Algorithms: Solutions 3

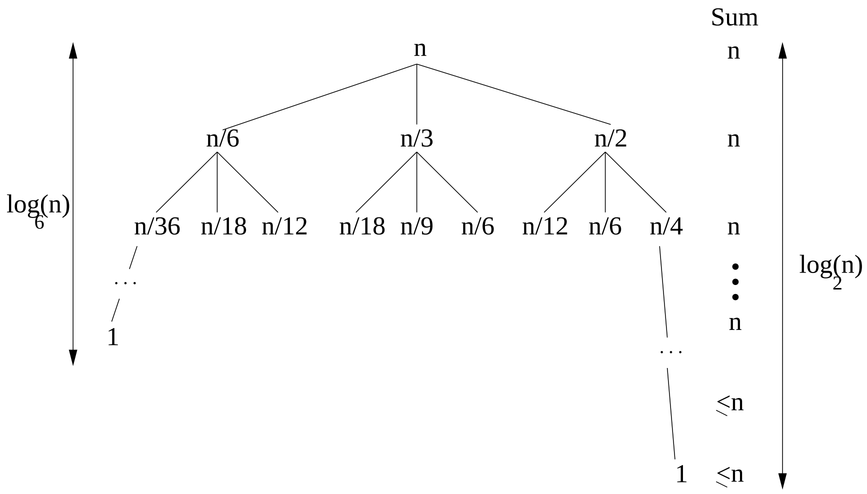
```
                                X
                            X   X
                            X   X
                            X   X
                        X   X   X
                    X   X   X   X
                    X   X   X   X
                X   X   X   X   X
        X       X   X   X   X   X
        X   X   X   X   X   X   X
    X   X   X   X   X   X   X   X
X   X   X   X   X   X   X   X   X
-----------------------------------
2   3   4   5   6   7   8   9   10
             grades
```

The histogram shows the distribution of grades.

Problem 1
Determine asymptotic upper and lower bounds for each of the following recurrences.

(a) $T(n) = T(n/6) + T(n/3) + T(n/2) + n$.

We use the iteration method, which leads to the following tree:



The summation gives an upper and lower bound for $T(n)$:

$$n \cdot \log_6 n \leq T(n) \leq n \cdot \log_2 n,$$

which implies that

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

(b) $T(n) = T(n-1) + n$.

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= T(n-2) + (n-1) + n \\ &\quad \cdots \\ &= 1 + 2 + 3 + ... + (n-1) + n \\ &= \frac{n(n+1)}{2} \\ &= \Theta(n^2) \end{aligned}$$

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

$$\begin{aligned} T(n) &= T(n-1) + \frac{1}{n} \\ &= T(n-2) + \frac{1}{n-1} + \frac{1}{n} \\ &\quad \cdots \\ &= 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n-1} + \frac{1}{n} \\ &= \ln n + O(1) \qquad \triangleright \text{ using Equality 3.5 from the textbook} \\ &= \Theta(\lg n) \end{aligned}$$

(d) $T(n) = T(\sqrt{n}) + 1$.

We "unwind" the recurrence until reaching some constant value of n, e.g. until $n \leq 2$:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 2 \\ T(\sqrt{n}) + 1 & \text{if } n > 2 \end{cases}$$

For convenience, assume that $n = 2^{2^k}$, for some natural value k:

$$\begin{aligned} T(n) &= 1 + T(\sqrt{2^{2^k}}) \\ &= 1 + T(2^{2^{k-1}}) \\ &= 1 + 1 + T(\sqrt{2^{2^{k-1}}}) \\ &= 1 + 1 + T(2^{2^{k-2}}) \\ &= 1 + 1 + 1 + T(\sqrt{2^{2^{k-2}}}) \\ &= 1 + 1 + 1 + T(2^{2^{k-3}}) \\ &\quad \cdots \\ &= 1 + 1 + 1 + ... + 1 + T(2) \qquad \triangleright \text{ the sum is of length } k \\ &= k + \Theta(1) \\ &= \Theta(k) \end{aligned}$$

Finally, note that $k = \lg \lg n$ and, hence,

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

(e) $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n.$

We assume for convenience that $n = 2^{2^k}$ and $T(4) = 4$, and use induction to prove the following equality:

$$T(2^{2^k}) = 2^{2^k} \cdot k.$$

This equality holds for $k = 1$:

$$T(2^{2^1}) = T(4) = 4 = 2^{2^1} \cdot 1,$$

and the induction step is as follows:

$$\begin{aligned}
T(2^{2^{k+1}}) &= \sqrt{2^{2^{k+1}}} \cdot T(\sqrt{2^{2^{k+1}}}) + 2^{2^{k+1}} \\
&= 2^{2^k} \cdot T(2^{2^k}) + 2^{2^{k+1}} \\
&= 2^{2^k} \cdot (2^{2^k} \cdot k) + 2^{2^{k+1}} \\
&= (2^{2^k})^2 \cdot k + 2^{2^{k+1}} \\
&= 2^{2^{k+1}} \cdot k + 2^{2^{k+1}} \\
&= 2^{2^{k+1}} \cdot (k+1)
\end{aligned}$$

We now note that $k = \lg \lg n$, which implies that

$$T(n) = n \cdot \lg \lg n.$$

Problem 2

The standard analysis of MERGE-SORT(A, p, q) is based on the assumption that we pass $A[1..n]$ by a pointer. If a language does not allow passing an array by a pointer, we may have two other options; for each option, determine the running time of MERGE-SORT.

(a) Copy all elements of the array $A[1..n]$, which takes $\Theta(n)$ time.

Let n be the size of the array $A[1..n]$, and m be the size of the segment $A[p..q]$, sorted by the recursive call MERGE-SORT(A, p, q). The time of copying the array is $\Theta(n)$, and the time of the MERGE operation is $\Theta(m)$, which leads to the following recurrence:

$$T(m) = 2 \cdot T(m/2) + \Theta(n) + \Theta(m).$$

Since $m \leq n$, we conclude that

$$T(m) = 2 \cdot T(m/2) + \Theta(n) = 2 \cdot T(m/2) + c \cdot n,$$

and unwind this recurrence as follows:

$$\begin{aligned}
T(m) &= 2 \cdot T(m/2) + c \cdot n \\
&= 4 \cdot T(m/4) + 2 \cdot c \cdot n + c \cdot n \\
&= 8 \cdot T(m/8) + 2^2 \cdot c \cdot n + 2 \cdot c \cdot n + c \cdot n \\
&\quad \cdots \\
&= 2^{\lg m} \cdot c \cdot n + 2^{\lg m - 1} \cdot c \cdot n + ... + 2^2 \cdot c \cdot n + 2 \cdot c \cdot n + c \cdot n \\
&= (2^{\lg m + 1} - 1) \cdot c \cdot n \\
&= (2 \cdot m - 1) \cdot c \cdot n \\
&= \Theta(m \cdot n)
\end{aligned}$$

Thus, the running time of MERGE-SORT(A, p, q) is $\Theta(m \cdot n)$, where m is the size of the segment $A[p..q]$. The top-level call to the sorting algorithm is MERGE-SORT$(A, 1, n)$; for this call, we have $m = n$, which means that the time complexity is

$$T(n) = \Theta(n^2).$$

(b) Copy the elements of the segment $A[p..q]$, which takes $\Theta(q - p + 1)$ time.

The complexity of copying the segment is $\Theta(m)$, which is the same as the time of the MERGE procedure; hence, copying does not affect the complexity of the algorithm. The recurrence is the same as the standard recurrence for MERGE-SORT, and the overall time is $\Theta(n \cdot \lg n)$.

Problem 3

Suppose that $A[1..n]$ and $B[1..m]$ are sorted arrays, and $n \leq m$. Write an algorithm that finds their smallest common element; if they have no common elements, it should return 0.

The intuitive idea is to divide $B[1..m]$ into segments, each of size $k = m/n$, and perform binary search in each segment. We need to use a version of binary search, BIN-SEARCH(B, p, r, k), which searches for an element k in a segment $B[p..r]$. If this version finds k, it returns the corresponding index of B; if not, it returns the index of the next larger element. For example, if $k = 6$ and $B[p..r] = \langle 3, 5, 7, 9 \rangle$, the search returns the index of 7. The following algorithm calls BIN-SEARCH on k-element segments of B.

COMMON-ELEMENT(A, B, n, m)
$k \leftarrow \lfloor m/n \rfloor$
$i \leftarrow 1$
$j \leftarrow 1$
while $i \leq n$ and $j \leq m$
    do **if** $A[i] = B[j]$
            then return $A[i]$
        if $A[i] < B[j]$
            then $i = i + 1$
            else repeat $j = j + k$
                    until $j > m$ or $A[i] \leq B[j]$
                $j \leftarrow$ BIN-SEARCH$(B, j - k + 1, \min(j, m), A[i])$
return 0

The running time of COMMON-ELEMENT is $O(n \cdot (1 + \lg \frac{m}{n}))$. In particular, if A and B are of about the same size, then the time is $O(m)$. On the other hand, if A is much smaller than B, the running time is significantly better than $O(m)$.