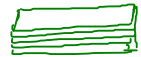
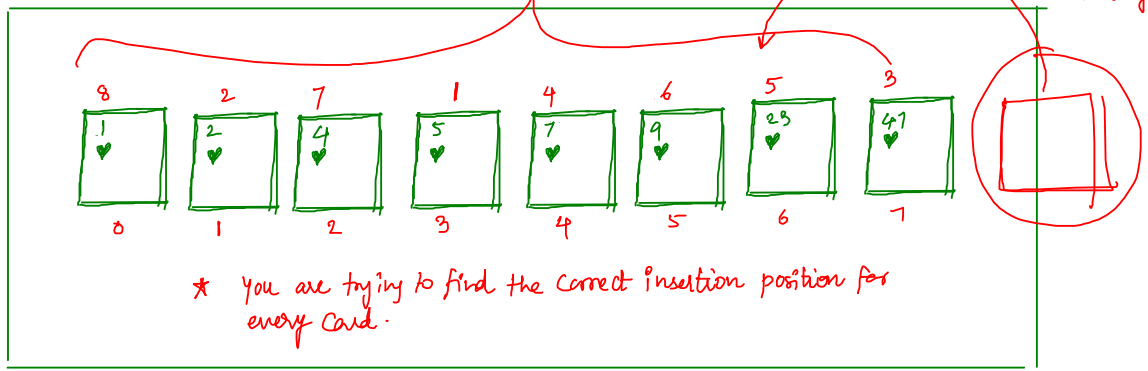
★ Insertion sort :



deck of card

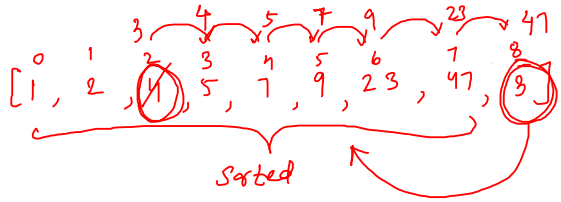
Sorted

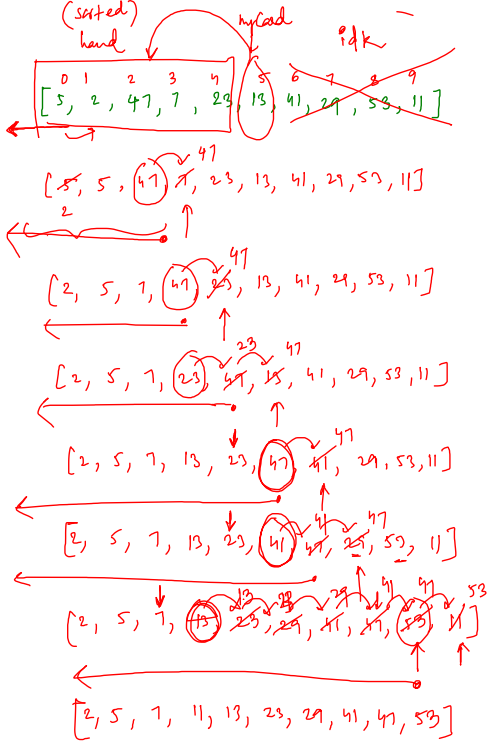
★ You trying to insert an element in a sorted array.



★ You are trying to find the correct insertion position for every card.

hand





5 > 2
2 < 5 → shift towards right

47 < 5 ✗

7 < 47 →
7 < 5 → ✗

23 < 47 → ✓
23 < 7 → ✗

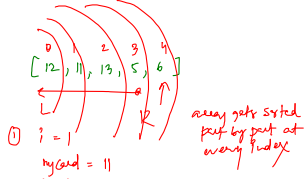
13 < 47 → ✓
13 < 23
13 < 7 → ✗

41 < 47 → ✓
23 < 47 → ✗

29 < 47 → ✓
29 < 41
29 < 23 → ✗

53 < 47
11 < 53 11 < 41
11 < 47

```
// Insertion sort
function insertionSort(arr) {
  // We are getting cards one by one
  for (let i = 1; i < n; i++) {
    // I got ith card place it in the right position
    const myCard = arr[i];
    let j = i - 1;
    while (j >= 0 && myCard < arr[j]) {
      arr[j + 1] = arr[j]; // shifting towards right
      j--;
    }
    arr[j + 1] = myCard;
  }
}
```



array gets sorted part by part at every index

① $i = 1$

myCard = 11

$j = i - 1 = 0$

a) $j = 0$ && $11 < 12$ → shift

arr[0+1] = arr[0]

arr[1] = arr[0]

arr[j] = 12

j-- ⇒ $j = -1$

[12, 12, 13, 5, 6]

b) $j = -1$ → break

arr[j+1] = myCard

arr[-1+1] = 11 ⇒ arr[0] = 11

[11, 12, 13, 5, 6]

② $i = 2$

myCard = 13

$j = i - 1 = 2 - 1 = 1$

~~a)~~ ① $= 0$ && $13 < 12$

arr[j+1] = myCard

arr[1+1] = 13

arr[2] = 13

[11, 12, 13, 5, 6]

③ $i = 3$

myCard = 5

$j = i - 1 = 2$

a) $j >= 0$ && $5 < 13$

arr[j+1] = arr[j]

arr[3] = arr[2]

arr[3] = 13

j--;

[11, 12, 13, 13, 6]

b) $j >= 1$, $5 < 12$

arr[2] = arr[1]

arr[2] = 12

j--;

[11, 12, 12, 13, 6]

c) $j >= 0$, $5 < 11$

arr[1] = arr[0]

arr[1] = 11

j--;

~~d)~~ $j = -1$

[11, 11, 12, 13, 6]

arr[j+1] = myCard

arr[-1+1] = 5 ⇒ arr[0] = 5

[5, 11, 12, 13, 6]

$i = 0 \rightarrow 0$

* T.C : $i = 1,\ j = 0 \rightarrow 0$ (1) $j = 0$

$i = 2,\ j = 1 \rightarrow 0$ (2) $j = 1, 0$

$i = 3,\ j = 2 \rightarrow 0$ (3) $j = 2, 1, 0$

$i = 4,\ j = 3 \rightarrow 0$ (4) $j = 3, 2, 1, 0$

$i = 5,\ j = 4 \rightarrow 0$ (5) $j = 4, 3, 2, 1, 0$

$\Rightarrow 1 + 2 + 3 + 4 + 5 + \cdots + n-1$

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

$\Rightarrow O(N^2)$ → worst case : [6, 5, 4, 3, 2, 1]

is it the same with best case ?

⇒ when arr is completly sorted : [1, 2, 3, 4, 5, 6]

TC : O(N)

$j = 0 \rightarrow 0$ $(2 < 1) \times$

$j = 1 \rightarrow 0$ $(3 < 2) \times$

$j = 2 \rightarrow 0$ $(4 < 3) \times$

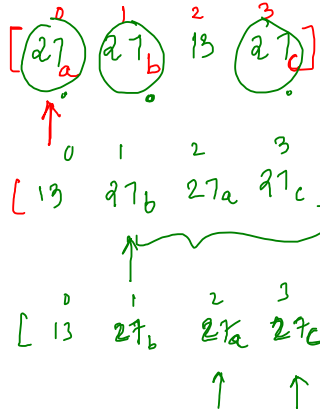
$j = 3 \rightarrow 0$ $(5 < 4) \times$

$j = 4 \rightarrow 0$ $(6 < 5) \times$

* inner while loop is not running

bubble / selection / insertion
→ bubble, selection $O(N^2)$
→ insertion → $O(N^2)$ → $O(N)$
* insertion perform much better on partially sorted or sorted arrays.

Selection sort is not stable :

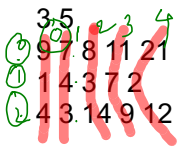


* order of equal elements is changed

```
if (minEle < arr[i])
    minEle = arr[i]
```

⇒ [13, 27a, 27b, 27c] (expected order after sorting)

* AS sorting 1 :



⇒

	0	1	2	3	4
0	1	3	3	7	2
1	4	4	8	9	12
2	3	7	14	11	21

(Col-wise sorted matrix)

⇒ arr.sort()

c=0 ①

col = [9, 1, 4]

= [1, 4, 9]

(0)(0) = col(0) = 1

(1)(0) = col(1) = ④

(2)(0) = col(2) = 9

0 1 2

[1, 4, 9]

c=0 ① col = [9, 1, 4]

c=1 ② col = [7, 4, 3]

c=2 ③ col = [8, 3, 14]

c=3 ④ col = [11, 7, 9]

c=4 ⑤ col = [21, 2, 12]

```
for(let c = 0; c < cols; c++) {
    let column = [];
    for(let r=0; r<rows; r++) {
        column.push(mat[r][c]);
    }
    column.sort() → (no. of ele = rows)
    for(let r=0; r<rows; r++) {
        mat[r][c] = column[r];
    }
}
```

Debugging arr.sort():

arr: [3, 8, 14]

arr.sort()

⇒ ["3", "8", "14"]

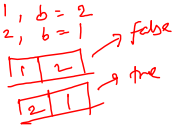
⇓ sort

["14", "3", "8"]

⇒ ["21", "2", "12"]

⇒ ["12", "2", "21"]

a = 1, b = 2
a = 2, b = 1



"1" "11" "111" "12" "13" "14" "21" "211" "212"

"a" "aa" "aaa" "ab" "aba" "abb" "ba" "bca" "bba"

* We can pass functions as parameters
* In JS, sort() can accept a function.
* This function acts as comparision between two elements.

```
function compare(a,b) {
    return a - b;
}
```

Should I swap or not

<= 0 (false) → no swap

> 0 (true) → swap

a = 1, b = 2 ⇒ a - b = -1 ⇒ X

a = 2, b = 1 ⇒ a - b = 1 ⇒ swap

* before sort ⇒ all elements are converted to string then dictionary order wise sorted.

* replace your compare function with my compare function.

```
552  function sortCol(mat, n, m) {
553    for (let c = 0; c < m; c++) {
554      const col = [];
555      for (let r = 0; r < n; r++) {
556        col.push(mat[r][c]);
557        // same as col[r] = mat[r][c];
558      }
559      col.sort((a, b) => Number(a) - Number(b));
560      for (let r = 0; r < n; r++) {
561        mat[r][c] = col[r];
562      }
563    }
564    printMatrix(mat);
565  }
```

m → n log n → n → m * n

* TC: $(m * (n + n\log n + n)) + m * n$

$\Rightarrow (m*n) + m*n\log n + m*n + m*n$

$\Rightarrow O(m * n\log n)$

* SC: $O(n)$ — len(col) = no. of rows = n

$\Rightarrow$ col array

2nd Smallest :

run selection sort until 2 iteration (2 fix indices)

return arr[1];

after 1st itr → arr[0] is fixed (1st min) → O(n)
after 2nd itr → arr[1] is fixed (2nd min) → O(n)
} O(2n) ⇒ O(n)

2nd Largest :

run bubblesort until 2 iterations.