Work with your neighbor. (This will be graded for participation only.)

1. Download https://www.obagy.com/cs120/LECTURES/sumv1.py and https://www.obagy.com/cs120/LECTURES/sumv2.py.

   Add calls to run each version for the values 10,000, 100,000, and 1,000,000. Answer the questions below.

   ANS:

   What observations do you have about the efficiency of `sumv1.py` vs. `sumv2.py`?

   `sumv1.py` is less efficient than `sumv2.py` as the input gets larger

   Do you notice a pattern when increasing the input size for `sumv1.py`?

   - For `sumv1.py`, as we increase n, the running time increases. The running time increases in proportion to n.
   - For `sumv2.py`, as we increase n, the running time stays the same.

2. Consider these two algorithms for searching for a word, `my_word`, in a dictionary (a physical dictionary, which is of course sorted):

   - Algo 1: search from the beginning

     start at the first word in the dictionary

     if the word is not my_word, then go to the next word

     continue in sequence until my_word is found

   - Algo 2: start at the middle of the dictionary

     if my_word is greater than the word in the middle,

     start with the middle word and continue from there to the end of the dictionary

     if my_word is less than the word in the middle,

     start with the middle word and search from there to the beginning of the dictionary

ANS:

a) Which is better, Algo 1 (search from the beginning) or Algo 2 (search from the middle)? Explain your answer.

Most of the time, Algo 2 will be more efficient because for all search words that lie in the second half of the (physical) dictionary, you will eliminate having to check all the words in the first half of the dictionary.

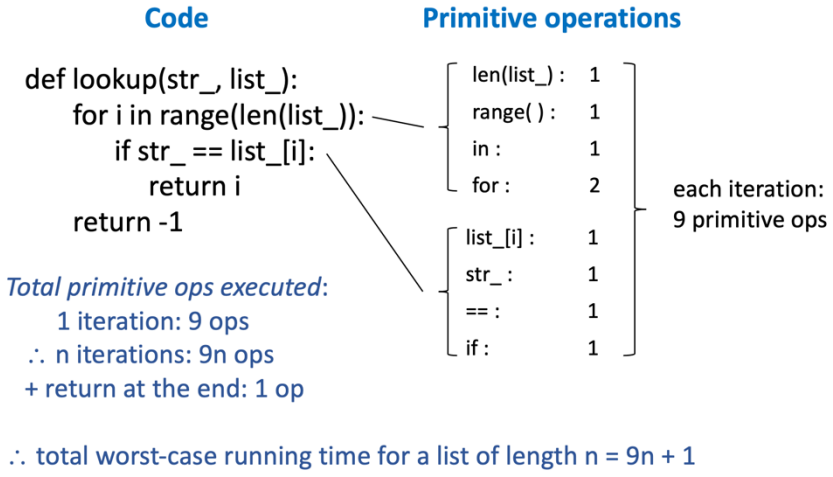
b) Regardless of which algorithm that you chose, is there ever a scenario where the other one is better? If so, explain the scenario.

For words that lie in the first quarter of the dictionary, Algo 2 would be less efficient because you will first check all the words from the midpoint to the beginning of the second quarter of words before getting to the first quarter of the dictionary.

c) When considering which is better, what measure are we using?

We are measuring—implicitly counting—the number of comparisons (does `my_word` equal the current word in the dictionary?) that are made before finding the search word.

3. Use the analysis of the lookup() example as reference to answer the questions below:



∴ total worst-case running time for a list of length n = 9n + 1

a) What is the worst-case running time of the following code fragment expressed in terms of n?

```
for i in range(n):
    k = 2 + 2
```

ANS:

for i in range(n):	n:	1
	range():	1
	in:	1
	for:	2
k = 2 + 2	+:	1
	=:	1

1 iteration: 7 ops
n iterations: 7n ops

Therefore, the worst-case running time for n numbers is 7n

b) What is the worst-case running time of the following code fragment expressed in terms of n?

```
a = 5
b = 10
for i in range(n):
    x = i * b
for j in range(n):
    z += b
```

ANS:

a = 5	=:	1	
b = 10	=:	1	total of 2 ops
for i in range(n):	n:	1	
	range():	1	
	in:	1	
	for:	2	
x = i * b	i:	1	
	b:	1	
	*:	1	
	=:	1	
for j in range(n):	n:	1	
	range():	1	
	in:	1	
	for:	2	
z += b	z:	1	

	b:	1
	+:	1
	=:	1
at the top	2 ops	
1st loop:	1 iteration: 9 ops	
	n iterations: 9n ops	
2nd loop	1 iteration: 9 ops	
	n iterations: 9n ops	

Therefore, the worst-case running time for n numbers is 18n + 2