CMPS 201  
Midterm 1 Review  
Solutions to Selected Problems

1. Let $T(n)$ satisfy the recurrence $T(n) = aT(n/b) + f(n)$, where $a \geq 1$, $b > 1$ and $f(n)$ is a polynomial satisfying $\text{deg}(f) > \log_b(a)$. Prove that case (3) of the Master Theorem applies, and in particular, prove that the regularity condition necessarily holds.

Proof:
Let $d = \text{deg}(f)$ and replace $f(n)$ by the asymptotically equivalent function $n^d$. We compare the polynomials $n^d$ and $n^\log_b(a)$. Let $\epsilon = d - \log_b(a)$, which is positive since $d > \log_b(a)$. Therefore $d = \log_b(a) + \epsilon$, and $n^d = \Omega(n^d) = \Omega(n^{\log_b(a)+\epsilon})$, verifying the first hypothesis of case (3).

Observe $d > \log_b(a) \Rightarrow b^d > a \Rightarrow a/b^d < 1$. Pick any $c$ in the range $a/b^d \leq c < 1$. Then for any $n \geq 1$, we have $a(n/b)^d = (a/b^d)n^d \leq cn^d$, verifying the regularity condition. ■

2. The $n$th harmonic number is defined to be $H_n = \sum_{k=1}^{n} \left( \frac{1}{k} \right)$. Use induction to prove that

$$\sum_{k=1}^{n} H_k = (n + 1)H_n - n$$

for all $n \geq 1$. (Hint: Use the fact that $H_n = H_{n-1} + \frac{1}{n}$)

Proof:
I. If $n = 1$, then $H_1 = 1$ and $\sum_{k=1}^{1} H_k = 1 = 2 - 1 = (1 + 1) - 1 - 1 = (1 + 1)H_1 - 1$, so the base case is satisfied.

II. Let $n > 1$ be chosen arbitrarily, and assume $\sum_{k=1}^{n-1} H_k = ((n - 1) + 1)H_{n-1} - (n - 1)$. We must show that $\sum_{k=1}^{n} H_k = (n + 1)H_n - n$. We have

$$\sum_{k=1}^{n} H_k = \sum_{k=1}^{n-1} H_k + H_n$$

$$= ((n - 1) + 1)H_{n-1} - (n - 1) + H_n$$

by the induction hypothesis

$$= nH_{n-1} - n + 1 + H_n$$

$$= nH_n - nH_n + nH_{n-1} - n + 1 + H_n$$

$$= (n + 1)H_n - n + 1 - n(H_n - H_{n-1})$$

$$= (n + 1)H_n - n + 1 - n \left( \frac{1}{n} \right)$$

by the definition of $H_n$

$$= (n + 1)H_n - n,$$

as required. It follows that $\sum_{k=1}^{n} H_k = (n + 1)H_n - n$ for all $n \geq 1$. ■
3. Define the sequence $S_n$ by the recurrence $S_n = (n - 1) + \frac{n-1}{n^2} \cdot \sum_{k=1}^{n-1} S_k$. Use induction to prove $S_n \leq 2n$ for all $n \geq 1$.

**Proof:**

I. Observe $S_1 = (1 - 1) + \frac{1-1}{1^2} \cdot (\text{empty sum}) = 0 \leq 2 = 2 \cdot 1$, establishing the base case.

II. Let $n > 1$, and assume for all $k$ in the range $1 \leq k < n$ that $S_k \leq 2k$. We must show that $S_n \leq 2n$. We have

$$S_n = (n - 1) + \frac{n-1}{n^2} \cdot \sum_{k=1}^{n-1} S_k$$

$$\leq (n - 1) + \frac{n-1}{n^2} \cdot \sum_{k=1}^{n-1} 2k \quad \text{by the induction hypothesis}$$

$$= (n - 1) + \frac{(n-1)^2}{n}$$

$$= (n - 1) \left( 1 + \frac{n-1}{n} \right)$$

$$= (n - 1) \left( 1 + 1 - \frac{1}{n} \right)$$

$$= (n - 1) \left( 2 - \frac{1}{n} \right)$$

$$= 2n - 2 - 1 + \frac{1}{n}$$

$$= 2n - 3 + \frac{1}{n}$$

$$\leq 2n \quad \text{since} \; n > 1 \Rightarrow \frac{1}{n} \leq 1 \Rightarrow -3 + \frac{1}{n} \leq 0$$

as required. It follows that $S_n \leq 2n$ for all $n \geq 1$. ■

4. The following sorting algorithm, called BadSort() is a modified version of StoogeSort() from the 2nd edition of CLRS, which seems to have been left out of the 3rd edition.

```
BadSort(A, p, r)  pre: p ≤ r
3. else
4.   return
5.   q = [(r - p + 1)/3]
6.   BadSort(A, p, r - q)
7.   BadSort(A, p + q, r)
8.   BadSort(A, p, r - q)
```

a. Use induction on the length \( m = r - p + 1 \) of \( A[p \cdots r] \) to prove the correctness of \( \text{BadSort}(\cdot) \).

**Proof:**

I. If \( m = 1 \), then \( p = r \) so the test on line (1) is false and that on line (3) is true, so the algorithm returns with no changes to the array. Indeed, an array of length 1 is already sorted and no changes are necessary. If \( m = 2 \), then \( p + 1 = r \). Lines (1) and (2) insure that \( A[p] \) and \( A[p + 1] \) are arranged in increasing order. The test on line (3) is true so the algorithm returns with no other action. The base cases are therefore satisfied.

II. Let \( m > 2 \), and assume that \( \text{BadSort}(\cdot) \) correctly sorts any subarray of length less than \( m \). We must show that if \( m = r - p + 1 \), then \( \text{BadSort}(A, p, r) \) correctly sorts \( A[p \cdots r] \). After placing \( A[p] \) and \( A[r] \) in increasing order, the test on line (3) will be false (since \( m > 2 \Rightarrow r - p + 1 > 2 \Rightarrow r > p + 1 \)) so lines (6)-(9) will be executed. Line (6) sets \( q = \lfloor m/3 \rfloor \), and since \( m \geq 3 \) we have \( q \geq 1 \). Therefore

\[
\text{length}(A[p \cdots (r - q)]) = r - q - p + 1 = m - q < m
\]

and

\[
\text{length}(A[(p + q) \cdots r]) = r - p - q + 1 = m - q < m.
\]

By our induction hypothesis, the effect of the recursive calls on lines (7)-(9) is to correctly sort the corresponding subarrays. It remains to show that this sequence of calls has the effect of sorting the subarray \( A[p \cdots r] \). To simplify the discussion, we define \( X, Y \) and \( Z \) to be the subarrays

\[
X = A[p \cdots (p + q - 1)] \quad \text{1st third}
\]

\[
Y = A[(p + q) \cdots (r - q)] \quad \text{2nd third}
\]

\[
Z = A[(r - q + 1) \cdots r] \quad \text{3rd third}
\]

After line (7) is executed, the subarray \( A[p \cdots (r - q)] = (X, Y) \) is sorted. Thus every element in \( X \) is less than or equal to every element in \( Y \), which we signify by writing \( X \leq Y \). After line (8) is executed, \( A[(p + q) \cdots r] = (Y, Z) \) is sorted, whence \( Y \leq Z \). Also \( X \leq Z \) since any element that was in \( X \) before the sort, and which belongs in \( Z \), was placed in \( Y \) by line (7), then placed in \( Z \) by line (8). In other words, all elements that ultimately belong in \( Z \) are placed there by the time (8) is executed. However \( X \leq Y \) may no longer be true at this point since some element that was originally in \( Z \), and is now in \( Y \), may be smaller than some element of \( X \). After line (9), we again have \( X \leq Y \), so the subarrays \( X, Y \) and \( Z \) are sorted and \( X \leq Y \leq Z \). Therefore \( A[p \cdots r] = (X, Y, Z) \) is now sorted, as required.

b. Write a recurrence relation for the number of array comparisons performed by \( \text{BadSort}(\cdot) \) on an array of length \( n \).

**Solution:**
At the top level of the recurrence, the sub-arrays have length \( n - q = n - \lfloor n/3 \rfloor = \lfloor 2n/3 \rfloor \). The (best, worst and average case) run time \( T(n) \) of \( \text{BadSort}(\cdot) \) therefore satisfies the recurrence

\[
T(n) = \begin{cases} 
1 & 1 \leq n < 3 \\
3T(\lfloor 2n/3 \rfloor) + 1 & n \geq 3 
\end{cases}
\]
c. Use the Master Theorem to find an asymptotic solution to this recurrence, and explain what is bad about BadSort().

Solution:
Simplifying the above recurrence for the Master Theorem gives \( T(n) = 3T \left( \frac{n}{3^{1/2}} \right) + 1 \). We compare \( 1 = n^0 \) to \( n^{\log_{3/2}(3)} \). Observe \( 3 > 1 \Rightarrow \log_{3/2}(3) > 0 \), so setting \( \epsilon = \log_{3/2}(3) \), we have \( 1 = n^0 = O(n^0) = O(n^{\log_{3/2}(3)-\epsilon}) \). Case (1) yields \( T(n) = \Theta(n^{\log_{3/2}(3)}) \).

The runtime of most other sorting algorithms is no worse than \( \Theta(n^2) \). For instance MergeSort() and HeapSort() run in \( \Theta(n \log n) \) time, while InsertionSort() and QuickSort() run in time \( \Theta(n^2) \) (again worst case). But \( \log_{3/2}(3) = 2.7095 \ldots \), so BadSort() runs in \( \Theta(n^{2.7095\ldots}) \) time. This is considerably worse than any standard sorting algorithm, making BadSort() aptly named.

5. Simplify the recurrence for MergeSort() by assuming that \( n \) is an exact power of 2; \( n = 2^k \) for some integer \( k \geq 0 \).

\[
T(n) = \begin{cases} 
0 & n = 1 \\
2T \left( \frac{n}{2} \right) + (n - 1) & n \geq 2, n = 2^k
\end{cases}
\]

Use the iteration method to find an exact solution to this recurrence.

Solution:
Recurring down to depth \( k \) yields:

\[
T(n) = (n - 1) + 2T \left( \frac{n}{2} \right) = (n - 1) + 2 \left[ \left( \frac{n}{2} - 1 \right) + 2T \left( \frac{n}{2^2} \right) \right] = (n - 1) + (n - 2) + 2^2 T \left( \frac{n}{2^2} \right) = (n - 1) + (n - 2) + 2^2 \left[ \left( \frac{n}{2^2} - 1 \right) + 2T \left( \frac{n}{2^3} \right) \right] = (n - 1) + (n - 2) + (n - 2^2) + 2^3 T \left( \frac{n}{2^3} \right) \]

\[
\vdots
\]

\[
= \sum_{i=0}^{k-1} (n - 2^i) + 2^k T \left( \frac{n}{2^k} \right).
\]

The recursion halts when \( k \) satisfies: \( \frac{n}{2^k} = 1 \Leftrightarrow n = 2^k \Leftrightarrow k = \log n \). For this \( k \) we have

\[
T(n) = \sum_{i=0}^{k-1} (n - 2^i) + 2^k T(1)
\]
\[
= \sum_{i=0}^{k-1} n + \sum_{i=0}^{k-1} 2^i + 2^k \cdot 0
\]
\[
= k \cdot n + \frac{2^k - 1}{2 - 1}
\]
\[
= n \lg n + 2^k - 1,
\]
and hence \( T(n) = n \lg n + n - 1 \).

One can check directly that this function solves the above recurrence. First \( T(1) = 1 \cdot 0 + 1 - 1 = 0 \).

For the recursive branch, observe that
\[
\text{RHS} = 2T\left(\frac{n}{2}\right) + (n - 1)
= 2 \left[ \left(\frac{n}{2}\right) \lg \left(\frac{n}{2}\right) + \left(\frac{n}{2}\right) - 1 \right] + (n - 1)
= n(\lg n - \lg 2) + n - 2 + n - 1
= n \lg n - n \lg 2 + 2n - 1
= n \lg n - n + 1
= \text{LHS},
\]
showing that \( T(n) = n \lg n + n - 1 \) solves the recurrence in the special case that \( n \) is and exact power of two.  

\[\blacksquare\]