# DSC 40B - Discussion 03

**Problem 1.**

**a)** State (but do not solve) the recurrence relation describing this function's run time.

```
import random
def foo(n):
    if n <= 2:
    return
for i in range(n):
    for j in range(i,n):
        print(i)
return foo(n//2) + foo(n//2)
```

**b)** Suppose a binary search is performed on the following array using the implementation of *binary_search* from lecture. What is the worst case number of equality comparisons that would be made to search for an element in the array? That is, what is the greatest number of times that `arr[middle] == target` can be run?

```
[1, 4, 7, 8, 8, 10, 15, 51, 60, 65, 71, 72, 101]
```

**Problem 2.**

**a)** Suppose you are sorting an array using selection sort. At some point during sorting, a snapshot of the array is recorded:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 2]
```

In this snapshot of the array, what is the maximum number outer loop iterations ran?

```
def selection_sort(arr):
    """In-place selection sort."""
    n = len(arr)
    if n <= 1:
        return
    for barrier_ix in range(n-1):
            # find index of min in arr[start:]
        min_ix = find_minimum(arr, start=barrier_ix) #swap
        arr[barrier_ix], arr[min_ix] = (
            arr[min_ix], arr[barrier_ix]
        )
```

**b)** State the recurrence relation of the following code.

```
import math
def foo(arr):
    if len(arr) == 0:
        return 0

    if len(arr) == 1:
        return arr[0]

    one_third = math.floor(len(arr) / 3)
    left = arr[:one_third]
    middle = arr[one_third: one_third * 2]
    right = arr[one_third * 2:]

    for i in range(len(arr)):
        if i < one_third:
            arr[i] = 2 * arr[i] - sum(left)/len(left)
        if i >= one_third and i < one_third * 2:
            arr[i] = 2 * arr[i] - sum(middle)/len(middle)
        if i >= one_third * 2:
            arr[i] = 2 * arr[i] - sum(right)/len(right)

    total = sum(left) + sum(middle) + sum(right)

    return foo(left) + foo(middle) + foo(right) + total
```

**c)** Solve the recurrence relation describing the above function's run time

**Problem 3.**

We're given two sorted lists in ascending order, `A` and `B` and a target `t`, and our goal is to find an element $a$ of `A` and an element $b$ of `B` such that $a + b = t$.