**Bubble Sort – Time Complexity:**

* Bubble sort has a time complexity score almost always (**worst or average case**) of **O(n\*\*2)**.
* The **best case** score is **O(n)**, but this is misleading because the best case scenario is an already sorted list
* It’s a very inefficient algorithm due to how long it takes for a small element at the front of a list to “**bubble**” up toward the back (or vice versa)

**Bubble Sort – Space Complexity:**

* Bubble sort has a space complexity score of **O(1)** in the **worst case scenario**. This is because it is an **in-place** algorithm, meaning it uses no auxiliary memory.

**Bubble sort is a little more efficient than selection sort, but still very inefficient overall.**

* It is sometimes called the “**generic bad sorting algorithm**”.

**Odd-Even Sort (or Brick Sort)**:

* It is a simple sort, and a variation of **Bubble Sort**
* Its **time-complexity** is **O(n\*\*2)**
* Compares all odd/even indexed adjacent elements in the list
* Swaps if they are in the wrong order
* Next, it compares all **even/odd** adjacent elements in the list and swaps if they are mixed up
* It keeps doing this until there is a sorted list (odd/even, even/odd, odd/even, etc).
* Its **space-complexity** is **O(1)**….again, it’s an **in-place** algorithm

**Itertools.combinations():**

* Can be used to quickly find combinations of objects
* **Example:**
  + **return [pair for pair in itertools.combinations(sequence,2) if sum(pair) == 10]**
* combinations takes two arguments:
  + 1). An iterable
  + 2). r, a length variable (i.e. passing in 2 will return combinations in groups of two, etc.)
* **More Examples:**
* ***combinations('ABCD', 2) --> AB AC AD BC BD CD***
* ***# combinations(range(4), 3) --> 012 013 023 123***