Given Algorithms

Algorithm A2.1: ins(data, pos)
This algorithm inserts the data at the position specified in the argument.

1. Slice a list from index 0 till pos. Call it list1.
2. The list from pos till length of the initial list will be list 2.
3. Concatenate the element or sub-list with list 1.
4. Concatenate the list obtained from step 3 with list 2.
5. Print the final list

Algorithm A2.2: dele(data)
This algorithm removes the data if available in the list.

1. For `i` in the range of length of the list, check whether `data` is there at any index position.
2. If there is a match, slice the list from 0 till index value `i`, call it list 1.
3. The list from `i+1` till length of the initial list will be list 2.
4. Concatenate the list obtained from step 2 with list 2.
5. Print the final list.

Algorithm A2.3: search(data)
This algorithm finds the data in the list and returns the position of the data value.

1. For `i` in the range of length of the list, check whether `data` is there at any index position.
2. If there is a match, return the index value `i`.
3. Print the value of the index.

Exercise

1. Implement algorithms A2.1 to A2.3 in the form of functions. Compare the implemented functions with built-in functions available for list data structure in terms of execution time. Note down your observations.

```python
import timeit

# USER DEFINED FUNCTIONS

def insertion(original_list,data,position):
    l1 = original_list[:position]
    l2 = original_list[position:]
    return l1 + [data] + l2


def delete(original_list,data):
    for i in range(len(original_list)):
        if original_list[i] == data:
            list1 = original_list[:i]
            list2 = original_list[i+1:]
            return list1 + list2


def search(original_list,data):
    for i in range(len(original_list)):
        if original_list[i] == data:
            return "data search at Index value of list:",i



my_list = [10, 20, 30, 40, 50]
# Execution time
print("\nExecution Time (Custom vs Built In):")

print("Insert Custom:", timeit.timeit(lambda: insertion(my_list, 25, 2), number=10000),"seconds") # lambda to avoid immediate execution
print("Insert Built-in:", timeit.timeit(lambda: my_list.insert(25,2,),number=10000),"seconds") # use insert built-in function

print("Delete Custom:", timeit.timeit(lambda: delete(my_list, 30), number=10000),"seconds")
print("Delete Built-in:", timeit.timeit(lambda: my_list.copy().remove(30), number=10000),"seconds") # use copy to avoid modifying original list

print("Search Custom:", timeit.timeit(lambda: search(my_list, 40), number=10000),"seconds")
print("Search Built-in:", timeit.timeit(lambda: my_list.index(40), number=10000),"seconds")# use index built-in function

```

```
OUTPUT:

Execution Time (Custom vs Built In):
Insert Custom: 0.0038931999999931577 seconds
Insert Built-in: 0.018789100000049075 seconds
Delete Custom: 0.5297610999998597 seconds
Delete Built-in: 0.2695017999999436 seconds
Search Custom: 0.004037299999936295 seconds
Search Built-in: 0.0009231000001364009 seconds
```

2. Develop algorithms for insertion, deletion and searching for dictionary and tuple data structure. Implement all the algorithms and compare them with built-in functions available for respective operations in terms of execution time. Note down your observations.

Dictionary Part

```python
import timeit

# USER DEFINED FUNCTIONS

def dic_insertion(dic, key, value):
    dic[key] = value
    return dic

def dic_dele(dic,key):
    if key in dic:
        del dic[key]
    return dic

def dic_search(dic,key):
    for k,v in dic.items():
        if k == key:
            return v
    return None


my_dic = {1:"one", 2:"two", 3:"three", 4:"four", 5:"five"}
# Execution time
print("\nExecution Time (Custom vs Built In):")

print("Insert Custom:", timeit.timeit(lambda: dic_insertion(my_dic, 6, "six"), number=10000),"seconds") # lambda to avoid immediate execution
print("Insert Built-in:", timeit.timeit(lambda: my_dic.update({6:"six"}),number=10000),"seconds") # use update built-in function

print("Delete Custom:", timeit.timeit(lambda: dic_dele(my_dic, 6), number=10000),"seconds")
print("Delete Built-in:", timeit.timeit(lambda: my_dic.copy().pop(6, None), number=10000),"seconds") # use pop built-in function to remove key safely

print("Search Custom:", timeit.timeit(lambda: dic_search(my_dic, 4), number=10000),"seconds")
print("Search Built-in:", timeit.timeit(lambda: my_dic.get(4), number=10000),"seconds")# use get built-in function
```

```
Output:

Execution Time (Custom vs Built In):
Insert Custom: 0.0013057999999546155 seconds
Insert Built-in: 0.0020296999996389786 seconds
Delete Custom: 0.0020502000002124987 seconds
Delete Built-in: 0.0016466000001855718 seconds
Search Custom: 0.0036857999998574087 seconds
Search Built-in: 0.0007633999998688523 seconds
```

Tuple Part

```
Output:

Execution Time (Custom vs Built In):
Insert Custom: 0.012448700000277313 seconds
Insert Built-in: 0.009713800000099582 seconds
Delete Custom: 0.020112199999857694 seconds
Delete Built-in: 0.021778399999675457 seconds
Search Custom: 0.02414480000061303 seconds
Search Built-in: 0.005986399999528658 seconds
```

```python
import timeit

# USER DEFINED FUNCTIONS

def tup_insertion(tup,value,pos):
    tup1 = tup[:pos]
    tup2 = tup[pos+1:]
    return tup1 + (value,) + tup2

def tup_delete(tup,value):
    j = None
    for i in range(len(tup)):
        if tup[i] == value:
            j = i # tuple are immutable, so we need to create a new one
            break
    if j is None:
        # Value not found, return original tuple or raise an error
        return tup
    tup1 = tup[:j]
    tup2 = tup[j+1:]
    return tup1 + tup2

def tup_search(tup,value):
    for i in range(len(tup)):
        if tup[i] == value:
            return i



my_tuple = (1, 2, 3, 4, 5)
# Execution time
print("\nExecution Time (Custom vs Built In):")

print("Insert Custom:", timeit.timeit(lambda: tup_insertion(my_tuple, 6,5 ), number=10000),"seconds") # lambda to avoid immediate execution
print("Insert Built-in:", timeit.timeit(lambda: list(my_tuple).copy().insert(5, 6), number=10000),"seconds") # use insert built-in function

print("Delete Custom:", timeit.timeit(lambda: tup_delete(my_tuple, 6), number=10000),"seconds")
print("Delete Built-in:", timeit.timeit(lambda: [x for x in my_tuple if x != 6], number=10000), "seconds") # create a new list without the value 6 done  by using list comprehensio

print("Search Custom:", timeit.timeit(lambda: tup_search(my_tuple, 4), number=10000),"seconds")
print("Search Built-in:", timeit.timeit(lambda: my_tuple.index(4), number=10000),"seconds")# use index built-in function

```