Today's Content:

1) Towers of Hanoi
2) Sorting Basics
   - → Sorting Basics
   - → Stable Sorting
   - → Selection Sort
   - → Bubble Sort
   - → Insertion Sort → [If Time Permits]

Recursion Basics

Ass: Decide what your function does

Main: Solving assumption with subproblems

Base: When to stop

Towers of Hanoi :

→ Given 3 Towers (A, B, C)

→ There are N discs placed in Tower A,

→ Move all discs from A → C (using B)

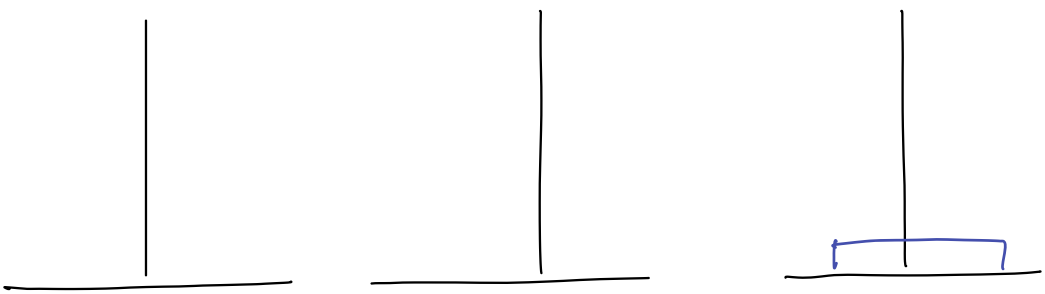
(Note): 1) Only 1 disc can be moved at a time

2) larger disc cannot be placed on smaler discs

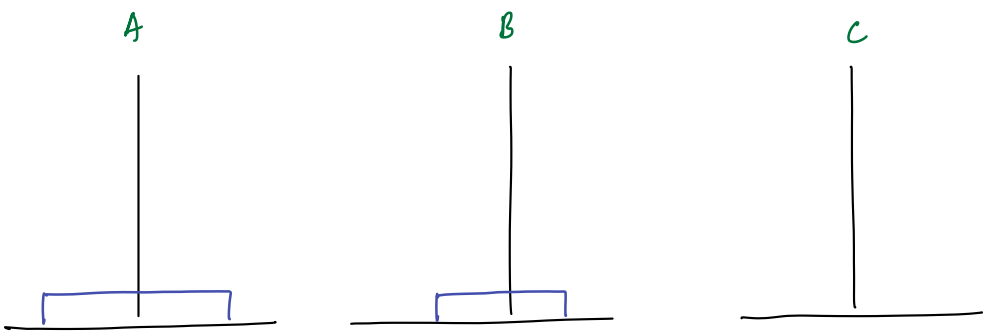
Q): Print movement of discs

N=1

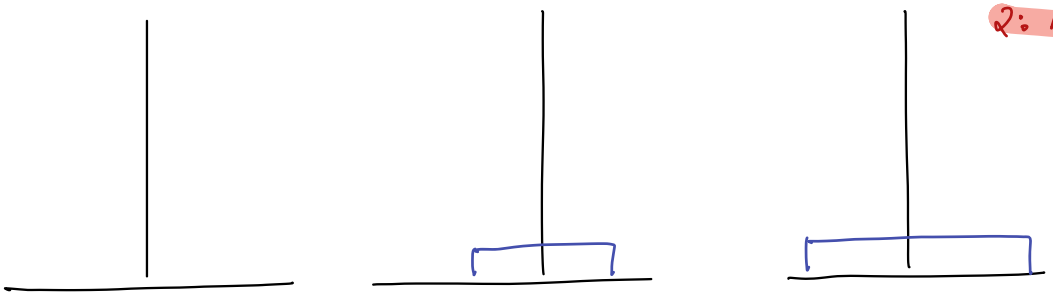
: 1 A → C



N=2



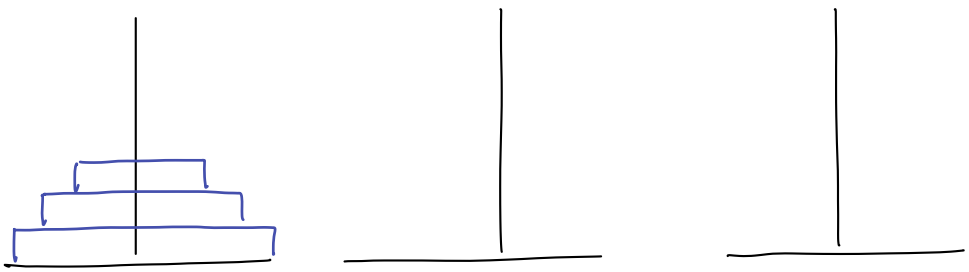
1 : A → B

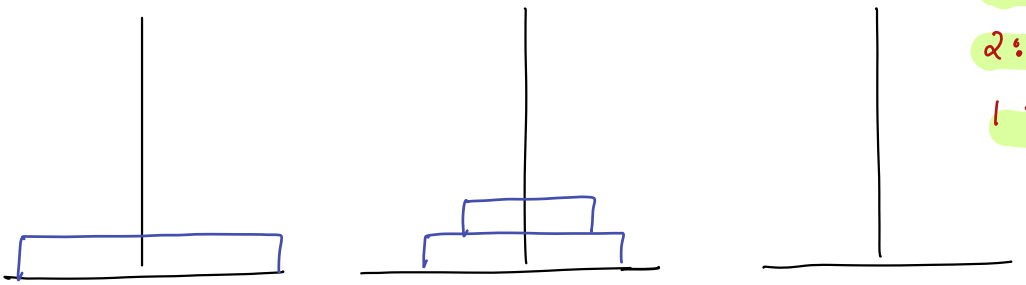


2 : A → C

1 : B → C

N=3

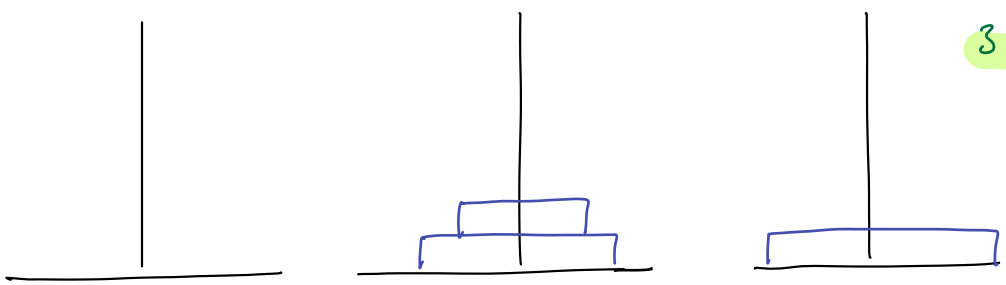




1 : A → C

2 : A → B

1 : C → B



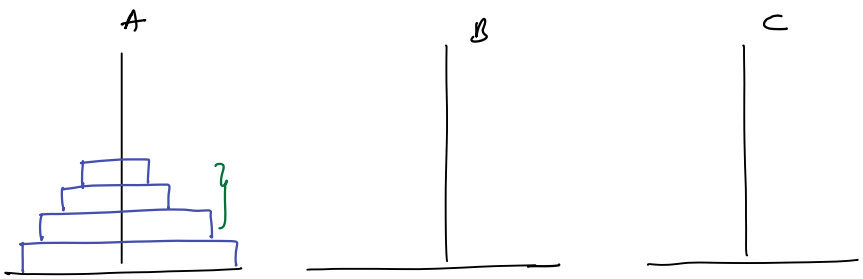
3 : A → C

1 : B → A

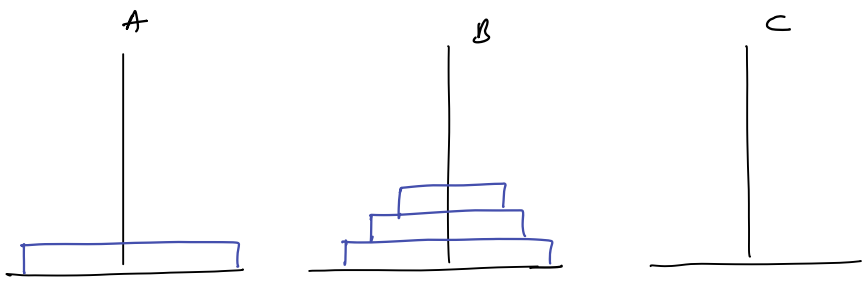
2 : B → C

1 : A → C

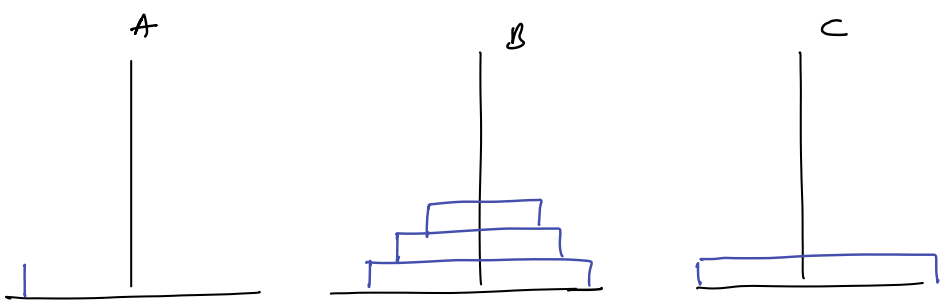
// 4 discs from A → C using B



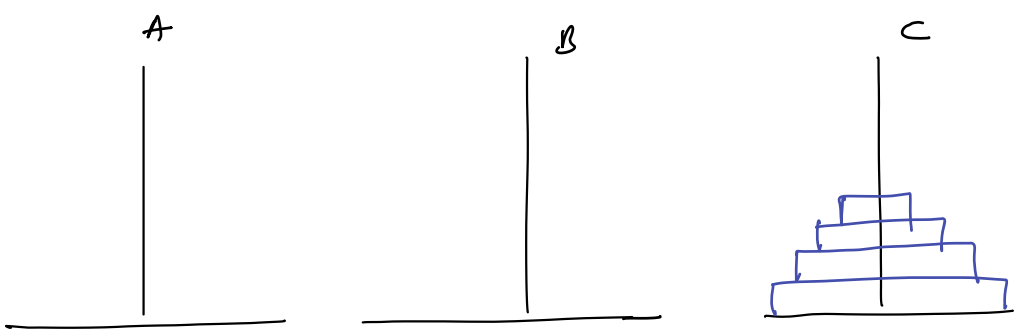
→ 3 discs from A → B using C



→ 4th disc from A → C // movement



→ 3 discs from B → C using A



// Ass: Move N discs from S → D using Helper


```
void TOH ( int N, char S, char H, char D ) {
    if ( N == 0 ) { return }
    1) TOH ( N-1, S, D, H )
    2) print ( N^th : S → D )
    3) TOH ( N-1, H, S, D )
}
```

main logic

Recursive Relation:

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

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

$T(0) = 1$

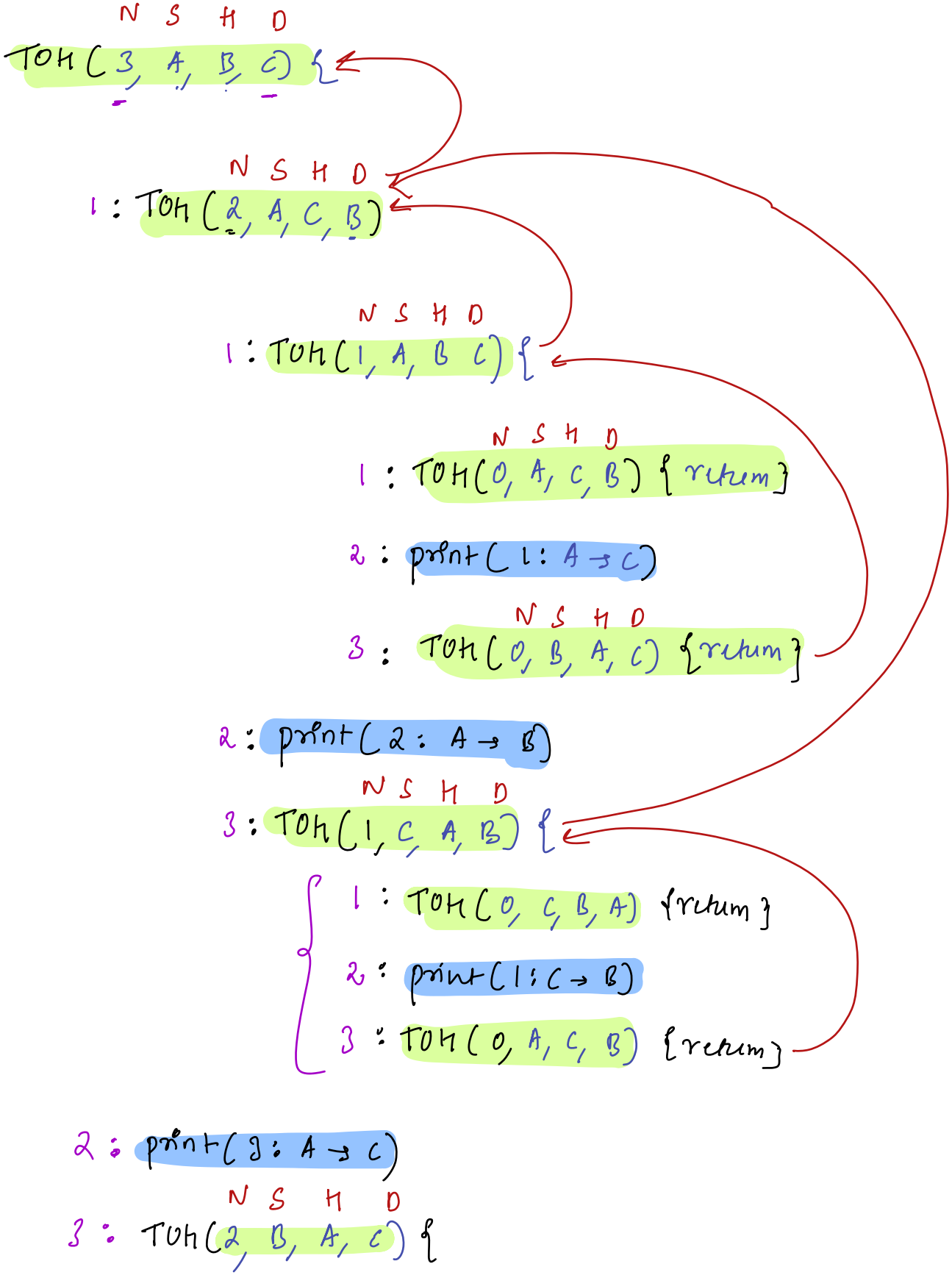
```
// main() {
    TOH ( 3, A, B, C )
}
```

S H D

output

1: A → C
2: A → B
1: C → B
// 2 ops from A → B

3: A → C } 3rd A → C


N S H D
TOH(3, A, B, C) {
N S H D
1 : TOH(2, A, C, B)
N S H D
1 : TOH(1, A, B C) {
N S H D
1 : TOH(0, A, C, B) {return}
2 : print(1 : A → C)
N S H D
3 : TOH(0, B, A, C) {return}
2 : print(2 : A → B)
N S H D
3 : TOH(1, C, A, B) {
1 : TOH(0, C, B, A) {return}
2 : print(1 : C → B)
3 : TOH(0, A, C, B) {return}
2 : print(3 : A → C)
N S H D
3 : TOH(2, B, A, C) {

// Recursive Relation:

$$T(N) = 2T(N-1) + 1, \quad T(0) = 1$$

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

$$= 2[2T(N-2) + 1] + 1$$

$$= 4T(N-2) + 3 \rightarrow 2^2 T(N-2) + 2^2 - 1$$

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

$$= 4[2T(N-3) + 1] + 3$$

$$= 8T(N-3) + 7 \rightarrow 2^3 T(N-3) + 2^3 - 1$$

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

$$= 8[2T(N-4) + 1] + 7$$

$$= 16T(N-4) + 15 \rightarrow 2^4 T(N-4) + 2^4 - 1$$

// generalized expression

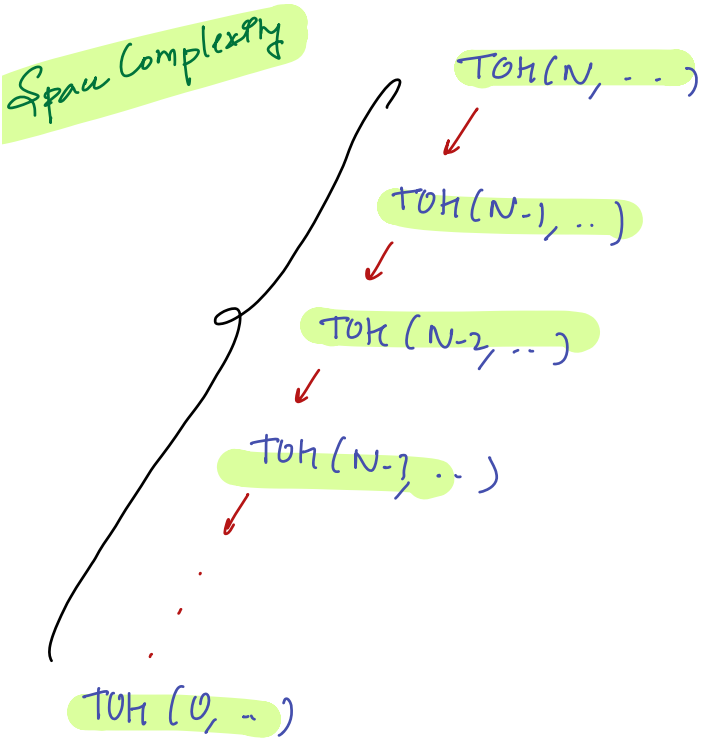
$$= 2^k T(N-k) + 2^k - 1 \quad \} T(0) = 1$$

$k = N$: SC: $O(N)$

$$= 2^N T(0) + 2^N - 1$$

$$= 2^N + 2^N - 1 \rightarrow 2 \cdot 2^N - 1 \rightarrow O(2^N)$$

Space Complexity



At max in our stack at any given point we can have N+1 functions calls

SC: O(N)

Sorting Basics : (10mins break) → 10:30pm

↳ ? : arranging data in inc/dec order based on parameter

Ex1: 4 8 10 14 24 → inc

Ex2: 40 24 16 8 2 → dec

Ex3:

1	3	5	7	4	9	6	10	12
1	2	2	2	3	3	4	4	6

} inc based on factors

Sorting Algorithms:

Stable Sorting ?

In place Algorithm?

SC: O(1) / without extra space

Eg:

Actors	Remuneration
Srk	40crs
NTR	30cr
Yash	20cr
Salaman	40cr
Allu Arjun	50cr
Anger khan	50cr
Saif	25cr
Sonam	40cr
Anushka	50cr

Based on Remuneration (parameter) Increasing

Yash	20cr
Saif	25cr
NTR	30cr
Srk	40crs
Salaman	40cr
Sonam	40cr
Allu Arjun	50cr
Anger khan	50cr
Anushka	50cr

Stable: 2 data points having same parameter value, their relative order before sorting and after sorting should be same → {Stable Sorting}

Ex: ar[] → 10 2 3 1 2 9

Stable sorting Inc Based on value →

Before & After sorting Relative order between 2 & 2 is same

1 2 2 3 9 10

Obs: In Stable Sort, order is preserved in case of conflict

Q) Given N array elements, find k^{th} smallest element

→ Q): If array is sorted, get $k-1$ index element

									k=1	k=2	k=3	k=4
ar[8]:	2	8	4	-1	6	7	5	10	-1	2	4	5

Idea:

↳ Sort in inc & get $k-1^{th}$ index } → TC: O(N log N)

1^{st} smallest → 0^{th} index

2^{nd} smallest → 1^{st} index

:

k^{th} smallest → $k-1^{st}$ index

ar[6] = 3 5 5 6 5 8

k = 2 → 3 5 5 5 6 8

↳ 2nd index in sorted array: 0 1 2

ans = 5

	0	1	2	3	4	5	6	7
ar[8] :	~~-1~~	~~2~~	4	~~-1~~	7	10	~~8~~	6
	-1	2		~~2~~			8	
				~~5~~				
After 4 :	-1	2	4	5	7	10	8	6

lastIndex

Map Iterate ↑ Ind →

1st : i=0 [0 - 7] 3 — Swap (ar[3], ar[0])

2nd : i=1 [1 - 7] 3 — Swap (ar[3], ar[1])

3rd : i=2 [2 - 7] 2 — Swap (ar[2], ar[2])

4th : i=3 [3 - 7] 6 — Swap (ar[6], ar[3])

5th : [0 4] is sorted

6th : [0 5] is sorted

7th : [0 6] is sorted

8th : [0 7] is sorted

→ Entire array is sorted

1) Selection Sort ✓

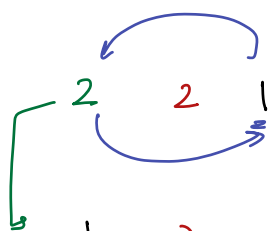
```
SelectionSort(int ar[], int N){
    for(i=0; i<N; i=i+1){ // i: [0, N-1]  N times
        minval = ar[i], minInd = i
        for(j=i; j<N; j=j+1){ // minval, minInd
            if(ar[j] < minval){
                minval = ar[j]
                minInd = j
            }
        }
        Swap ar[minInd] with ar[i]
    }
}
```

TC: $O(N^2)$ SC: $O(1)$ → Inplace ✓

// Stable or not: TODO

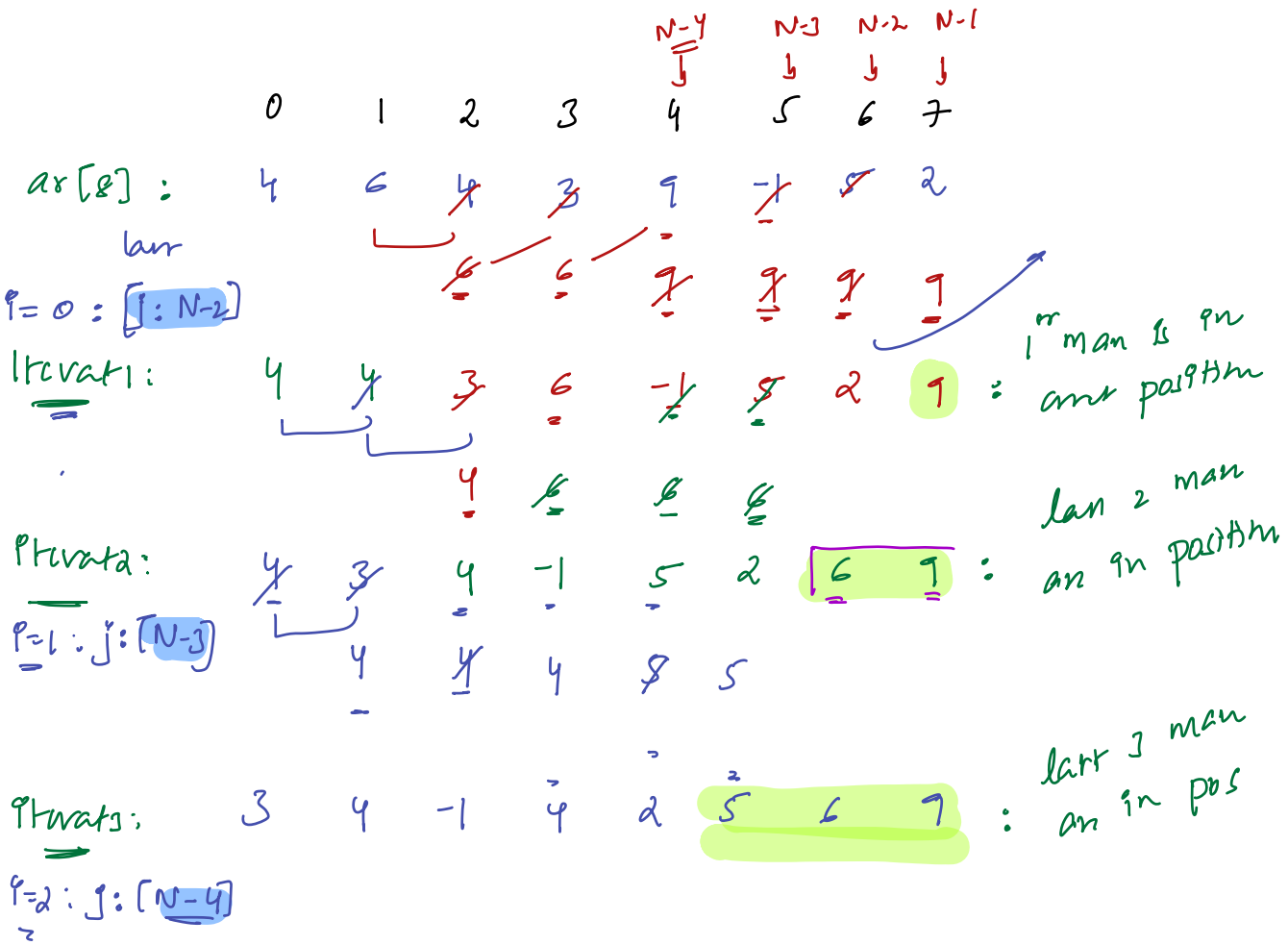
Data: 2 2 1 4 —If Stable→ 1 2 2 4

Algo



1 2 2 4 —Not Stable→

2) // Given an array, Sort in asc order but we can only adjacent Elements.



// $j < N - i - 1$ (max j)

i=0 : $j < N-1$ ⇒ N-2

i=1 : $j < N-2$ ⇒ N-3

i=2 : $j < N-3$ ⇒ N-4

Pseudocode:

```
BubbleSort(int ar[], int N){
    for(i=0; i<N; i=i+1){
        int c=0
        for(j=0; j<N-1-i; j=j+1){        // j=(N-1) : ar[N-1] > ar[N]  => error
            if(ar[j] > ar[j+1]){         // order is wrong
                swap ar[j] & ar[j+1]
                c=c+1
            }
        }
        if(c==0){ Break }                // Already sorted
    }
}
```

TC: $O(N^2)$ SC: $O(1)$

2 2 1 4

→ 2 1 2 4

relative order is maintained

→ 1 2 2 4 } → Stable Sorting

→ SC: O(1)

arr[7] : 10 8 2 4 5 6 7

8 10 10 10 10 10

iterat1 : 8 2 4 5 6 7 10

2 8 8 8 8 8

iterat2 : 2 4 5 6 7 8 10

iterat3 : 2 4 5 6 7 8 10 : 0 swaps

if 0 swaps in any iteration array is sorted

if already sorted break it

// Lake

---

0 ↑ bigger bubble will rain first

// permutations with repeatition →

$a_1\ \underbrace{a_2\ a_3}\ a_4 \rightarrow \frac{4!}{2!} \rightarrow \left\{\frac{4!}{2!} \rightarrow \frac{24}{2} \rightarrow 12\right\}$

$\underbrace{a_1\ a_2}\ a_3\ \underbrace{a_4} \rightarrow \frac{4!}{3!} \rightarrow 4$

$\underbrace{a_1\ a_2}\ a_3\ \underbrace{a_4\ a_5} \rightarrow \frac{5!}{2! \times 2!} \rightarrow \frac{120}{4} \rightarrow 30$

$a_1\ \underbrace{a_2}\ \underbrace{a_3\ a_4}\ \underbrace{a_5}\ \underbrace{a_6} \rightarrow \frac{6!}{3! \times 2!} \rightarrow \frac{720}{12} \rightarrow 60$

$a_1\ \underbrace{a_2}\ \underbrace{a_3}\ \underbrace{a_4}\ \underbrace{a_5}\ \underbrace{a_6\ a_7\ a_8} \rightarrow \frac{8!}{2! \times 2! \times 3!} \rightarrow$

// Stable Algorithm :