Problem Solutions

1. (a) Using the Principle of Optimality, we can express $L_j$ recursively as

$$L_j = \begin{cases} 1 & \text{if } j = 1, \\ \max \{L_{j-1}, \max_{1 \leq i < j} \{1 + L_i\}\} & \text{otherwise.} \end{cases}$$

Of course the values of $A_j$ and $B_j$ must be maintained in this recursive definition (that is, which of the various possibilities in the max was maximum with the lowest value of $A_j$ and what the corresponding value of $B_j$ is).

Without the hint, we might be tempted to express the recurrence looking at all increasing subsequences that have $a_i$ as a member, recursively obtain longest increasing subsequences on the left and on the right, and the choose the combination with the overall maximum length. Writing the recurrence (and hence recursive code) is tricky because the element we insist on, $a_i$, must be larger than the largest element of the increasing subsequence on the left and smaller than the least element of the increasing subsequence on the right. This approach would mean adding two parameters to the recurrence so that it works out properly—a value that is an upper bound for a possible left increasing sequence and a a value that is a lower bound for a possible right increasing sequence.

(b) Let $c_j$ be the rate of growth of the cost of evaluating $L_j$ using the recurrence in (a). We have

$$c_j = \begin{cases} 1 & \text{if } j = 1, \\ 1 + \sum_{1 \leq i < j} c_i & \text{otherwise.} \end{cases}$$

Looking at the difference between successive values (as we did in the Quicksort recurrence to eliminate the summation), we find that $c_j - c_{j-1} = c_{j-1}$ implying $c_j = 2c_{j-1}$ and hence $c_j = 2^{j-1}$; we could also guess at this solution by computing the first few values and then verifying the guess by induction. Thus computing $L_n$ would cost $\Theta(2^n)$.

(c) Following the hint, to memoize (a) we use an array $L[j]$ to tell us the length of the LIS among $a_1, a_2, \ldots, a_j$, an array $A[j]$ to tell us the index of the smallest possible largest (last) element of the LIS among $a_1, a_2, \ldots, a_j$, and an array $B[j]$ to tell us the index of
the smallest possible second largest element of the LIS among $a_1, a_2, \ldots, a_j$. Define a dummy element $a_0 = -\infty$; then $L[1] = 1$, $A[1] = 1$, and $B[1] = 0$. We have:

1: $a_0 = -\infty$
2: $L[1] = 1$
4: $B[1] = 0$
5: for $j = 2$ to $n$ do
6: \[ L[j] \leftarrow L[j - 1] \]
7: \[ A[j] \leftarrow A[j - 1] \]
8: \[ B[j] \leftarrow B[j - 1] \]
9: for $i = 1$ to $j - 1$ do
10: if $L[j] = L[i]$ and $a_{A[j]} > a_{A[i]} > a_{B[j]}$ then
11: // $a_{A[i]}$ gives us a lower largest value for $L[j]$
12: \[ A[j] \leftarrow A[i] \]
13: else if $L[j] = L[i]$ and $a_{A[j]} > a_j > a_{B[j]}$ then
14: // $a_j$ gives us a lower largest value for $L[j]$
15: \[ A[j] \leftarrow j \]
16: else if $L[j] \leq L[i]$ and $a_j > a_{A[i]}$ then
17: // we can add $a_j$ to end of $L[i]$
18: \[ L[i] \leftarrow L[i] + 1 \]
19: \[ B[j] \leftarrow A[j] \]
20: \[ A[j] \leftarrow j \]
21: end if
22: end for
23: end for

(d) The cost of this double-nested loop computation is \( \sum_{1 < j \leq n} \sum_{1 \leq i < j} 1 = \sum_{1 < j \leq n} (j - 1) = n(n - 1)/2 = \Theta(n^2) \).

(e) We recover the LIS itself from the $A[j]$ and $B[j]$ values as follows:

1: procedure WriteLIS($i$)
2: if $i > 0$ then
3: \[ A[i] = i \] then
4: WriteLIS($B[i]$)
5: SystemPrint($a_i$)
6: else
7: WriteLIS($A[i]$)
8: end if
9: end if

2. (a) Take $r_1 = 2$, $r_2 = 3$, and $r_3 = 4$. Buying the equipment in increasing order has a total cost $100(2 + 3^2 + 4^3) = 7500$, but in reverse order the total cost is $100(4 + 3^2 + 2^3) = 2100$.

(b) Following the hint, suppose there is an optimal solution $O$ that differs from our highest-cost-first solution $S$. Then $O$ must contain an “inversion” in which two successive
months, $t$ and $t + 1$, in which the increase rate of the equipment bought in month $t$, $r_t$ is less than the rate for the equipment bought in month $t + 1$, $r_{t+1}$; that is, $r_t < r_{t+1}$. Swapping these two purchases saves us

$$100(r^t_t + r^{t+1}_{t+1}) - 100(r^t_{t+1} + r^{t+1}_{t}).$$

We want to show this is a positive amount; that is,

$$r^t_t + r^{t+1}_{t+1} > r^t_{t+1} + r^{t+1}_{t}$$

Rearranging, we need

$$r^{t+1}_{t+1} - r^t_{t+1} > r^{t+1}_{t} - r^{t}_{t}$$

or

$$r^{t+1}_{t+1}(r_{t+1} - 1) > r^{t}_{t}(r_{t} - 1),$$

which is true because $r_i > 1$ for all $i$ and $r_t < r_{t+1}$.

(c) The running time of the algorithm is dominated by $\Theta(n \log n)$, the cost of sorting the equipment into decreasing order by rate; the rest of the algorithm is just outputting the result.

3. Let $n$ be the number of elements on the stack before an operation. The calculations are nearly identical to those done in chapter 17 of CLRS (lecture notes of February 29):

$$\text{AMORT}_{\text{POP}} = \text{ACTUAL}_{\text{POP}} + |\text{stack}|_{\text{after}}^2 - |\text{stack}|_{\text{before}}^2$$
$$= 1 + (n - 1)^2 - n^2 = -2(n - 1) \leq 0 = O(n).$$

$$\text{AMORT}_{\text{PUSH}} = \text{ACTUAL}_{\text{PUSH}} + |\text{stack}|_{\text{after}}^2 - |\text{stack}|_{\text{before}}^2$$
$$= 1 + (n + 1)^2 - n^2 = 2(n + 1) = O(n).$$

$$\text{AMORT}_{\text{MULTIPOP}}(k) = \text{ACTUAL}_{\text{MULTIPOP}}(k) + |\text{stack}|_{\text{after}}^2 - |\text{stack}|_{\text{before}}^2$$
$$= k(k - 1)/2 + (n - k)^2 - n^2$$
$$= k^2/2 - k/2 - 2kn$$
$$= k(k/2 - 2n - 1/2) < 0 = O(n),$$

because $k \leq n$.

4. We want a complexity of $\Omega(m \log n)$, therefore a reasonable guess is that the disjoint-set tree generated with union by rank has a depth $\Omega(\log n)$. Such a tree can be easily achieved. Suppose that, without the loss of generality, $n$ is a power of 2 and we have $\{x_1, x_2, \ldots, x_n\}$ since there are $n$ MAKE-SET operations. Then, because our UNION is a union by rank, we need to union small trees as soon as possible so that we won’t have a tree where all the small trees are linked to one single tree. In an extreme case, we may end up with a tree of depth 1, where $n-1$ nodes are linked to one node with is the root. Therefore, after preparing the set by
running $n$ \textsc{Make-Set} operations on $\{x_1, x_2, \ldots, x_n\}$, we run \textsc{Union}(x_1, x_2), \textsc{Union}(x_3, x_4), \textsc{Union}(x_5, x_6) and so forth in the first round. After all individual nodes are unioned, we union the trees by running \textsc{Union}(x_1, x_3), \textsc{Union}(x_5, x_7) and so forth by unioning each pair of consecutive trees in the second round. This is repeated until there is only a single tree (set) left, at which time there were $\log n$ rounds of unions. It is easy to see we have performed $n - 1$ \textsc{Union} operations ($\frac{n}{2} + \frac{n}{4} + \cdots + 2 + 1$), and the tree has a depth $O(\log n)$ because at each round the depth increase by 1. So far, we have $n$ \textsc{Make-Set} operations and $n - 1$ \textsc{Union} operations.

Suppose we have $k$ \textsc{Find-Set} operations. If we perform \textsc{Find-Set} on the node of greatest depth, each \textsc{Find-Set} will be $\Theta(\log n)$. Then, the total complexity of $m = 2n - 1 + k$ operations will be $\Omega(n + n - 1 + k \log n)$. When $k \geq n$, $k = \Theta(m)$, and we have $\Omega(2n - 1 + k \log n) = \Omega(m \log n)$. 