

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 of `sum` for the values 10,000, 100,000, and 1,000,000.

Answer the questions below:

- a) What observations do you have about the efficiency of `sumv1.py` vs. `sumv2.py`?
 - b) Do you notice a pattern when increasing the input size for `sumv1.py`?
-
2. Consider these two algorithms from the lecture slides 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

- a) Which is better, Algo 1 (search from the beginning) or Algo 2 (search from the middle)? Explain your answer.
- b) Regardless of which algorithm that you chose, is there ever a scenario where the other one is better? If so, explain the scenario.
- c) When considering which is better, what measure are we using?

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

Code	Primitive operations																
<pre>def lookup(str_, list_): for i in range(len(list_)): if str_ == list_[i]: return i return -1</pre>	<table> <tr> <td>len(list_) :</td><td>1</td></tr> <tr> <td>range() :</td><td>1</td></tr> <tr> <td>in :</td><td>1</td></tr> <tr> <td>for :</td><td>2</td></tr> <tr> <td>list_[i] :</td><td>1</td></tr> <tr> <td>str_ :</td><td>1</td></tr> <tr> <td>== :</td><td>1</td></tr> <tr> <td>if :</td><td>1</td></tr> </table>	len(list_) :	1	range() :	1	in :	1	for :	2	list_[i] :	1	str_ :	1	== :	1	if :	1
len(list_) :	1																
range() :	1																
in :	1																
for :	2																
list_[i] :	1																
str_ :	1																
== :	1																
if :	1																
<i>Total primitive ops executed:</i> 1 iteration: 9 ops ∴ n iterations: 9n ops + return at the end: 1 op	each iteration: 9 primitive ops																

∴ 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
```

- 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
```