# Homework 2 solutions

**Problem 1** Give the Big-O performanc of the following code fragment:

```
for i in range(n):      # (1)  n * (steps (2) and (3))
    for j in range(n):  # (2)  n * (step (3))
        k = 2 + 2       # (3)  O(1)
```

$$n \cdot n \cdot O(1) = O(n^2) \tag{1}$$

**Problem 2** Give the Big-O performance of the following code fragment:

```
for i in range(n):    # (1)  n * (step 1(2))
    k = 2 + 2         # (2)  O(1)
```

The time complexity of the code above becomes

$$n \cdot O(n) = O(n) \tag{2}$$

**Problem 3** Give the Big-O performance of the following code fragment:

```
i = n             # (1)  O(1)
while i > 0:      # (2)  log_2(n) * (steps (3) and (4))
    k = 2 + 2     # (3)  O(1)
    i = i // 2    # (4)  O(1)
```

Because step (4) divides i by 2 on each iteration, this causes the i reach 0 in $log_2(n)$ divisions. Thus the time complexity of the code above in big-O notation becomes

$$O(1) + log_2(n) \cdot (O(1) + O(1)) \tag{3}$$

Because $log_2(n)$ grows with $n$ while the $O(1)$ terms do not, the $log_2(n)$ becomes the dominant term as $n$ grows. Thus Forumula~3 becomes

$$O(log_2(n)) \tag{4}$$

**Problem 4** Give the Big-O performance of the follwoing code fragment:

```
for i in range(n):          # (1)  n * (steps (2) through (4))
   for j in range(n):       # (2)  n * (steps (3) through (4))
       for k in range(n):   # (3)  n * (step (4))
           k = 2 + 2        # (4)  O(1)
```

Due to the triple nesting of for loops the code block above has time complexity give by

$$n \cdot n \cdot n \cdot O(1) = O(n^3) \tag{5}$$

**Problem 5**: (2 points) The Bag class from lectures 6 and 7 can be found in the class repository at

`https://git.cs.olemiss.edu/harrison/csci-356`

in

`lectures6and7/bag/bag.py`

Add an iterator class to Bag. The iterator class must pass the unit tests committed in the repository in the directory `hw2/bag/test_bag.py`. You will receive *partial credit if you do not write unit tests for your iterator class, or if the code lacks comments or type hints.* Write to test more conditions than the tests given in `test_bag.py`. The tests should cover edge conditions like the iterator should work as expected on an empty list.

**Answer 5**

See hw2/bag/bag.py

These should pass the tests in hw2/test_bag.py as well as the additional tests in hw2/test_bag_extended.py.

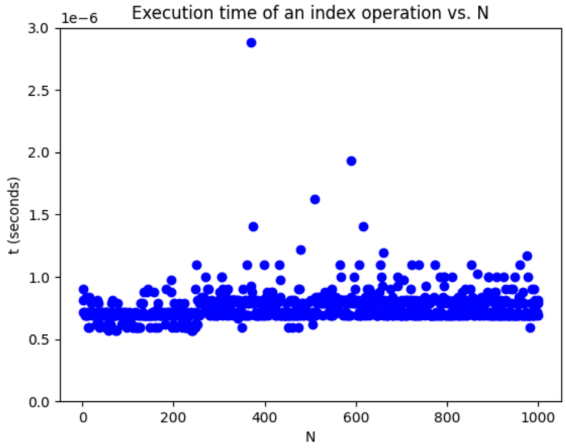
The additional tests just test a couple edge cases.

**Problem 6**: (2 points) Write a program that verifies that the list index operator is O(1). The program must plot the run time of the list index operator as a function of n using matplotlib.

**Answer 6**

See p6.py

Possible plot:

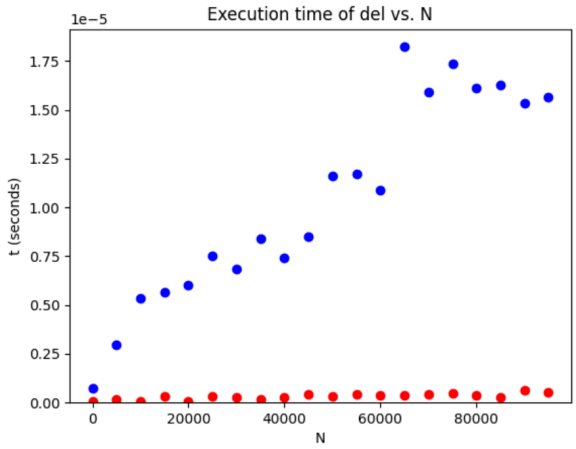



In this plot, there doesn't appear to be any correlation between $n$ and the time to execute an index operation. This would be the case if the operation takes $O(1)$ time.

**Problem 7**: (2 points) Write a program that compares the performance of the `del` operator on lists and dictionaries. The main program should plot the run time of each on the same plot as a function of n. Also plot functions that bound the time complexity and print out what you think is the time complexity of `del` operators for lists and dictionaries using big-O notation. When measuring performance on the list `del` operator, be sure to delete items at random locations from the list.

**Answer 7**

See p7.py



Red denotes average execution time per delete from a dict containing $n$ items. Blue denotes the same thing but for lists.

From this it looks like the time complexity of deletion from a dict is O(1), but deletion from a list is O(n).