CSE 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 $\deg(f) > \log_b(a)$. Prove that case (3) of the Master Theorem applies, and in particular, prove that the regularity condition necessarily holds.

   Proof:
   Let $d = \deg(f)$ 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 $d > \log_b(a)$. Therfore $d = \log_b(a) + \epsilon$, 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 $\sum_{k=1}^{1} H_k = 1 = 2 - 1 = (1+1)\cdot 1 - 1 = (1+1)H_1 - 1$, 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

$$\begin{aligned}
\sum_{k=1}^{n} H_k &= \sum_{k=1}^{n-1} H_k + H_n \\
&= \big((n-1)+1\big)H_{n-1} - (n-1) + H_n && \text{by the induction hypothesis} \\
&= nH_{n-1} - n + 1 + H_n \\
&= nH_n - nH_n + nH_{n-1} - n + 1 + H_n \\
&= (n+1)H_n - n + 1 - n(H_n - H_{n-1}) \\
&= (n+1)H_n - n + 1 - n\left(\frac{1}{n}\right) && \text{by the definition of } H_n \\
&= (n+1)H_n - n,
\end{aligned}$$

   as required. If 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 $S_n \le 2n$ for all $n \ge 1$.

Proof:

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

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

$$
\begin{aligned}
S_n &= (n-1) + \frac{n-1}{n^2} \cdot \sum_{k=1}^{n-1} S_k \\
&\le (n-1) + \frac{n-1}{n^2} \cdot \sum_{k=1}^{n-1} 2k && \text{by the induction hypothesis} \\
&= (n-1) + \frac{n-1}{n^2} \cdot 2 \cdot \frac{n(n-1)}{2} \\
&= (n-1) + \frac{(n-1)^2}{n} \\
&= (n-1)\left(1 + \frac{n-1}{n}\right) \\
&= (n-1)\left(1 + 1 - \frac{1}{n}\right) \\
&= (n-1)\left(2 - \frac{1}{n}\right) \\
&= 2n - 2 - 1 + \frac{1}{n} \\
&= 2n - 3 + \frac{1}{n} \\
&\le 2n && \text{since } n > 1 \Rightarrow \frac{1}{n} \le 1 \Rightarrow\ -3 + \frac{1}{n} \le 0
\end{aligned}
$$

as required. It follows that $S_n \le 2n$ for all $n \ge 1$. ∎

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
if  A[p] > A[r]
     A[p] ↔ A[r]  (swap)
if  p + 1 ≥ r
     return
else
     q = ⌊(r − p + 1)/3⌋
     BadSort(A, p, r − q)
     BadSort(A, p + q, r)
     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 BadSort().

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 BadSort() 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

$X = A[p \cdots (p + q - 1)]$	1st third
$Y = A[(p + q) \cdots (r - q)]$	2nd third
$Z = A[(r - q + 1) \cdots r]$	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 BadSort() on an array of length n.

Solution:

At the top level of the recurrence, the sub-arrays have length $n - q = n - \lfloor n/3 \rfloor = \lceil 2n/3 \rceil$. The (best, worst and average case) run time $T(n)$ of BadSort() therefore satisfies the recurrence

$$T(n) = \begin{cases} 1 & 1 \leq n < 3 \\ 3T(\lceil 2n/3 \rceil) + 1 & n \geq 3 \end{cases}$$

■

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 $T(n) = 3T\left(\frac{n}{3/2}\right) + 1$. 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 $1 = n^0 = O(n^0) = O\left(n^{\log_{3/2}(3)-\epsilon}\right)$. Case (1) yields $T(n) = \Theta\left(n^{\log_{3/2}(3)}\right)$.

The runtime of most other sorting algorithms is no worse than $\Theta(n^2)$. For instance MergeSort() and HeapSort() run in (worst case) $\Theta(n \log n)$ 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. Define $T(n)$ by the recurrence

$$T(n) = \begin{cases} 0 & n = 1 \\ 2T(\lfloor n/2 \rfloor) + n \lg(n) & n \geq 2 \end{cases}$$

Here lg means $\log_2$.

a. Prove that the Master Theorem cannot be applied to this recurrence.

Proof:
We first simplify the recurrence to $T(n) = 2T(n/2) + n\lg(n)$. Comparing $n \lg(n)$ to $n^{\log_2(2)} = n$, we see that $\frac{n\lg(n)}{n} = \lg(n) \longrightarrow \infty$ as $n \to \infty$, so that $n\lg(n) = \omega(n)$. Since $n\lg(n)$ is the winner, the only possible case of the Master Theorem that could apply is case 3. But although $n\lg(n)$ is the winner, it does not win by a polynomial factor. Indeed, let $\epsilon > 0$ be chosen arbitrarily. Then

$$\frac{n\lg(n)}{n^{\log_2(2)+\epsilon}} = \frac{n\lg(n)}{n^{1+\epsilon}} = \frac{\lg(n)}{n^{\epsilon}} \longrightarrow 0 \text{ as } n \to \infty,$$

which yeilds $n\lg(n) = o\left(n^{\log_2(2)+\epsilon}\right)$. Exercise 6 on page 5 of the handout on asymptotic growth rates implies $o\left(n^{\log_2(2)+\epsilon}\right) \cap \Omega\left(n^{\log_2(2)+\epsilon}\right) = \emptyset$, and hence $n\lg(n) \notin O\left(n^{\log_2(2)+\epsilon}\right)$. Since $\epsilon > 0$ was arbitrary, this holds for *all* such ϵ. Therefore we are not in case 3, and the Master Theorem cannot be applied. ■

b. Use the Substitution method to prove that $T(n) = O(n(\lg n)^2)$.

Proof:
We show by induction that $T(n) \leq n(\lg n)^2$ for all $n \geq 1$, from which $T(n) = O(n(\lg n)^2)$ follows.

I. For $n = 1$ we have $T(1) = 0 \leq 0 = 1 \cdot (\lg 1)^2$, and the base case is satisfied.

II. Let $n > 1$ be arbitrary, and assume $T(k) \leq k(\lg k)^2$ for any k in the range $1 \leq k < n$. We must show that $T(n) \leq n(\lg n)^2$.

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n\lg(n) \\ &\leq 2 \cdot \lfloor n/2 \rfloor (\lg\lfloor n/2 \rfloor)^2 + n\lg(n) \qquad \text{By the induction hypothesis with } k = \lfloor n/2 \rfloor. \end{aligned}$$

$$\le 2 \cdot (n/2)(\lg(n/2))^2 + n\lg(n) \qquad \text{Since } \lfloor x \rfloor \le x \text{ for any } x \in \mathbb{R}$$

$$= n(\lg(n) - 1)^2 + n\lg(n)$$

$$= n((\lg(n))^2 - 2\lg(n) + 1) + n\lg(n)$$

$$= n(\lg n)^2 - 2n\lg(n) + n + n\lg(n)$$

$$= n(\lg n)^2 - n\lg(n) + n$$

$$\le n(\lg n)^2$$

The last inequality follow from

$$n > 1 \;\Rightarrow\; n \ge 2 \;\Rightarrow\; \lg(n) \ge 1 \;\Rightarrow\; n\lg(n) \ge n \;\Rightarrow\; -n\lg(n) + n \le 0.$$

Therefore $T(n) \le n(\lg n)^2$ for all $n \ge 1$ by the 2nd PMI. ■

7. Given $A = (A_1, A_2, \dots, A_n)$, a pair of indices (i, j) is called an *inversion* iff both $i < j$ and $A_i > A_j$. Write a recursive algorithm that determines the number of inversions in its input array A. Do this in such a way that the worst case number of comparisons performed is $T(n) = \Theta(n \log n)$. (Hint: modify MergeSort() so that it counts inversions as it sorts.)

Solution:
We alter both Merge() and MergeSort() to return an integer, as well as perform their previous sorting functions. Merge(A, p, q, r) returns the number of inversions between the two subarrays $A[p \cdots q]$ and $A[(q+1) \cdots r]$, i.e. it returns a count of the number of times an element in $A[(q+1) \cdots r]$ is less than an element in $A[p \cdots q]$.

```
Merge(A,p,q,r) (Pre: A[p⋯q] and A[(q+1)⋯r] are sorted)
n_1 = (q − p + 1)
n_2 = (r − q)
create arrays L[1⋯(n_1 + 1)] and R[1⋯(n_1 + 1)]
for i = 1 to n_1
        L[i] = A[p + i − 1]
for j = 1 to n_2
        R[j] = A[q + j]
L[n_1 + 1] = ∞,  R[n_2 + 1] = ∞
i = 1, j = 1, count = 0
for k = p to r
        if L[i] ≤ R[j]
                A[k] = L[i]
                i = i + 1
        else
                A[k] = R[j]
                j = j + 1
                count = count + (n_1 − i + 1)
return count
```

Observe that each time the test in line 11 is false, count is incremented by the length of the subarray $L[i \cdots n_1]$, which is the set of elements that $R[j]$ must pass over in order to reach its proper location in subarray $A[p \cdots r]$. By the time the algorithm is complete, count is precisely the number of inversions of the form (x, y) in the subarray $A[p \cdots r]$, where $p \leq x \leq q$ and $q + 1 \leq y \leq r$. $\text{MergeSort}(A, p, r)$ returns the total number of inversions in the subarray $A[p \cdots r]$.

```
MergeSort(A, p, r)
1. if p < r
2.         q = ⌊(p+r)/2⌋
3.         a = MergeSort(A, p, q)
4.         b = MergeSort(A, q + 1, r)
5.         c = Merge(A, p, q, r)
6.         return (a + b + c)
7. else
8.         return 0
```

If $p < r$, the number of inversions in $A[p \cdots r]$ is the sum of the number of inversions in $A[p \cdots q]$, plus the number of inversions in $A[(q + 1) \cdots r]$, plus the number of inversions between the two sub-arrays $A[p \cdots q]$ and $A[(q + 1) \cdots r]$. This is exactly what is returned on line 6. If $p \geq r$, then $A[p \cdots r]$ has length at most 1, and therefore contains no inversions. In this case 0 is returned on line 8. The asymptotic run time of this modified MergeSort() is the same as the original, namely $\Theta(n \lg n)$, by the same analysis as before. ∎