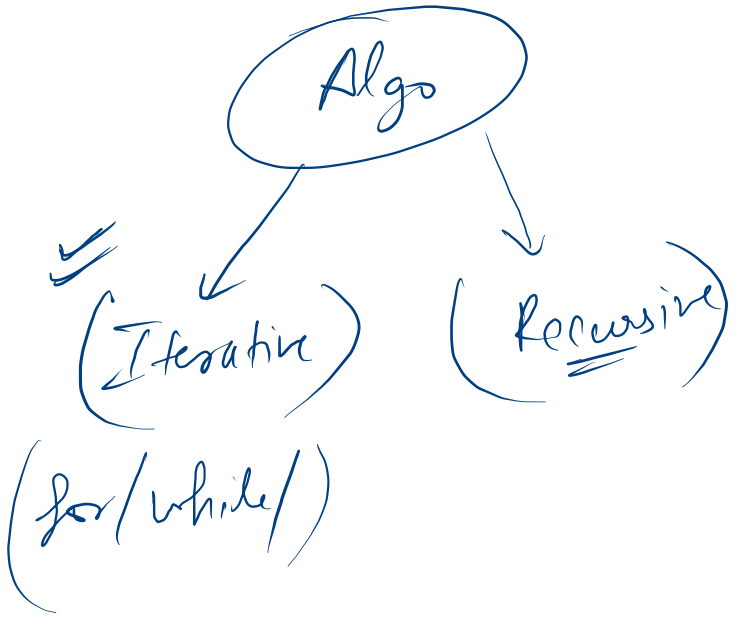

Algo
Iterative
for/while
Recursive

Search Algo

Linear Search

Find

Input

Array

K (element to be found)

$\Omega(1)$

$T.O. = (1)$

Best Case

K is present @ 0^{th} idx

$T.O. = (n)$

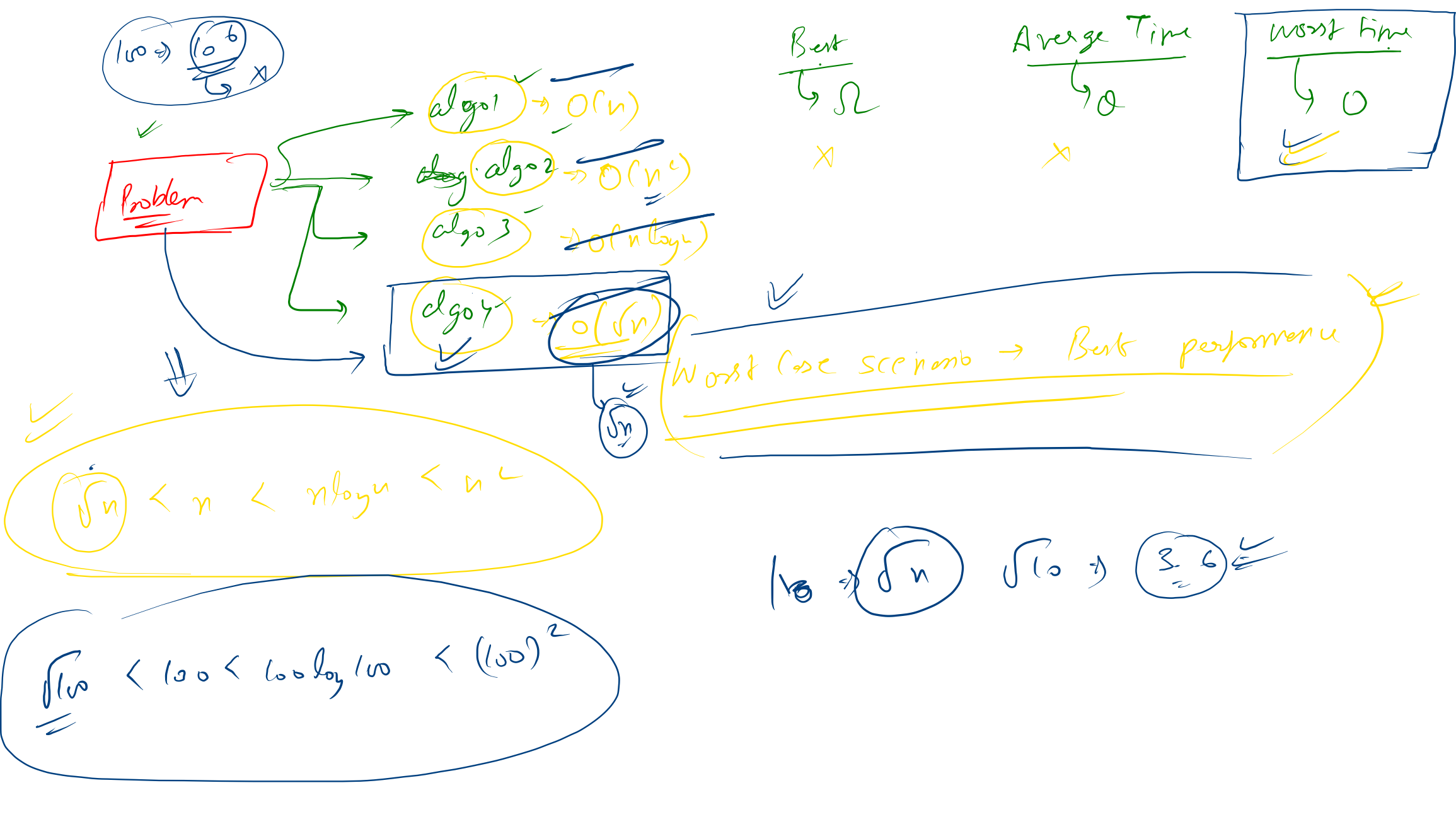
Worst Case

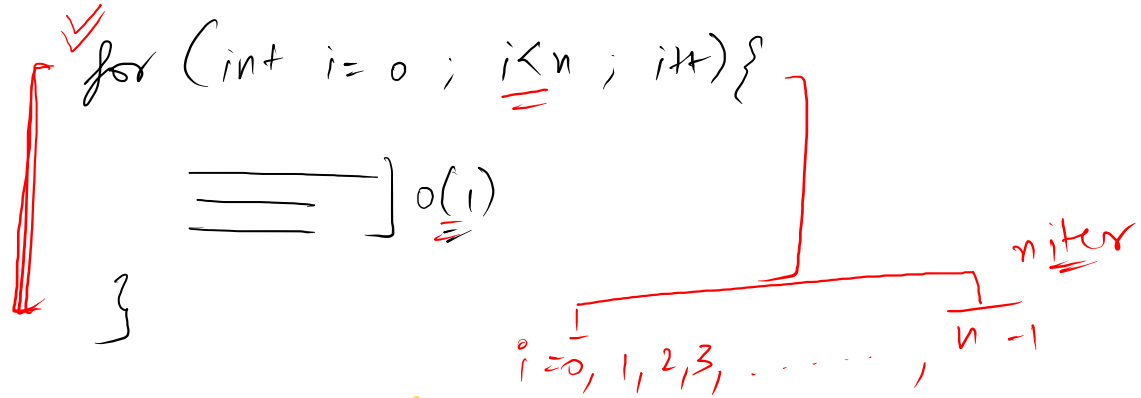
K ∉ Array

$O(n)$

K = 18

0	1	2	3	4	5	6	7
10	5	6	1	44	14	18	19


Best
Ω
Average Time
Θ
worst time
O
Problem
algo1 → O(n)
algo2 → O(n²)
algo3 → O(n log n)
algo4
O(√n)
√n
Worst case scenario → Best performance
√n < n < n log n < n²
√100 < 100 < 100 log 100 < (100)²
100 → √n √100 → 3. 6



O(1) → Constant time

1 itr → Constant 1

n itr → (n times)

T.C. ⇒ O(n)

absolute time

Time Complexity

① T.O. $\Rightarrow$ $n^2 + n + 1$, T.O. $\propto n^2$

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

Accurate ग्रुप $\Rightarrow$ Absolute time

$O(n^2)$

$\{ n^2+1 \;,\; \frac{n^2+4}{2} \;,\; n^2+n \;,\; n^2+n\log n \;,\; \frac{n^2+n}{2} \}$

Q) T.O. ⇒ $1 + 2 + 3 + 4 + \ldots\ldots\ldots + n$

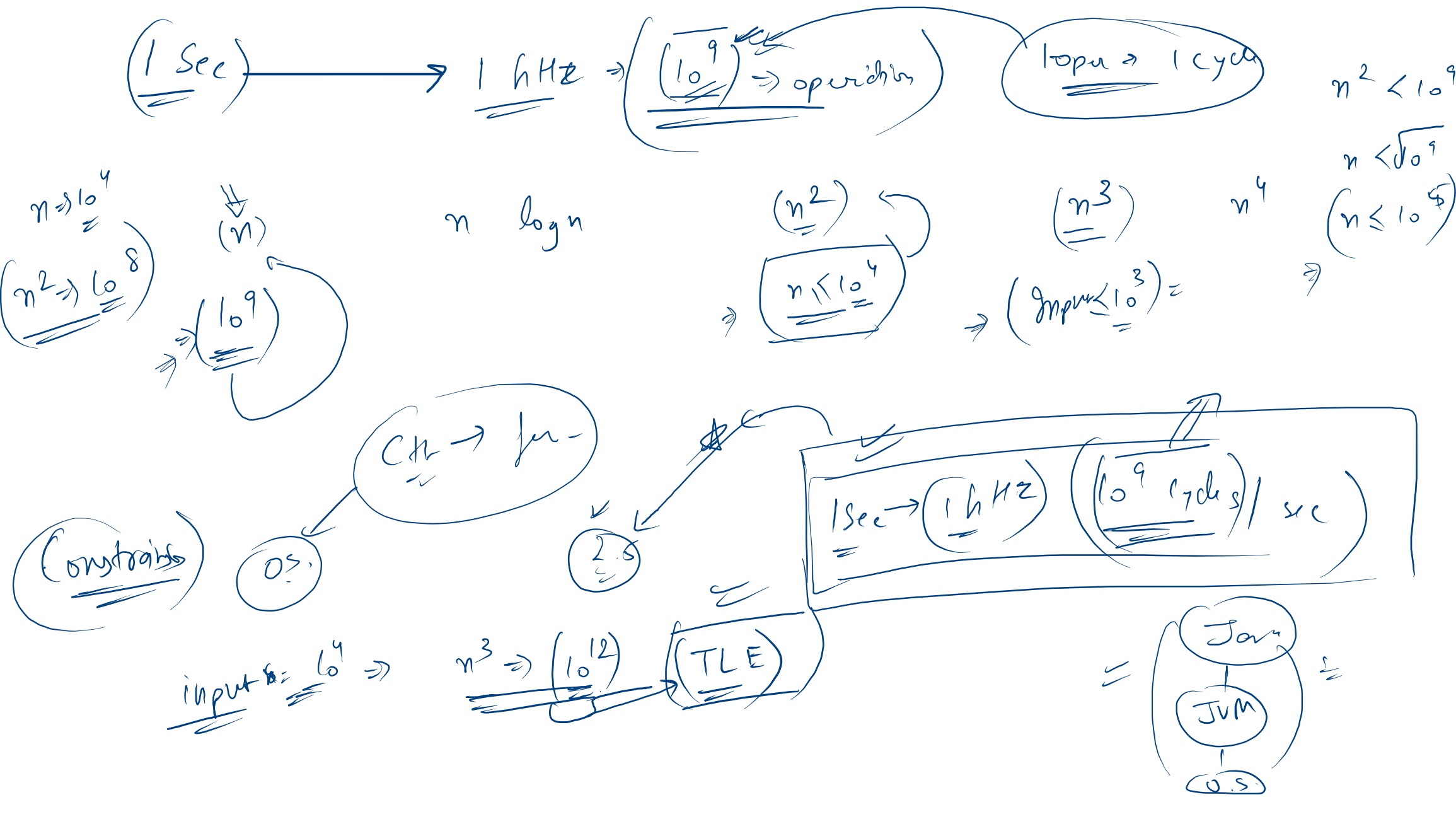
$\Rightarrow \frac{n(n+1)}{2} \Rightarrow \frac{(n^2)+n}{2}$

T.C. ⇒ $O(n^2)$

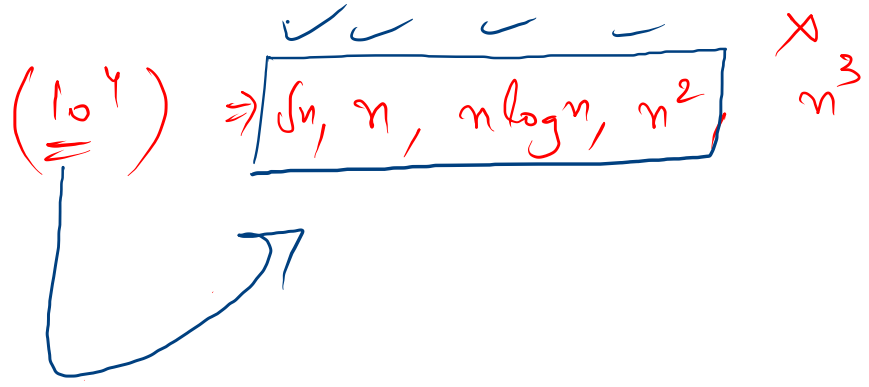
Q) T.O. ⇒ $1 + 2 + 4 + 8 + 16 + \ldots\ldots$ (n terms)

$\Rightarrow \frac{a[r^n - 1]}{r-1} \Rightarrow 1\frac{[2^n - 1]}{2-1} \Rightarrow 2^n - 1$

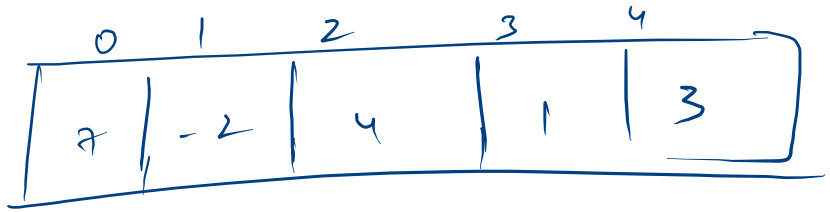
T.C. ⇒ $O(2^n)$

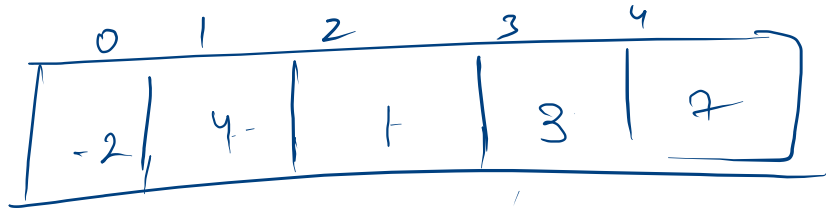

1 Sec
1 GHz
10^9
operations
1 op → 1 cycle
$n^2 < 10^9$
$n < \sqrt{10^9}$
$n \leq 10^4$
$n \geq 10^4$
$n^2 \geq 10^8$
n
10^9
$n \log n$
n^2
$n \leq 10^4$
n^3
n^4
input $\leq 10^3$
C++ → for
OS
constraints
2 s
1 sec → 1 GHz
10^9 cycles / sec
input: 10^4
$n^3 \Rightarrow 10^{12}$
TLE
Java
JVM
O.S

- $1 <= intervals.length <= 10^4$
- $intervals[i].length == 2$
- $0 <= start_i <= end_i <= 10^4$

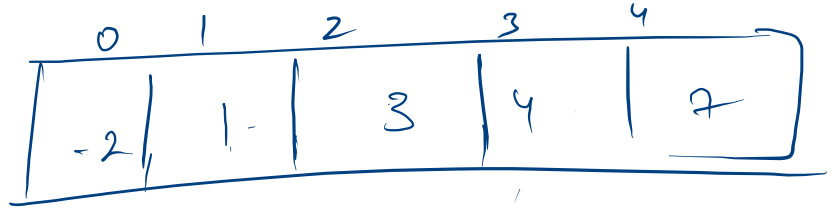


(Bubble Sort) (Sort → inc)





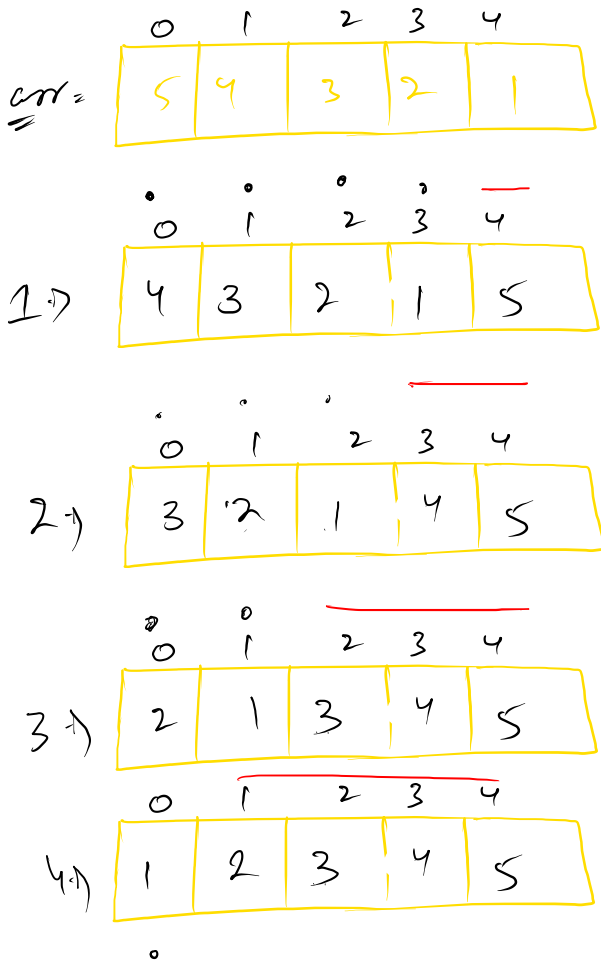
1 (itr →)



2 (itr →)

(x < y)

Bubble (X , Y)



n => 5

```
int n = arr.length;

for(int itr = 1 ; itr <= n-1 ; itr++){
    for(int j = 0 ; j <= n-itr-1 ; j++){
        if(arr[j+1] < arr[j]){
            //swap
            int tmp = arr[j+1];
            arr[j+1] = arr[j];
            arr[j] = tmp;
        }
    }
}
```

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

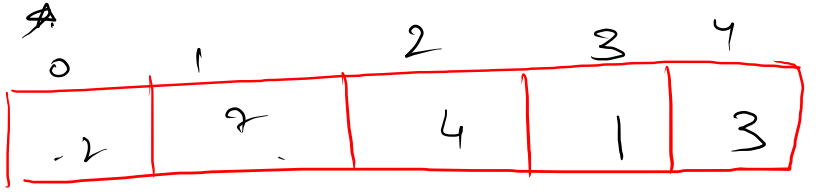
itr	j	n - itr - 1	opr
1	0 1 2 3		4
2	0 1 2	5 - 2 - 1 => 2	3
3	0 1	5 - 3 - 1 = 1	2
4	0	5 - 4 - 1 => 0	1

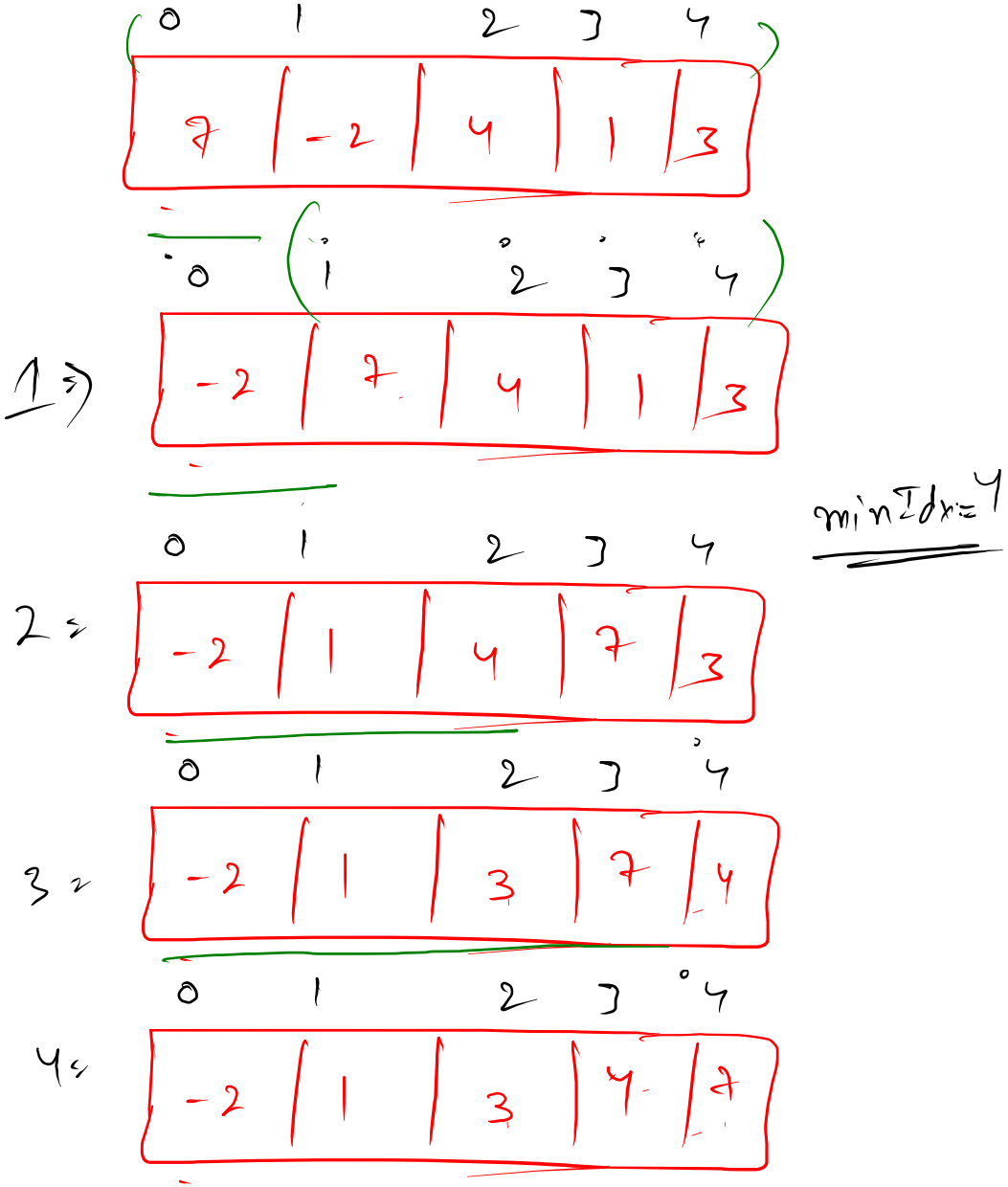
4 + 3 + 2 + 1

$T_{0.} \Rightarrow (n-1) + (n-2) + \dots + 3 + 2 + 1$

$T_{.0.} \Rightarrow \frac{n(n-1)}{2} \propto n^2 \Rightarrow (T_{.1.} \Rightarrow O(n^2))$

(midIdx = 1)






```
int n = arr.length;

for(int itr = 1 ; itr <= n-1 ; itr++){
    int minIdx = itr-1;

    for(int j = itr ; j < n ; j++){
        if(isSmaller(arr,j,minIdx)){
            minIdx = j;
        }
    }

    swap(arr,itr-1,minIdx);
}
```

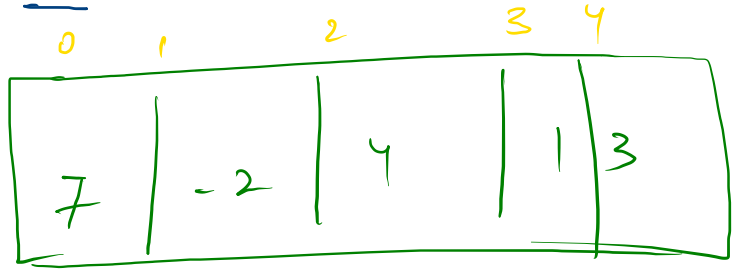
itr	itr-1	j
1	0	~~1~~ ~~2~~ ~~3~~ 4
2	1	~~2~~ ~~3~~ 4
3	2	~~3~~ 4
4	3	4

Iter
4
3
2
1

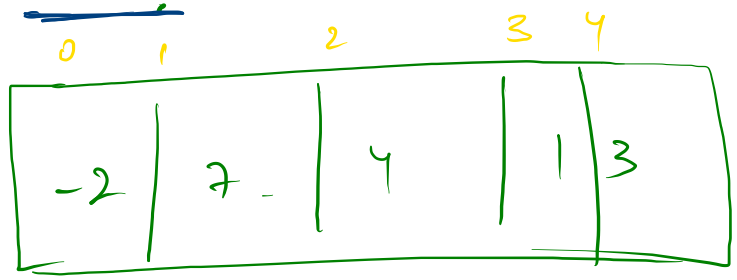
T.C. => $\frac{n(n-1)}{2}$

T.C. => $O(n^2)$

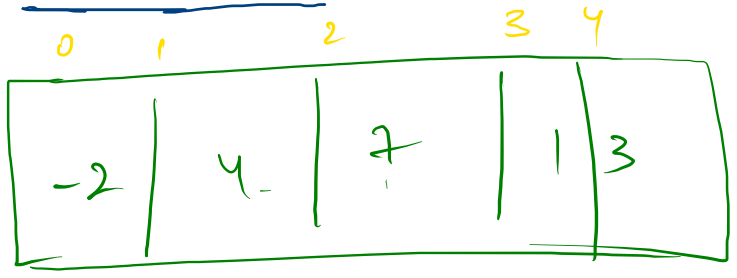
arr :-



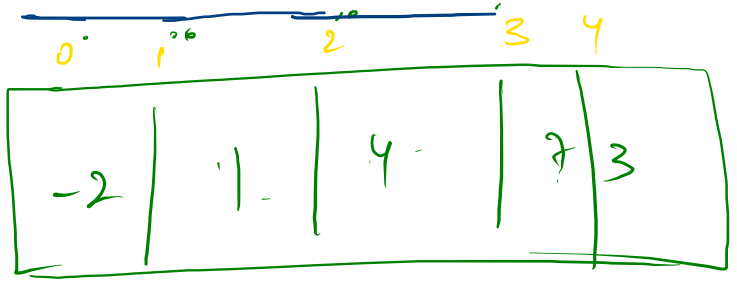
1st

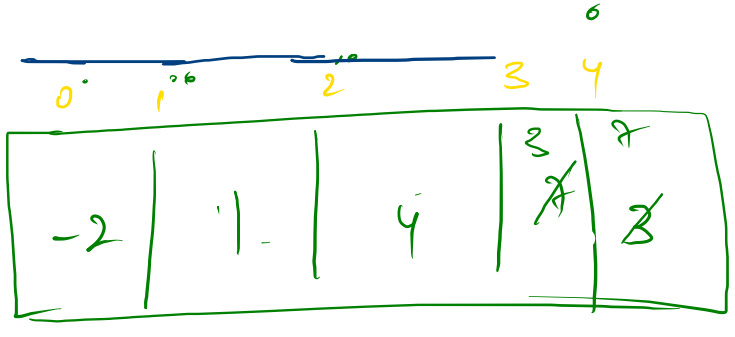


2nd

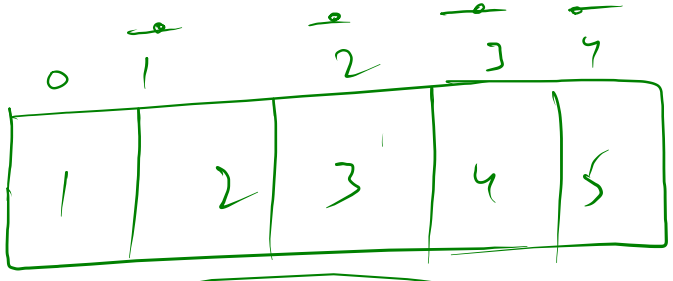


3rd





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



Sorted

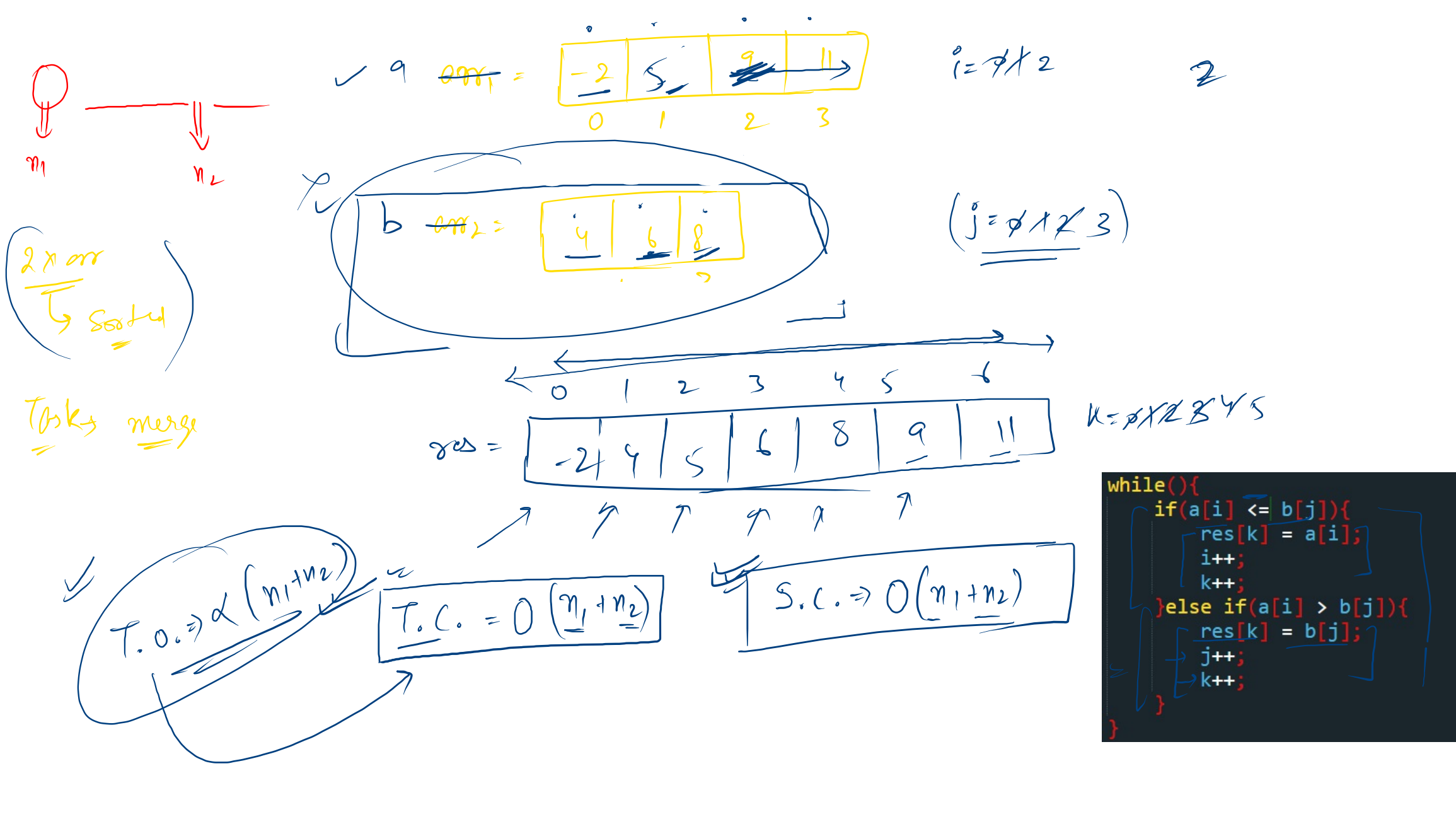
Insertion Sort → $\Omega(n)$

$n-5$

itr	T.C.
1	1
2	1
3	1
4	1

4 (n-1)

T.C. → $\Omega(n)$

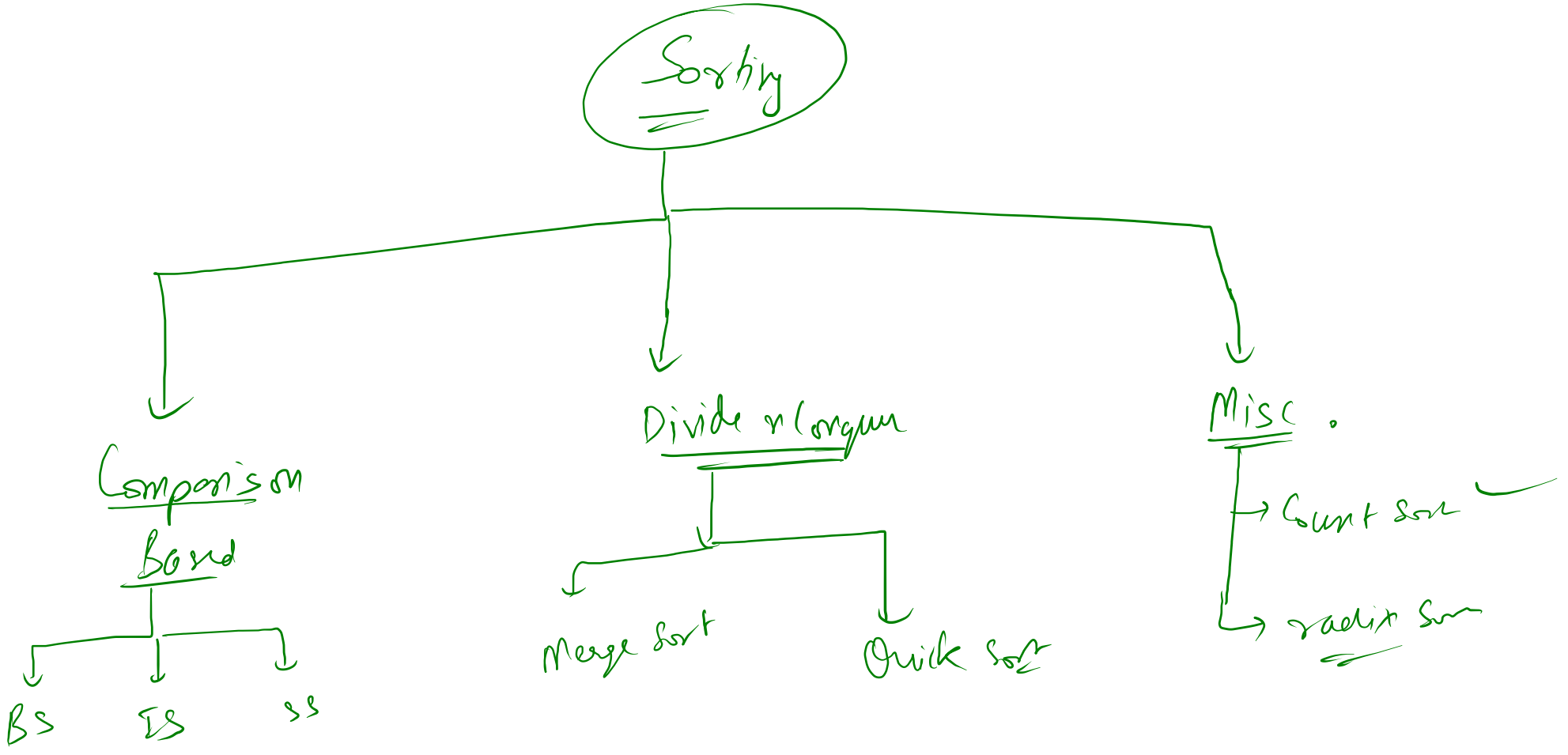


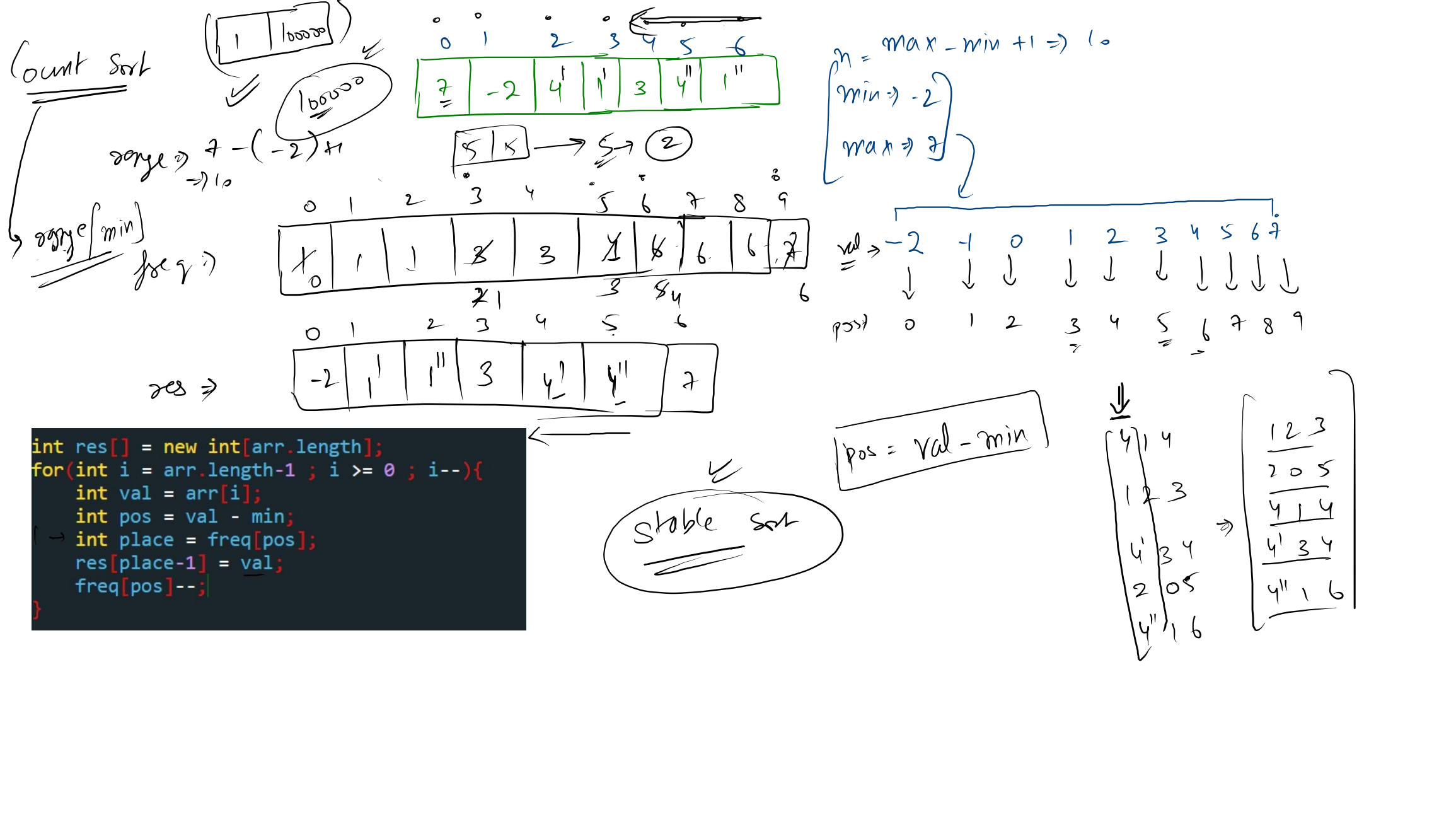

```
while(){
    if(a[i] <= b[j]){
        res[k] = a[i];
        i++;
        k++;
    }else if(a[i] > b[j]){
        res[k] = b[j];
        j++;
        k++;
    }
}
```

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

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

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


Sorting
Comparison Based
BS
IS
SS
Divide n Conquer
Merge sort
Quick sort
Misc.
Count sort
radix sort




```
int res[] = new int[arr.length];
for(int i = arr.length-1 ; i >= 0 ; i--){
    int val = arr[i];
    int pos = val - min;
    int place = freq[pos];
    res[place-1] = val;
    freq[pos]--;
}
```

```java
int range = max-min+1;

int freq[] = new int[range];
for(int val : arr){
    int pos = val - min;
    freq[pos]++;
}

for(int i = 1 ; i < range ; i++){
    freq[i] += freq[i-1];
}

int res[] = new int[arr.length];
for(int i = arr.length-1 ; i >= 0 ; i--){
    int val = arr[i];
    int pos = val - min;
    int place = freq[pos];
    res[place-1] = val;
    freq[pos]--;
}


for(int i = 0 ; i < arr.length ; i++){
    arr[i] = res[i];
}
```

range

range

arr.length

arr.length

S.C. = O [maxval - minval + 1] + [arr.length]

T.C. => O [(range) + (arr.length)]

Count Sort

(range >>> arr.length)

↳ S.C. = O(range) T.C. = O(range)

(range <<< arr.length)

↳ S.C. => O(n) & T.C. => O(n)