Insertion Sort

By:

Dustin Smith

Insertion sort is a simple sorting algorithm. Unfortunately, insertion sort is much less efficient when compared to quick sort, heap sort, and merge sort. There are some advantages to insertion sort though:

1. easy to implement,
2. performs well on small sets,
3. adaptive–better on partially sorted sets,
4. stable–doesn't change the relative order of the same keys,
5. in place additional memory, $\theta(1)$, and
6. online–can be sorted as values received.

How do we measure complexity for insertion sort? In the best case, it would take linear time to run through a sorted array, $\theta(n)$. In general, if we are inserting into a subarray with k elements, all k elements may have to shift over one unit. Let's say a comparison cost c. It could take up to $c \cdot k$ to insert into a subarray of k elements. If we were inserting into this array a value that is always less than every element, the index of k_0 would be at one the first time, two, three, ..., $n-1$. The total time spend would be

$$c + 2 \cdot c + 3 \cdot c + \cdots + (n-1) \cdot c = c \cdot (1 + 2 + 3 + \cdots + (n-1)).$$

We can see we have an arithmetic series where $a_n = a_1 + (n-1) \cdot c$ where c is the common difference between elements. The sum of this series is

$$\begin{aligned} S_n &= \frac{n}{2}[2 \cdot a_1 + a_n] \\ &= \frac{n}{2}[2 \cdot c + c \cdot (n - 1 + 1)] \\ &= \frac{c \cdot n^2}{2} + c \cdot n \end{aligned}$$

Thus, we have $\theta(n^2)$.

```python
def insertion_sort(array: List[int]) -> None:
  for idx in range(1, len(array)):
    curr = array[idx]
    curr_idx = idx

    while curr_idx > 0 and array[curr_idx - 1] > curr:
      array[curr_idx] = array[curr_idx - 1]
      curr_idx = curr_idx - 1

    array[curr_idx] = curr
```