CS170–Spring 2017 — Homework 1Solutions

Ninh Hai DO, SID 25949105

Collaborators: NONE

1. Course Syllabus

(a) Not OK

(b) Not OK

(c) Not OK

(d) OK

(e) Not OK

2. Shortcut Question: Asymptotic Complexity Comparisons

(a) The new order is: $f_3,\ f_7,\ f_2,\ f_5,\ f_4,\ f_9,\ f_8,\ f_6,\ f_1$

(b) Consider the limit:

$$L = \lim_{x\to\infty} \frac{\log x}{x^{\varepsilon}} \tag{1}$$

Since both nominator and denominator go to ∞ for $x \to \infty$, using De L'Hopital rule gives:

$$L = \lim_{x\to\infty} \frac{\log x}{x^{\varepsilon}} = \lim_{x\to\infty} \frac{(\log x)'}{(x^{\varepsilon})'} = \lim_{x\to\infty} \frac{\frac{1}{x\ln a}}{x^{\varepsilon-1}} = \lim_{x\to\infty} \frac{1}{x^{\varepsilon}\ln a} = 0 \tag{2}$$

where a is the base of logarithm. This means the quotient $\dfrac{\log x}{x^{\varepsilon}} < 1$ for large x. Therefore, $\log x \in O(x^{\varepsilon})$

3. Recurrence Relations

(a) Using Master theorem with $a = 2,\ b = 2,\ d = \dfrac{1}{2}$, we have $\log_b a = 1 > \frac{1}{2} = d$, so:

$$T(n) = \Theta(n^{\log_2 2}) = \Theta(n) \tag{3}$$

(b) Expanding T gives:

$$\begin{aligned} T(n) &= T(n-1) + c^n \\ &= T(n-2) + c^{n-1} + c^n \\ &= T(1) + c^2 + c^3 + \cdots + c^{n-1} + c^n \\ &= T(1) + c^2(1 + c + c^2 + \cdots + c^{n-2}) \\ &= T(1) + c^2 \times \frac{c^{n-1} - 1}{c - 1} \\ &= \Theta(c^n) \end{aligned}$$

(c) Expanding T(n) tree, we see that the tree has $\log\log n$ layers, layer i-th has $2^i \times 3$ operations as below:
Layer 0: $T(n)$ $2^0 \times 3$ operations
Layer 1: $T(\sqrt{n})$ $T(\sqrt{n})$ $2^1 \times 3$ operations
Layer 2: $T(\sqrt[4]{n})$ $T(\sqrt[4]{n})$ $T(\sqrt[4]{n})$ $T(\sqrt[4]{n})$ $2^2 \times 3$ operations
etc.
The number of operation is:

$$\begin{aligned} T(n) &= 3 \times \left(1 + 2 + 4 + \cdots + 2^{\log\log n}\right) \\ &= 3 \times \left(2^{\log\log n + 1} - 1\right) \\ &= \Theta(\log n) \end{aligned}$$

4. Bound the Running Time

(a) Replace n by 2^k we see the algorithm $f2$ expands in $\Theta(k)$ levels similar to Fibonacci algorithm $f1$ expanding in $\Theta(n)$ levels in example. That is $f2(n/2) = f2(2^{k-1})$, $f2(n/2) = f2(2^{k-2})$, corresponding to $f1(n-1)$, $f1(n-2)$.
Therefore, the answer is the Fibonacci answer with replacing n by $k = \log n$. That is
$\Theta\left(\left(\frac{1}{2}\left(1+\sqrt{5}\right)\right)^{\log n}\right)$

(b) In the best case $n = 2^k$, the runtime is $\Theta(\log n)$. It is the best case because in each recursion call, the value of argument always reduces by half, never increases.
In the worst case $n = 2^k + 1$, the runtime is $\Theta(\log n + \log n)$ $\Theta(\log n)$. The first $\log n$ is the number of times where argument adding up with 1, after each time, the argument becomes even and is reduced by half in next recursion call. Since it alternates between odd and even in each recursion call, the number of times to call $f3(n/2)$ is approximately equal to the number of times to call $f3(n+1)$, that is reason why we have two $\log n$. This is the worst case because there is no case of n that yields two consecutive $f(n+1)$ recursion calls
The algorithm $f3$ is lower and upper bounded by $\Theta(\log n)$, so its runtime is $\Theta(\log n)$.

(c) In the best case $n = 2^k$, the runtime is $\Theta(\log n)$.
In the worst case $n = 2^k + 1$, the runtime is $\Theta(\log n + 2 * \log n + 2 * \log n)$ $\Theta(\log n)$. The first $\log n$ is the number of times where argument reduced by half, the second $2 * \log n$ is the number of times after which the argument turning in to $3^k - 1$, the third $2 * \log n$ is the number of times the argument alternates between $f4(n/2)$ and $f4(3n+1)$.
The algorithm $f4$ is lower and upper bounded by $\Theta(\log n)$, so its runtime is $\Theta(\log n)$.

5.

YOUR ANSWER GOES HERE

6. Merged Median

Main idea:

1. Find medians of k sorted arrays in $\Theta(k)$, we have the array M of k medians. Find the median mM of M in $O(k)$ using the devide-and-conquer algorithm in lecture.
2. For each original array, devide it into two subarrays, one smaller or equal to mM, the other is larger than mM. After doing it to all k arrays, eliminate the parts that add up less than a half number of kl elements.
3. Repeat step 1 and 2 until the number of eliminated element on one side equal to a half number of kl elements.

Pseudocode:

$$stillworking \tag{4}$$

Proof of Correctness:

It is correct because we always use a single number mM, aka the median of medians, in each recursion call to divide the k arrays, so no matter that they are k separate arrays or 1 big array, as long as we keep track of the number of eliminated elements (to make sure it reach a half of n on one side) we obtain the middle element, which is the median of all array.

Running time:

For each recursion call, finding medians of k sorted arrays takes $\Theta(k)$, finding the median of the medians' array takes $O(k)$ time (using the randonm devide-and-conquer algorithm in lecture), dividing array in step 2 using binary search takes $O(k \log l)$ time. Therefore, the runtime for each iteration is $O(k + k \log l)$. Since the argument of each recursion call reduces by half, there is around $O(\log l)$ recursion layers.

The total runtime is: $O(k \log l + k \log^2 l)$ or $O(k \log^2 l)$