1 ▸ **Problem** *17-2*, parts ***a*** and ***b***, on pg. 473 of our text.

SOLUTION: ***a*** Let $k = \lceil \lg(n+1) \rceil$, and there are $O(\lg n)$ arrays. Searching each A_i using BINARYSEARCH, the worst-case search time is $(k-1)+(k-2)+...+2+1 \in O(\lg^2 n)$.

b To INSERT a new element *x* into this data structure, we make a new (sorted) array A_0^* containing only *x*. We then MERGE this new data structure with the old structure consisting of sorted arrays, analogous to the operation on binomial heaps. To merge two sorted arrays into one sorted array, we use the MERGE procedure on pg. 29 of our text, which operates in linear time. In the worst case, the data structure contains arrays $A_0,...,A_{k-1}$, and the worst-case execution time is $\Theta(n)$. In general, each element which is INSERTed into the data structure, gets MERGEd and costs an operation each time it moves from an A_i to an A_{i+1}. Thus, it moves $O(k)$ or $O(\lg n)$ times. Thus, when an element is INSERTed, \$*k* is spent, \$1 to pay for the INSERTion, and \$(*k*-1) to pay for future MERGEs.

****

2 ▸ Assume that we are implementing a binomial heap with eager UNION.

a. Describe a class of binomial heaps which cause EXTRACT-MIN to run in time $\Omega(\lg n)$.

b. Describe a class of binomial heaps which cause INSERT to run in time $\Omega(\lg n)$.

SOLUTION: ***a***. For any positive integer *k*, performing EXTRACT-MIN on a binomial heap of $n = 2^k - 1$ elements where the minimum element is the root of binomial tree B_{k-1} exposes *k*-1 new binomial trees. A new set of *k*-1 links must be formed, in time $\Omega(\lg n)$.

b. For any positive integer *k*, performing INSERT on a binomial heap of $n = 2^k - 1$ elements links the new element with the roots of *k*-1 binomial trees, in time $\Omega(\lg n)$.

****

3 ▸ **Exercise *17.1-2*** on pg. 456 of our text.

SOLUTION: If the counter contains $2^k - 1$ (all 1's), then each operation in the sequence

INCREMENT, DECREMENT, INCREMENT, DECREMENT, INCREMENT, DECREMENT,...

of *n* operations takes $\Theta(k)$ time, and the sequence takes $\Theta(nk)$ time.

****

4 ▸ **Exercise *17.1-3*** on pg. 456 of our text.

****

5 ▸ **Exercise *17.2-1*** from pg. 458 of our text.

SOLUTION: Assume that COPY is invoked automatically after every sequence of *k* PUSHs and POPs. Charge \$2 for each PUSH and POP, and COPY is free, \$0. We charge \$1 to execute a PUSH, and we store the extra \$1 with the item. Likewise, we charge \$1 to execute a POP, and store the extra \$1 with the stack. Since *k* PUSHs and POPs are executed between successive pairs of COPYs, we know that \$*k* has been stored, either with items in the stack or with the stack (from POPs). So the cost of *n* operations is $O(n)$.

****

6 ▸ **Exercise *17.2-3*** on pg. 459 of our text.