ECS 122A Homework 2 Solution (Subset)

Only Question 8 was graded by correctness, break-down of points is given below. The rest are credited on whether you attempted or not and is worth 10 pts each.

1. **Solution**:

   We guess $T(n) = O(n^2)$.

   Assume $\forall k < n, T(k) \leq ck^2$. Thus, $T(n-1) \leq c(n-1)^2$.

   Then, we do the substitution as follows:

$$\begin{aligned} T(n) &= T(n-1) + n \\ &\leq c(n-1)^2 + n \\ &= c(n^2 - 2n + 1) + n \\ &= cn^2 + (1-2c)n + c \end{aligned}$$

   In order to show $cn^2 + (1-2c)n + c \leq cn^2$, we have to prove $(1-2c)n + c \leq 0$. If $1 - 2c > 0$, then $n \leq \frac{-c}{1-2c}$, and this contradicts with our goal to find n_0 such that when $n \geq n_0$, $T(n) \leq cn^2$. Therefore, $1 - 2c < 0$. Let $c = 1$, $n \geq \frac{-c}{1-2c} = 1$. Let $n_0 = 2$, $T(n) = T(2) = T(1) + 1 = 2 \leq 1 \cdot 2^2$. (We assume $T(1) = 1$)

2. **Solution**:

   We guess $T(n) = O(\log n)$.

   Assume $\forall k < n, T(k) \leq c \log n$. Thus, $T(\lceil n/2 \rceil) \leq c \log \lceil n/2 \rceil$. **Notice that base 2 logarithmic function is monotonically increasing**, so $\forall n > 1, c \log \lceil n/2 \rceil < c \log 2n/3$.

   Then, we do the substitution as follows:

$$\begin{aligned} T(n) &= T(\lceil n/2 \rceil) + 1 \\ &\leq c \log \lceil n/2 \rceil + 1 \\ &< c \log 2n/3 + 1 \\ &= c(\log 2n - \log 3) + 1 \\ &= c(\log 2 + \log n - \log 3) + 1 \\ &= c \log n + c(1 - \log 3) + 1 \end{aligned}$$

   To show $c \log n + c(1 - \log 3) + 1 \leq c \log n$, we have to prove $c(1 - \log 3) + 1 \leq 0$. $c \geq \frac{1}{\log 3 - 1}$. Let $c = 2, n_0 = 2, T(n) = T(2) = T(1) + 1 = 2 \leq 2 \cdot \log 2$. (We assume $T(1) = 1$)

4. **Solution**: Recall that the recursion tree method is essentially to calculate the extra cost ($f(n)$ part in the recurrence relation) at each level of the tree plus the cost of the leaves (and we treat each leaf as constant time). In that sense, the cost of the leaves is equivalent to the number of leaves.

   Let's try to analyze the extra cost at the first few levels to find a pattern. The root (level 0) is simple, just n. The first level is $4(\frac{n}{2}+2)$. The second level is $4^2(\frac{n}{2^2}+2/2+2)$. The third level is $4^3(\frac{n}{2^3}+1/2+2/2+2)$. Do you find the pattern? Inside the parentheses, other than the term involving n, we have a geometric series. If i denotes the number of level, then the extra cost at each level is:

$$\begin{aligned} Cost(i) &= 4^i(\frac{n}{2^i}+\sum_{j=1}^{i}(\frac{1}{2})^{j-2}) \\ &= 2^i n + 4^{i+1} - (2)^{i+2} \end{aligned}$$

   The height of this recursion tree is approximately $\log n + 1$ (including root). For simplicity, we can ignore the lower term 2. And the number of leaves is $4^{\log n} = n^2$. The reason is that at each level, a node will create four children at the next level.

   To bring every part together, we have:

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log n - 1} (2^i n + 4^{i+1} - (2)^{i+2}) + n^2 \\ &= (n-1)n + \frac{4(n^2-1)}{3} - 4(n-1) + n^2 \\ &= \Theta(n^2) \end{aligned}$$

   It's also okay to ignore the fractions in the geometric series at each level since they're the lower terms.

   I leave the verification by substitution to you.

8. **Solution**: 15 pts = 2.5 pts $\times$ 6.

   Mind the difference between uppercase N and lowercase n, especially M2 and M3.

   (a) Recall that the recurrence of binary search is $T(n) = T(n/2) + \Theta(1)$.

M1 will have recurrence $T(n) = T(n/2) + \Theta(1)$. By master theorem, $T(N) = \Theta(\log N)$.

If we use **M2** to pass the array, then the recurrence is given by $T(n) = T(n/2) + \Theta(N)$, since the entire array will be copied for each recursive call. By tree method, $T(N) = \Theta(N \log N)$.

Finally, since **M3** only copies the length of the subproblem, the recurrence for this method is given by $T(n) = T(n/2) + \Theta(n)$. By master theorem, $T(N) = \Theta(N)$.

(b) Recall that the recurrence of merge sort is $T(n) = 2T(n/2) + \Theta(n)$.

For **M1**, if the array is passed by pointer, then the linear merge operation dominates ($\Theta(n) > \Theta(1)$) and the procedure has recurrence $T(n) = 2T(n/2) + n$, which runs in time $\Theta(N \log N)$.

For **M2**, the parameter passing operation dominates the linear merge at each subproblem($\Theta(N) > \Theta(n)$). The recurrence for this method is $T(n) = 2T(n/2) + 2N$, since there are two recursive calls that copy the entire input array. Using the tree method, $T(N) = \Theta(N^2)$.

For **M3**, since we are only copying the size of the subproblem, the recurrence is $T(n) = 2T(n/2) + n$ and the program runs in time $\Theta(N \log N)$.

10. **Solution**

(a) We first merge any two of the three subarrays into a merged array M_1. This will take $\frac{2n}{3}$ comparisons. (Because whenever there's a comparison, we assign a value to an element in M_1 of length $\frac{2n}{3}$). Then we merge M_1 with the remaining subarray, and this will cost n comparisons (same reasoning). In total, we need $\frac{2n}{3} + n = \frac{5n}{3}$ comparisons.

Alternatives are also accepted.

(b) $T(n) = 3T(\frac{n}{3}) + \frac{5n}{3} = \Theta(n \log n)$

11. **Solution**: $T(n) = 2T(\frac{n}{2}) + n^2 = \Theta(n^2)$