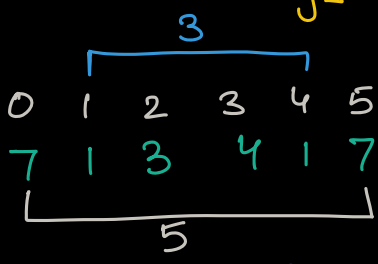
Today's Content:

1) Shaggy & Distances (from prev class)
2) Tower of Hanoi
3) Sorting
   - → Sorting Basics.
   - → Stable sorting / In-place sorting.
   - → Selection Sort.
   - → Bubble Sort. (revisited)

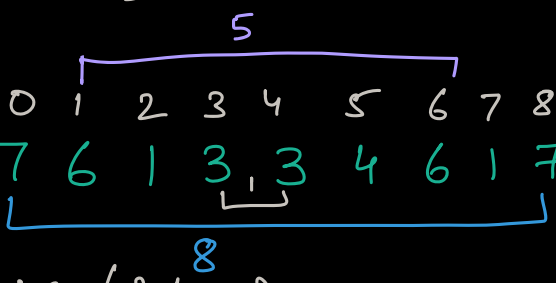
(Hashing contd...)

Q: Given a[n] having integers. Find any pair (i, j) such that $j > i$ and $A[i] == A[j]$ & $(j - i)$ is minimum.

Eg: a[] = { 7 1 3 4 1 7 } (0 1 2 3 4 5) ans = 3



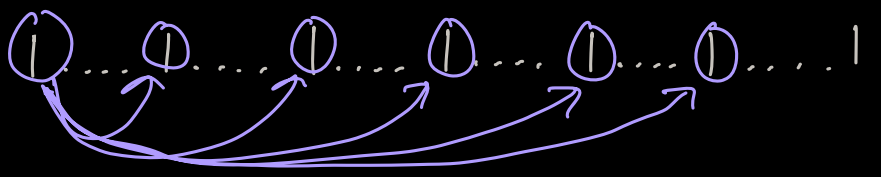
a[9] = { 7 6 1 3 3 4 6 1 7 } (0 1 2 3 4 5 6 7 8) ans = 1



idea 1: Consider all pairs (2 loops)

TC: $O(n^2)$, SC: $O(1)$

Problem with Bruteforce: →



obs: Compare with nearest element only. => We just need prev. index.

Hashmap <int, int> hm

Eg: a[] = { 1 2 3 6 1 2 3 2 1 } (0 1 2 3 4 5 6 7 8)

ans = ~~∞~~ ~~4~~ 2

Hashmap <int, int> hm

<1 : 8> <2 : 7>

<3 : 6> <6 : 3>

1 : ans = min (ans, 4 - 0) = 4

2 : ans = min (ans, 5 - 1) = 4

3 : ans = min (ans, 6 - 2) = 4

2 : ans = min (ans, 7 - 5) = 2

1 : ans = min (ans, 8 - 4) = 2

Pseudo code :

```
int shaggy (int a[]) {
    n = a.length
    Hashmap<int, int> hm
    ans = ∞ (INT_MAX)
    for (i = 0; i < n; i++) {
        if (hm.search(a[i]) == true) {
            ans = min(ans, i - hm[a[i]])
            hm[a[i]] = i
        }
        else {
            hm.insert(a[i], i)
        }
    }
    return ans
}
```

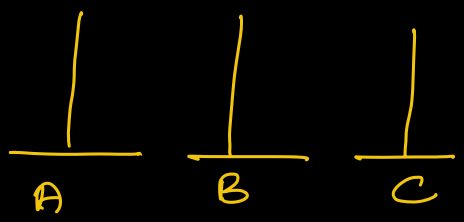
TC : O(n)

SC : O(n)

Recursion :→

Qn. Tower of Hanoi

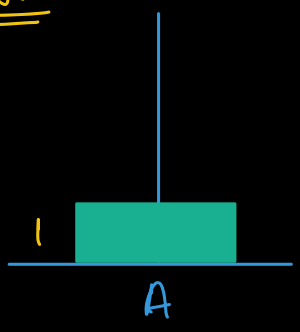
- Given 3 towers A, B, C
- There are N discs placed on tower A
- Move all discs from A to C (using B)

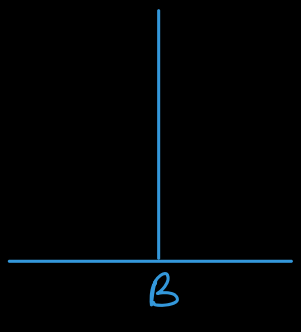


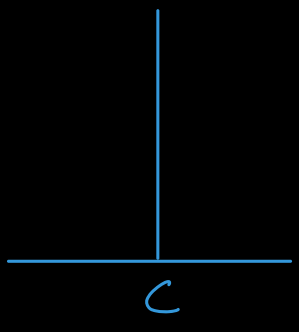
Note: * Only 1 disc can be moved at a time
* Larger disc can't be placed on top of smaller disc.

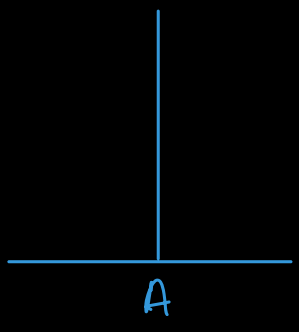
→ Print the movement of the discs.

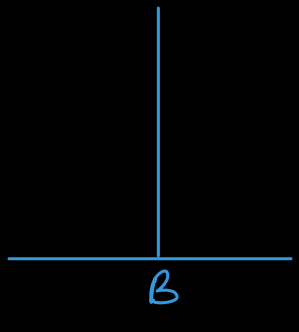
N = 1



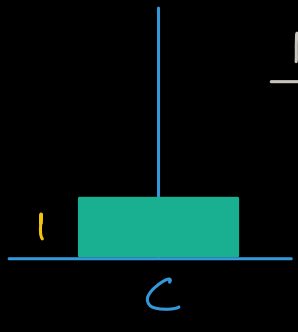




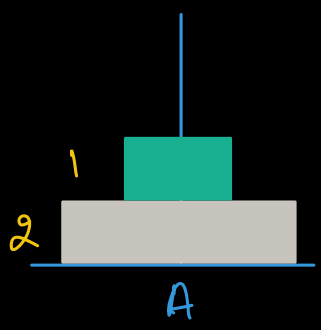


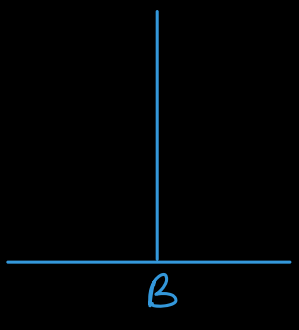


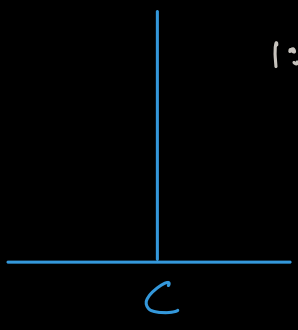
1 : A → C



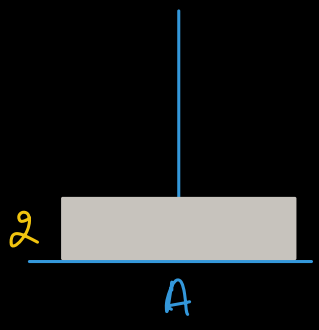
N=2

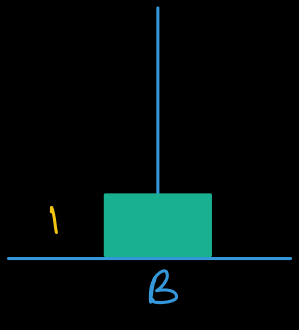


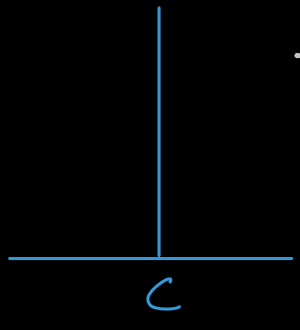




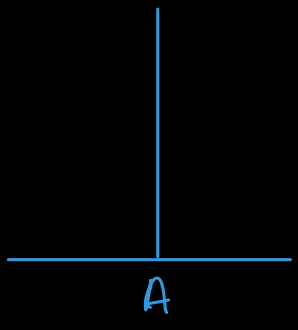
1: A→B

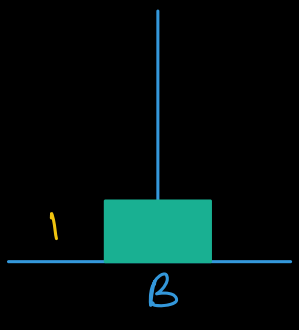


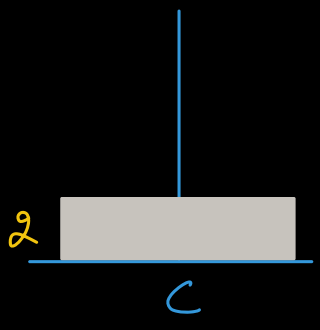




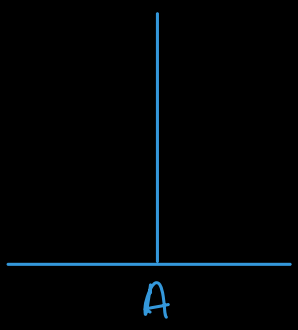
2: A→C

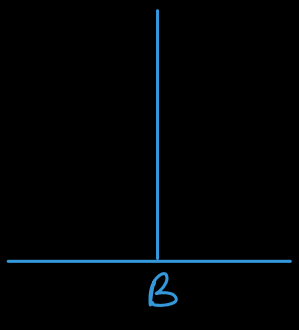


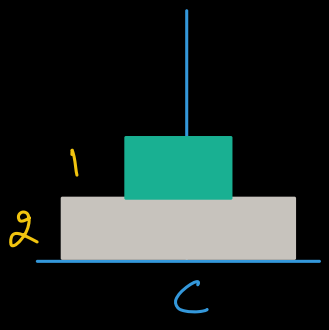




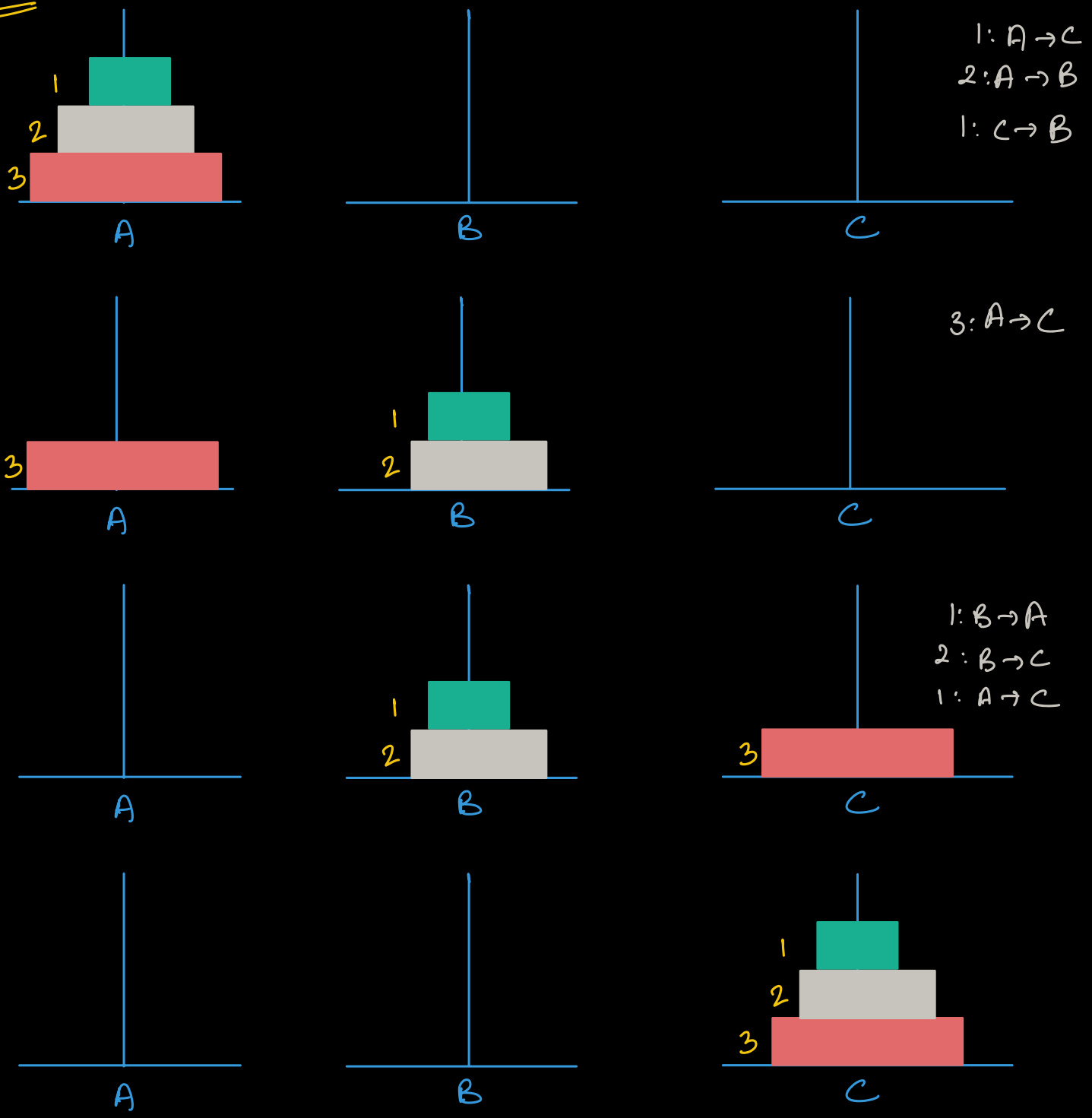
1: B→C







N=3



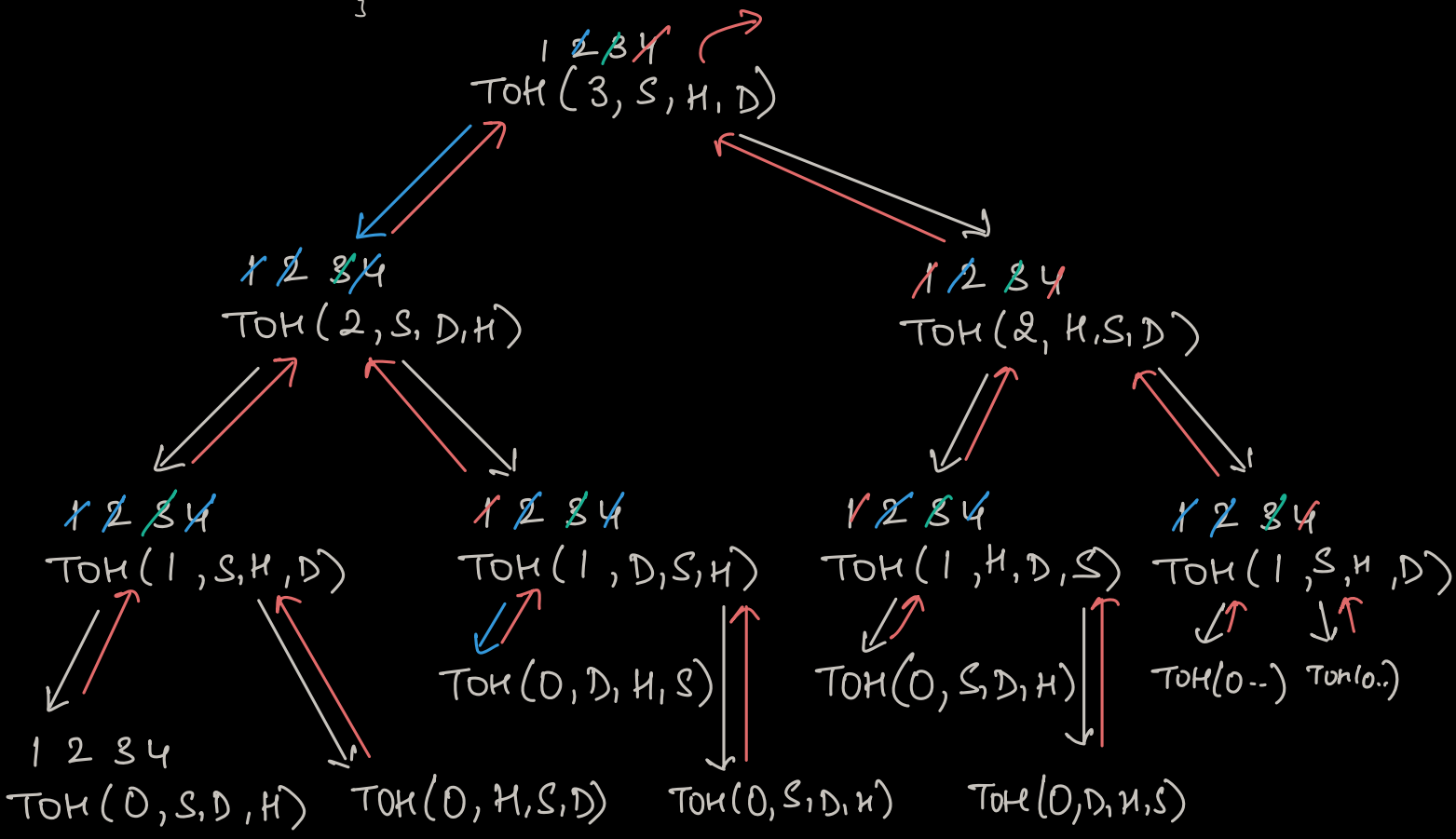
// Pseudo Code: // Assumption: Move N discs from S → d.

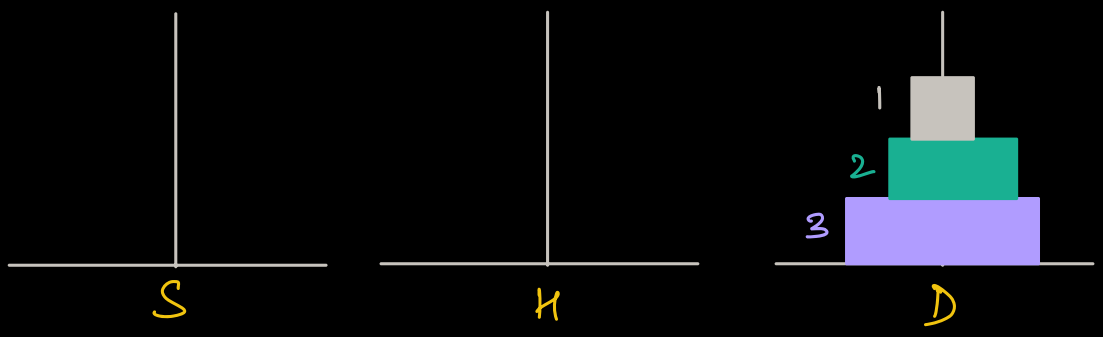
```
void TOH (int n, char S, char h, char d) {
    if (n==0) {return}
    TOH (n-1, S, d, h)
    print (n: S→d)
    TOH (n-1, h, S, d)
}

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

Dry run N = 3

```
void ToH (int n, char s, char h, char d) {
    ① if (n==0) {return
    ② ToH (n-1, s, d, h)
    ③ print (n : s→d)
    ④ ToH (n-1, h, s, d)
}
```





O/p:
1 : S → D
2 : S → H
1 : D → H
3 : S → D
1 : H → S
2 : H → D
1 : S → D

Break till 8:48am

Time Complexity : Recurrance Relation

$T(n) = 2T(n-1) + 1$ $\quad 2^1 T(n-1) + 2^1 - 1$

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

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

$T(n) = 4T(n-2) + 3$ $\quad 2^2 T(n-2) + 2^2 - 1$

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

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

$T(n) = 8T(n-3) + 7$ $\quad 2^3 T(n-3) + 2^3 - 1$

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

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

$T(n) = 16T(n-4) + 15$ $\quad 2^4 T(n-4) + 2^4 - 1$

$\therefore \quad T(n) = 2^K T(n-K) + 2^K - 1 \quad , \; T(0) = 1$

$n - K = 0$

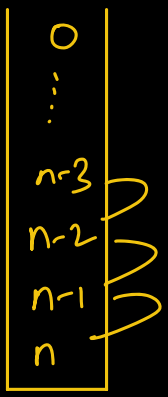
$\therefore \boxed{n = K}$

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

$= 2^n + 2^n - 1$

$T(n) = 2^{n+1} - 1 \Rightarrow \underline{O(2^n)}$

$SC \Rightarrow O(n)$



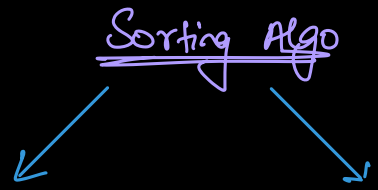
Sorting :

arranging data in some inc / dec order based on parameters.

Eg:- 4 8 10 14 24 (inc.)

Eg:- 24 14 10 8 4 (dec.)

Eg: 1(1) 3(2) 5(2) 7(2) 4(3) 9(3) 6(4) 10(4) 12(6) (no. of factors)



where SC : O(1)

w/o extra memory.

Eg: Famous Movie Star Names

	Fees
Deepika	100 cr
SrK	100 cr
Rajni sir	100 cr
Hritik	40 cr
Salman	10 cr
Vijay	120 cr
M Babu	70 cr

Salman	10 cr
Hritik	40 cr
M Babu	70 cr
Deepika	100 cr
SrK	100 cr
Rajni sir	100 cr
Vijay	120 cr

* Relative order is preserved after sorting ⇒ Stable sort

Stable Sort: 2 data points having same parameter value, their relative order before & after sorting remains same

⇒ STABLE SORT

Eg: arr[] : 10 2 3 1 2 9

Sort: 1 2 2 3 9 10 ⇒ Stable Sort

Qn: Given a[N], find K^{th} smallest element.

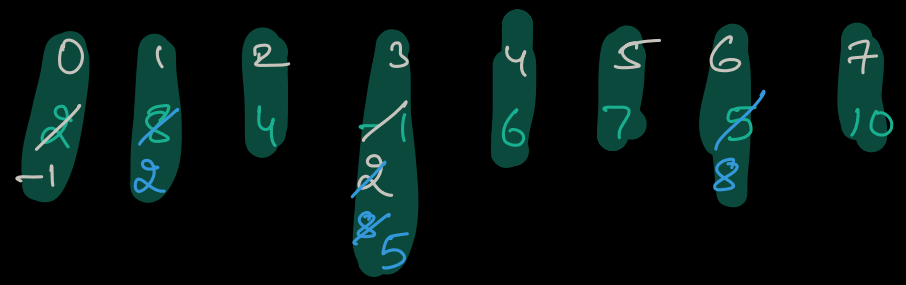
Eg: a[8] = 2 8 4 -1 6 7 5 10

K=1	K=2	K=3	K=4
-1	2	4	5

idea: Sort the array & return $(K-1)^{th}$ index. TC: O(n log n)

Another idea:

Eg: a[8] =



Min idx	Range	idx	
0	[0-7]	3	swap(a[0], a[3])
1	[1-7]	3	swap(a[1], a[3])
2	[2-7]	2	swap(a[2], a[2])
3	[3-7]	6	swap(a[3], a[6])
4	[4-7]	4	swap(a[4], a[4])
5	[5-7]	5	swap(a[5], a[5])
6	[6-7]	6	swap(a[6], a[6])
7	[7-7]	7	swap(a[7], a[7])

* This is called **Selection sort**

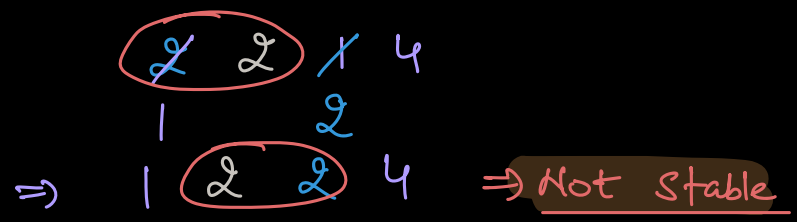
↳ Select min in the range.

```
int[] selectionSort (int a[]) {
    for (i=0; i<n; i++) {
        // ith iteration
        minVal = a[i], minIdx = i
        for (j = i; j<n; j++) {
            if (a[j] < minVal) {
                minVal = a[j]
                minIdx = j
            }
        }
        swap (a[i], a[minIdx])
    }
}
```

TC : $O(n^2)$

SC : $O(1)$

↳ in-place



⇒ Not Stable

Qn: Given a[N]. Sort array by swapping adjacent elements.

Eg:

	0	1	2	3	4	5	6	7
	4	6	4	3	9	-1	5	2
		4	6 3	6	-1	9 5	9 2	9
i=0	4	4	3	6	-1	5	2	9
j=n-2		3	4	-1	6 5	6 2	6	
i=1	4	3	4	-1	5	2	6	9
j=n-3	3	4	-1	4	2	5		
i=2	3	4	-1	4	2	5	6	9
j=n-4		-1	4	2	4			
i=3	3	-1	4	2	4	5	6	9
j=n-5								

$i+j = n-2$

$j = n-i-2$

Continue the process ... everything will be sorted.

```
bubbleSort(int a[]) {
    for(i=0; i<n; i++) {
        c=0
        for(j=0; j<=n-i-2; j++) {
            if(a[j] > a[j+1]) {
                swap(a[j], a[j+1])
                c = c+1
            }
        }
        if (c==0) { break }
    }
}
```

TC: $O(n^2)$

SC: $O(1)$

↳ In-place + Stable

Best Case TC: $O(n)$