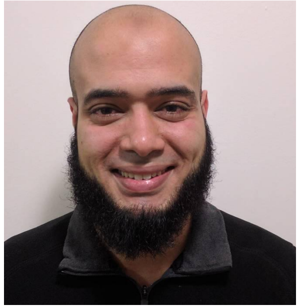
Algorithms

Insertion Sort 1

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



The Sorting problem

- Given an array of numbers, order them from small to large
  - Input: [9,2, 10, 0, 5, 3, 90, 85]
  - Output: [0, 2, 3, 5, 9, 10, 85, 90]
  - Observe: A[i] <= A[i+1]
  - Observe: If A <= B and B <= C, then A <= C (**transitivity**)
- Why Sorting?
  - It makes many computational problems easier to solve
  - Many applications involve sorting items
    - The LeetCode website allows ordering the tasks based on “acceptance ratio”

About elements order

- 1, 2, 3, 4, 5, 6
  - This is an **increasing** sequence. Sometimes we use the term **strictly increasing**
- 1, 2, **3, 3, 3**, 4, 5, 6, 6
  - This is not strictly increasing, as we have **duplicates**
  - We call it a **non-decreasing** sequence
  - Some sources sometimes still call it increasing. So check out the definition.
- Similarly decreasing and non-increasing concepts
  - 6, 5, 4, 3, 2, 1 is a decreasing sequence
  - 6, 6, 6, 4, 3, 3, 3, 2, 1 is a non-increasing sequence
  - What about: 9, 2, 10, 0, 5, 3, 90, 85? Not ordered
- **Ascending order** means order (numbers/words) from smallest to largest from left to right. **Descending** order means order from largest to smallest

Sorting Algorithms

n	$n^2/4$	$n \lg n$
10	*25*	33
100	2,500	*664*
1,000	250,000	*9,965*
10,000	25,000,000	*132,877*
100,000	2,500,000,000	*1,660,960*

- There are many sorting algorithms!
- **Simple but inefficient**: Insertion, Selection, Bubble
  - O(n^2) algorithms
- **Efficient**: Merge sort, Quicksort, Heapsort
  - HeapSort is based on the **heap** data structure. We mentioned it in the DS course
  - O(n Logn) algorithms. Logn is $Log_2 n$. $log_2(256) = 8$. $log_2(4{,}294{,}967{,}296) = 32$
- Distribution-based techniques: Counting sort, Bucket sort, Radix sort
  - What if all the values are in the range [0, 100]?

Incremental Thinking

- Insertion sort is based on a simple and effective thinking technique
- It is called **Incremental thinking**
- The idea is simple. Let’s use a simple setup for it here
    - Assume we want to apply some function(array), for example sort(array)
    - The array has N elements
- You think this way
    - Assume, we know the answer for the first N-1 elements
    - How can we update it for N?
- If the previous question is applicable, we can apply it simply by starting from the first element in the array
    - Think of the first element as our base case
    - With each new element, the solution builds upon the solution found by all of the previous elements

Incremental Thinking: Let’s Apply It

- Given the array [9, 2, 10, 0, 5, 3, 90, 85], where N = 8. How to sort the array?!
- Q1) What is the sorting answer for the first N-1 elements?
- The first 7 values are [9, 2, 10, 0, 5, 3, 90].
- Sorting them gives [0, 2, 3, 5, 9, 10, 90]
- Q2) How can we **update** the sorted array, but also include the value **85**?
- Simply iterate from the *end of the array* and find the **first** element, where 85 is <= to it. In our case it is 90. Put it before it ⇒ [0, 2, 3, 5, 9, 10, **85**, 90]
- Then?
- Now start from the **2nd element**, and for each number put in its right place with the previous M-1 numbers!
- Done!

Let’s simulate: [9, 2, 10, 0, 5, 3, 90, 85]

- The first number is sorted. Start from idx = 1, at value 2
- Sorted so far [9]. Next number is 2. Remaining is [10, 0, 5, 3, 90, 85]
- Where to insert 2 to get [2, 9]? Directly before 9

Let’s simulate: [2, 9, **10**, 0, 5, 3, 90, 85]

- Sorted so far [2, 9]. Next number is 10. Remaining is [0, 5, 3, 90, 85]
- Where to insert 10 to get [2, 9, 10]? It is bigger than previous ones
  - Hence it stays in its location!

Let’s simulate: [2, 9, 10, 0, 5, 3, 90, 85]

- Sorted so far [2, 9, 10]. Next number is 0. Remaining is [5, 3, 90, 85]
- Where to insert 0 to get [0, 2, 9, 10]? In the first index
  - 0 vs 10. Smaller next
  - 0 vs 9. Smaller next
  - 0 vs 2. Smaller next
  - None. Put it here

Let’s simulate: [0, 2, 9, 10, **5**, 3, 90, 85]

- Sorted so far [0, 2, 9, 10]. Next number is 5. Remaining is [3, 90, 85]
- Where to insert 5 to get [0, 2, 5, 9, 10]? After 2
  - 5 vs 10. Smaller next
  - 5 vs 9. Smaller next
  - 5 vs 2. Greater than it.
  - Put it here

Let's simulate: [0, 2, 5, 9, 10, **3**, 90, 85]

- Sorted so far [0, 2, 5, 9, 10]. Next number is 3. Remaining is [90, 85]
- Where to insert 3 to get [0, 2, 3, 5, 9, 10]? After 2
    - 3 vs 10. Smaller next
    - 3 vs 9. Smaller next
    - 3 vs 5. Smaller next
    - 3 vs 2. Greater than it.
    - Put it here
- Do you realize now why we call it **insertion** sort?
    - We take an element and insert it in the **right** location!

Let’s simulate: [0, 2, 3, 5, 9, 10, **90**, 85]

- Clearly, 90 is already in the correct location for it

Let's simulate: [0, 2, 3, 5, 9, 10, 90, **85**]

- Remember the concept of incremental thinking?
  - Given that we sorted the first N-1 numbers: [0, 2, 3, 5, 9, 10, 90]
  - Where to **insert 85** to make it sorted? Before 90
- Your turn
  - Think about the correctness of what we did. Anything missing?
  - Code it!
  - Trace your code for the different potential test cases
    - Can you find a case where the code perform the fewest number of iterations? largest?
  - Analyze your approach, being mindful of time and space complexity
- Tip: Algorithms usually challenge several skills simultaneously
  - Thinking, coding, testing, debugging and proving.

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."