CS 457, Data Structures and Algorithms I
Second Problem Set

October 9, 2018

1. (24 pts) Solve the following recurrence equation (tight upper *and* lower bounds!)

$$T(n) = \begin{cases} 1 & \text{if } n < 1 \\ 2T(n/2) + 23 & \text{otherwise.} \end{cases}$$

   (a) Using the *master theorem*. Make sure to explain which case applies and why.

   Solution: We examine the recurrence case and identify a, b and $f(n)$. We see that $a = 2$, $b = 2$, and $f(n) = 23 = \Theta(1)$.

   So we can apply the first case of the master theorem since we see that $n^{\log_2(2)-\epsilon} = n^{1-\epsilon}$. We can pick $\epsilon = \frac{1}{2}$ and we have that $23 = O(\sqrt{n})$.

   Applying the first case of the master theorem we see that we can conclude the asymptotic bound directly:

$$T(n) = \Theta(n).$$

   (b) Using the *recursion tree* method. Do not use asymptotic notation for the depth of the recursion tree; use exact upper and lower bounds instead.

   Solution: We observe that at each level of the recursion tree, the input size is halved and that the tree does not reach leaves until the input size to $T(n)$ is less than 1. Thus, we can find the exact depth of the tree as

$$d = \lfloor \lg(n) \rfloor + 1.$$

   We must then find the cost at every level of the tree. From the recurrence equation, we see that for every non-leaf vertex in the recursion tree we will have cost 23 and for every leaf in the recurrence tree we will have cost 1. To determine the number of vertices at each level of depth in the tree, we observe that based on the leading coefficient of our recurrence equation, the number of vertices doubles at each level in the recursion tree. In fact, the number of vertices at depth d is exactly 2^d since none of the branches "die out" before any others (i.e. our tree is perfectly balanced). Thus we have for our tree a cost of:

$$\sum_{d=0}^{\lfloor \lg n \rfloor} \left(23(2^d)\right) + 2^{\lfloor \lg n \rfloor + 1}.$$

Rewriting this sum and then applying algebra gives us an upper bound:

$$\begin{aligned} T(n) &= 23\sum_{d=0}^{\lfloor \lg n \rfloor} \left(2^d\right) + 2^{\lfloor \lg n \rfloor + 1} \\ &\leq 23\sum_{d=0}^{\lg n} 2^d + 2^{\lg n + 1} \\ &= 23\frac{2^{\lg n + 1} - 1}{2 - 1} + 2(2^{\lg n}) \\ &= 23(2n - 1) + 2n \\ &= 48n - 23 \\ &\Rightarrow \\ T(n) &= O(n) \end{aligned}$$

Similarly, we may find a lower bound:

$$\begin{aligned} T(n) &= 23\sum_{d=0}^{\lfloor \lg n \rfloor} \left(2^d\right) + 2^{\lfloor \lg n \rfloor + 1} \\ &\geq 23\sum_{d=0}^{\lg n - 1} \left(2^d\right) + 2^{\lg n} \\ &= 23\frac{2^{\lg n} - 1}{2 - 1} + 2^{\lg n} \\ &= 23(n - 1) + n \\ &= 24n - 23 \\ &\Rightarrow \\ T(n) &= \Omega(n) \end{aligned}$$

(c) Using the *substitution* method. Make sure to show that the boundary conditions hold as well; choose the constants c and n_0 appropriately.

Based on the previous two sections we guess that $T(n) = \Theta(n)$. We begin by showing $T(n) = O(n)$. We assume that $T(n') \leq cn' - d$ for all $n' < n$. We seek to show $T(n) \leq cn - d$. We have the following:

$$\begin{aligned} T(n) &= 2T(n/2) + 23 \\ &\leq 2(c\tfrac{n}{2} - d) + 23 \\ &= cn - 2d + 23 \qquad \text{Select c = 2, d = 23} \\ &= cn - d \end{aligned}$$

Thus, we have that $T(n) = O(n)$.

We now seek to show the lower bound, i.e. $T(n) = \Omega(n)$. We assume that $T(n') \geq cn'$ for all $n' < n$. We seek to show $T(n) \geq cn$. We have the following:

$$\begin{aligned} T(n) &= 2T(n/2) + 23 \\ &\geq 2c\tfrac{n}{2} + 23 \\ &= cn + 23 \\ &\geq cn \end{aligned}$$

Thus, we have $T(n) = \Omega(n)$ and therefore $T(n) = \Theta(n)$.

2. (30 pts) For the following algorithm calls, prove a *tight* asymptotic bound for their worst-case running time. Pay attention to the input in the algorithm *calls*!

   a) The call to RECURSIVE-ALGORITHM(n) for some $n > 1$.

```
1 RECURSIVE-ALGORITHM(a);
2 q = 0;
3 if a ≥ 1 then
4     for i = 1 to ⌊a⌋ do
5         q=q+1
6     RECURSIVE-ALGORITHM(a/2);
7     RECURSIVE-ALGORITHM(a/5);
8     RECURSIVE-ALGORITHM(a/9);
```

   Solution: As long as $a \geq 1$, the algorithm RECURSIVE-ALGORITHM(a) performs a sequence of operations that costs $\Theta(a)$ and then calls itself recursively three times. We can therefore represent the worst-case running time of the call to RECURSIVE-ALGORITHM(n) using the following equation: $T(n) = T(n/2) + T(n/5) + T(n/9) + \Theta(n)$.
   We guess that the solution to this equation is $T(n) = \Theta(n)$ and assume that $T(n') \leq cn'$ for some constant c and all $n' \leq n$. Upon substitution, this yields

$$T(n) \leq cn/2 + cn/5 + cn/9 + \Theta(n) \leq 73cn/90 + kn.$$

   We want to choose a value for c such that, for any constant k we have $73cn/90 + kn \leq cn$ which leads to $c \geq 90k/17$, and thus proves that $T(n) = O(n)$.
   For the lower bound, we assume that $T(n') \geq c'n'$ for some constant c' and all $n' \leq n$. Upon substitution, this yields

$$T(n) \geq c'n/2 + c'n/5 + c'n/9 + \Theta(n) \geq 73c'n/90 + k'n.$$

   We want to choose a value for c' such that, for any constant k' we have $73c'n/90 + k'n \geq c'n$ which leads to $c' \leq 90k'/17$, and thus proves that $T(n) = \Omega(n)$, which concludes the proof.

   b) The call to RECURSIVE-ALGORITHM2(n, n) for some $n > 2$.

```
1 RECURSIVE-ALGORITHM2(a, b);
2 if a ≥ 2  and  b ≥ 2 then
3     u = a/2;
4     v = b − 1;
5     RECURSIVE-ALGORITHM2(u, v);
```

Solution: While $a \geq 2$ and $b \geq 2$, the RECURSIVE-ALGORITHM2(a, b) algorithm performs some constant cost operations and then calls itself recursively. Since there is a single recursive call, we prefer to use the recursion tree method, because the tree will consist of a single path. As we mentioned above, the cost at each level of the tree is constant, so all that we need to figure out is what the length of this path may be. Note that the call whose worst-case running time we are asked to analyze is RECURSIVE-ALGORITHM2(n, n) for some $n > 2$, so the values of the two variables are initially the same. We then observe that after at most $\lg n$ recursive calls the value of the first variable will be less than 2, which will cause the termination of the recursive calls, and hence the depth of the recursion tree is at most $\lg n$. Since the cost at each level is constant, this also implies that the worst-case running time of the given call will be $O(\lg n)$. In fact, while $a \leq 2$ and $b \leq 2$ the second variable value shrinks slower than the first, so the depth and the cost will also be $\Omega(\lg n)$, which concludes the proof.

3. (30 pts) In solving the following recurrence equations, first try to use the master theorem. If it does not apply, explain why this is the case.

   a) Solve the following recurrence equation.

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ T(2n/3) + T(n/4) + \Theta(n) & \text{otherwise.} \end{cases}$$

   Solution: The master theorem does not work since there is multiple recursive terms of different size.

   We observe that the sum of $2n/3$ and $n/4$ equals $\frac{11}{12}n$, so the total cost at each level of the recursion tree drops by a factor of $\frac{11}{12}n$. In particular, the cost of the first level is cn, the cost of the second level is $\frac{11}{12}cn$, and the cost of the third level is $\left(\frac{11}{12}\right)^2 cn$. Using this observation, we guess that this will form a geometric series that does not exceed a linear growth, and assume that $T(n) \leq c_1 n$ for every $n' < n$ and some constant c_1. Upon substitution, this yields

$$T(n) \;\leq\; 2c_1 n/3 + c_1 n/4 + \alpha n \;\;\Rightarrow\;\; T(n) \;\leq\; \left(\frac{11c_1}{12} + \alpha\right) n.$$

   If we let $c_1 \geq 12\alpha$, i.e., $\alpha \leq c_1/12$, the inequality above becomes $T(n) \leq c_1 n$, which concludes the proof that $T(n) \in O(n)$. To verify that $T(n)$ is also $\Omega(n)$, we can just observe that, even if we disregard the recursive calls and look at the cost of the first level in the recursion tree, this cost is, by definition, $\Omega(n)$. We therefore conclude that $T(n) \in \Theta(n)$.

   b) Solve the following recurrence equation.

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 4T(n/2) + n^2/\log n & \text{otherwise.} \end{cases}$$

   Solution: The master theorem does not apply because when we find that $n^{\log_2 4} = n^2$, we observe that $n^2 > n^2/\log n$ but $n^{2-\epsilon} < n^2/\log n$ for any positive ϵ. We thus apply some algebra to find a solution.

$$\begin{aligned} T(n) &= 4T\left(\tfrac{n}{2}\right) + \tfrac{n^2}{\log n} \\ &= 4^2 T\left(\tfrac{n}{2^2}\right) + \tfrac{4\left(\frac{n}{2}\right)^2}{\log n/2} + \tfrac{n^2}{\log n} \\ &= 4^2 T\left(\tfrac{n}{2^2}\right) + n^2\left(\tfrac{1}{\log n/2} + \tfrac{1}{\log n}\right) \\ &= 4^3 T\left(\tfrac{n}{2^3}\right) + n^2\left(\tfrac{1}{\log n/2^2} + \tfrac{1}{\log n/2} + \tfrac{1}{\log n}\right) \\ &= ... \\ &= 4^{\log n} T(1) + n^2\left(\tfrac{1}{\log n/2^{(\log n)-1}} + ... + \tfrac{1}{\log n/2} + \tfrac{1}{\log n}\right) \end{aligned}$$

   Note that $\log n/2^{(\log n)-k} = k$. Therefore the sum in the parentheses is nothing more that the harmonic sum

$$1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{\log n/2} + \frac{1}{\log n} = \Theta(\log\log n).$$

   Thus, $T(n) = n^2 T(1) + n^2\Theta(\log\log n) = \Theta(n^2 \log\log n)$.

   c) For the following equation provide an exact (*not asymptotic*) closed form solution.

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 5T(n/2) & \text{otherwise.} \end{cases}$$

Solution: We cannot apply the master theorem since it does not provide exact solutions, only asymptotes.

Note that, unlike all the other recurrence equations that we have considered, this $T(n)$ suffers no cost between its recursive calls. Therefore, if one produced the recursion tree, there would be zero cost at all levels, except the leaves. According to the definition of this function, we know that its cost will be constant (1) for $n \leq 1$. If we assume that n is a power of 2, i.e., $n = 2^k$ for some $k \in \mathbb{N}$, then the depth of the tree would be exactly k, and the number of leaves would be 5^k, leading to $T(n) = 5^k$. If, on the other hand, n is not a power of 2, then the depth of the tree would be $\lceil \lg n \rceil$, and we would get $T(n) = 5^{\lceil \lg n \rceil}$.

4. (16 pts) Consider the following variation of Merge Sort: rather than dividing the array into two equal sized parts, recursively sorting each of them, and then merging them together, we divide the array into $\sqrt{n}$ equal sized parts instead. Once we recursively sort each one of these $\sqrt{n}$ parts of size $\sqrt{n}$ each, we need $\Theta(n \lg n)$ time in order to merge them together, leading to the following recurrence equation:

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ \sqrt{n}T(\sqrt{n}) + n \lg n & \text{otherwise.} \end{cases}$$

Solve this recurrence equation, providing tight upper and lower bounds

a) Using the *recursion tree* method.

Solution: Every node in the tree of size k has $\sqrt{k}$ children and contributes cost $k \lg k$. At depth 1, there are $n^{1/2}$ nodes with size $n^{1/2}$. Therefore, the total cost at the first level is $n^{1/2} \cdot n^{1/2} \lg n^{1/2} = \frac{1}{2} n \lg n$. Each of the $n^{1/2}$ nodes at depth 1 have $n^{1/4}$ children with size $n^{1/4}$. Therefore, at depth 2, there are $n^{3/4}$ nodes of size $n^{1/4}$, implying a total cost of $n \lg n^{1/4} = \frac{1}{4} n \lg n$. Generalizing this, we can see at depth d there are $n^{1-(1/2)^d}$ nodes with size $n^{(1/2)^d}$ which corresponds to a total cost at depth d of $\frac{1}{2^d} n \lg n$. Since the depth of the tree is finite (the depth of the tree is the number of times one must take the square root of a number to reach a constant size, which is given by $\Theta(\lg \lg n)$), the total cost in the tree is then bounded above by the geometric series: $\sum_{d=0}^{\infty} \frac{1}{2^d} n \lg n = 2n \lg n$. Therefore, we have that $T(n) = O(n \lg n)$. Since the cost at the first level is $n \lg n$, we also have $T(n) = \Omega(n \lg n)$.

b) Using the *master theorem* after applying an appropriate change of variables.

Solution: To apply the master theorem, we make the familiar change of variables $m = \lg n$. Doing so yields $T(2^m) = 2^{m/2} T\left(2^{m/2}\right) + m \cdot 2^m$. If we divide both sides with 2^m, we get

$$\frac{T(2^m)}{2^m} = \frac{T\left(2^{m/2}\right)}{2^{m/2}} + m.$$

We can now rename $S(m) = \frac{T(2^m)}{2^m}$, which reduces our initial recurrence equation to $S(m) = S(m/2) + m$, which is much easier to solve. To use the master theorem to solve $S(m)$, we must identify a, b, and $f(m)$ in our new recurrence equation. We have $a = 1$, $b = 2$ and $f(m) = m$, so $m^{\log_b a} = m^0 = 1$. It appears that case 3 applies. Selecting $\epsilon = 1/2$, we have $f(m) = m \in \Omega(m^{1/2}) = \Omega(m^{(\log_b a)+1/2})$. Further, choosing $c = 0.9$ gives $\frac{m}{2} \leq \frac{9m}{10}$ for all $m > 1$, so we have confirmed that case 3 does apply. This means that we have $S(m) \in \Theta(m)$. Since $S(m) = \frac{T(2^m)}{2^m}$, this implies that $T(2^m) = 2^m S(m) \in \Theta(m \cdot 2^m)$. Finally, since $m = \log n$, we conclude that

$$T(n) = \Theta(n \log n).$$