Problem 1.

Write a function that takes in a function and an input to the function, as well as the number of repetitions to perform, and returns the average time the function took on the given input across the given number of repetitions. You can use the `time` module to time your function.

```
def time_func(f, x, n):
  # return the average time it takes to run f(x),
  # across n repetitions
```

You can use the following to graph runtime vs. input size (you will need to modify the code to make it work – it is not complete)

```
import matplotlib.pyplot as plt
# ns = [1, 100, 10000, ... 100000]
# inputs = []
# for n in ns:
#  inputs.append(generate_worst_case(n))
plt.plot(ns, runtimes)
plt.xlabel('input size')
plt.ylabel('runtime')
plt.show()
```

For counting sort and bubble sort, generate a plot with two graphs that shows how the worst-case runtime increases with the size of the input.

Problem 2.

Without using loops, the `*` operator, or any function in the `math` module, write a function with the signature

```
times(a, b)
```

which returns $a \times b$.

Use recursion.

Use the fact that $a \times b = (a \times (b - 1)) + a$.

Problem 3.

Using recursion, implement "linear search": return the index of the element e in the list L. Do not use built-in functions or loops.

Hint: if L[0] is already e, you know the answer. If you know the index in L[1:], you can transform it to be the answer for L.

Problem 4.

Without using loops, write the function with the signature

```
interleave(L1, L2)
```

which takes `L1` and `L2`, two lists of the same length, and returns a list which consists of `L1` and `L2` interleaved, i.e., `[L1[0], L2[0], L1[1], L2[1], ..., L1[n-1], L2[n-1]]` (here, `n == len(L1) == len(L2)`).

Problem 5.

Without using loops or slicing, write a function that reverses a list in place. That is, the effect of calling

```
reverse_rec(L)
```

should be that `L` is reversed.

Here is how you might do this *with* loops:

```
def reverse_loop(L):
  for i in range(len(L)//2):
    L[i], L[-1-i] = L[-1-i], L[i]
```

You need to write a helper function that calls itself.

Problem 6.

Fun fact: this question is taken from the final exam that I wrote in my first year.

Without using loops and without ever using `print` with a list (as opposed to individual elements of a list), write a function that, given a list `L` of size n (assume n is odd), prints the elements of `L` in the following order:

`L[n//2] L[n//2-1] L[n//2+1] L[n//2-2] L[n//2+2] L[n//2-3] L[n//2+3] ... L[n-1]`

Hint: here is a recursive function that prints the following sequence for a list `L` of size n:

`L[0] L[n-1] L[1] L[n-2] L[2] L[n-3] ... L[n//2]`

```
def zigzag1(L):
  if len(L) == 0:
    print('')
  elif len(L) == 1:
    print(L[0], end = "")
  else:
    print(L[0], L[-1], end = "")
    zigzag1(L[1:-1])
```

Problem 7.

This is the last ESC180 lab this semester – say goodbye and thank you to your lab TAs!