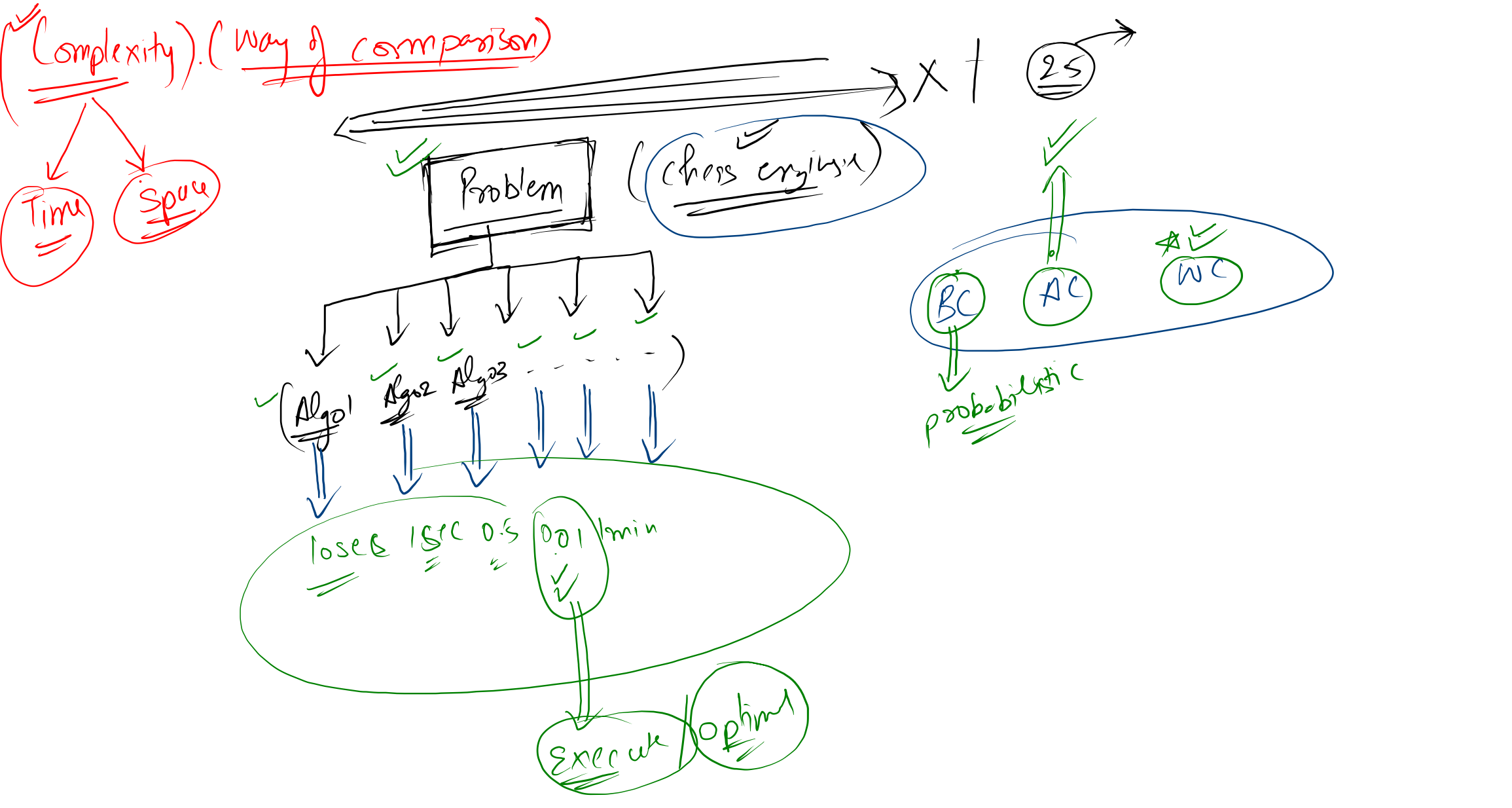
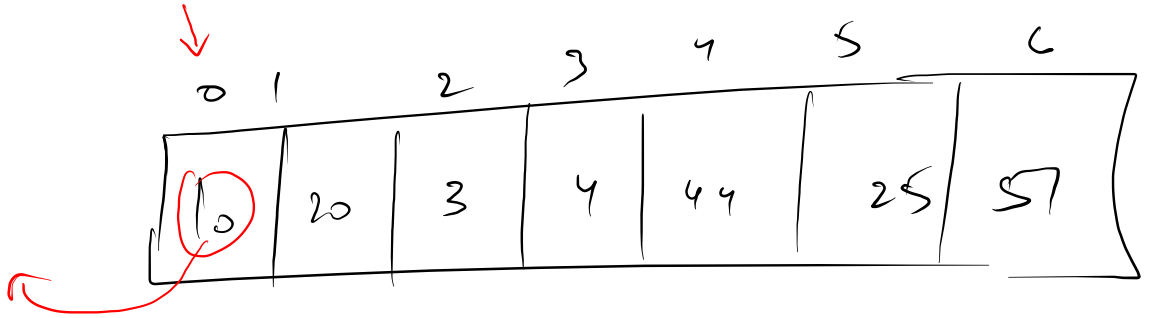

(complexity).(way of comparison)
Time
Space
Problem
(chess engine)
X
25
Algo1
Algo2
Algo3
loses 1sec 0.5
0.01
1min
Execute
optimal
BC
AC
WC
probabilistic

Linear Search

find

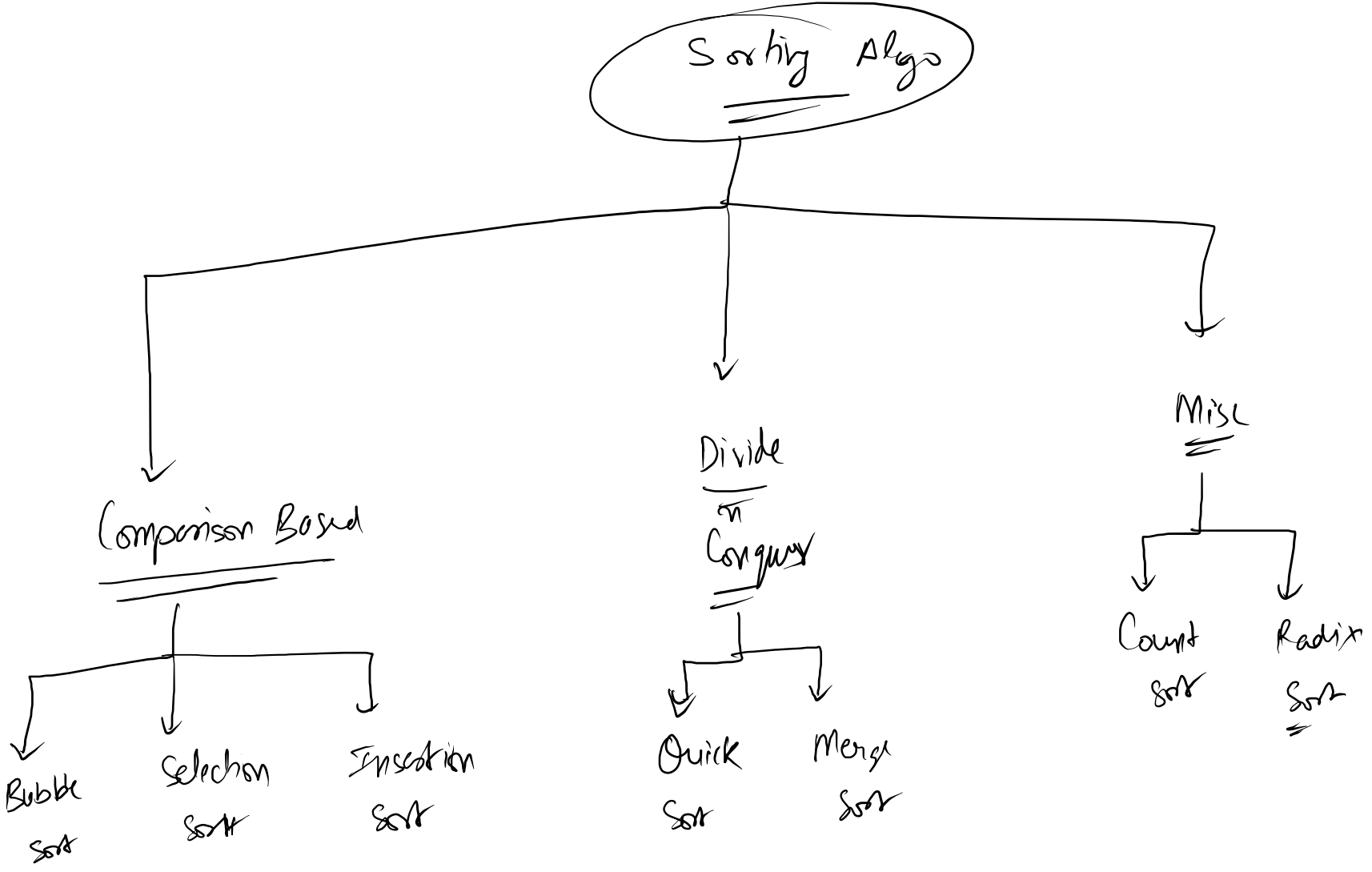


Best Case ⇒ X = 10

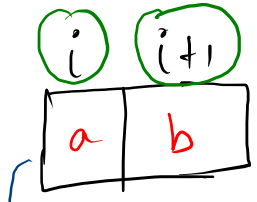
T.C. ⇒ 1 → T.C. ⇒ $\Omega(1)$ ⇒ Constant time

Worst Case ⇒ X = 100

T.C. ⇒ n → T.C. ⇒ $O(n)$

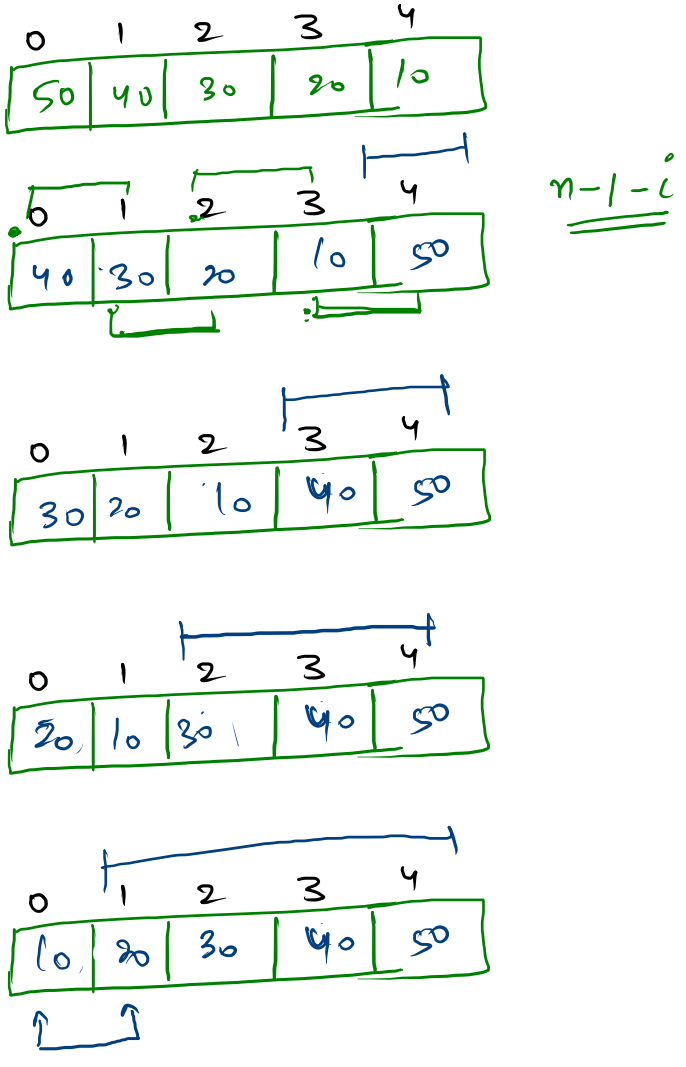

Sorting Algo
Comparison Based
Divide
or
Conquer
Misc
Bubble
Sort
Selection
Sort
Insertion
Sort
Quick
Sort
Merge
Sort
Count
Sort
Radix
Sort

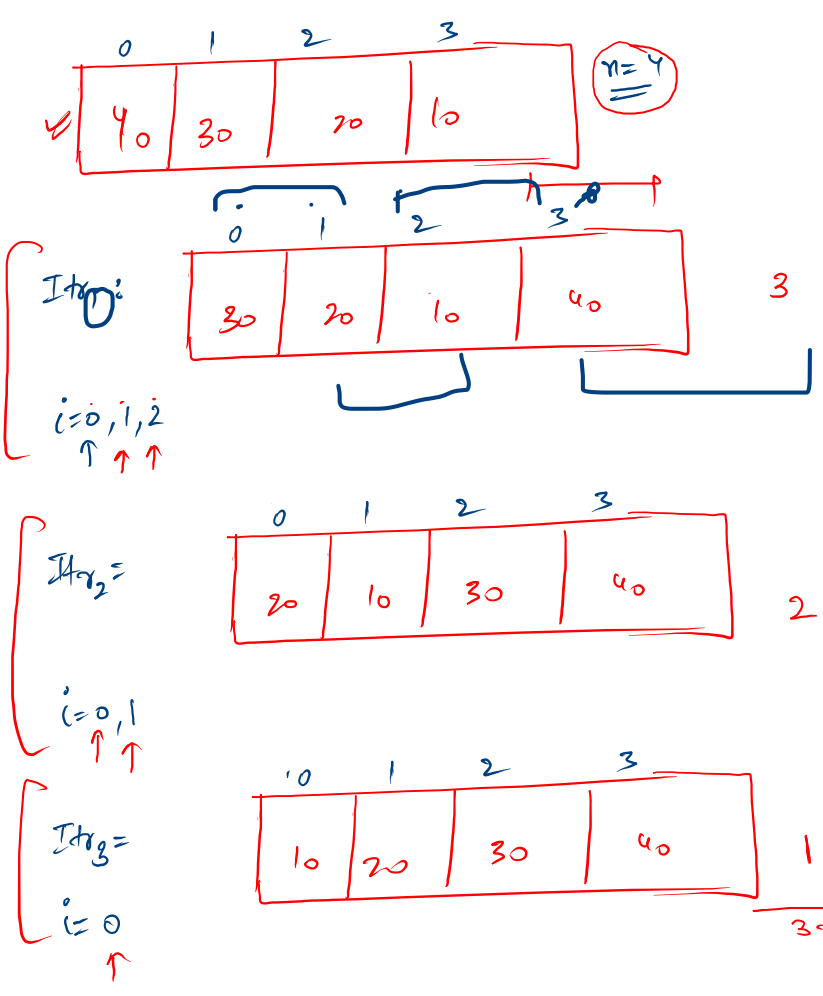
Bubble sort → Output [data → sort → inc order]



[a <= b → do nothing

a > b → swap]






```java
public static void bubbleSort(int[] arr) {
  int n = arr.length;
  for(int itr = 1 ; itr <= n-1 ; itr++){
      for(int i = 0 ; i <= n-itr-1 ; i++){
          // arr[i] arr[i+1]
          if(isSmaller(arr,i+1,i)){
              swap(arr,i+1,i);
          }
      }
  }

}
```

→ O(1)

```java
// return true if ith element is smaller than jth element
public static boolean isSmaller(int[] arr, int i, int j) {
  System.out.println("Comparing " + arr[i] + " and " + arr[j]);
  if (arr[i] < arr[j]) {
    return true;
  } else {
    return false;
  }
}
```

$O(n^2)$

$\{ n^2+n+1,\ \frac{n^2+n}{2},\ n^2+\log n,\ \frac{n^2-n}{2} \ldots \}$

T.O. :) $(n-1)+(n-2)+(n-3)+\ldots\ldots+3+2+1$

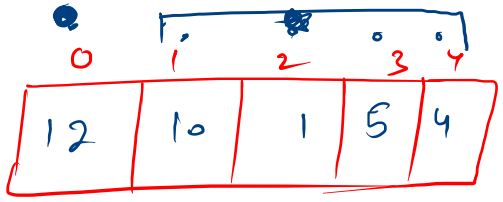
Actual hm

T.O. $\Rightarrow \frac{n(n-1)}{2} \Rightarrow \frac{n^2-n}{2}$

T.O. $\propto n^2$

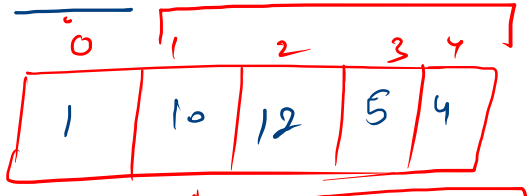
T.C. $\Rightarrow O(n^2)$ → Approximation

Selection Sort



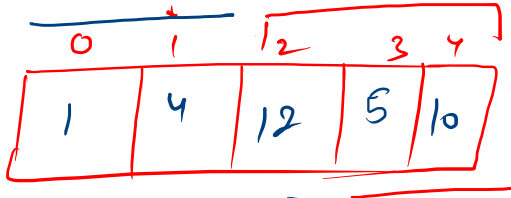
minIdx = ✗ 2

I_1 :



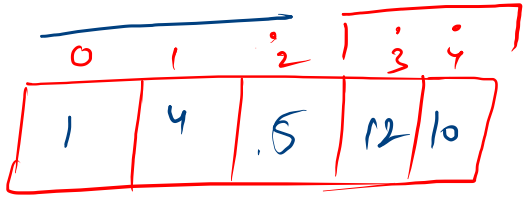
4

I_2 :



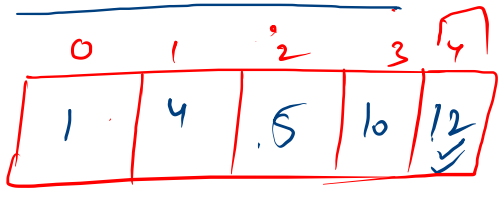
3

I_3 :



2

I_4 :



1

min = ✗ 4

```
int n = arr.length;
for(int itr = 1 ; itr <= n-1 ; itr++){
    int minIdx = itr-1;
    for(int i = itr ; i < n ; i++){
        if(isSmaller(arr,i,minIdx)){
            minIdx = i;
        }
    }
    swap(arr,itr-1,minIdx);
}
```

O(1)

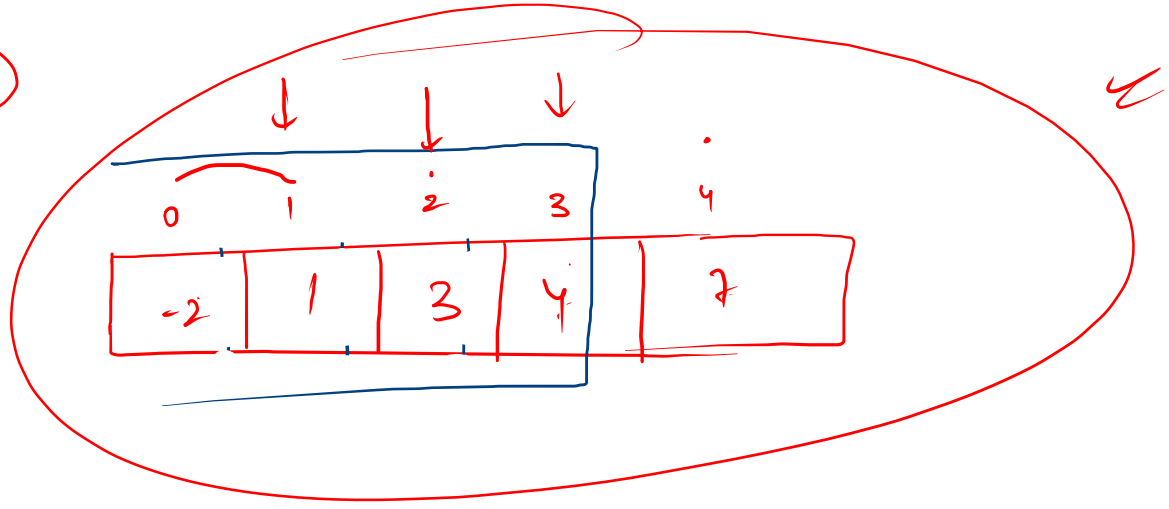
O(1)

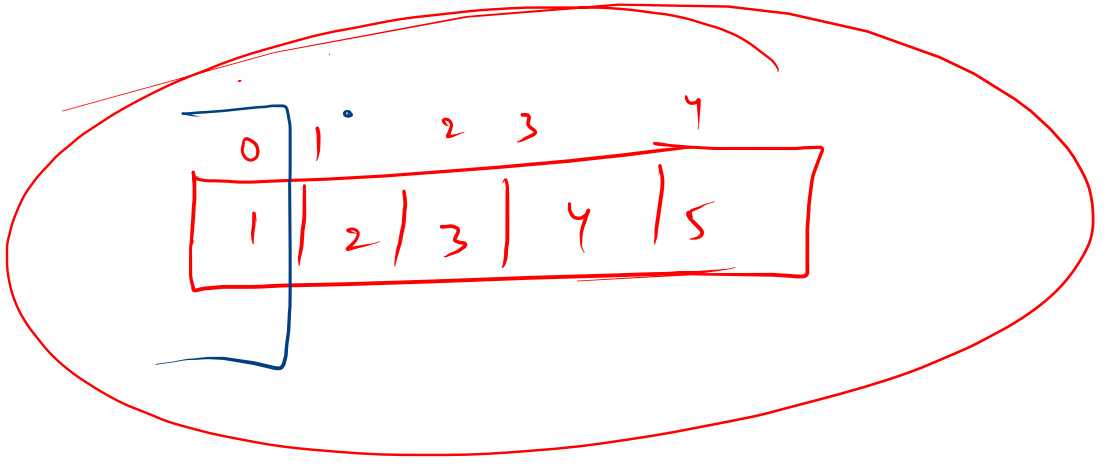
T.O. $\Rightarrow (n-1)+(n-2)+(n-3)+\ldots+3+2+1$

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

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

7 -2 4 1 3





$T(n) \Rightarrow \Omega(n)$

```java
int n = arr.length;
for(int itr = 1; itr <= n-1 ; itr++){
    for(int i = itr ; i > 0 ; i--){
        if(isGreater(arr,i-1,i)){
            swap(arr,i-1,i);
        }else{
            break;
        }
    }
}
```

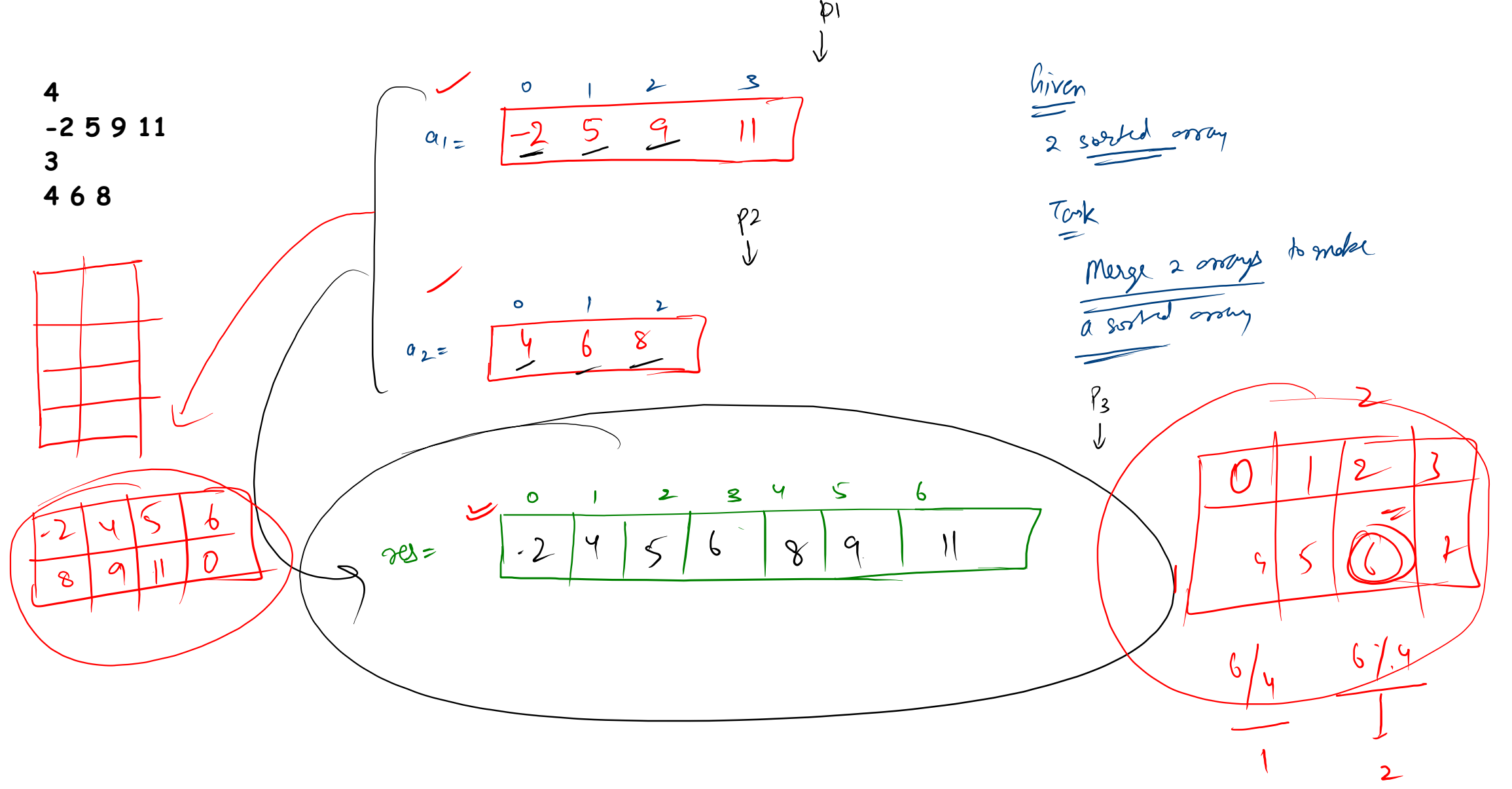
n-1 times

Constant

B.C. → T.O. → n-1 , T.C. → Ω(n)

W.C. → T.O. → $\frac{n(n-1)}{2}$, T.C. → $O(n^2)$

4
-2 5 9 11
3
4 6 8



Given

2 sorted array

Task

Merge 2 arrays to make a sorted array

```java
public static int[] mergeTwoSortedArrays(int[] a, int[] b){
    int res[] = new int[a.length + b.length];
    int p1=0 , p2=0 , p3=0;

    while(p1 < a.length && p2 < b.length){
        if(a[p1] < b[p2]){
            res[p3] = a[p1];
            p1++;
            p3++;
        }else{
            res[p3] = b[p2];
            p2++;
            p3++;
        }
    }

    while(p2 < b.length){
        res[p3] = b[p2];
        p2++;
        p3++;
    }

    while(p1 < a.length){
        res[p3] = a[p1];
        p1++;
        p3++;
    }

    return res;
}
```

a ⇒ n elements

b ⇒ m elements

n+m

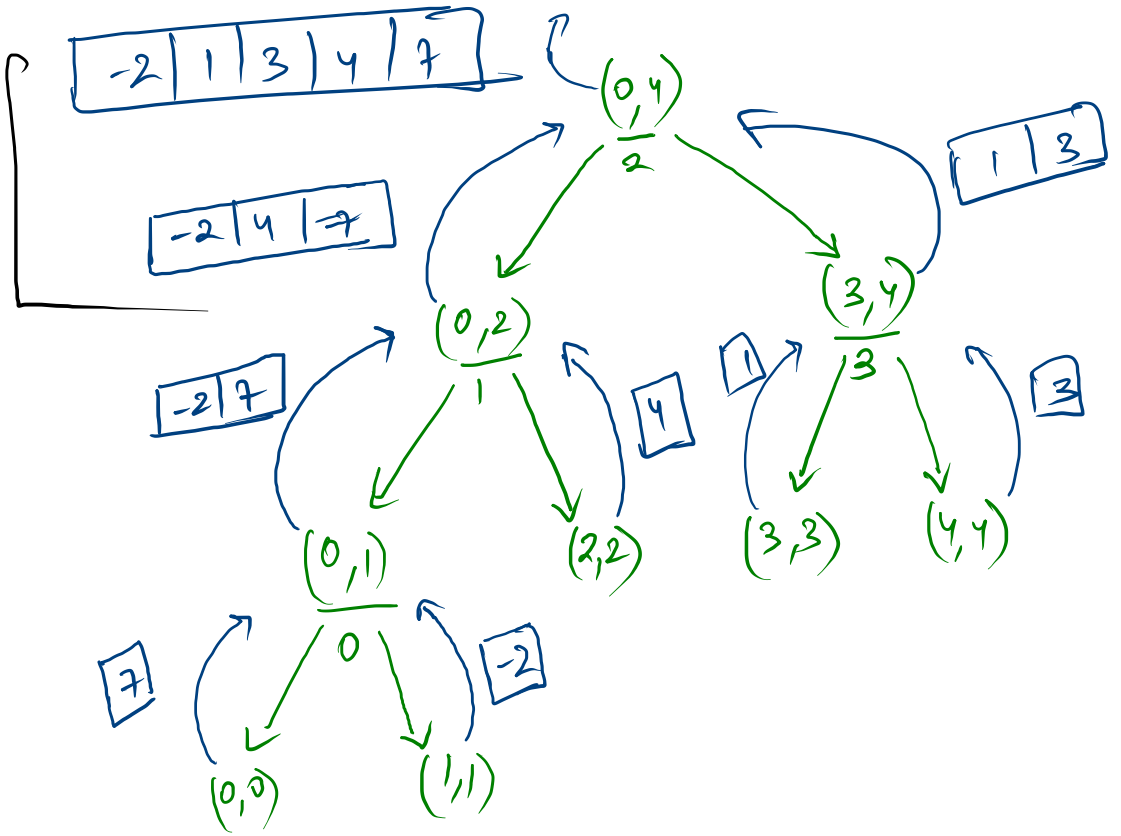
T.C. → O(n+m)

S.C. → O(n+m)

T.S = T(2,3)

arr =

0	1	2	3	4
7	-2	4	1	3



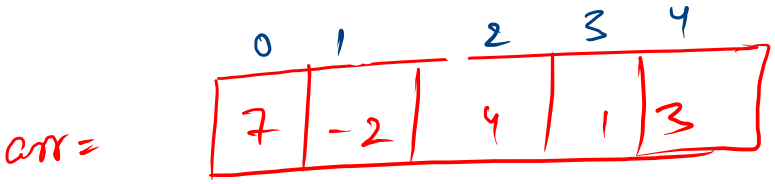
(lo, hi)

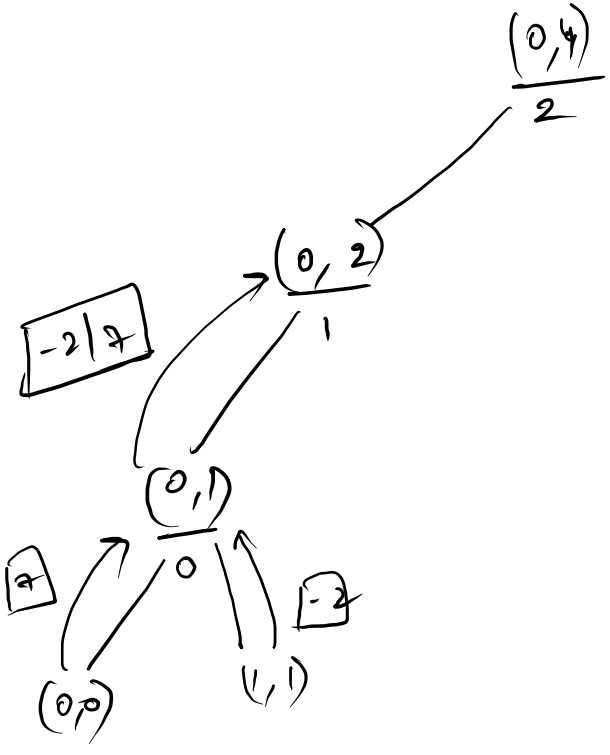
mid

(lo, mid) (mid+1, hi)

T.C. => O(nlogn)

S.C. => O(n)





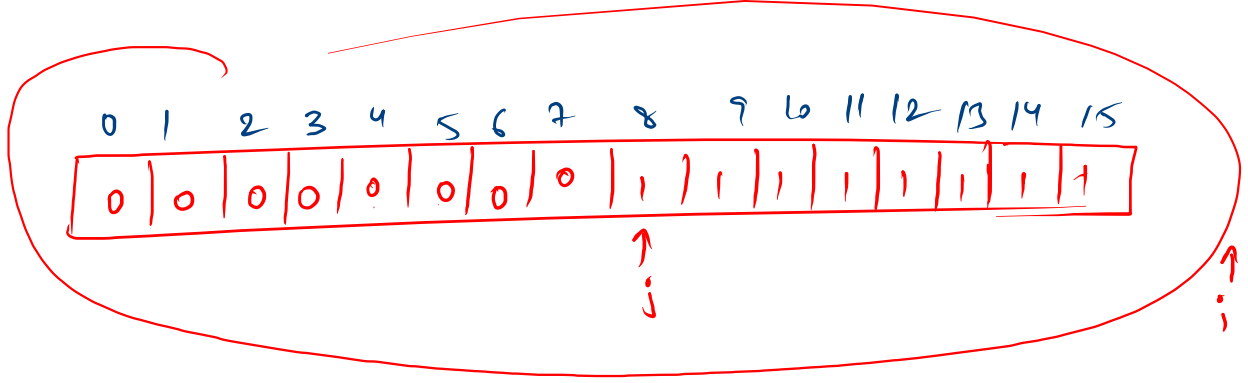

```java
public static int[] mergeSort(int[] arr, int lo, int hi) {
  if(lo == hi){
      int b[] = new int[1];
      b[0] = arr[lo];
      return b;
  }
  int mid = (lo+hi)/2;
  int lp[] = mergeSort(arr,lo,mid);
  int rp[] = mergeSort(arr,mid+1,hi);

  return mergeTwoSortedArrays(lp,rp);
}
```

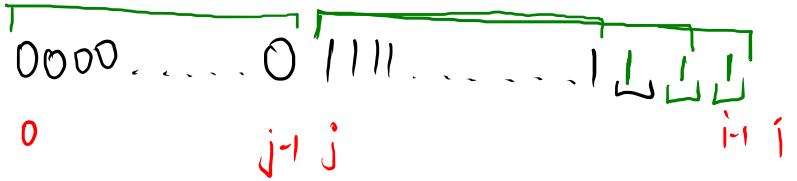
Sort-01 → Dividing Array in two parts

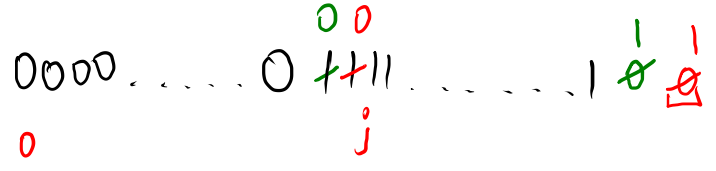
Sort 012 → Dividing Array in 3 parts

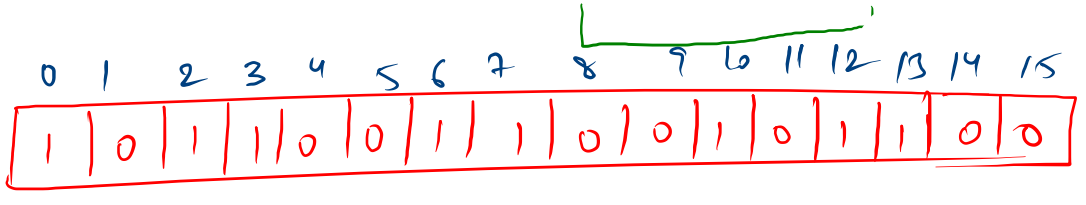
Sort



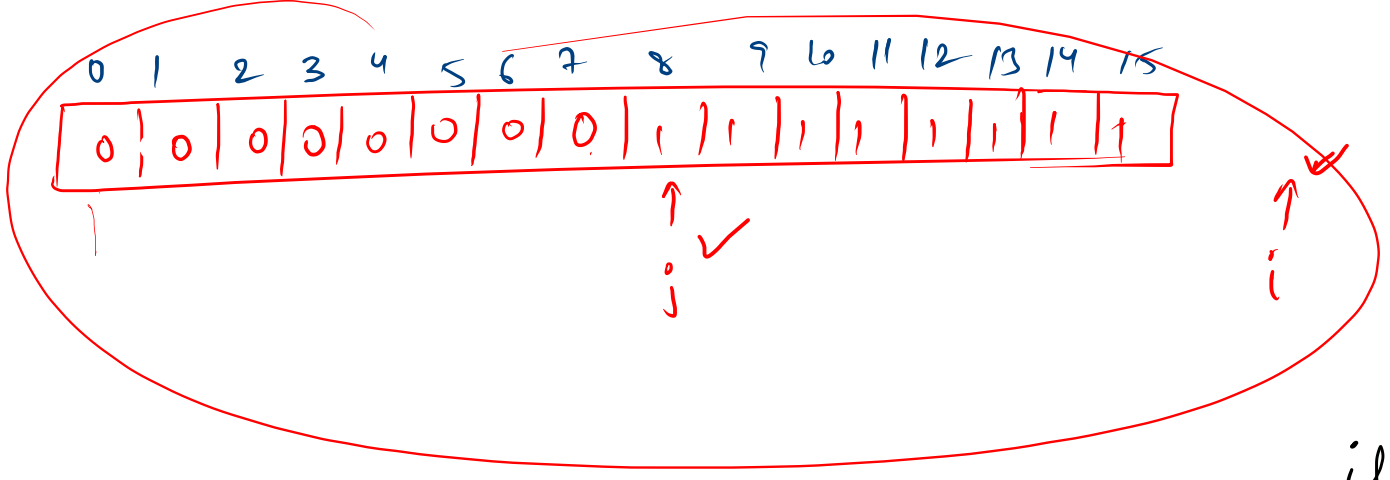
- $0 \to j-1$: 0's
- $j \to i-1$: 1's
- $i \to len-1$: unknown








```
if (arr[i] == 0){
    swap(arr, i, j);
    j++;
    j++;
}else{
    i++;
}
```



2 times

$n \rightarrow (2n)$

```
if (arr[i] == 0){
    swap(arr, i, j);
    j++;
    j++;
}else{
    i++;
}
```

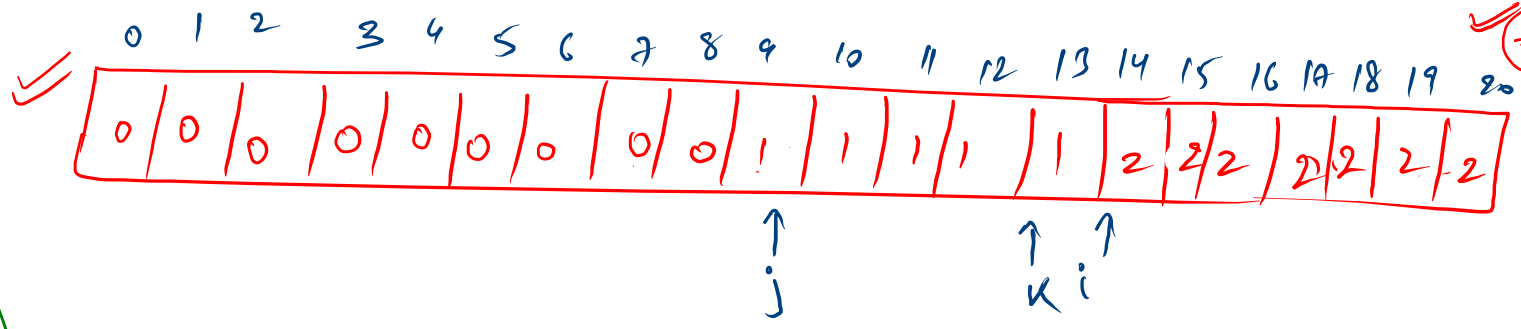
Constant

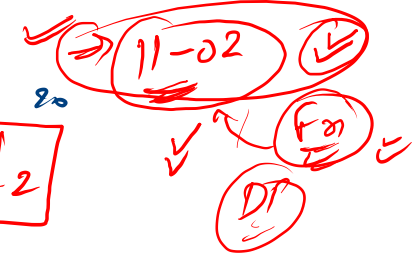
$T.C. \Rightarrow O(2n)$

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

$S.C. \Rightarrow O(1)$

Sort 012





(0, j-1) → 0's

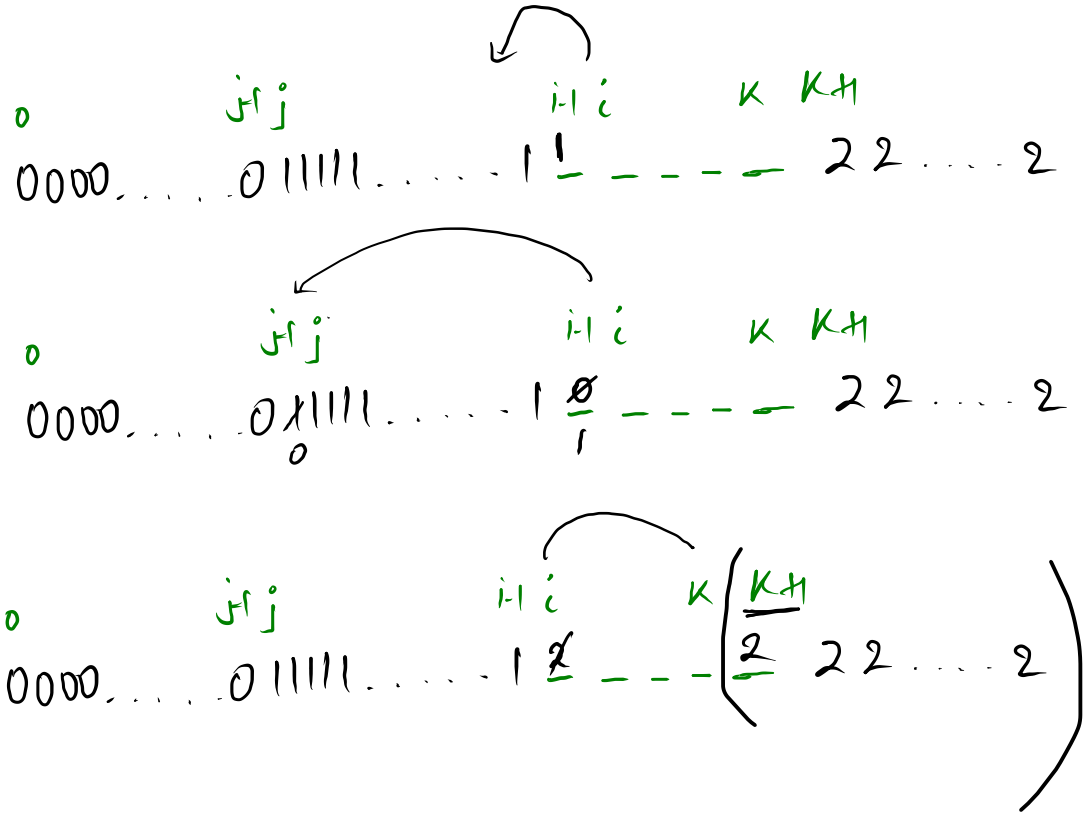
(j, i-1) → 1's

(k+1, len-1) → 2's

(i, k) → unknown

T.C. → O(n)

S.C. → O(1)




```
if(arr[i] == 0){
    swap(arr, i, j)
    i++, j++;
}else if(arr[i] == 1){
    i++;
}else{
    swap(arr, i, k)
    k--;
}
```

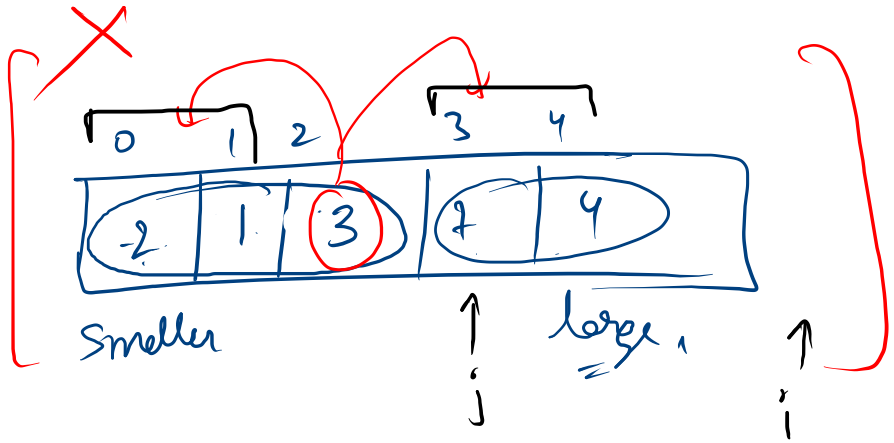
Sort → 01

partition → smaller - larger

partition → +ve -ve

partition → odd even

5 7 -2 4 1 3 3

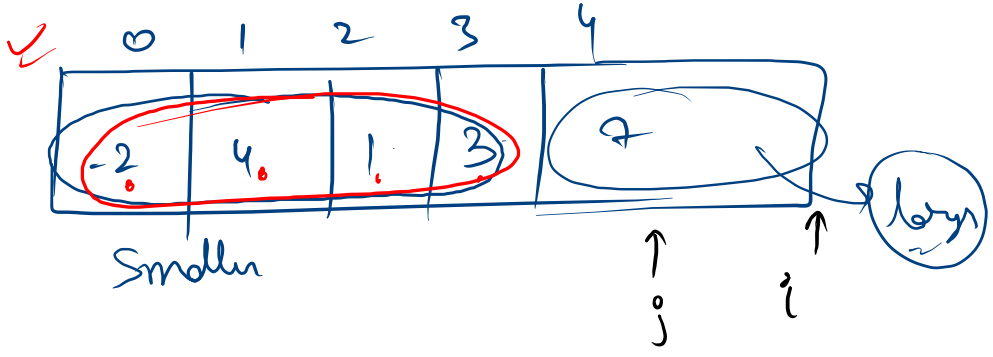


pivot = 3

Smaller → num <= pivot

larger → num > pivot

pivot = 4



j-1 j i-1 i

0 Smaller larger

```
if(arr[i] > pivot){
    c++;
}else{
    swap(i,j);
    i++;
    j++;
}
```