CSCI 3104 **Prof. Aaron Clauset**
Problem Set 3 **Spring 2017, CU-Boulder**

1. (5 pts) Why do we analyze the average-case performance of a randomized algorithm and not its worst-case performance? Succinctly explain.

2. (15 pts) Solve the following recurrence relations using the recurrence tree method; include a diagram of your recurrence tree. If the recurrence relation describes the behavior of an algorithm you know, state its name.

   (a) $T(n) = T(\lceil n/2 \rceil) + 1$

   (b) $T(n) = 2T(n/2) + n$

   (c) $T(n) = 2T(n/2) + 1$

3. (20 pts) Lemma 13.1 in CLRS states

   A red-black tree with n internal nodes has height at most $2\lg(n+1)$.

   For a tree with $n = 10$, give a sequence of positive integers $\sigma = [\sigma_1, \sigma_2, \ldots, \sigma_n]$ for $\sigma_i \in [1, 10]$, such that inserting that sequence into an empty red-black tree achieves the maximum height. Justify your claim.

4. (30 pts) Professor McGonagall asks you to help her with some arrays that are *peaked*. A peaked array has the property that the subarray $A[1..i]$ has the property that $A[j] < A[j+1]$ for $1 \leq j < i$, and the subarray $A[i..n]$ has the property that $A[j] > A[j+1]$ for $i \leq j < n$. Using her wand, McGonagall writes the following peaked array on the board $A = [7, 8, 10, 15, 16, 23, 19, 17, 4, 1]$, as an example.

   (a) Write a recursive algorithm that takes asymptotically sub-linear time to find the maximum element of A.

   (b) Prove that your algorithm is correct. (Hint: prove that your algorithm's correctness follows from the correctness of another correct algorithm we already know.)

   (c) Now consider the *multi-peaked* generalization, in which the array contains k peaks, i.e., it contains k subarrays, each of which is itself a peaked array. Let $k = 2$ and prove that your algorithm can fail on such an input.

   (d) (10 pts extra credit) Suppose that $k = 2$ and we can guarantee that neither peak is closer than $n/4$ positions to the middle of the array, and that the "joining point" of the two singly-peaked subarrays lays in the middle half of the array. Now write an algorithm that returns the maximum element of A in sublinear time. Prove that your algorithm is correct, give a recurrence relation for its running time, and solve for its asymptotic behavior.

5. (30 pts total) Professor Dumbledore needs your help. He gives you an array A consisting of n integers $A[1], A[2], \ldots, A[n]$ and asks you to output a two-dimensional $n \times n$ array B in which $B[i, j]$ (for $i < j$) contains the sum of array elements $A[i]$ through $A[j]$, i.e., the sum $A[i] + A[i+1] + \cdots + A[j]$. (The value of array element $B[i, j]$ is left unspecified whenever $i \geq j$, so it doesn't matter what the output is for these values.)

   Dumbledore suggests the following simple algorithm to solve this problem:

```
dumbledoreSolve(A) {
   for i=1 to n
      for j = i+1 to n
         s = sum of array elements A[i] through A[j]
         B[i,j] = s
      end
   end
}
```

   (a) For some function f that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).

   (b) For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)

   (c) Although Dumbledore's algorithm is a natural way to solve the problem—after all, it just iterates through the relevant elements of B, filling in a value for each—it contains some highly unnecessary sources of inefficiency. Give an algorithm that solves this problem in time $O(f(n)/n)$ (asymptotically faster) and prove its correctness.

6. (20 points extra credit) With a sly wink, Dumbledore says his real goal was actually to calculate and return the largest value in the matrix B, that is, the largest subarray sum in A. Butting in, Professor Hagrid claims to know a fast divide and conquer algorithm for this problem that takes only $O(n \log n)$ time (compared to applying a linear search to the B matrix, which would take $O(n^2)$ time).

   Hagrid says his algorithm works like this:

   - Divide the array A into left and right halves
   - Recursively find the largest subarray sum for the left half

- Recursively find the largest subarray sum for the right half
- Find largest subarray sum for a subarray that spans between the left and right halves
- Return the largest of these three answers

On the chalkboard, which appears out of nowhere in a gentle puff of smoke, Hagrid writes the following pseudocode for his algorithm:

```
hagridSolve(A) {
    if(A.length()==0) { return 0 }
    return hagHelp(A,1,A.length())
}

hagHelp(A, s, t) {
    if (s  > t) { return 0 }
    if (s == t) { return max(0, A[s]) }

    m = (s + t) / 2

    leftMax = sum = 0
    for (i = m, i > s, i--) {
            sum += A[i]
            if (sum >= leftMax) { leftMax = sum }
    }

    rightMax = sum = 0
    for (i = m, i <= t, i++) {
            sum += A[i]
            if (sum > rightMax) { rightMax = sum }
    }

    spanMax = leftMax + rightMax
    halfMax = max( hagHelp(s, m), hagHelp(m+1, t) )
    return max(spanMax, halfMax)
}
```

Hagrid claims that his algorithm is correct, but Dumbledore says "tut tut."

(i) Identify and fix the errors in Hagrid's code, (ii) prove that the corrected algorithm works, (iii) give the recurrence relation for its running time, and (iv) solve for its asymptotic behavior.