1 (Huma Shiekh)

Huma Sheikh,

(1) $T(n/2) + T(n/2) + \Theta(1)$

$= 2T(n/2) + \Theta(1)$

$= 2T(n/2) + c$

$a = 2 \quad b = 2$

$\log_b a \Rightarrow \log_2 2 = 1$

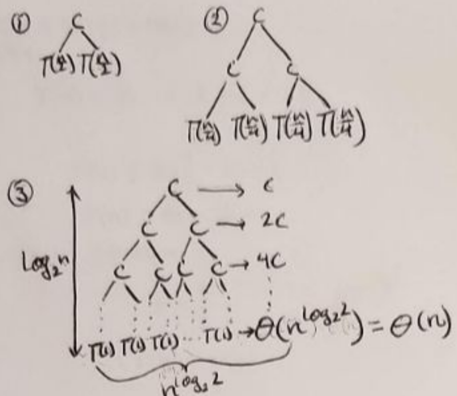
Compare: $n^{\log_b a}$ vs. $f(n)$; $f(n) = n^0$

$n^1 \qquad n^0$ ← case 1

f(n) is smaller polynomially.

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

2 Using Recursion Tree



From the tree we can see the following:

$$T(n) = 2^0c + 2^1c + \cdots + 2^{\log_2 n - 1}c + \Theta(n^{\log_2 2})$$

$$T(n) = c(2^0 + 2^1 + \cdots + 2^{\log_2 n - 1}) + \Theta(n)$$

$$T(n) = c\sum_{i=0}^{\log_2 n - 1} 2^i + \Theta(n)$$

$$T(n) = c(\frac{2^{\log_2 n} - 1}{2 - 1}) + n$$

$$T(n) = c(2^{\log_2 n} - 1) + n$$

$$T(n) = c(n - 1) + n$$

$$T(n) = n + cn - c$$

$$T(n) = n(c + 1) - c$$

Because $c > 0$, for instance, we can let $c = 1$ and show that:

$$T(n) = n(2 + 1) - c$$

$$T(n) = n(2 + 1) - 1$$

$$T(n) = 2n - 1$$

We can drop the constants,

$$\therefore \Theta(n)$$

2 (Daniel Santos)

3 (Guangxin Ye)

3. proof with substitution

(1) upper bound $T(n) \le 2T(\frac{n}{2}) + c$ (c is positive constant)

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

$T(n) \le dn - e$ (d, e are positive constants)

$T(\frac{n}{2}) \le d\frac{n}{2} - e$

$$T(n) \le 2 \cdot (\frac{dn}{2} - e) + c$$
$$= dn - 2e + c = dn - e - e + c$$
$$\le dn - e \text{ if } c - e \le 0 \Rightarrow c \le e$$

$\therefore T(n) = O(n)$

(2) lower bound $T(n) \ge 2T(\frac{n}{2}) + c$

Guess $T(n) = \Omega(n)$

$T(n) \ge dn$

$T(\frac{n}{2}) \ge d\frac{n}{2}$

$$T(n) \ge 2 \cdot \frac{dn}{2} + c$$
$$= dn + c$$
$$\ge dn$$

$\therefore T(n) = \Omega(n)$

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

4 Huma Sheikh

(4) four sub problems ⇒ which means $a = 4$

of size $n/3 = n/b$

$f(n) = 2^{\lg n}$

$T(n) = 4(n/3) + 2^{\lg n}$

$a = 4 \quad b = 3, \quad f(n) = 2^{\lg n} \Rightarrow n$

$\log_b a = \log_3 4 = \frac{\log 4}{\log 3} \approx 1.26$

compare $n^{\log_b a}$ vs. $f(n)$

$\log_3 4 \approx 1.26$ vs. n^1

case 1, since $f(n)$ is smaller polynomially.

$T(n) = \Theta(n^{\log_3 4}) \approx \boxed{\Theta(n^{1.26})}$

5 (Kyle Dokus)

5.1 $T(n) = 4T(\frac{n}{4}) + n$

$\log_b a = \log_4 4 = 1$

Compare n vs n

We have case 2, so $T(n) = \Theta(n\lg n)$

5.2 $T(n) = 3T(\frac{n}{2}) + \sqrt{10}n^2$

$\log_b a = \log_2 3 \approx 1.58$

Compare $n^{\log_2 3}$ vs $\sqrt{10}n^2$

We have case 3: $\sqrt{10}n^2 = \Omega(n^{\log_2 3})$, So, $T(n) = \Theta(n^2)$

5.3 $T(n) = T(n-1) + 10$

Cannot use master method with this type of recurrence.

5.4 $T(n) = 3T(\frac{3n}{2}) + n$

Cannot have b less than 1, here it is 2/3, which means each recursive call is on a larger problem set. Cannot use master method here.

5.5 $T(n) = 2nT(\frac{n}{3}) + n$

Cannot have a equal to a function of n, must be a constant greater than or equal to 1. Cannot use master method here.

6 a) Saara Luna

1) Pseudocode:

```
FIND_MODE(A, n)
1)   if n == 1
2)       return 1
3)   if A[⌊n/2⌋] > A[⌊n/2⌋+1]
4)       index = FIND_MODE(A[1..⌊n/2⌋], ⌊n/2⌋)
5)   else
6)       index = FIND_MODE(A[⌊n/2⌋+1..n], n-⌊n/2⌋) + ⌊n/2⌋
7)   return index
```

2) Find a recurrence for the running time of the algorithm:

Line	Cost	# executions (maximum)
1	c_1	1
2	c_2	1
3	c_3	1
4	$T(\frac{n}{2})$	1
5	c_4	1
6	$T(\frac{n}{2})$	1
7	c_5	1

Only one of these lines can be executed in the function, so the total from these two lines is $T(\frac{n}{2})$.

$T(n) = c_1 + c_2 + c_3 + T(\frac{n}{2}) + c_4 + c_5$

$\boxed{T(n) = T(\frac{n}{2}) + \theta(1)}$

6- Part b (Buhrle, Etienne) Recursion Tree

For the running time, we get

$$T_{Mode}(n) = \begin{cases} c_1 + c_2 & n = 1 \\ c_1 + c_3 + c_4 + T(n/2) & n > 1 \end{cases} = \begin{cases} \Theta(1) & n = 1 \\ T(n/2) + \Theta(1) & n > 1 \end{cases}$$

Each call generates only one recursive call, so the tree looks like in figure 3, with layers $i = 0...l$ and cost c per layer. For the deepest level, we have (according to the base case of the implementation) $n/2^l = 1 \Rightarrow l = \lg n$, making the total running time $T_{Mode}(n) = c \cdot (\lg n + 1) = \Theta(\lg n)$.

c
|
c
|
c
⋮
c

Figure 3: Recursion tree for the mode algorithm

6 b(Master Method), c (Kyle Dokus)

So $T(n) = T(n/2) + c$. There is no $2T(n/2)$ because only one of the recurrence calls is made, since it is an if else statement. By the master method we compare $n^{\log_2 1}$ vs c, we have: $c = \Theta(1)$ which is case 2 so: $T(n) = \Theta(lgn)$

Proof with substitution:

Upper Bound:

$T(n) = T(n/2) + c$ where c > 0

Use $T(n) = O(lgn)$, $T(n) \le dlgn$ where d > 0.

$T(n/2) \le dlg(n/2) = d(lgn - lg2) = dlgn - d$

$T(n) \le dlgn - d + c$

So, $T(n) \le dlgn$ if $c \le d$, which shows $T(n) = O(lgn)$

Lower Bound:

$T(n) = T(n/2) + c$ where c > 0

Use $T(n) = \Omega(lgn)$, $T(n) \ge dlgn$ where d > 0.

$T(n/2) \ge dlg(n/2) = d(lgn - lg2) = dlgn - d$

$T(n) \ge dlgn - d + c$

So, $T(n) \ge dlgn$ if $c \ge d$, which shows $T(n) = \Omega(lgn)$

Therefore $T(n) = \Theta(lgn)$

6 b, c Guangxin Ye

(2) $T(n) = T(\frac{n}{2}) + c = T(\frac{n}{2}) + \Theta(1)$

Since $T(n) = c_1 + c_2 + T(\frac{n}{2}) + c_3 = T(\frac{n}{2}) + c$ in the code

In this recurrence, $a=1$ $b=2$ $\log_b a = \log_2 1 = 0$

ompare $n^{\log_b a}$ with $f(n)$ $\Rightarrow$ $n^0 = 1$

$\therefore$ by case 2 in the Master Theorem

$T(n) = \Theta(1 \cdot lgn) = \Theta(lgn)$

(3) upper bound $T(n) \le T(\frac{n}{2}) + c$

Guess $T(n) \le dlgn$ $T(n) = O(lgn)$

$T(\frac{n}{2}) \le dlg\frac{n}{2}$

$T(n) \le d(lgn - lg2) + c = dlgn - d + c$

$\le dlgn$ if $c - d \le 0 \Rightarrow c \le d$

$\therefore T(n) = O(lgn)$

lower bound $T(n) \ge T(\frac{n}{2}) + c$

Guess $T(n) \ge dlgn$ $T(n) = \Omega(lgn)$

$T(\frac{n}{2}) \ge dlg\frac{n}{2}$

$T(n) \ge d(lgn - lg2) + c = dlgn - d + c$

$\ge dlgn$ if $c - d \ge 0 \Rightarrow c \ge d$

$\therefore T(n) = \Omega(lgn)$

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