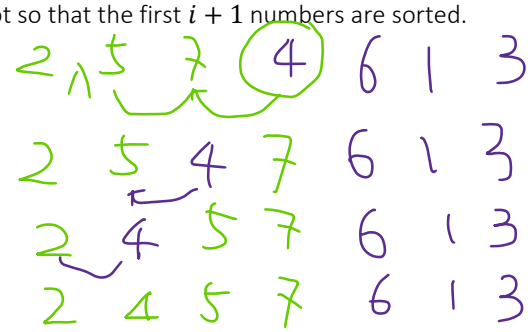
$\Theta(n^2)$ Sorting Algorithms

- **Insertion Sort**: if the first i numbers are sorted, we can insert the $(i+1)^{th}$ number in the array/list to the correct spot so that the first $i+1$ numbers are sorted.

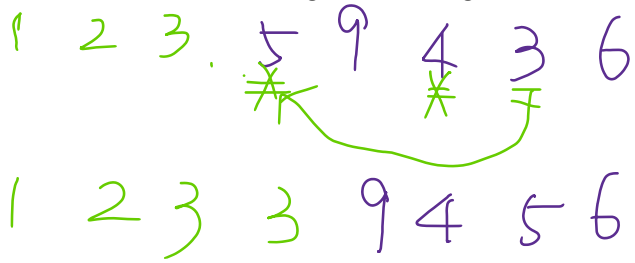


INSERTION SORT (A)

```
1   for j in range (1, A.length)
2       key = A[j]
3       // Insert A[j] into the sorted sequence A[1 ... j − 1]
4       i = j − 1
5       while i ≥ 0 and A[i] > key
6           A[i + 1] = A[i]
7           i = i − 1
8       A[i + 1] = key
```

1. What is the worst-case time complexity of insertion sort on a list/array of size n?
   - Line 2 to 8 run $n-1$ times.
   - Line 6 to 7 run at most $j-1, j=1 \dots n$ times, (this happens when the input list/array is sorted in decreasing order).
   - The worst-case time complexity of insertion sort is $\sum_{j=1}^{n-1} c_1 + c_2(j-1) = \frac{(c_1+c_1+c_2(n-2))(n-1)}{2} = (c_1 - c_2 + \frac{c_2 n}{2})(n-1)$, which is $\Theta(n^2)$. Here, c_1 and c_2 are some constants.

- **Selection Sort:** if the first i numbers are sorted and they are the i smallest numbers in the whole array, then we can select the smallest number among the remaining $n-i$ numbers and swap it to the $i+1$'s spot.



SELECTION-SORT (A)

```
1   for i in range (0, A.length)
2       key = A[i]
3       spot = i
4       for j in range (i + 1, A.length)
5           if A[j] > key then key = A[j], spot = j
```

6 $A[spot] = A[i]$
7 $A[i] = key$

2. What is the worst-case time complexity of selection sort on a list/array of size n?

- There are two layers of loops, the outer layer runs n iterations and the inner layer runs $n - i, i = 1 \ldots n$ iterations (when does this happen?), thus it is easy to see that the worst-case time complexity is $\Theta(n^2)$.

2 3 4 5 1

- **Bubble Sort:** keep "bubbling up" the smallest number to the front of the array by swapping adjacent elements.

1

5 2 1 4 6 3 → 5 2 1 4 3 6 → 5 2 1 3 4 6 → 5 2 1 3 4 6 → 5 1 2 3 4 6 →
1 5 2 3 4 6

2

1 5 2 3 4 6

BUBBLE-SORT (A)
1 **for** i **in range** $(0, A.length - 1)$
2 **for** $j = A.length - 1$ **downto** $i + 1$
3 **if** $A[j] < A[j - 1]$
4 **then** swap $A[j]$ with $A[j - 1]$

List, Array, and Arraylist

- The "list" we've been using so far is a built-in data structure in Python, it is a fully implemented class of the *abstract data type* "list".
- A **list**, as an abstract data type, is a collection of items where each item holds a relative position with respect to the others. It should support the following methods:
  - `List()` creates a new list that is empty. It needs no parameters and returns an empty list.
  - `add(item)` adds a new item to the list. It needs the item and returns nothing. Assume the item is not already in the list.
  - `remove(item)` removes the item from the list. It needs the item and modifies the list. Assume the item is present in the list.
  - `search(item)` searches for the item in the list. It needs the item and returns a boolean value.
  - `isEmpty()` tests to see whether the list is empty. It needs no parameters and returns a boolean value.
  - `size()` returns the number of items in the list. It needs no parameters and returns an integer.
  - `append(item)` adds a new item to the end of the list making it the last item in the collection. It needs the item and returns nothing. Assume the item is not already in the list.
  - `index(item)` returns the position of item in the list. It needs the item and returns the index. Assume the item is in the list.
  - `insert(pos,item)` adds a new item to the list at position pos. It needs the item and returns nothing. Assume the item is not already in the list and there are enough existing items to have position pos.
  - `pop()` removes and returns the last item in the list. It needs nothing and returns an item. Assume the list has at least one item.
  - `pop(pos)` removes and returns the item at position pos. It needs the position and returns the item. Assume the item is in the list.

- An **array** is a data structure that stores items of similar types at contiguous memory locations.
  - Array usually supports the following methods: `Array(n)`, `get(i)`, `set(i,item)`, `search(item)`, `sort()`.
  - Comparing to a list, we can access to an item at a certain index very quickly.
  - In python, these methods of array are realized in the list class.

- Many methods in list are not supported in an array. In other languages, one can implement the list abstract data type into a class with the use of arrays, such class is called an **ArrayList.** We do not need an ArrayList in Python anymore.