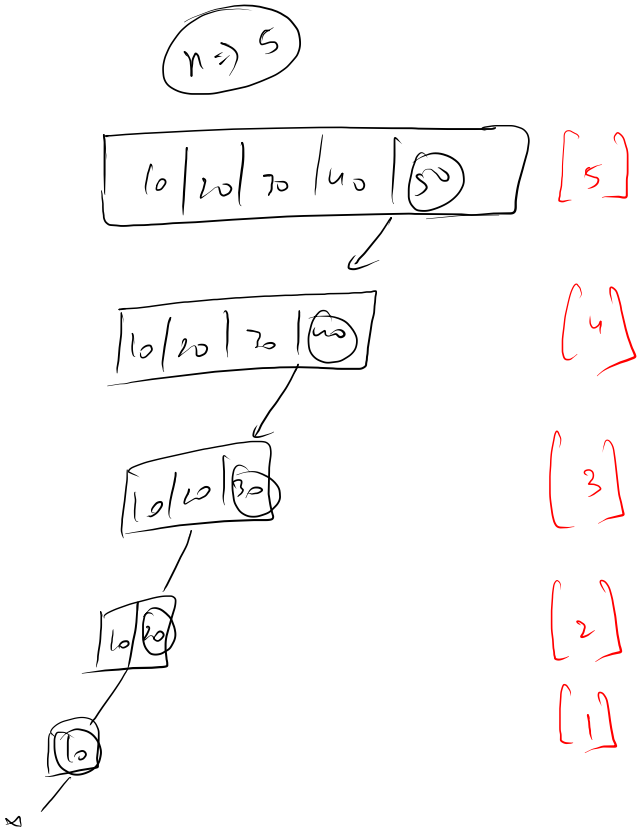
average / best

$T(n) = n + T(\frac{n}{2}) + T(n/2)$

$T(n) = 2 \cdot T(n/2) + n$

$n \log n + n$

$T.C. \Rightarrow O(n \log n)$



[5]

[4]

[3]

[2]

[1]

$T(n) \Rightarrow n + (n-1) + (n-2) + (n-3) + (n-4) \dots + 1$

$f(n) \Rightarrow \frac{n(n+1)}{2}$

$T.C. \Rightarrow O(n^2)$

$T(n) = n + T(n-4) + T(k)$

$T(n) = n + T(n-1)$

$T(n) = n + T(n-1)$

$T(n-1) = n-1 + T(n-2)$

$T(n-2) = (n-2) + T(n-3)$

$T(1) = (1) + T(0)$

$T(0) =$

$T(n) \Rightarrow n + (n-1) + (n-2) + (n-3) + \dots + 1$

$T(n) \Rightarrow \frac{n(n+1)}{2}$

$T.C. \Rightarrow O(n^2)$

MS

S.C. → O(n)

O/Ω/Θ → (n log n)

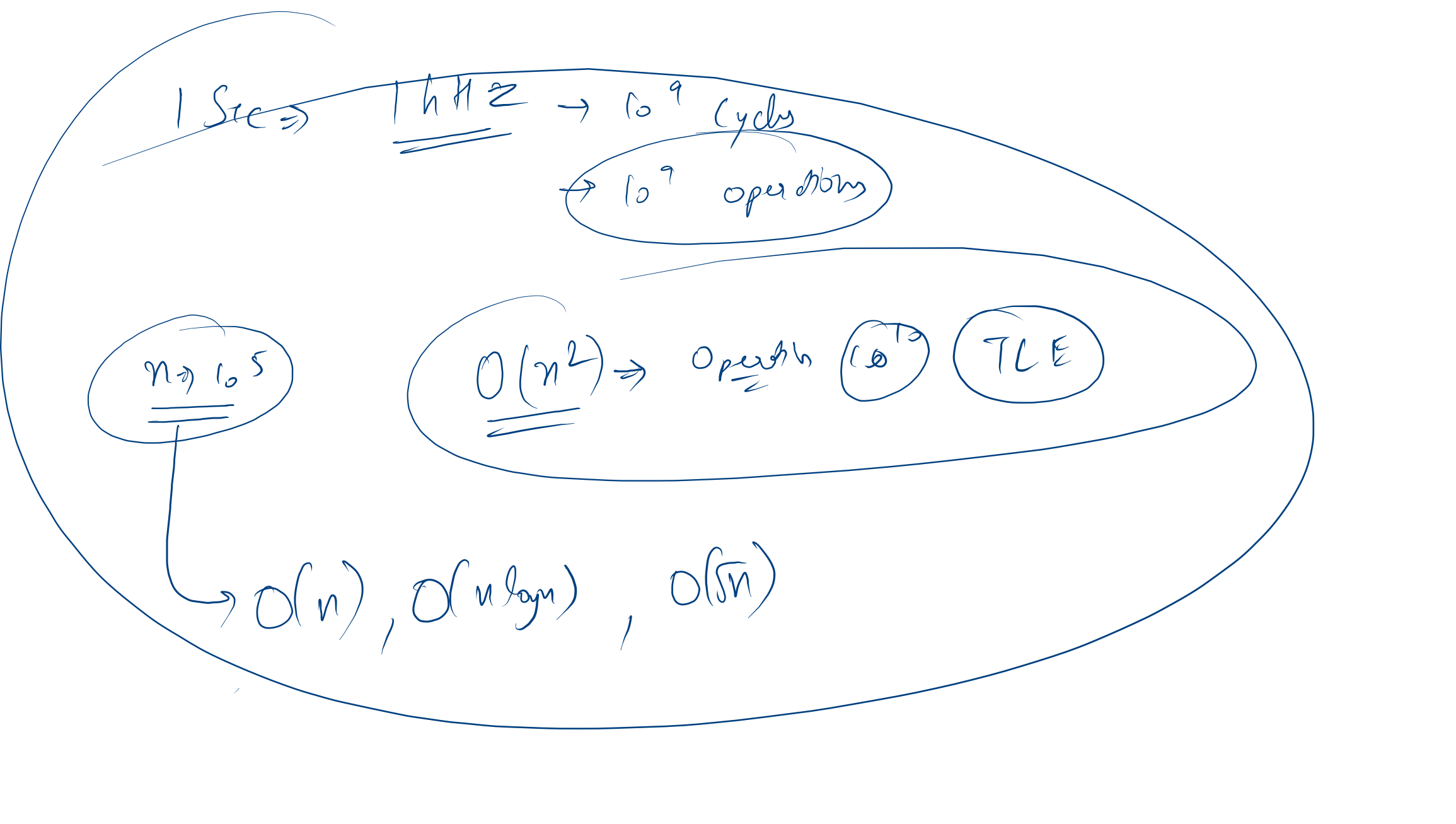
QS

S.C. → O(1)

Inplace

O/Ω → (n log n)

O → (n²)


1 Sec ⇒ 1 GHz → 10^9 cycles
→ 10^9 operations
$n \Rightarrow 10^5$
$O(n^2) \Rightarrow$ operations 10^9 TLE
$O(n)$, $O(n \log n)$, $O(\sqrt{n})$