Submission by: Abhijit Kumar Baruah

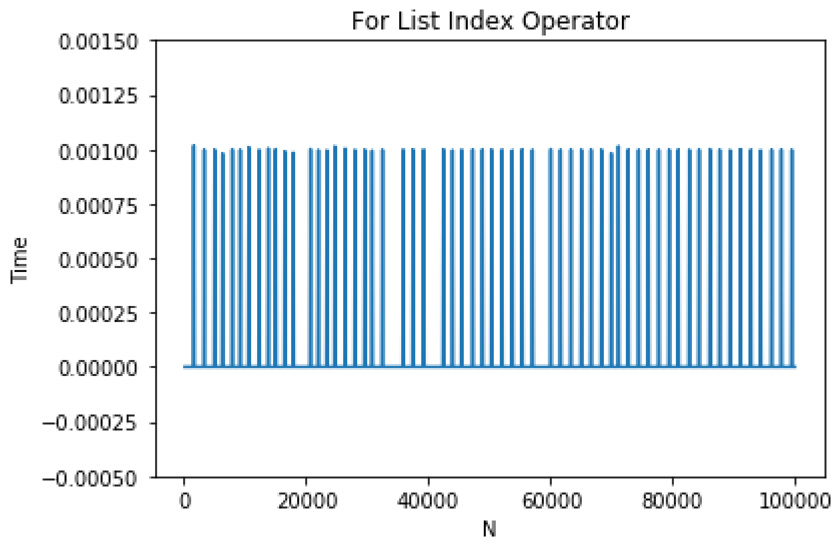
GitHub username: abhijit-baruah

Group Members: Abhijit Baruah, Jen Rowe, Sunny Liu

WS Problem #2: Devise an experiment to verify that the list index operator is $O(1)$.

In [11]:

```
import time # we use time instead of timeit for the simplicity provided by the time package
from matplotlib import pyplot as plt
import random

# first we create a function that will return the value associated with any index from a li
# our inputs here are the list and index, while output is the single associated value
def list_index_operation(alist,index):
    return alist[index]

# creating empty lists to store max value of N and time-values for y-axis
N1 = []
y1 = []

# create a list that mathematically signifies N approaching a large value from an initial v
for i in range(100, 100000, 100):
    N1.append(i)

# 2. Devise an experiment to verify that the list index operator is O(1)
# In the experiment we ask the list index operator to return us the value at a random index
# The list keeps growing in size to signify an increase in N, ofcourse upto a certain large
for n in N1:
    alist = range(n) # this is the list that we populated to select index from
    random_index = random.randint(0, n-1) # uniform random selection of one index from the

    start1 = time.time() # this is our starting time for list index operation

    # we can choose to not repeat the following list index operation in a loop
    # but the time difference would be too small to consider for a graph later on
    # Since added time due to repetition would only add a fixed constant towards the order
    # this would essentially still be bounded by O(1) going with the mathematics behind it.
    for repeat in range(200):
        list_index_operation(alist, random_index)

    end1 = time.time() # end time after 100 iterations of retreiving value from a list inde

    y1.append(end1-start1) # calculating the difference in time and storing as a list of ti

# PLotting time vs N
plt.plot(N1, y1)
plt.ylim(-0.0005,0.0015)
plt.xlabel('N')
plt.ylabel('Time')
plt.title('For List Index Operator')
plt.show()

import numpy as np
print(np.shape(y1)) # check the shape of y1
print(np.max(y1)) # check maximum time needed to implement index operation 100 times
```



```
(999,)
0.0010190010070800781
```

Explanation of graph

As seen from the graph, the time taken by the list index operation is stritly bounded in $(0, 0.0012)$ for large N. There is no linear growth in time consumed for a list index operation which indicates that the list index operator is strictly bounded above by $O(n)$. At the same time, since we have a strict finite upper bound for the time taken by the list index operator, the list index operator is time bounded and is infact $O(1)$.

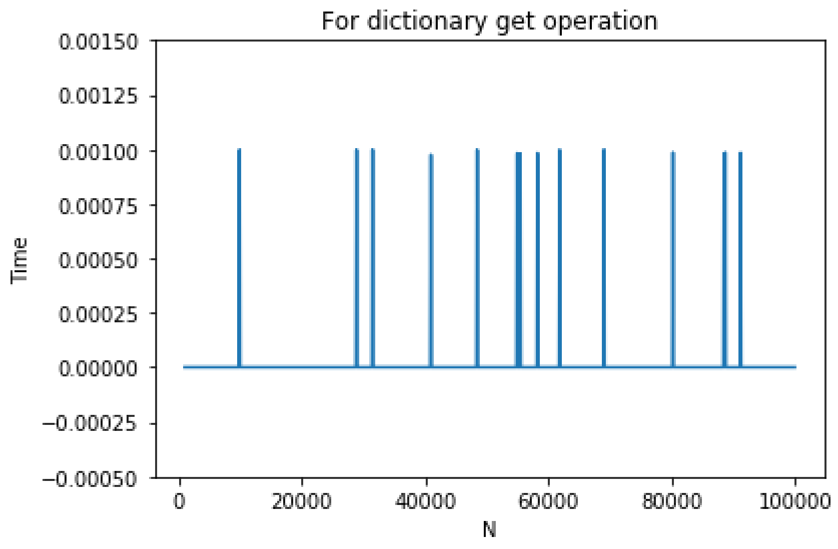
WS Problem #3: Devise an experiment to verify that the dictionary operations get, and set item are $O(1)$

In [2]:

```
# This experiment is similar to the one in Problem 2 with minor modifications to incorporat

import time # we use time instead of timeit for the simplicity provided by the time package
from matplotlib import pyplot as plt
import random

# first we create a function that will return the value associated with any index/key from
# our inputs here are the dictionary and index/key, while output is the single associated v
def dict_get_operation(adict, index):
    adict.get(index)

#defining a dictionary set operation
def dict_set_operation(adict, index):
    adict[index]=10 # we are assigning a fixed value 10 for a randomly selected index/key

# creating empty lists to store max value of N and time-values for y-axes
N2 = []
y2 = []
y3 = []

# create a list that mathematically signifies N approaching a large value from an initial v
for i in range(1000, 100000, 100):
    N2.append(i)

# 3. Devise an experiment to verify that the dictionary operations get, and set item are O(
for n in N2:
    adict = {j:None for j in range(n)} # this is the dictionary that we populated to select
    # we set the keys to integers and values to None, since, we are not concened with what
    # Rather we are interested in the time needed to get or set values relative to the keys

    random_index = random.randint(0, n-1) # uniform random selection of one index from the

    start2 = time.time() # this is our starting time for dictionary get operation

    # Again we can choose to not run this in a loop
    # but the time difference would be too small to consider while making graphs
    for repeat in range(50):
        dict_get_operation(adict, random_index)

    end2 = time.time() # end time after 50 iterations of retreiving value from a dictionary
    y2.append(end2-start2)

    start3 = time.time() # this is our starting time for dictionary set operation
    for repeat in range(50):
        dict_set_operation(adict, random_index)
    end3 = time.time() # end time after 50 iterations of setting a certain value to the ran

    y3.append(end3-start3) # calculating the difference in time and storing as a list of ti
```

In [9]:

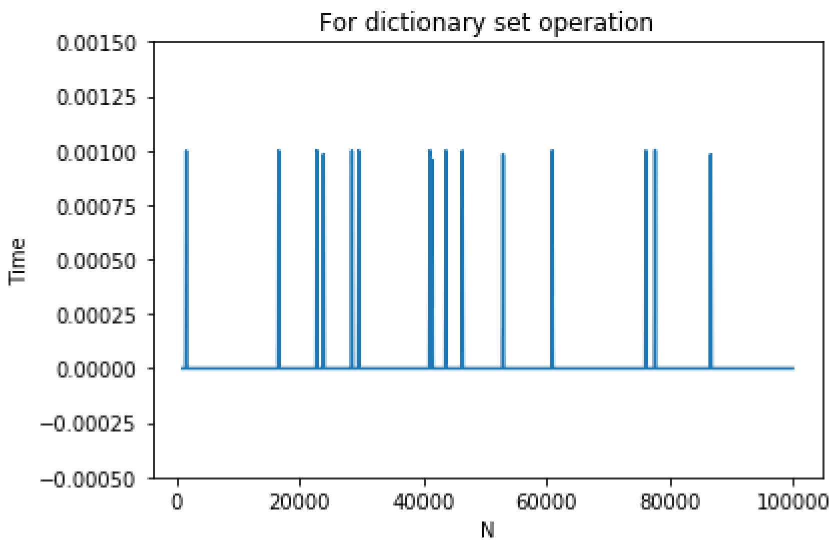
```
plt.plot(N2,y2)
plt.ylim(-0.0005,0.0015)
plt.xlabel('N')
plt.ylabel('Time')
plt.title('For dictionary get operation')
plt.show()

import numpy as np
print(np.shape(y2)) # check the shape of y2
print(np.max(y2)) # check maximum time needed to implement dictionary get operation
```



```
(990,)
0.0010004043579101562
```

In [10]:

```
plt.plot(N2, y3)
plt.ylim(-0.0005,0.0015)
plt.xlabel('N')
plt.ylabel('Time')
plt.title('For dictionary set operation')
plt.show()
import numpy as np
print(np.shape(y3)) # check the shape of y3
print(np.max(y3)) # check maximum time needed to implement dictionary set operation.
```



```
(990,)
0.001001596450805664
```

Explanation of the graphs

As seen previously in the case of List Index Operator in problem #1, the graphs for dictionary operations get and set items behave similar to that of the list index operator. Here both the timed graphs for dictionary get and set operations are strictly bounded in $(0, 0.0012)$ and there is no linear growth in time over large N. This also implies that both of these dictionary key operations are strictly finitely bounded. Hence, dictionary operations get and set items are $O(1)$.

In [ ]: