Problem 3-1. (Deterministic-Quicksort)

(a) Recall from lecture the Randomized-Select algorithm to the select a rank(i) element from an array.

```
function rand_select(A, p, q, i):
    r = rand_partition(A, p, q)
    k = r - p + 1 // rank of A[r]
    if i == k:
        return A[r]
    else if i < k:
        return rand_select(A, p, r - 1, i)
    else:
        return rand_select(A, r + 1, q, i - k)
```

What is the best, worst, and expected asymptotic runtime of this algorithm?

(b) Deterministic-Select, also from lecture, has a worst case runtime of $\theta(n)$. Now, consider Quicksort below.

```
function rand_quicksort(A, p, q):
    if p < q:
        i = rand_int(p, q) // choose a pivot
        r = partition(A, p, q, i)
        rand_quicksort(A, p, r - 1)
        rand_quicksort(A, r + 1, q)
```

Given Deterministic-Select, how would you modify Quicksort to bound the worst-case runtime on any input to $\theta(n \lg n)$?

(c) Why is the above algorithm typically not used in practice?

Problem 3-2. (Deterministic-Select)

In lecture, we covered Deterministic-Select for groups of size 5. In this problem we generalize the algorithm to groups of size k. Consider the pseudocode below:

```
pseudocode deterministic-select(A, k, i):
    1. Divide A into groups of size k, and find group medians.
    2. Recursively call deterministic-select to find the
       median, x, of the n/k group medians
    3. Partition around x. Let r = rank(x).
       if r == i:
           return x
       else if i < r:
```

```
        Recurse on left. A[:r-1].
    else:
        Recurse on right. A[r+1:].
```

(a) Give a recurrence for Deterministic-Select with groups of size 7.

(b) Argue that the algorithm with groups of size 7 runs in $\theta(n)$.

(c) Give a recurrence for Deterministic-Select with groups of size 3. Argue that the algorithm is $\omega(n)$.

Problem 3-3. (Super Slow Search...)

You are given an array of A[1...n] of distinct integers. We will now consider various search algorithms to find an element x.

(a) Define Random-Search:

```
function rand_search(A, n, x):
    while True
        i = rand_int(0, n)
        if A[i] == x:
            return i
```

What is the best, expected, and worst case runtime?

(b) Define Linear-Search:

```
function linear_search(A, n, x):
    /* A = shuffle(A) */
    for i in 1...n:
        if A[i] == x:
            return i
    throw exception
```

What is the best, expected, and worst case runtime?

(c) Define Shuffle-Search. Uncomment the first line from Linear-Search in the previous part. What is the best, expected, and worst case runtime?

(d) Which searching algorithm do you prefer?

Problem 3-4. (Iterative Randomized-Select)

(a) What is the space complexity of Randomized-Select?

(b) Can we do better? Give an iterative algorithm that runs with $O(1)$ space.