EECS 340: Assignment 3

Shaochen (Henry) ZHONG, `sxz517`
Yuhui ZHANG, `yxz2052`

Due and submitted on 02/20/2020
EECS 340, Dr. Koyutürk

Problem 1

(a) $T(n) = bT(n/a) + \Theta(n)$

For this recurrance, we have a comparsion between $n^{\log_a b}$ and n. Since it is given that $1 < a < b$, there must be $\log_a b > 1$. Therefore we may say that there must be a $n^\epsilon = \frac{n^{\log_a b}}{n}$ where $0 < \epsilon = \log_a b - 1$.

Now we have $f(n) = n = O(n^{(\log_a b) - \epsilon})$, where $0 < \epsilon = \log_a b - 1$, we can apply *case 1* of the master theorem and conclude that the solution is $T(n) = \Theta(n^{\log_a b})$.

(b) $T(n) = a^2T(n/a) + \Theta(n^2)$

For this recurrance, we have a comparsion between $n^{\log_a a^2}$ and n^2, thus n^2 and n^2. As now we have $f(n) = \Theta(n^2)$, we can apply *case 2* of the master theorem and conclude that the solution is $T(n) = \Theta(n \cdot \log n)$

(c) $T(n) = T(\lambda n) + n^\lambda$

We may rewrite it as $T(n) = T\left(\frac{n}{\frac{1}{\lambda}}\right) + n^\lambda$. Thus, we have a comparsion between $n^{\log_{\frac{1}{\lambda}} 1}$ and n^λ, which is equivalent as n^0 and n^λ, then we have $f(n) = \Omega(n^{\log_b a + \epsilon})$ for $\epsilon = \lambda$. We may also show $af(\frac{n}{b}) \leq cf(n)$ for $1 \cdot f\left(\frac{n}{\frac{1}{\lambda}}\right) = f(\lambda n)$. Combined, together, we can apply *case 3* of the master theorem and conclude that the solution is $T(n) = \Theta(n^\lambda)$.

(d) $T(n) = aT(\frac{n}{a}) + \Theta(n^\lambda(\log n)^b)$

For this recurrance, we have a comparsion between $n^{\log_a a}$ and $n^\lambda(\log n)^b$, which is equivalent to comparing n and $n^\lambda(\log n)^b$. We may prove that n is polynomially larger than $n^\lambda(\log n)^b$ by analyzing:

$$
\begin{aligned}
\text{W.T.S.} \quad \lim_{n\to\infty} \frac{n^{\epsilon}\cdot n^{\lambda}(\log n)^{b}}{n} &= 0 && (1)\\
\lim_{n\to\infty} \frac{n^{\lambda}(\log n)^{b}}{n^{1-\epsilon-\lambda}} &= 0\\
\Longrightarrow 1-\epsilon-\lambda > 0 \Rightarrow \epsilon &< 1-\lambda && (2)
\end{aligned}
$$

Thus, we have $f(n) = O(n^{\log_b a - \epsilon})$ for $\epsilon < 1-\lambda$. Then, we can apply *case 1* of the master theorem and conclude that the solution is $T(n) = \Theta(n)$.

Problem 2

(a) For any constant $0 < \alpha < 1$, if $T(n) = T(\alpha n) + T((1-\alpha)n) + \Theta(n)$, then $T(n) = O(n \log n)$

Guess $T(n) = O(n \log n)$

Thus there must be constants c, c' for $c, c' \in \mathbb{Z}^+$ s.t. $T(n) \le cn\log n - c'n$.

Given $T(n) = T(\alpha n) + T((1-\alpha)n) + \Theta(n)$

Proof. We may rewrite it as $T(n) = T(\alpha n) + T((1-\alpha)n) + dn$ for $d \in \mathbb{Z}^+$. Assume the claim $T(k) = ck\log k - c'k$ holds true for $T(k)$ for $k \in [1, n)$, and without loss of generality assume that $\alpha \ge 0.5$, we have:

$$
\begin{aligned}
T(k+1) &= T(\alpha(k+1)) + T((1-\alpha)(k+1)) + d(k+1) && (3)\\
&\le [c\cdot\alpha(k+1)\log(\alpha(k+1)) - c'(k+1)] +\\
&\quad [c\cdot((1-\alpha)(k+1))\log((1-\alpha)(k+1)) - c'(k+1)] + d(k+1)\\
&\le c\cdot\alpha(k+1)\log(\alpha(k+1)) + c(1-\alpha)(k+1)\log(\alpha(k+1)) + d(k+1) - 2c'(k+1)\\
&\le (c\alpha + c(k+1) - c\alpha)\cdot\log(\alpha(k+1)) + d(k+1) - 2c'(k+1)\\
&\le c(k+1)\cdot\log(\alpha(k+1)) + (d-2c')(k+1)\\
&\le c(k+1)\cdot\log(k+1) + (d-2c')(k+1)\\
\Longrightarrow T(k+1) &\le c(k+1)\cdot\log(k+1) \quad \text{for } c' = \frac{1}{2}d && (4)
\end{aligned}
$$

As now we have $T(k+1) \le c(k+1)\cdot\log(k+1)$ for $c' = \frac{1}{2}d$, we may say it is true for $T(k)$ for all $k \in [1, n)$. □

(b) For any constant $k > 0$, if $T(n) = \Theta(n) + \sum_{i=1}^{k} T(\frac{n}{2^i})$, then $T(n) = O(n)$

Guess $T(n) = O(n\log n)$

Thus, there must be a constant c for $c \in \mathbb{Z}^+$ s.t. $T(n) \le cn$.

Given $T(n) = \Theta(n) + \sum_{i=1}^{k} T(\frac{n}{2^i})$

Proof. As there must be a constant d for $d \in \mathbb{Z}^+$ s.t. $\Theta(n) \leq dn$ – and therefore causes $T(n) \leq dn + \sum_{i=1}^{k} T(\frac{n}{2^i})$ – with d for $0 < d \leq c - \sum_{i=1}^{k} c(\frac{1}{2^i})$. Now we may connect the two equations and get

$$\begin{aligned} T(n) &\leq dn + \sum_{i=1}^{k} T(\frac{n}{2^i}) \leq cn && (5) \\ &\leq dn + \sum_{i=1}^{k} c \cdot (\frac{n}{2^i}) \leq cn \end{aligned}$$

Assume the claim $T(n) \leq dn + \sum_{i=1}^{k} T(\frac{n}{2^i}) \leq cn$ holds true for $T(m)$ for $m \in [1, n)$, consider:

$$\begin{aligned} T(m+1) &\leq d(m+1) + \sum_{i=1}^{k} T(\frac{m+1}{2^i}) && (6) \\ &\leq d(m+1) + \sum_{i=1}^{k} c \cdot (\frac{m+1}{2^i}) \\ &\leq \underbrace{dm + \sum_{i=1}^{k} c \cdot (\frac{m}{2^i})}_{=T(m) \leq cm} + \underbrace{d + \sum_{i=1}^{k} c \cdot (\frac{1}{2^i})}_{=T(1) \leq c(1)} && (7) \\ \implies T(m+1) &\leq c(m+1) && (8) \end{aligned}$$

As now we have $T(m+1) \leq c(m+1)$ for $c \in \mathbb{Z}^+$, we may say it is true for $T(m)$ for all $m \in [1, n)$. □

Problem 3

Problem 4

(a)

Algorithm 1 QuickMiss(C, D, p, r, missLeft)

```
procedure QuickMiss(C, D, p, r, missLeft)
    if p < r then
        q ← Partition(C, p, r)
        if C[q] == D[q] then
            return QuickMiss(C, D, q + 1, r, False)
        else
            return QuickMiss(C, D, p, q − 1, True)
    if missLeft == True then
        return D[p]
    else
        return D[r + 1]
```

Algorithm 2 Partition(A, p, r)

```
procedure Partition(A, p, r)
    q ← p
    for i ← p to r do
        if Compare-Strings(A[i], A[r]) then
            Swap(A[i], A[q])                    ▷ Exchange the two elements.
            q ← q + 1
    Swap(A[q], A[r])
    return q
```

(b)

It is known that QuickSort has an average runtime of $O(n \log n)$ due to the fact that it is considered *case 2* of the master theorem: as it has n nodes on each level and with a depth of $\log n$, thus $O(n \log n)$. Our QuickMiss algorithm has a runtime of $O(n)$ due to it only calls Partition on either left portion or right portion to the `pivot`, but never both. Thus, on each level it will always has less than n nodes, therefore forming a *case 3* of the master theorem. The master theorem suggest, in *case 3* the roots dominate the leaves and therefore determine the runtime of the algorithm – as QuickSort has n roots at the beginning, we may conclude it has a runtime of $\Theta(n)$.