Problem 3-1. (Quicksort Median-of-3 Partition)

One way to improve the randomized quicksort procedure is to partition around a pivot that is chosen more carefully than by simply picking a random element from the subarray. In the median-of-3 method, the pivot is chosen as the median of a set of 3 elements randomly selected from the subarray.

(a) Assume the elements in the input array A[1..n] are distinct and $n \geq 3$. We denote the sorted output array by A'[1..n]. Using the median-of-3 method to choose the pivot element x, define $p_i = Pr\{x = A'[i]\}$. Give an exact formula for p_i as a function of n and i for $i = 2, 3, ..., n-1$.

Solution:

$$p_i = \frac{6(i-1)(n-i)}{n(n-1)(n-2)}$$

The three randomly selected elements must include one smaller than $A'[i]$, one larger than $A'[i]$, and $A'[i]$ itself. These elements can be chosen in any order.

(b) By what amount have we increased the likelihood of choosing the pivot as the median of A[1..n] compared to the ordinary implementation where the pivot is chosen uniformly randomly? Assume $n \to \infty$, and give the limiting ratio of these probabilities.

Solution:

$$\lim_{n\to\infty} \frac{6(i-1)(n-i)}{n(n-1)(n-2)} \Big/ \frac{1}{n} = \lim_{n\to\infty} \frac{6(\frac{n}{2}-1)(n-\frac{n}{2})}{(n-1)(n-2)} = \lim_{n\to\infty} \frac{\frac{3}{2}(n^2-2n)}{(n^2-3n+2)} = \frac{3}{2}$$

(c) Argue intuitively that in the $\Omega(n \lg n)$ running time of quicksort, the median-of-3 method affects only the constant factor.

Solution: Even if the best pivot (the median of the subarray) is chosen every time, the lower bound on the runtime is still only $\Omega(n \lg n)$. So, a technique that increases the probability of a good pivot being chosen such as median-of-3 cannot improve the asymptotic runtime beyond $\Omega(n \lg n)$.

Problem 3-2. (Deterministic-Quicksort)

(a) Recall from lecture the Randomized-Select algorithm to 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?

Solution: The best and expected runtimes are both $\theta(n)$. The worst case is $\theta(n^2)$.

(b) In lecture, we saw a worst-case linear-time algorithm for selection that involved recursively choosing the median of $\lfloor \frac{n}{5} \rfloor$ group medians to be pivot. Given this algorithm, how would you modify the Quicksort code below to bound the worst-case runtime on any input to $\theta(n \lg n)$?

```
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)
```

Solution: Use the deterministic selection algorithm to find the median. Take the median as the pivot and partition around it. Now, recurse on both sides.

The recurrence for Deterministic-Quicksort is $T(n) = 2T(n/2) + \theta(n)$. Apply Master Theorem case 2 to obtain $T(n) = \theta(n \lg n)$.

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

Solution: The worst-case linear-time selection algorithm runs slowly due to large constant factors.

Problem 3-3. (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:

```
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.

Solution: Half of the $n/7$ groups have at least 4 elements greater than the pivot.
Omit the group containing the pivot and the group of $n \mod 7$ elements.
The number of elements greater than the pivot is at least: $4(\lceil \frac{1}{2}\lceil n/7 \rceil \rceil - 2) \geq \frac{2n}{7} - 8$
The same holds true for the number of elements less than the pivot.
So, each time we recurse on at most $n - (\frac{2n}{7} - 8) = \frac{5n}{7} + 8$ elements.
$T(n) \leq T(\lceil n/7 \rceil) + T(5n/7 + 8) + \theta(n)$

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

Solution: $T(n) = \Omega(n)$ is trivial.
Guess $T(m) \leq cm$ for $m < n$.

$$\begin{aligned} T(n) &\leq c(\lceil n/7 \rceil) + c(5n/7 + 8) + n \\ &\leq c(n/7) + c + 5cn/7 + 8c + n \\ &= 6cn/7 + 9c + n \\ &\leq cn = O(n) \end{aligned} \tag{1}$$

The inequalities hold for large c, n. Try, $c \geq 20, n \geq 100$.

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

Solution: Half of the $n/3$ groups have at least 2 elements greater than the pivot.
Omit the group containing the pivot and the group of $n \mod 3$ elements.
The number of elements greater than the pivot is at least: $2(\lceil \frac{1}{2}\lceil n/3 \rceil \rceil - 2) \geq \frac{n}{3} - 4$

The same holds true for the number of elements less than the pivot.
So, each time we recurse on at most $n - (\frac{n}{3} - 4) = \frac{2n}{3} + 4$ elements.
$T(n) \leq T(\lceil n/3 \rceil) + T(2n/3 + 4) + \theta(n)$

Problem 3-4. (Super Slow Search...)

You are given an array 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?

Solution:
Best: $O(1)$
Worst: ...
Expected: $O(n)$, from geometric distribution if we know the x is in the array. Otherwise ...

(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?

Solution:
Best: $O(1)$
Worst: $O(n)$
Expected: On random inputs, $\frac{1}{n}\sum_{i=1}^{n} i = \frac{1}{n} \cdot \frac{n(n+1)}{2} = \frac{n+1}{2} = \theta(n)$

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

Solution:
Best: $O(n)$, from shuffling
Worst: $O(n)$
Expected: $O(n)$, same as before.

(d) Which searching algorithm do you prefer?

Solution: Linear-Search. (Shuffle-Search makes an extra pass through the array.)

Problem 3-5. (Iterative Randomized-Select)

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

Solution: Consider the number of necessary stack frames. Expected, $O(\lg n)$. Worst-case, $O(n)$.

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

Solution: Yes. Use the same tail recursion technique from problem set 2.

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