Solution 4

1. (5 points) The idea of the greedy algorithm is to merge two arrays with the smallest sizes each time. The merge pattern for the greedy algorithm can be represented by binary trees.

   //The algorithm has input a list L of n trees.
   //Each node of tree has three fields: LCHILD, RCHILD and WEIGHT.
   //Initially, each tree in L has exactly one node, which represents one array. And LCHILD and RCHILD are zeros, and WEIGHT is the size of array.
   1. for i ← 1 to n ← 1 do
      2. call GETNODE(T) //allocate a new root
      3. LCHILD(T) ← LEAST(L) //find a tree with the least root weight as the left child of the new root, then delete the tree from list
      4. RCHILD(T) ← LEAST(L) //find another tree with the least root weight as the right child of the new root, then delete the tree from list
      5. WEIGHT(T) ← WEIGHT(LCHILD(T)) + WEIGHT(RCHILD(T)) // get the weight for the the new root, and the weight is the sum of the weights of left and right children
      6. call INSERT(L,T) // insert the tree with new root to the list
      7. repeat //repeat n-1 times until there is only one tree
         8. return (LEAST(L)) // return the tree

2. (2 points) Time complexity:

   The main loop is executed n – 1 times. If L is kept in nondecreasing order according to the WEIGHT value in the roots, then LEAST(L) requires only O(1) time and INSERT(L,T) can be done in O(n) time. Hence the total time taken is $O(n^2)$.

   If we use the heap, then LEAST(L) and INSERT(L,T) can be done in $O(n \log n)$ time. Hence the total time taken is $O(n \log n)$.

3. (3 points) Proof. (By induction on n). For $n = 1$, it is clear. For the induction hypothesis, assume the greedy algorithm in (1) generates an optimal merge pattern for all $(q_1,q_2,\cdots,q_m)$, where $q_i$ is the size of array $i$ and $1 \leq m < n$. We will show that the greedy algorithm in (1) generates an optimal merge pattern for all $(q_1,q_2,\cdots,q_n)$. Without loss of generality, we may assume $q_1 \leq q_2 \leq \cdots \leq q_n$ and that $q_1$ and $q_2$ are the values of the WEIGHT fields of the trees found by algorithm LEAST in lines 3 and line 4 during the first iteration of the "for" loop. Now, the subtree $T$ with new root of weight $(q_1+q_2)$ is created. Let $T'$ be the binary tree of an optimal merge pattern for $(q_1,q_2,\cdots,q_n)$. Let $P$ be an internal node of maximum distance from the root in $T'$. If the children of $P$ are not $q_1$ and $q_2$, then we may interchange the present children with $q_1$ and $q_2$ without increasing the weight of the root. Hence, $T$ is also a subtree in an optimal merge tree. Now, in $T'$ if we replace $T$ by an external node with weight $q_1 + q_2$ then the resulting tree $T''$ is an optimal merge tree for $(q_1+q_2,q_3,\cdots,q_n)$. From the induction hypothesis, after replacing $T$ by the external node with value $q_1 + q_2$, proceeds to find an optimal merge tree for $(q_1+q_2,q_3,\cdots,q_n)$. Hence, the greedy algorithm in (1) generates an optimal merge pattern for all $(q_1,q_2,\cdots,q_n)$.

(Note: form the above (1), we know a merge pattern can be represented as a binary tree whose leaf nodes correspond to original given arrays, internal nodes correspond to the
merged arrays, and the root corresponds to the final single sorted array. If let $d_i(T)$ be the distance from root to the external node for the array $i$, and $q_i$ be the size of array $i$, then the total number of datum moves $B(T)$ is

$$\sum_{i=1}^{n} d_i(T)q_i$$

So, in order to find the minimum total number of datum moves, we need to construct a binary tree with minimum value of $B(T)$, which is the same as Huffman Codes problem in page 385-391).