In-Place Algorithm

An **in-place** function modifies data structures or objects outside of its own stack frame (i.e.: stored on the process heap or in the stack frame of a calling function). Because of this, the changes made by the function remain after the call completes.

> In-place algorithms are sometimes called **destructive**, since the original input is "destroyed" (or modified) during the function call.

> **Careful: "In-place" does *not* mean "without creating any additional variables!"** Rather, it means "without creating a new copy of the input." In *general*, an in-place function will only create additional variables that are $O(1)$ space.

An **out-of-place** function doesn't make any changes that are visible to other functions. Usually, those functions copy any data structures or objects before manipulating and changing them.

In many languages, **primitive** values (integers, floating point numbers, or characters) are copied when passed as arguments, and more complex **data structures** (lists, heaps, or hash tables) are passed by reference. This is what Python does.

Here are two functions that do the same operation on a list, except one is in-place and the other is out-of-place:

Python 3.6

```python
def square_list_in_place(int_list):
    for index, element in enumerate(int_list):
        int_list[index] *= element

    # NOTE: no need to return anything - we modified
    # int_list in place


def square_list_out_of_place(int_list):
    # We allocate a new list with the length of the input list
    squared_list = [None] * len(int_list)

    for index, element in enumerate(int_list):
        squared_list[index] = element ** 2

    return squared_list
```

Working in-place is a good way to save time and space. An in-place algorithm avoids the cost of initializing or copying data structures, and it usually has an $O(1)$ space cost.

But be careful: an in-place algorithm can cause side effects. Your input is "destroyed" or "altered," which can affect code *outside* of your function. For example:

Python 3.6

```python
original_list = [2, 3, 4, 5]
square_list_in_place(original_list)

print("original list: %s" % original_list)
# Prints: original list: [4, 9, 16, 25], confusingly!
```

Generally, out-of-place algorithms are considered safer because they avoid side effects. You should only use an in-place algorithm if you're space constrained or you're *positive* you don't need the original input anymore, even for debugging.

Programming interview questions by company:

- Google interview questions
- Facebook interview questions
- Amazon interview questions
- Uber interview questions
- Microsoft interview questions
- Apple interview questions
- Netflix interview questions
- Dropbox interview questions
- eBay interview questions
- LinkedIn interview questions
- Oracle interview questions
- PayPal interview questions
- Yahoo interview questions

Programming interview questions by topic:

- SQL interview questions
- Testing and QA interview questions
- Bit manipulation interview questions
- Java interview questions
- Python interview questions
- Ruby interview questions
- JavaScript interview questions
- C++ interview questions
- C interview questions
- Swift interview questions
- Objective-C interview questions
- PHP interview questions
- C# interview questions

228 Park Ave S #82632, New York, NY US 10003 (862) 294-2956