📌 What is Sorting?

🎒 Indian Classroom Sorting Analogy

Picture this: It's the first day of school, and the teacher walks in dramatically — like she's entering a TV serial 🎭

She looks at the classroom full of kids running wild and says:

> "Roll number ke hisaab se line mein lag jao! Abhi!" 📢👩‍🏫

Now you've got:

- Rahul (Roll No. 12) standing next to Astha (Roll No. 3)
- Chintu (Roll No. 7) scrolling reels at the back
- And that one kid with Akarsh (Roll No. 1) just chilling by the window 😎

So what happens?

Everyone starts finding their correct spot — **Roll No. 1 in front**, **Roll No. 2 next**, and so on, until the **line is perfectly sorted** 🔢✅

💡 That's Sorting!

Just like the teacher wanted the students in **order of roll number**, sorting in programming arranges data in a meaningful way — like **ascending or descending**.

💡 Definition of Sorting

Sorting is the process of arranging data or items in a specific order— usually **ascending** (smallest to largest 🔼) or **descending** (largest to smallest 🔽), such as:

- **Numerical order** (e.g.,1️⃣2️⃣3️⃣..)
- **Alphabetical order** (e.g., A, B, C...)
- **Chronological order** (e.g., by date or time ⏰)

For example:

- Sorting numbers: `3, 1, 4, 2` ➡️ `1, 2, 3, 4`
- Sorting words: `"banana", "apple", "cherry"` ➡️ `"apple", "banana", "cherry"`

Sorting helps organize data so it's easier to read, search, and analyze. It's widely used in computers for things like:

- 🔍 Searching more efficiently
- 📊 Displaying reports or results
- 🧮 Making calculations faster

It's commonly used in:

- 🗃️ Databases
- 📊 Spreadsheets
- 🔍 Searching algorithms
- 📚 Alphabetizing names or lists

📌 Stories to remeber?

Suppose you are either an educated person or someone not very aware of programming or algorithms, and you're given the marks of 1,000 students. You're asked to arrange them in order of rank, where the student with the highest marks gets Rank 1, the next highest gets Rank 2, and so on.

Now, how would you do that?

🧠 First Approach (Backbencher Style)

Since I'm a backbencher and want to play a game, I'll do things a little differently — I'll give **Rank 1000** to the student with the **highest marks**, Rank 999 to the next highest, and so on.

Here's how I'd do it:

- First, I would find the student with the **highest marks** and place them at the **last position**.
- Assuming there are `n` students, I would now have `n - 1` students left.
- From the remaining list, I would again find the student with the **next highest marks** and place them at the **second last position**.
- I would keep repeating this process — always picking the highest from what's left and placing it one step earlier — until all students are arranged in **ascending order of marks** (from lowest to highest).
  Once that's done, I would start assigning ranks from **1 to 1000**, starting with the student who has the **lowest marks** (now at the first position).
- This way, the **top scorer** ends up with **Rank 1000**, and the **lowest scorer** gets **Rank 1** — exactly how I like it.

😎 Second Approach (Also Simple and Fun)

This one's even easier and more straightforward:

- From the entire dataset, I simply pick the **student with the lowest marks** and place them at the **first position**.
- Now I have `n - 1` students left.
- From this remaining list, I again find the **lowest marks** and place that student at the **second position**.
- I keep repeating this process — always picking the minimum and placing it in the next available position — until all students are sorted in **ascending order of marks**.
- Once sorted, I can assign ranks from **1 to 1000** directly — first position gets Rank 1, second gets Rank 2, and so on.

⏰ Third Approach

- When I was in school, I used to be late most of the time. By the time I arrived, the morning assembly had already started. Since I was late, I couldn't join my class line in the usual way. So, the teacher advised me to simply stand at the end of the row.
- Each row represented a class, and usually, students were lined up by height. Since I was out of order (**the "fringe element" disturbing the sorted line**), **I used to compare myself with the student standing in front of me. If their height was greater than mine, I would swap places with them**. I kept doing this until I reached a position where the person in front of me had a height less than mine, and the person behind me had a height greater than mine.
- That's when I stopped, knowing I had found the right place based on height. I also knew the rest of the students were already in the correct (sorted) order—only I was the one out of place initially.

🧼 Bubble Sort

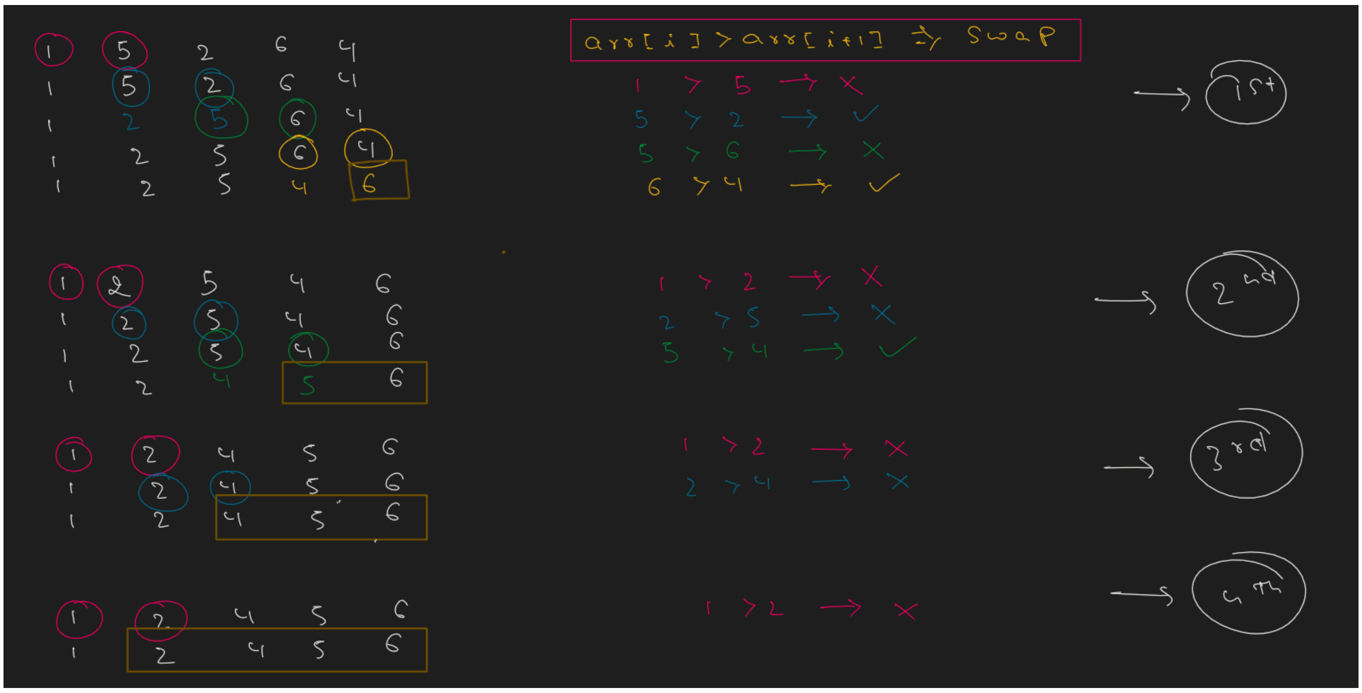
“Bubble the maximum element at last”

🧠Soap Bubbles Rising

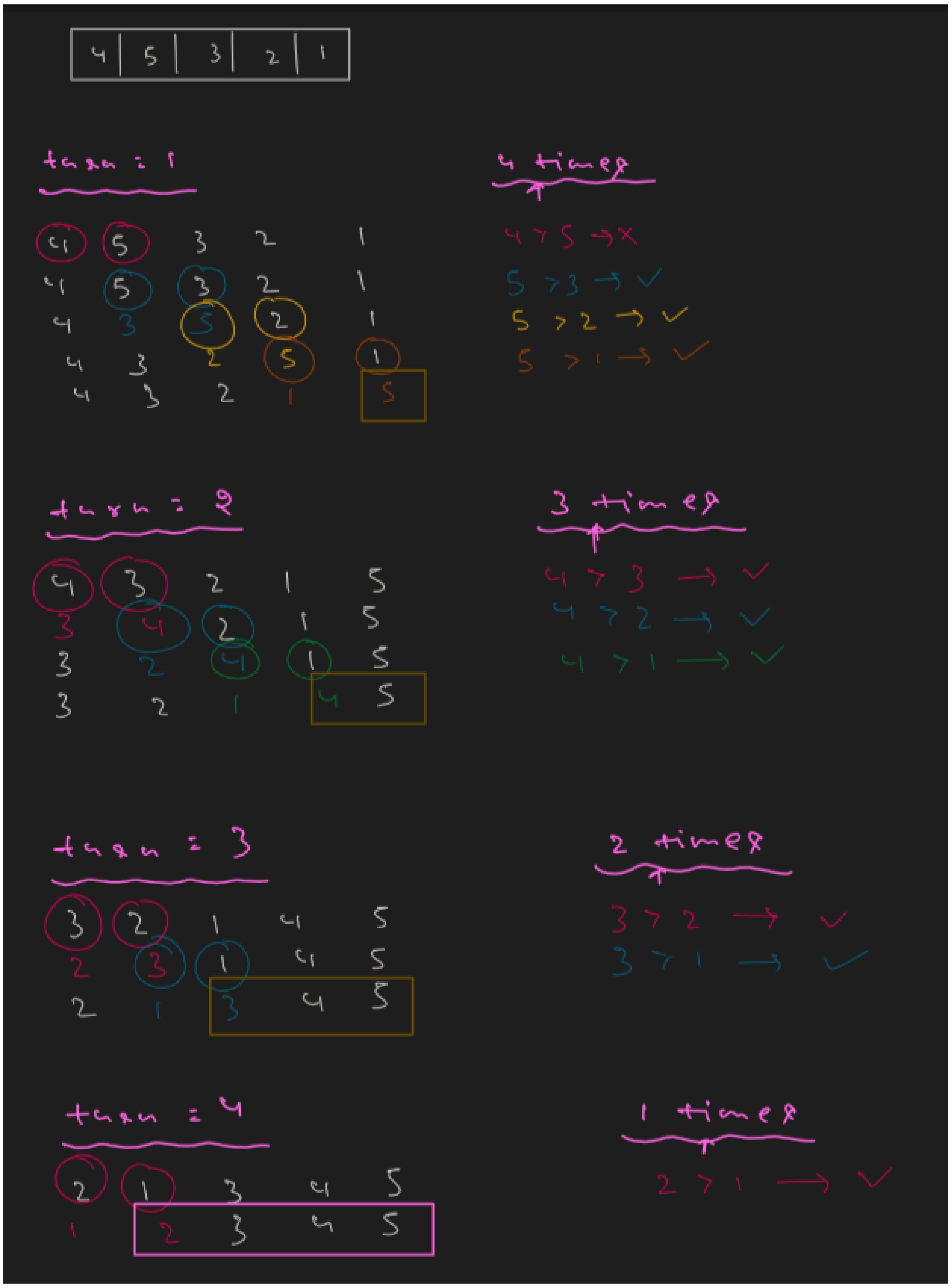
Imagine soap bubbles in a water tank. The largest bubbles rise to the top with each pass.

- Each pass compares adjacent bubbles.
- If a bigger bubble is below a smaller one, they swap.
- After each full pass, the largest unsorted bubble reaches its correct position at the top.

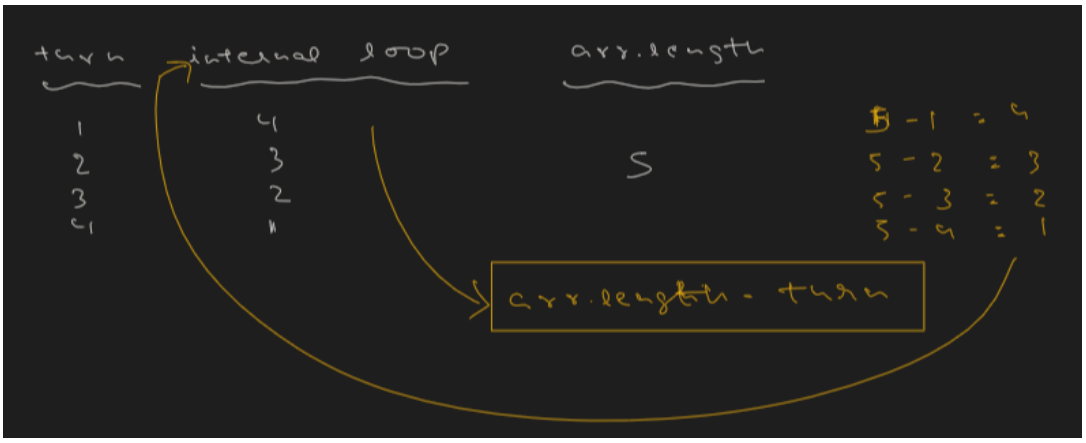
This mirrors how Bubble Sort pushes the largest elements to the end of the array in each iteration.



```java
public class Main {

      public static void main(String[] args) {
        int[] arr = { 4, 5, 3, 2, 1 };
          bubbleSort(arr);
            for (int i = 0; i < arr.length; i++) {
                  System.out.print(arr[i] + " ");
            }
      }

      public static void bubbleSort(int[] arr) {
            for (int turn = 1; turn < arr.length; turn++) {
                  for (int i = 0; i <arr.length-turn ; i++) {
                        if(arr[i]>arr[i+1]) {
                              int temp=arr[i];
                              arr[i]=arr[i+1];
                              arr[i+1]=temp;
                        }
                  }
            }
      }

}
```



4 | 5 | 3 | 2 | 1
turn = 1
4 times
4 > 5 → ×
5 > 3 → ✓
5 > 2 → ✓
5 > 1 → ✓
4 5 3 2 1
4 5 3 2 1
4 3 5 2 1
4 3 2 5 1
4 3 2 1 5
turn = 2
3 times
4 > 3 → ✓
4 > 2 → ✓
4 > 1 → ✓
4 3 2 1 5
3 4 2 1 5
3 2 4 1 5
3 2 1 4 5
turn = 3
2 times
3 > 2 → ✓
3 > 1 → ✓
3 2 1 4 5
2 3 1 4 5
2 1 3 4 5
turn = 4
1 times
2 > 1 → ✓
2 1 3 4 5
1 2 3 4 5



turn
internal loop
arr.length
1 2 3 4
4 3 2 1
5
5 - 1 = 4
5 - 2 = 3
5 - 3 = 2
5 - 4 = 1
arr.length - turn

🧮 Selection Sort?

"Put minimum element at first position"

🎯 Minimum index

🥔 Analogy: Searching for the Cheapest Aloo in the Market

Imagine you're in a vegetable market (sabzi mandi), and you're standing at **shop number 5** in a long row of **n aloo vendors**. Each vendor is shouting prices for their potatoes (aloo): ₹20, ₹18, ₹25, ₹15, etc.

You decide:

> "Bas! From this shop onwards, I want to find **the cheapest aloo**—and note which shop is selling it."

So you walk from shop 5 to the end, checking each price and writing on my hand **which shop number has the cheapest one, and what is the price**. Each time you get a shop which has a price lesser than the price written on your hand, you will overwrite both the values, initially you have considered shop 5 is the cheapest aloo shop.

You don't care about vendors before shop 5—because you already walked past them. 😅

At the end, you shout:

> "Shop number **k** is selling the cheapest aloo!"

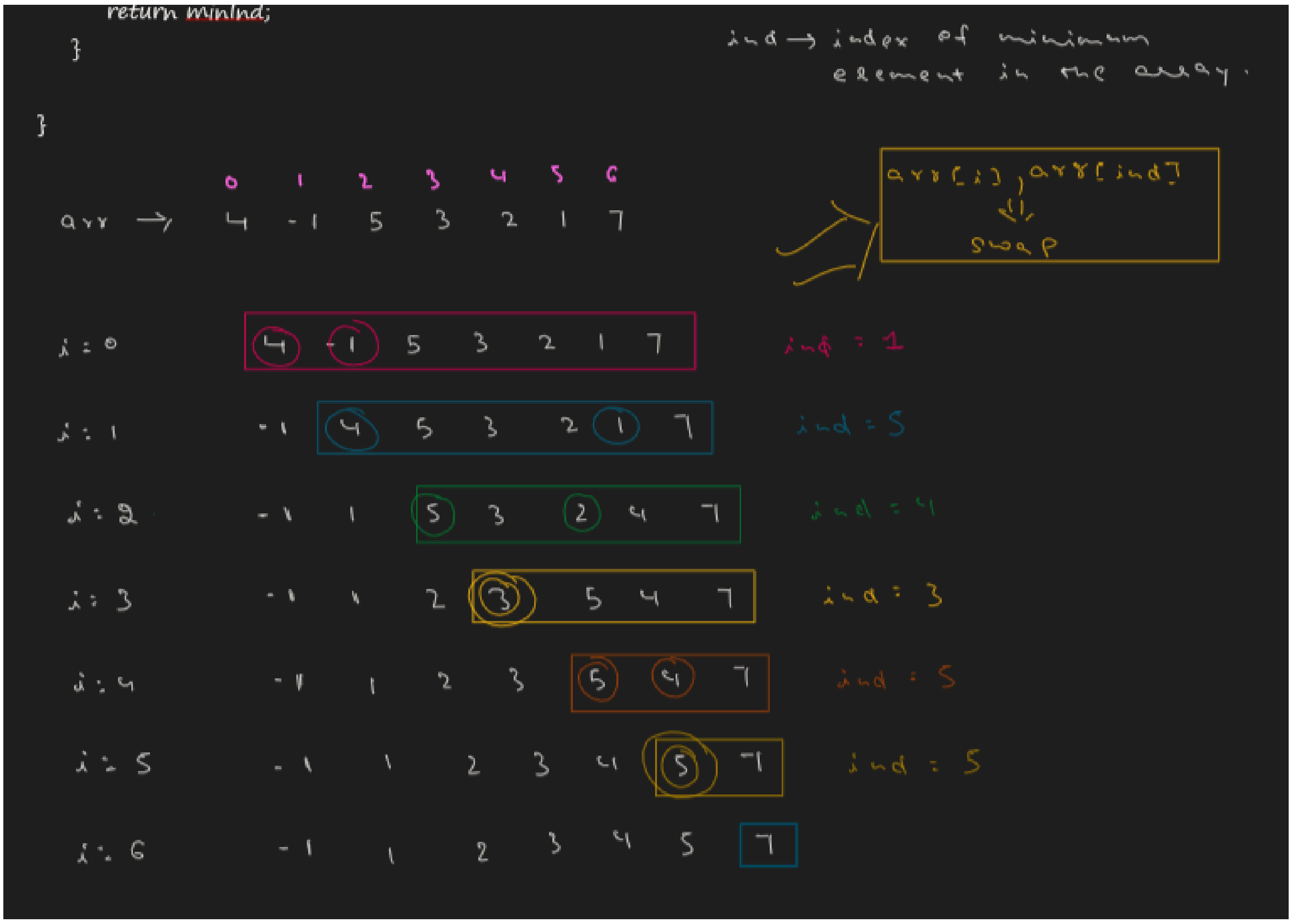
That's exactly what you're doing in the array:

- Each shop = an array element.
- Aloo price = value at that index.
- Shop number = array index.
- You return the index (shop number) with the cheapest price (minimum value), starting from your current location (index 5).

Can you find the index of minimum value in an array of length n, starting from a given index?

For example, if I give you index 5, you need to find the minimum value in the subarray starting from index 5 to the end of the array, and return the index of that minimum value.

```java
public class Main {

      public static void main(String[] args) {
        int[] arr = { 4, -1, 5, 3, 1, 2 };
            System.out.println(minimumElementIndex(arr, 2));
      }

      public static int minimumElementIndex(int[] arr, int idx) {
            int minIndex = idx;
            for (int i = idx + 1; i < arr.length; i++) {
                  if (arr[minIndex] > arr[i]) {
                        minIndex = i;
                  }
            }
            return minIndex;
      }

}
```



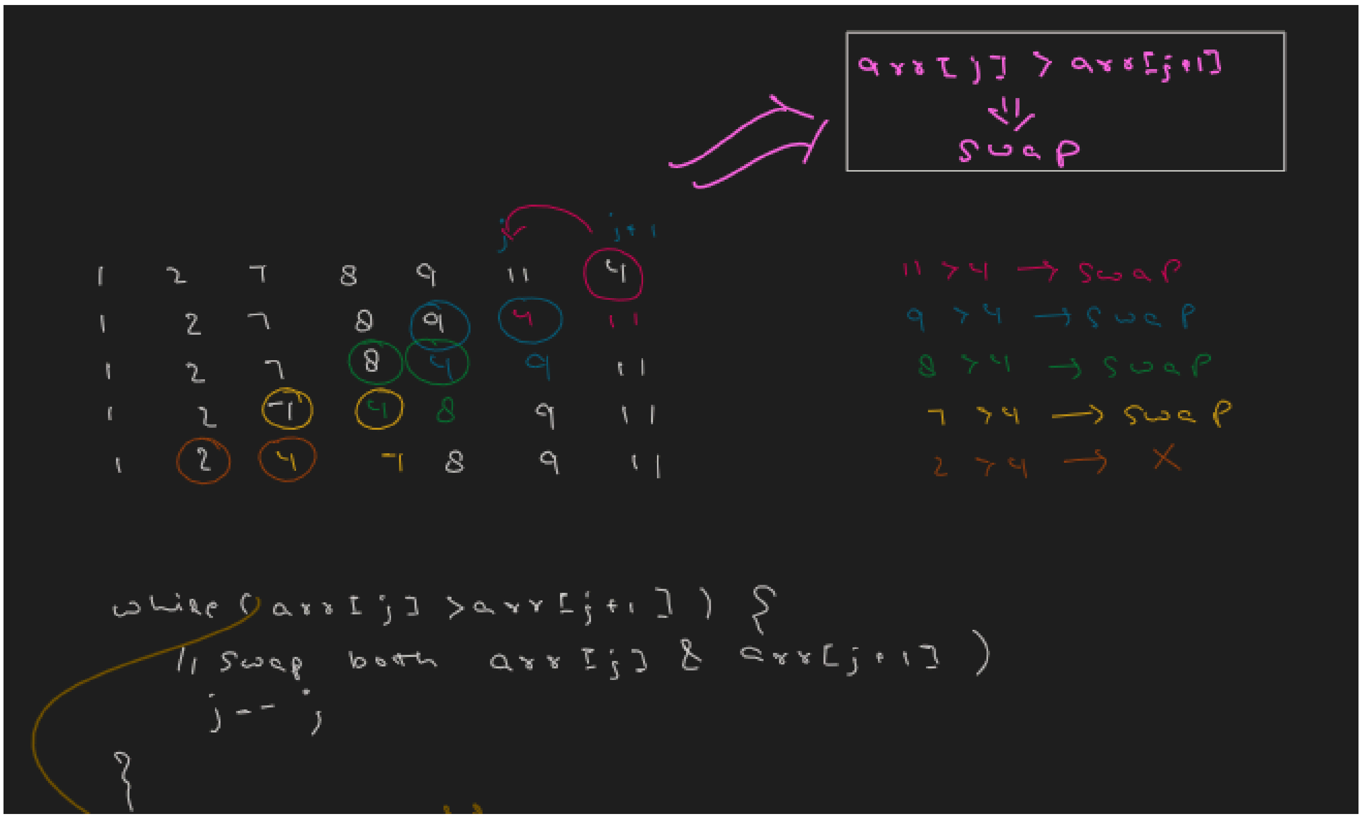
```java
public class Main {

      public static void main(String[] args) {

            int[] arr = { 4, -1, 5, 3, 1, 2 };
            selectionSort(arr);
            for (int i = 0; i < arr.length; i++) {
                  System.out.print(arr[i] + " ");
            }

      }

      public static void selectionSort(int[] arr) {

            for (int i = 0; i < arr.length; i++) {
                  int idx=minimumElementIndex(arr, i);
                  int temp=arr[i];
                  arr[i]=arr[idx];
                  arr[idx]=temp;
            }

      }

      public static int minimumElementIndex(int[] arr, int idx) {

            int minIndex = idx;
            for (int i = idx + 1; i < arr.length; i++) {
                  if (arr[minIndex] > arr[i]) {
                        minIndex = i;
                  }
            }
            return minIndex;

      }

}
```

If you observe carefully, you'll notice that selection sort performs fewer element swaps than bubble sort.

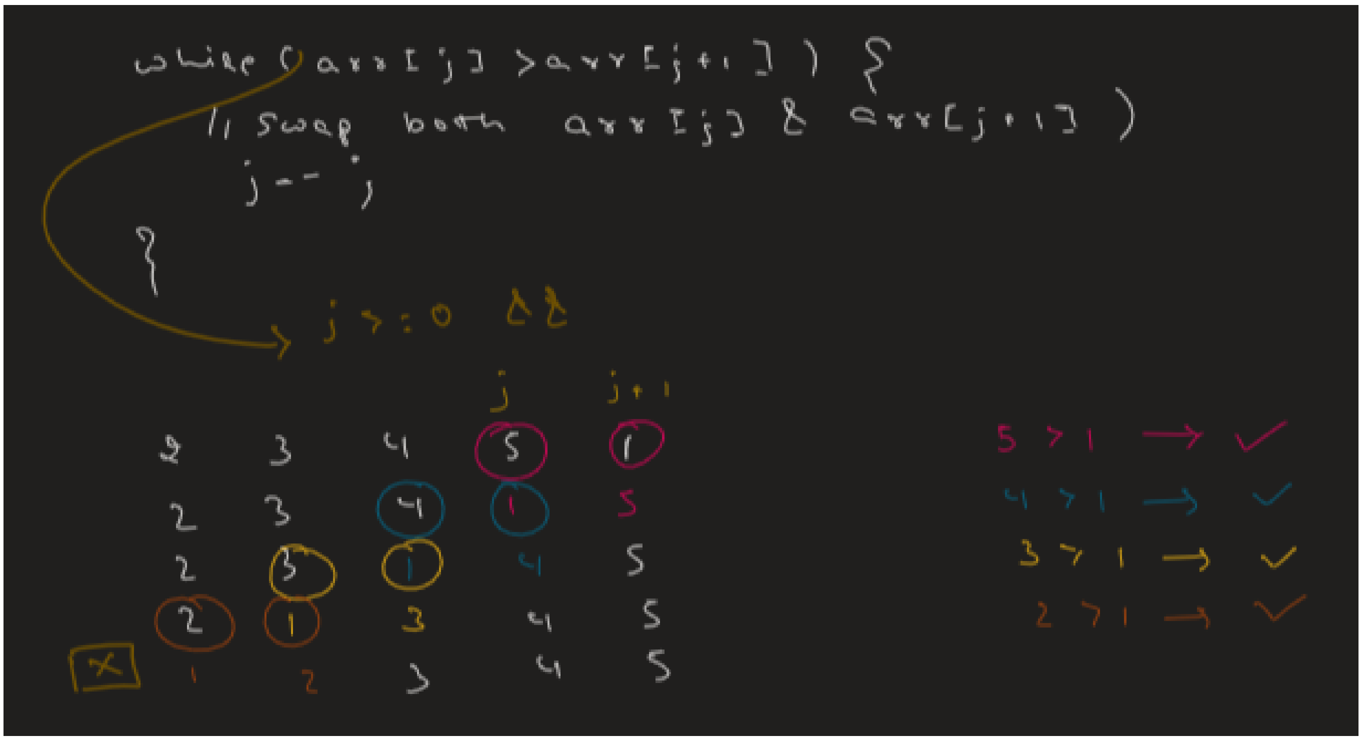
📥 Insertion Sort?

So basically, in insertion sort, you start with the whole array is sorted. The last element is at an unsorted position. The idea is to **take that element and shift towards its correct position in the sorted part of the array**. That's the core concept of insertion sort.

Can you help me **move the last element in the array to its correct sorted position**?
The rest of the array is already sorted; only the last element is out of place.



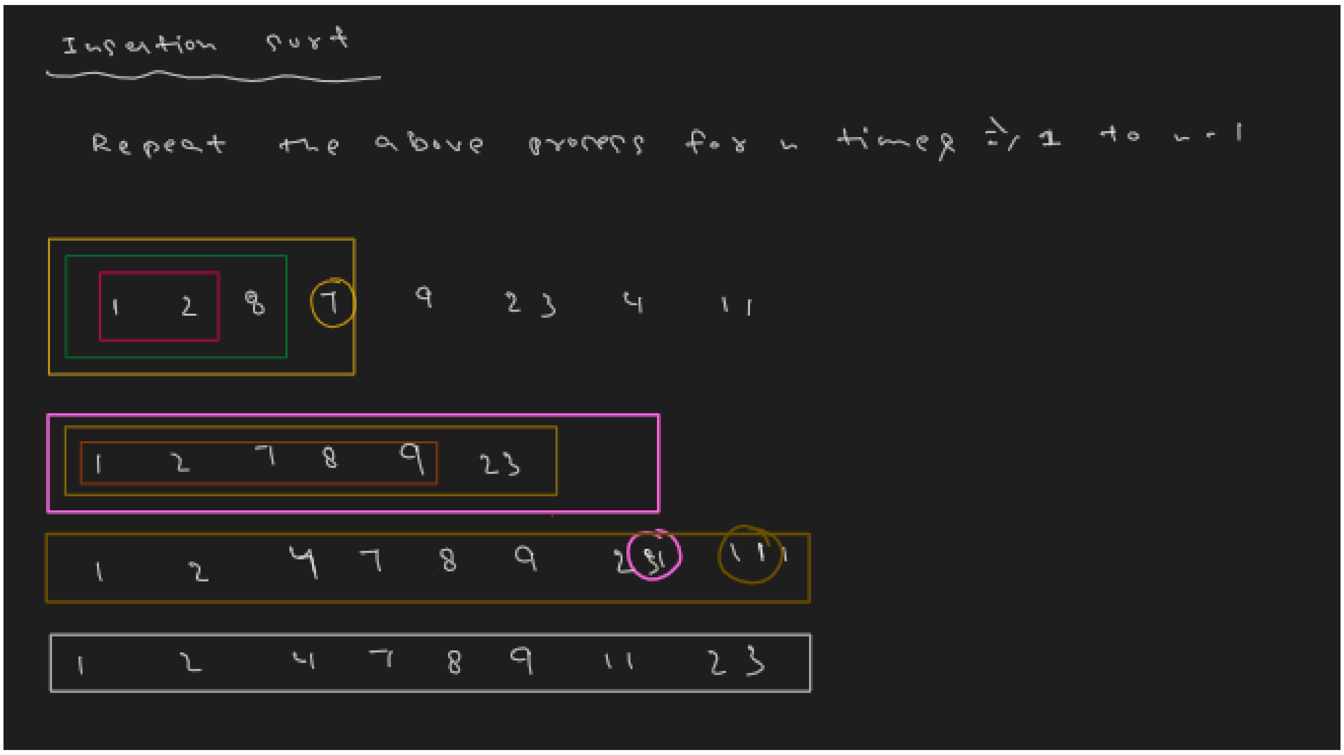
```
while (arr[j] > arr[j+1]) {
    // swap both arr[j] & arr[j+1] )
    j--;
}
```



```
while (arr[j] > arr[j+1]) {
    // swap both arr[j] & arr[j+1] )
    j--;
}
```

```java
public class Main {

	public static void main(String[] args) {

		int[] arr = { 4, 1, 3, 4, 5, 9, 6, 2 };
		insertLastElement(arr,arr.length-1);
		for (int i = 0; i < arr.length; i++) {
			System.out.print(arr[i] + " ");
		}
	}

	public static void insertLastElement(int[] arr, int i) { // i-> last index
		int j = i - 1;
		int item = arr[i];
		while (j >= 0 && arr[j] > item) {
			arr[j + 1] = arr[j];
			arr[j] = item;
			j--;
		}
		// pos = j+1
	}

}
```

Will repeat the above process for all elements from 1 to n-1 index to get a sorted array.😎



```java
public class Main {

      public static void main(String[] args) {

        int[] arr = { 4, 1, 3, 4, 5, 9, 6, 2 };
            insertionSort(arr);
            for (int i = 0; i < arr.length; i++) {
                  System.out.print(arr[i] + " ");
            }

      }

      public static void insertionSort(int[] arr) {

            for (int i = 1; i < arr.length; i++) {
                  insertLastElement(arr, i);
            }

      }

      public static void insertLastElement(int[] arr, int i) { // i-> last index

            int j = i - 1;
            int item = arr[i];
            while (j >= 0 && arr[j] > item) {
                  arr[j + 1] = arr[j];
                  arr[j] = item;
                  j--;
            }
            // pos = j+1

      }

}
```

Note: Dry run once.