***Problem*** 1. (30%)

`http://www.csie.ntu.edu.tw/~hsinmu/courses/lib/exe/fetch.php?media=ada_13fall:hw1_problem1.zip`

***Problem*** 2. (16%)

a. (2%) $T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$

Use the substitution method to prove that $T(n) = O(n \log n)$

---

Assume that $T(n) \le cn \log n,\ \forall \left\lfloor \frac{k}{2} \right\rfloor \le n < k$

$$\begin{aligned}
T(k) &= 2T\left(\left\lfloor \frac{k}{2} \right\rfloor\right) + k \\
&\le 2 \cdot \left(c\left\lfloor \frac{k}{2} \right\rfloor \log \left\lfloor \frac{k}{2} \right\rfloor\right) + k \qquad by\ T\left(\left\lfloor \frac{k}{2} \right\rfloor\right) \le c\left\lfloor \frac{k}{2} \right\rfloor \log \left\lfloor \frac{k}{2} \right\rfloor \\
&\le 2 \cdot \left(c\frac{k}{2} \log \frac{k}{2}\right) + k \qquad by\ \left\lfloor \frac{k}{2} \right\rfloor \le \frac{k}{2}\ and\ \log \left\lfloor \frac{k}{2} \right\rfloor \le \log \frac{k}{2} \\
&= ck \log \frac{k}{2} + k \\
&= ck \log k - ck \log 2 + k \\
&= ck \log k - ck + k \\
&\le ck \log k,\ as\ long\ as\ c \ge 1
\end{aligned}$$

$T(2) = 2T(1) + 2 = 4 \le c \cdot 2 \log 2,\ if\ c \ge 2$

$T(3) = 2T(1) + 3 = 5 \le c \cdot 3 \log 3,\ if\ c \ge 2$

By induction proof, $T(n) \le cn \log n,\ for\ c \ge 2,\ n \ge 2.$

Therefore $T(n) = O(n \log n)$.

---

**Common Mistake 1. 忘記檢查邊界條件**

Substitution method 就像高中學的數學歸納法一樣，

原理是

①先證出「若小於 n 的 case 皆成立，則等於 n 的 case 必成立」

②證明小於 n 的 case 皆會成立 (在高中時，通常是 n=1 就能成立)

而事實上，不用「所有」小於 n 的 case 都要成立

以這題來說，在證明①時，用到的是 $T\left(\left\lfloor \frac{k}{2} \right\rfloor\right) \le c\left\lfloor \frac{k}{2} \right\rfloor \log \left\lfloor \frac{k}{2} \right\rfloor$

因此，只要在 $\left\lfloor \frac{k}{2} \right\rfloor$ 時有成立，那麼 by①, 在 $k$ 時就會成立

由以上可知，第②部分的 boundary condition check 是證明的一部份。若缺少②，則證明的正確性就無法確立。所以沒有 check boundary condition 的人都無法滿分。

**Common Mistake 2. 只檢查 T(2), 沒檢查 T(3)**

必須要讓 $\forall k \geq n_0, T\left(\left\lfloor\frac{k}{2}\right\rfloor\right) \leq c\left\lfloor\frac{k}{2}\right\rfloor \log\left\lfloor\frac{k}{2}\right\rfloor$ 的假設都要能成立

因為 $T\left(\left\lfloor\frac{3}{2}\right\rfloor\right)$ 時，也就是 T(1) 時，假設並不成立

所以 T(3) 不能從①來證明，必須挑出來，以②的方式檢查

而 T(4) 時，$T\left(\left\lfloor\frac{4}{2}\right\rfloor\right) = T(2)$ 是成立的，所以 T(4) 能夠被①證

接著，所有 $k \geq 4$ 的 case 都可以從①得證了，因為 $\forall k >= 4, \left\lfloor\frac{k}{2}\right\rfloor$ 的 case 都一定是對的

---

**Common Mistake 3. 「設 T(n)=O(nlogn)」**

應該明確提出「假設 T(n)≤...」，因為這句才是 induction proof 的關鍵假設，只「假設 $T(n) = O(nlogn)$」是無法作 induction proof 中的推導的

並且「假設 T(n)≤...」之後，最後要記得證出「T(n)≤...」，兩者要吻合。有不少人會寫錯其中一邊，這樣是不對的。

b. (2%) $T(n) = T\left(\left\lceil\frac{n}{2}\right\rceil\right) + T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + 1$

Use the substitution method to prove that $T(n) = O(n)$

---

Assume that $T(n) = cn - 1,\ \forall \left\lfloor\frac{k}{2}\right\rfloor \leq n < k$

Then $T(k) = T\left(\left\lceil\frac{k}{2}\right\rceil\right) + T\left(\left\lfloor\frac{k}{2}\right\rfloor\right) + k$

$= \left(c\left\lceil\frac{k}{2}\right\rceil - 1\right) + \left(c\left\lfloor\frac{k}{2}\right\rfloor - 1\right) + k$

$= ck - 2 + k$

$\leq ck - 1,\ as\ long\ as\ k \leq 1$

$T(1) = 1 \leq c \cdot 1 - 1,\ as\ long\ as\ c \geq 2$

By induction proof, $T(n) \leq cn - 1,\ for\ c \geq 2,\ n \geq 1.$

Therefore $T(n) = O(cn - 1) = O(cn) = O(n).$

---

**Common Mistake. 只有檢查 T(2) 沒檢查 T(3)**

有兩種可行的檢查 boundary cases 的方式

① T(2) 和 T(3) 皆檢查

② 只檢查 T(1)

詳細說明請見 **problem a 的 Common Mistake 2.**

c. (2%) $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + \Theta(n)$

Draw a recursion tree, and use it to prove that $T(n) = \Omega(nlgn)$

---

(Please draw a recursion tree.)

The height of the tree $\geq \log_3 n$.

The sum of every layer of the tree $= \Theta(n) = cn,\ c$ is a constant .

$T(n) \geq cn \times \log_3 n = \Omega(nlgn)$

---

**Common Mistake.**

**用 induction proof 證明，認為無法以 recusion tree 算出的結果證明**

只要以 recursion tree 得出的結果是精準的 bound，那麼就可以直接證明；但如果 recursion tree 只是寬鬆的大概算的話，就不能用其直接證明。

可參考課本第 88 頁：「If you are very careful when drawing out a recursion tree and summing the costs, however, you can use a recursion tree as a direct proof of a solution to a recurrence.」

事實上 Master Theorem 也是由 recusion tree 的方法證明的，有興趣的同學可參考課本該章第 6 小節。

這題只要有「畫出 recursion tree」、給出「$\geq cn \times \log_3 n$」接著「$\Omega(nlgn)$」，就能拿滿分；反之，則扣 1 分。

由於時間有限，這題寫 induction proof 的人，induction 的那段都完全沒改；如果有很想要請助教過目的可以寄信給助教。

d. (3%) $T(n) = 3T\left(n^{\frac{1}{3}}\right) + \log n$

---

$Let\ m = \log n$

$\Rightarrow 2^m = n$

$\Rightarrow T\left(2^m\right) = 3T\left(2^{\frac{m}{3}}\right) + m$

$Let\ S(m) = T\left(2^m\right)$

$\Rightarrow S(m) = 3S\left(\frac{m}{3}\right) + m$

$a = 3 \geq 1,\ b = 3 > 1,\ f(m) = m$

$m^{\log_3 3} = m^1 = m$

$f(m) = m = \Theta(m)$

`By case 2 of the master theorem,` $S(m) = \Theta(m^{\log_b a} \log m) = \Theta(m \log m)$

$S(m) = T\left(2^m\right) = T(n) = \Theta(m \log m) = \Theta(\log n \log\log n)$

e. (2%) $T(n) = 7T\left(\frac{n}{4}\right) + \Theta(n^2)$

---

$a = 7 \geq 1,\ b = 4 > 1,\ f(n) = \Theta(n^2)$
$n^{\log_b a} = n^{\log_4 7} \approx n^{1.4}$
$f(n) = \Theta(n^2) = \Omega(n^{1.4+\epsilon}) = \Omega(n^{\log_b a+\epsilon})$, where $\epsilon = 0.1$ (or any $0 \leq \epsilon < 2 - \log_4 7$ will do)
$af\left(\frac{n}{b}\right) - cf(n) = 7f\left(\frac{n}{4}\right) - cn^2 = 7\left(\frac{n^2}{16}\right) - cn^2 = \left(\frac{7}{16} - c\right)n^2 \leq 0$
, for $\frac{7}{16} \leq c < 1$ and all sufficiently large $n$
By case 3 of the master theorem, $T(n) = \Theta(f(n)) = \Theta(n^2)$

f. (3%) $T(n) = 2T\left(\frac{n}{2}\right) + n\log_2 n$

---

$$
\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + n\log_2 n \\
&= 2[2T\left(\frac{n}{4}\right) + \frac{n}{2}\log_2\frac{n}{2}] + n\log_2 n \\
&= 4T\left(\frac{n}{4}\right) + n\log_2\frac{n}{2} + n\log_2 n \\
&= 4T\left(\frac{n}{4}\right) + 2n\log_2 n - n\log_2 2 \\
&= 4[2T\left(\frac{n}{8}\right) + \frac{n}{4}\log_2\frac{n}{4}] + 2n\log_2 n - n \\
&= 8T\left(\frac{n}{8}\right) + 3n\log_2 n - 3n \\
&= nT\left(\frac{n}{n}\right) + (\log_2 n)\cdot n\cdot(\log_2 n) - \frac{(1+(\log_2 n - 1))(\log_2 n - 1)}{2}\cdot n \\
&= n + n{\log_2}^2 n - \frac{n{\log_2}^2 n}{2} + \frac{n\log_2 n}{2} \\
&= \Theta(n\log^2 n)
\end{aligned}
$$

---

$a = 2 \geq 1,\ b = 2 > 1,\ f(n) = n\log_2 n$
$n^{\log_b a} = n^{\log_2 2} = n$
$f(n) = n\log_2 n = \Omega(n^1) = \Omega(n^{\log_b a})$
But $f(n) = n\log_2 n \neq \Omega(n^{1+\epsilon})$ :
The ratio $\frac{f(n)}{n^{\log_b a}} = \frac{n\log_2 n}{n} = \log_2 n$ is asymptotically less than $n^\epsilon$ for any constant $\epsilon$
Therefore, master theorem can't be applied in this problem.
(note: $f(n) \neq O(n^{\log_b a-\epsilon}),\ f(n) \neq \Theta(n^{\log_b a}),\ f(n) \neq \Omega(n^{\log_b a+\epsilon})$ )
For detailed description, you can refer to page 95 of the text book.

g. (2%) $T(n) = 2T(n-1) + 1$

---

$$\begin{aligned}
&T(n) = 2T(n-1) + 1 \\
&= 2[2T(n-2) + 1] + 1 = 4T(n-2) + 1 + 2 \\
&= 4[2T(n-3) + 1] + 1 + 2 = 8T(n-3) + 1 + 2 + 4 \\
&= 2^{n-1}T(n-(n-1)) + \frac{1(2^{n-1}-1)}{2-1} \\
&= 2^{n-1} + 2^{n-1} - 1 = 2^n - 1 = \Theta(2^n)
\end{aligned}$$

評分標準：

1. 該小題必須全對才能得滿分，除非可明顯判斷為抄寫時的筆誤，且不影響答案及推導過程，那就不會扣分。
2. 只要有寫一定程度的計算過程，不論過程及答案的對錯程度，都會給基本分 1 分。
3. 多寫了不必要的過程，不會扣分。

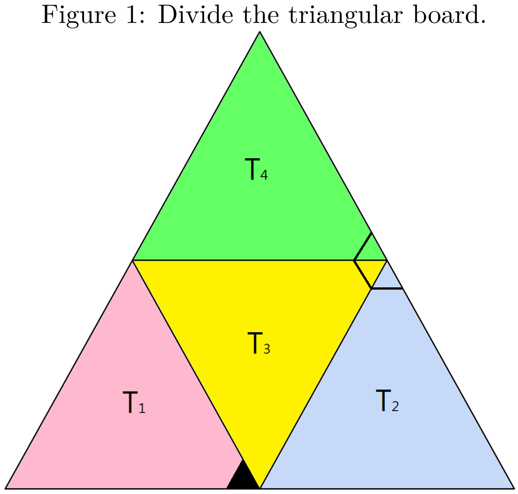
(a) 只要「證明」+「兩個 boundary 都有 check」就可以兩分
常數範圍算錯不扣分 (因為有些人沒寫出常數範圍)

(f)
- * 使用大師定理的 - 得 1 分
- * 把 T(n) 算出來的
  若算錯 - 得 2 分
  算對 - 得 3 分
- * 用 induction proof 的
  連一邊都沒 prove 出來 - 得 1 分
  兩邊都有嘗試 prove，整體觀念還 OK，只錯小地方的 - 得 2 分
  全班只有一人以 induction proof 完美證出這題

(g) 必須把 T(n) 算對 ($T(n) = 2^n - 1$)
並且寫出 $T(n) = \Theta(2^n)$

(d)-(g) 題目要求給 tight bound，即 Θ(...)，如果寫 big-O 的都會扣 1 分

Figure 1: Divide the triangular board.




***Problem*** 3. (15%)

We can divide the input triangular board $T$ into four half-length triangles, $T = \{T_1, T_2, T_3, T_4\}$. First, we find out the $T_t$ which the removed tile locates in. Next we put an trapezoid piece on the joint of other three half-length triangles. Then four half-length triangulars all contain a removed tile on the edge (see Figure **??**). Thus, we can divide the original problem into four subproblems with half-length. Suppose $n$ is the length of the board. The recurrence form of the algorithm is $T(n) = 4T(\frac{n}{2}) + O(1)$. Therefore, $T(n) = \Theta(n^2)$ by Master theorem.

***Problem*** 4. (15%)

The following pseudo code implements a functoin that computes the $kth$ smallest element in the union of two sorted lists with size $m$ and $n$.

```
int find(list M, list N, k)
    if m == 0 return N[k]
    if n == 0 return M[k]
    if k == 1 return min(M[1], N[1])
    midM = ⌈m/2⌉ midN = ⌈n/2⌉
    case k > midM+midN
// the midNth in N < the (midM+midN)th in M+N < the kth in M+N
// so the elements <= the midNth in N would not be the kth in M+N
        if M[midM] >= N[midN]
            return find(M, N.subList(midN+1, n), k-midN)
        else
            return find(M.subList(midM+1, m), N, k-midM)
    case k < midM+midN
// the midMth in M >= the (midM+midN)th in M+N > the kth in M+N
// so the elements >= the midMth in M would not be the kth in M+N
        if M[midM] >= N[midN]
            return find(M.subList(1, midM-1), N, k)
        else
            return find(M, N.subList(1, midN-1), k)
    case k == midM+midN
// the midNth in N < the (midM+midN)th in M+N == the kth in M+N
// so the elements <= the midNth in N would not be the kth in M+N
// the midMth in M >= the (midM+midN)th in M+N == the kth in M+N
// so the elements > the midMth in M would not be the kth in M+N
        if M[midM] >= N[midN]
            return find(M.sub(1, midM), N.sub(midN+1, n), k-midN)
        else
            return find(M.sub(midM+1, m), N.sub(1, midN), k-midM)
```

To compute the Time Complexity of the algorithm:

$$T(m+n) = \begin{cases} case1: T(m+\frac{n}{2}) + O(1), \begin{cases} \text{if } k > \lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil \text{ and } M[\lceil \frac{m}{2} \rceil] \geq N[\lceil \frac{n}{2} \rceil] \\ \text{if } k < \lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil \text{ and } M[\lceil \frac{m}{2} \rceil] < N[\lceil \frac{n}{2} \rceil] \end{cases} \\ \\ case2: T(\frac{m}{2}+n) + O(1), \begin{cases} \text{if } k > \lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil \text{ and } M[\lceil \frac{m}{2} \rceil] < N[\lceil \frac{n}{2} \rceil] \\ \text{if } k < \lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil \text{ and } M[\lceil \frac{m}{2} \rceil] \geq N[\lceil \frac{n}{2} \rceil] \end{cases} \\ \\ case3: T(\frac{m}{2}+\frac{n}{2}) + O(1), \text{ if } k == \lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil \\ \\ case4: O(1), \text{ if } m = 0 \text{ or } n = 0 \text{ or } k = 1 \end{cases}$$

From the above Analysis, we can find that if the program go through in only one case and finally reach case 4, then the running time would be $O(\log m)$ or $O(\log n)$. But if the program go through not in only one case, but switch to another case several times until it's reach case 4, then the running time would be $O(\log m + \log n)$.

***Problem*** 5. (24%)

1.

```
multiply(value1, value2)
    length1 = number of digits of value1
    length2 = number of digits of value2
    if length1 < 5 && length2 < 5
        // It's small enough that we can do
        // arithmetic operations in constant time.
        return value1 * value2
    k = max(length1, length2) / 2 + max(length1, length2) % 2
    add (2*k - length1) '0's to value1
    add (2*k - length2) '0's to value2
    value1_high = first k digits of value1
    value1_low = last k digits of value1
    value2_high = first k digits of value2
    value2_low = last k digits of value2
    z1 = multiply(value1_high, value2_high)
    z2 = multiply(value1_high + value1_low,
                  value2_high + value2_low)
    z3 = multiply(value1_low, value2_low)
    z2 = z2 - z1 - z3;
    add 2*k trailing '0's to z1
    add k trailing '0's to z2
    return z1+z2+z3
```

$T(n) = 3T(\frac{n}{2}) + O(n)$.

By Master Theorem, $O(n) = O(n^{\log_2 3-\epsilon})$ where $\epsilon = 0.1$, the procedure can run in $O(n^{log_2 3})$ time.

2.

```
power_of_ten_to_binary(n)
    if n == 0
        return "1"
    if n == 1
        return "1010"
    z = power_of_ten_to_binary((int)(n/2))
    z = multiply(z, z)
    if n % 2 == 1
        z = multiply(z, "1010")
    return z
```

$T(n) = T(\frac{n}{2}) + O(n^{\log_2 3})$.

By Master Theorem, $O(n^{\log_2 3}) = \Omega(n^{\log_2 1+\epsilon})$ where $\epsilon = 0.1$ and $(\frac{n}{2})^{\log_2 3} < cn^{\log_2 3}$ where $c = 2^{-0.1}$, the procedure can run in $O(n^{\log_2 3})$ time.

3.

```
decimal_to_binary(x)
    if x < 10
        a[0...9] = {"0", "1", "10", "11", "100",
                    "101", "110", "111", "1000", "1001"}
        return a[x]
    n = number of digits of x
    k = (int)(n / 2) + n % 2
    high = (int)(x / (10 ^ k))
    low = x % (10 ^ k)
    high_binary = decimal_to_binary(high)
    low_binary = decimal_to_binary(low)
    ten_of_k = power_of_ten_to_binary(k)
    return multiply(high_binary, ten_of_k) + low_binary
```

$T(n) = 2T(\frac{n}{2}) + O(n^{\log_2 3})$.

By Master Theorem, $O(n^{\log_2 3}) = \Omega(n^{\log_2 2+\epsilon})$ where $\epsilon = 0.1$ and $2(\frac{n}{2})^{\log_2 3} < cn^{\log_2 3}$ where $c = 2^{-0.1}$, the procedure can run in $O(n^{\log_2 3})$ time.