A: Searching for an index

In this part of the assignment, you should write a function `where_is` that finds an element in a list. More precisely:

Input: A list `l` and an element `x` to search for.

Output: An index `idx` such that `l[idx] == x`, or `None` if `x` is not present in `l`. If `x` occurs more than once, the first occurrence is reported.

Note: You should not use built-in Python functions or methods to solve this. Your implementation will have to loop over all the elements in the list `l`.

The following has to work:

```
> where_is([1, 3, 5], 3)
1
> where_is([1, 2, 4, 5, 4], 4)
2
> where_is([1, 2, 3], 0)
None
```

1. Break down the problem: What is the first thing you need?
   - This sub-problem should **not be more than 1-2 lines of code!**
2. Solve that first problem **until you have something that works**
3. Identify the next sub-problem and continue with 1.

- Hint: Start by writing a function that takes the requested input and simply prints back `None`.
  - This should only be 2 lines of code

B: Best- and worst-case runtimes (without running the code)

Open the file `search.py` and read the code. In it is a function called `find_element_in_list`.

1. What value of `element` is the best case for `find_element_in_list` when the `data_list` is `[0, 1, ..., 9999]`?
2. What value of `element` is the worst case for `find_element_in_list` when the `data_list` is `[0, 1, ..., 9999]`?
3. Let's say `find_element_in_list` is run with a list of size `n`. The worst-case runtime $O(n)$ $O(n)$ is one of the following choices a-d, write down which one:

```
a. constant
b. linear
c. quadratic
d. exponential
```

C: Runtime analysis: Looking for elements

Consider the following variant of `find_element_in_list`:

```
def find_element_in_list2(data_list, element):
    for idx, el in enumerate(data_list):
        if el == element:
            first_result_idx = idx
            break
    for idx, el in enumerate(data_list):
        if el == element and idx == first_result_idx:
            return idx, el
```

1. Let's say `find_element_in_list2` is run with a list of size `n`. The worst-case runtime $O(n)$ is one of the following choices a.-d. But which one?

    a. constant
    b. linear
    c. quadratic
    d. exponential

D: Runtime analysis: Turtles

- Take the following turtle program and measure how long it takes to complete:

```
import turtle



turtle = turtle.Turtle()
n = 8
moves = [100] * n
for move in moves:
    turtle.forward(move)
    turtle.left(45)
```

- What happens with the runtime if you double `n` (set it to 16)?
- What happens with the runtime if you double `n` again (set it to 32)?
- What is the limiting factor of the turtle program? And what is the big-O notation for that?

E: Sort performance

In the file `sort_algos.py` are three sorting algorithms: bubble, selection and merge sort.

1. Using the four lists below, your job is to measure the runtimes of the three algorithms. Taking one list at the time, go through the three sorting algorithms and record their runtimes in the table below. You can record the runtimes from python using the `timeit` module as seen in class.

```
import random
list1 = list(range(1000))
list2 = list(range(10000))
list3 = list(range(100000))
list4 = list3[:] # Copy the full list3 into list4 ([:] is a slicing opera
tion that copies ALL elements)
random.shuffle(list4) # Shuffles list4, so the elements are now in random
positions
```

Algorithm	Runtime for list1	Runtime for list2	Runtime for list3	Runtime for list4
Bubble sort				
Selection sort				
Merge sort				

2. (Optional) Draw and hand in a graph of the runtimes, where the x axis is time and the y axis is the size of the list.

3. What are the worst-case ($O(n)$ $O(n)$) runtime for the the three sorting algorithms? Can you tell us why?

    - Bubble sort
    - Selection sort
    - Merge sort

Note: All inputs are worst-case inputs. The algorithms does not take into account that they could terminate early.

A: Searching for an index

In this part of the assignment, you should write a function `where_is` that finds an element in a list. More precisely:

Input: A list `l` and an element `x` to search for.

Output: An index `idx` such that `l[idx] == x`, or `None` if `x` is not present in `l`. If `x` occurs more than once, the first occurrence is reported.

Note: You should not use built-in Python functions or methods to solve this. Your implementation will have to loop over all the elements in the list `l`.

The following has to work:

```
> where_is([1, 3, 5], 3)
1
> where_is([1, 2, 4, 5, 4], 4)
2
> where_is([1, 2, 3], 0)
None
```

1. Break down the problem: What is the first thing you need?
   - This sub-problem should **not be more than 1-2 lines of code!**
2. Solve that first problem **until you have something that works**
3. Identify the next sub-problem and continue with 1.

- Hint: Start by writing a function that takes the requested input and simply prints back `None`.
  - This should only be 2 lines of code

B: Best- and worst-case runtimes (without running the code)

Open the file `search.py` and read the code. In it is a function called `find_element_in_list`.

1. What value of `element` is the best case for `find_element_in_list` when the `data_list` is `[0, 1, ..., 9999]`?
2. What value of `element` is the worst case for `find_element_in_list` when the `data_list` is `[0, 1, ..., 9999]`?
3. Let's say `find_element_in_list` is run with a list of size `n`. The worst-case runtime $O(n)$ is one of the following choices a-d, write down which one:

```
a. constant
b. linear
c. quadratic
d. exponential
```

C: Runtime analysis: Looking for elements

Consider the following variant of `find_element_in_list`:

```python
def find_element_in_list2(data_list, element):
    for idx, el in enumerate(data_list):
        if el == element:
            first_result_idx = idx
            break
    for idx, el in enumerate(data_list):
        if el == element and idx == first_result_idx:
            return idx, el
```

1. Let's say `find_element_in_list2` is run with a list of size `n`. The worst-case runtime $O(n)$ is one of the following choices a.-d. But which one?

    a. constant
    b. linear
    c. quadratic
    d. exponential

D: Runtime analysis: Turtles

- Take the following turtle program and measure how long it takes to complete:

```python
import turtle

turtle = turtle.Turtle()
n = 8
moves = [100] * n
for move in moves:
    turtle.forward(move)
    turtle.left(45)
```

- What happens with the runtime if you double `n` (set it to 16)?
- What happens with the runtime if you double `n` again (set it to 32)?
- What is the limiting factor of the turtle program? And what is the big-O notation for that?

E: Sort performance

In the file `sort_algos.py` are three sorting algorithms: bubble, selection and merge sort.

1. Using the four lists below, your job is to measure the runtimes of the three algorithms. Taking one list at the time, go through the three sorting algorithms and record their runtimes in the table below. You can record the runtimes from python using the `timeit` module as seen in class.

```
import random
list1 = list(range(1000))
list2 = list(range(10000))
list3 = list(range(100000))
list4 = list3[:] # Copy the full list3 into list4 ([:] is a slicing opera
tion that copies ALL elements)
random.shuffle(list4) # Shuffles list4, so the elements are now in random
positions
```

Algorithm	Runtime for list1	Runtime for list2	Runtime for list3	Runtime for list4
Bubble sort				
Selection sort				
Merge sort				

2. (Optional) Draw and hand in a graph of the runtimes, where the x axis is time and the y axis is the size of the list.

3. What are the worst-case ($O(n)$) runtime for the the three sorting algorithms? Can you tell us why?

    - Bubble sort
    - Selection sort
    - Merge sort

Note: All inputs are worst-case inputs. The algorithms does not take into account that they could terminate early.