**1. Problem Discussion :**

We just have to sort the integers in increasing order. But we should use the insertion sort algorithm for that purpose. There are already some predefined functions in the template.

Example :

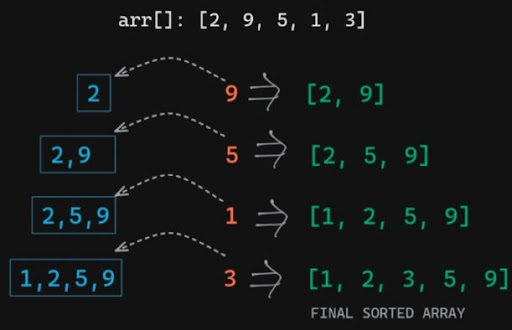
Input : [2 3 9 6 4 1] Output : [1 2 3 4 6 9]

**2. Approach :**

Let's start with a basic understanding of how insertion sort works. We'll use the following array as an example: 2, 9, 5, 1, 3. The insertion sort will be broken down into steps. But first, keep in mind that "an array of size one is always sorted." There is no other number to compare with if there is only one element, hence it is always sorted (both increasing and decreasing). As a result, we must conduct operations on the remaining four digits. Iterations will be n-1 for n integers.

1• We'll figure out where 9 is in only 2, 9 seconds. It will be placed after 9 because 9 is greater than 2. So [2, 9] will be the final order. 2• In just 2, 9, 5, we shall determine the position of 5. Because 5 is between 2 and 9, it will be positioned as [2, 5, 9]. HOW? We'll talk about it later. 3• The position of 1 in 2, 5, 9, 1 will have to be determined. Because 1 is the smallest, it will be placed first. As a result, the ultimate order is [1, 2, 5, 9]. 4• Now we'll figure out where 3 belongs in the numbers 1, 2, 5, and 9, and 3 will be between 2 and 5. So [1, 2, 3, 5, 9] is the final order.

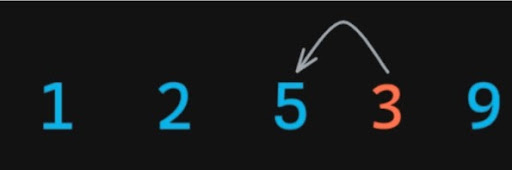
Here is an illustration demonstrating the same:

****

Let's take a closer look at one iteration in particular. Consider the previous iteration, in which we already had 1, 2, 5, and 9 and wish to "insert" 3 in the proper place. 1. 3 is compared with the last element 9.

****

Since 3 is less than 9, we will just swap 3 and 9.

****

Now 3 is still less than 5 we will swap and move on.

****

3 is now more than 2. Because the array is already sorted, 3 must be greater than all of the items prior to 2. As a result, we've discovered the perfect spot for 3 and will come to a halt here.

****

So this operation will be performed for all the numbers. I insist you take a pen and paper and do this yourself. If you are stuck at any point you can watch the video. We may, in the best-case scenario, have all of the pieces sorted. Only one procedure will be carried out in this situation. Because the preceding element in each insertion will be smaller than the current one. Let's go over the steps once more: Because the array of size 1 is already sorted, start at the first index. From i-1 to 0, we'll conduct an inner loop. Let's use the letter j to represent it. We shall switch if the j-th value is bigger than the j+1-th value. Otherwise, the inner loop will be broken. This will be repeated for all n-1 elements. Every iteration, it seems to be bubbling in the opposite direction. Because we swap and move in the bubble sort as well.

**3. Code :**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void insertionSort(int[] arr) {

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

for (int j = i - 1; j >= 0; j--) {

if (isGreater(arr, j, j + 1)) {

swap(arr, j, j + 1);

} else {

break;

}

}

}

}

// used for swapping ith and jth elements of array

public static void swap(int[] arr, int i, int j) {

System.out.println("Swapping " + arr[i] + " and " + arr[j]);

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

// return true if jth element is greater than ith element

public static boolean isGreater(int[] arr, int j, int i) {

System.out.println("Comparing " + arr[i] + " and " + arr[j]);

if (arr[i] < arr[j]) {

return true;

} else {

return false;

}

}

public static void print(int[] arr) {

for (int i = 0; i < arr.length; i++) {

System.out.println(arr[i]);

}

}

public static void main(String[] args) throws Exception {

Scanner scn = new Scanner(System.in);

int n = scn.nextInt();

int[] arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = scn.nextInt();

}

insertionSort(arr);

print(arr);

}

}

**4. Analysis:**

Time Complexity :

O(n^2) We're only running two loops. What do you believe the worst case scenario for Insertion sort will be? Yes, if all of the numbers are arranged backwards (in descending order). Then we'll have to do 1+ 2 + 3 +... swaps for each iteration of n-1. So, this is an Arithmetic Progression with the outcome n\*(n+1)/2 = n^2. Worst-case scenario: O (n^2). Best Case: O(n) [This was already explored]

Space Complexity :

O(1) Because the only core operation is a swap, we're working with constant auxiliary space. Hopefully, you have grasped the concept of this Algorithm. If you haven't already, I recommend watching the entire video and running the code by hand. That way, you'll be able to grasp the logic behind this lovely sorting method. Best wishes and keep learning!!