Solution 2.1 Recurrence Relations.

We conjecture that the closed form of the recurrence relation is the following series:

\[ T(n) = a^\log_b(n) + c n \sum_{i=0}^{\log_b(n)-1} \left( \frac{a}{b} \right)^i + d \sum_{i=0}^{\log_b(n)-1} a^i. \]  

(1)

We distinguish three cases:

Case 1: \( a \neq b, a \neq 1 \). In this case, our conjecture becomes

\[ T(n) = a^{\log_b(n)} + c n \cdot \left( \frac{a}{b} \right)^{\log_b(n)-1} - 1 \cdot \frac{a^{\log_b(n)-1}}{a-1} + d \cdot \frac{a^{\log_b(n)-1} - 1}{a-1}. \]  

(2)

Base step: The conjecture is true for \( n = 1 \), because \( T(1) = a^0 e + c \cdot 0 + d \cdot 0 = e \).

Inductive hypothesis: We assume our conjecture to be true for \( T(n/b) \), so (we have \( \log_b(n/b) = \log_b(n) - 1 \))

\[ T(n/b) = a^{\log_b(n)-1} e + c n \cdot \left( \frac{a}{b} \right)^{\log_b(n)-1} - 1 \cdot \frac{a^{\log_b(n)-1}}{a-1} + d \cdot \frac{a^{\log_b(n)-1} - 1}{a-1}. \]  

(3)

Inductive step: By using the inductive hypothesis we show that the conjecture holds for \( T(n) \):

\[ T(n) = aT(n/b) + cn + d \]  

Ind.hyp.

(4)

\[ = a \left( a^{\log_b(n)-1} e + c n \cdot \left( \frac{a}{b} \right)^{\log_b(n)-1} - 1 \cdot \frac{a^{\log_b(n)-1}}{a-1} \right) + cn + d \]  

(5)

\[ = a^{\log_b(n)} e + c n \cdot \left( \frac{a}{b} \right)^{\log_b(n)-1} - 1 \cdot \frac{a^{\log_b(n)-1}}{a-1} + cn \cdot \frac{a - 1}{a - 1} + d \cdot \frac{a^{\log_b(n)-1} - a}{a-1} + d \cdot \frac{a - 1}{a-1} \]  

(6)

\[ = a^{\log_b(n)} e + c n \cdot \left( \frac{a}{b} \right)^{\log_b(n)-1} - 1 \cdot \frac{a^{\log_b(n)-1}}{a-1} + d \cdot \frac{a^{\log_b(n)} - 1}{a-1}. \]  

(7)

Case 2: \( a \neq b, a = 1 \). In this case, our conjecture becomes

\[ T(n) = e + c n \cdot \left( \frac{1}{b} \right)^{\log_b(n)-1} - 1 \cdot \frac{1}{b-1} + d \log_b(n) = d \log_b(n) + cb \cdot \frac{1-n}{1-b} + e. \]  

(8)

Base step: The conjecture is true for \( n = 1 \), because \( T(1) = d \cdot \log_b(1) + c b \cdot \frac{1}{1-b} + e = 0 + 0 + 1 \).

Inductive hypothesis: We assume our conjecture to be true for \( T(n/b) \), so

\[ T(n/b) = d \log_b(n/b) + cb \cdot \frac{1-n}{1-b} + e. \]  

(9)
Inductive hypothesis: By using the inductive hypothesis we show that the conjecture holds for $T(n)$:

$$T(n) = T(n/b) + cn + d$$

Ind. hyp.

$$= d \log_b(n/b) + cb \cdot \frac{1 - \frac{n}{b}}{1 - \frac{1}{b}} + e + cn + d \log_b(b)$$

$$= d \log_b(n) + c \cdot \frac{b - n}{1 - b} + cn + e$$

$$= d \log_b(n) + c \cdot \frac{b - n + nb}{1 - b} + e$$

$$= d \log_b(n) + cb \cdot \frac{1 - n}{1 - b} + e.$$  (14)

Case 3: $a = b, (a \neq 1)$. In this case, our conjecture becomes

$$T(n) = ne + cn \log_b(n) + d \cdot \frac{n - 1}{a - 1}.$$  (15)

Base step: The conjecture is true for $n = 1$, because $T(1) = 1 \cdot e + c \cdot 1 \cdot 0 + d \cdot \frac{1 - 1}{a - 1} = e$.

Inductive hypothesis: We assume our conjecture to be true for $T(n/b)$, so

$$T(n/b) = \frac{n}{b} \cdot e + c \cdot \frac{n}{b} \cdot \log_b(n/b) + d \cdot \frac{n - 1}{a - 1}.$$  (16)

Inductive step: By using the inductive hypothesis we show that the conjecture holds for $T(n)$:

$$T(n) = aT(n/b) + cn + d$$

Ind. hyp.

$$= a \left( \frac{n}{b} \cdot e + c \cdot \frac{n}{b} \cdot \log_b(n/b) + d \cdot \frac{n - 1}{a - 1} \right) + cn \log_b(b) + d$$

$$= \frac{a}{b} \cdot ne + \frac{a}{b} \cdot cn \log_b(n) - \frac{a}{b} \cdot cn + cn + d \cdot \frac{a}{b} \cdot n - a + a - 1 \cdot \frac{a - 1}{a - 1}$$

$$= ne + cn \log_b(n) + d \cdot \frac{n - 1}{a - 1}.$$  (20)

Solution 2.2  \textit{Estimating asymptotic Running Time.}

a) To obtain an upper bound, we overestimate the running time. The inner loop has at most $\lceil n/2 \rceil$ iterations, the outer one at most $n$. Therefore the overall running time is at most $O(n^2)$. Now we underestimate the running time: the inner loop has at least $\lfloor n/8 \rfloor$ iterations, the outer one at least $\lfloor n/10 \rfloor$. Therefore the running time has a lower bound of $\Omega(n^2)$, thus it is in $\Theta(n^2)$.

b) The loop in the steps 2–3 is repeated exactly $\lceil \sqrt{n} \rceil$ times, the loop in the steps 4–5 exactly $\lfloor \log_2(n) \rfloor$ times. Since $\lceil \sqrt{n} \rceil \geq \lfloor \log_2(n) \rfloor$ for every $n \in \mathbb{N}$, the overall number of iterations in the steps 2–5 is at least $\sqrt{n}$ and at most $2\sqrt{n}$. Therefore the running time in these steps is in $\Theta(n \sqrt{n})$. Since the outer loop is repeated exactly $n$ times, the overall running time is in $\Theta(n \sqrt{n})$.

c) Without the recursive calls in step 3 the running time is constant, i.e. it is upper bounded by a constant $C$. If we consider also recursive calls, the overall running time is at most

$$T(n) = C + 2T(n/2), T(1) = C.$$  (21)

Setting $a = b = 2$, $c = 0$ and $d = e = C$, we can solve the recurrent relation using exercise 2.1, and we obtain $T(n) = C(2n - 1) \in \Theta(n)$. 

2
Solution 2.3  Cost Models.

a) We first observe that before the $k$-th iteration of the loop the value of $x$ is exactly $2^{2^k-1}$. We prove this by induction over $k$.

*Base step:* The statement is true for $k = 1$ since the value of $x$ before the first iteration is $2 = 2^1 = 2^0 = 2^{2^1-1}$.

*Inductive hypothesis:* Let $x = 2^{2^k-1}$ before the $k$-th iteration.

*Inductive step:* By inductive hypothesis we assume that $x = 2^{2^k-1}$ before the $k$-th iteration. After the multiplication we obtain $x = 2^{2^k-1} \cdot 2^{2^k-1} = 2^{2^k}$, thus, before the $(k+1)$-th iteration the value of $x$ is $2^{2^{(k+1)-1}}$.

Now we know that before the $n$-th (and last) iteration the value of $x$ is $2^{2^{n-1}}$. The multiplication finally computes $x = 2^{2^{n-1}} \cdot 2^{2^{n-1}} = 2^{2^n}$.

b) Since the loop is iterated $n$ times, we have $2n + 2$ assignments, $n$ multiplications, $n$ additions and $n + 1$ comparisons (namely “$k \leq n$” before each iteration, and one before the termination). Therefore the uniform costs are $(2n + 2) + n + n + (n + 1) = 5n + 3$, i.e. linear in $n$.

c) In the logarithmic cost model the cost of an operation is the logarithm of the largest number involved in this operation (strictly speaking we also had to add 1 which we ignore for now). We have seen in a) that $x = 2^{2^k-1}$ before the $k$-th iteration. Therefore the cost of just the multiplication in the $k$-th iteration is $\log_2(2^{2^k-1}) = 2^k-1$. If we consider only multiplications and ignore all other operations, then all iterations together have an overall cost of at least

$$2^0 + 2^1 + \cdots + 2^{n-1} = 2^n - 1.$$  \hspace{1cm} (22)

Since we ignored all operations other than multiplications, the overall cost of the code fragment is greater than $2^n - 1$, i.e., it is at least $2^n$.

Solution 2.4  Algorithm Design: Divide-and-Conquer.

With the “divide-and-conquer” paradigm we get to the following solution: the array is divided in two parts of equal size (for simplicity we will assume that $n$ is even). The key observation is the following: if a majority element exists in $A[1..n]$, i.e., if it occurs more than $n/2$ times, then we can find it in at least one of the two halves more than $n/4$ times (if it occurred at most $n/4$ times in both of the two halves, then it occurred in $A[1..n]$ only at most $n/2$ times). It follows that, if an element is a majority element in $A[1..n]$, then it is a majority element also in at least one of the two halves, either in $A[1..n/2]$ or in $A[n/2 + 1..n]$. The same considerations hold also if $n$ is odd. Therefore we can determine recursively whether a majority element exists in one of the two halves. This gives us one, two or no candidates (i.e., possible elements) for the majority element of $A[1..n]$. For each of these candidates we can easily check whether it is the majority element of $A[1..n]$ by going through the array and counting how many times each candidate appears.

*Analysis:* Let $T(n)$ be the running time of the algorithm for an array with $n$ elements. Then, we get $T(n) = 2T(n/2) + cn + d$, and $T(1) = e$, where $c,d,e$ are constants. We can solve this recurrence relation using exercise 2.1, and obtain $T(n) = ne + cn \log_2(n) + d(n - 1) \in \Theta(n \log n)$.  

3