Problem 17-2

17-2(a)

We do binary search list by list, i.e. if cannot find it in the first list, search it in the second list, and so on until it find the element. Since the list length is at most $\log(n)$ and there are at most $\log(n)$ lists. Therefore the run time is $O(\log^2(n))$

17-2(b)

When inserting an element so size increase from n to $n' = n+1$, compare the binary representation of n' with n, there always exists a lowest significant bit b_j changes from 0 to 1 and all $j = 0..i-1$ changes from 1 to 0. As it is trivial that each individual merge of 2 lists have run time linear to the total number of elements involved in the two lists, we merge lists two by two, that is merge list 0 with list 1, then merge the resulted list with list 2, then merge with list 3, repeat until we merge with list $i-1$, and we move the list as list i and clear all list $0..i-1$.

At step k we merge list k with the result of mergin list 0..k-1, the total number of elements involved $= \sum_{j=0}^{k} 2^j \leq 2^{k+1}$. Then we sum up runtime of all merges $\leq \sum_{k=0}^{i} 2^{k+1} \leq 2^{i+2} = O(2^i)$

17-2(c)

Using accounting method.

Charge each insertion an amortized cost of $O(\log(n))$, which pays the actual amount cost $O(\log(n))$ of inserting the element in sorted order, into list i as defined in (b), we insert the new element by insertion sort, as list i has length at most $O(\log(n))$, the insertion takes at most $O(\log(n))$, and associate a credit of $O(\log(n))$ onto the newly created list i that the insertion process generated, then the associated credit of each list is never less than its length $O(\log(n))$.

Hence in merging list $0..i-1$ to i, the total credit of list $0..i-1$ is at least the number of the element involved, i.e. $\Omega(2^i)$, then the credits are enough for paying the merging process which costs $O(2^i)$.

Hence together the insertion and merging, the amortized cost is $O(\log(n))$.

Problem 17-3

17-3(a)

Traverse the subtree in order and put the items into an array, since it is a BST thus the array is sorted. From the sorted array we construct a new BST with the element at the middle as root. Recursively do this to the left sub array and right sub array as the left subtree and right subtree respectively, until the array is empty list we return it as empty tree.

The BST rebuilding from array shall be runtime of $O(n)$. Denote the runtime is $T(n)$, $T(n) = 2T(n/2) + O(n) = O(n)$. As the in order traveral is of $O(n)$ also, the total running time required is $O(n)$.

17-3(b)

Lemma: α-balanced tree have height $O(\log(n))$
Proof: Define $v.bigger$ as the subtree of node v which have bigger height.

$$\begin{aligned}
& \alpha size(v) \geq size(v.bigger) \\
\implies & n = size(v) \geq \frac{1}{\alpha} size(v.bigger) \\
& \geq \frac{1}{\alpha^2} size(v.bigger.bigger) \geq ... \geq \alpha^{-d} size(leaf) = \alpha^{-d} \\
\implies & \log n \geq d \log(\frac{1}{\alpha}), \log \text{ is strictly increasing} \\
\implies & d \leq \log n / \log(\frac{1}{\alpha}), \log(\frac{1}{\alpha}) \text{ is positive since } \frac{1}{2} < \alpha < 1 \\
\implies & d = O(\log n)
\end{aligned}$$

Where d is the height.

Use the potential method.

Define potential on each node v, its potential is $p(v) = |size(v.left) - size(v.right)|$. Then potential of the whole tree becomes $O(\sum_{v \in V} p(v))$.

1. Insertion/ deletion:
   Actual cost: $O(\log(n))$
   Potential change: $O(\log(n))$
   Amortized cost: Actual + potential change = $O(\log(n))$

   When doing insertion or deletion, each node on path may increase in potential for at most 1 time, and there are at most $O(\log(n))$ nodes on the path. Then the amortized cost for paying the total potential change is at most $O(\log(n))$.

2. Rebuild:
   m is the number of elements involved in the rebuilding
   Actual cost: $O(m)$
   Potential change: $-\Omega(m)$
   Amortized cost = actual cost + change in potential = $O(m) - \Omega(n) \leq 0$,
   by using sufficiently large constant.

   When node u need to be reconstructed, that is u is the highest unbalanced node after insertion or deletion, without losing generality we assume $size(u.left) > \alpha size(u) > \frac{1}{2} size(u)$, $m = size(u)$ then we have

   $$size(u.left) - size(u.right) > \alpha size(u) - ((1-\alpha) size(u) - 1) = 2\alpha m - m + 1$$

   then we have potential at u $p(u) = \Omega(m)$ before the reconstruction.

   After rebuilding the BST at u, all internal node will be balanced, which the potential drop to zero, except node at the second bottommost layer may have potential at most 1 since the

size of tree is not always power of 2, but number of such node is $O(\log(m))$ as a binary tree, so the potential after reconstruction is $O(\log(m))$. Meanwhile potential at other location remains unchanged. Then the potential change is $-\Omega(m)$.

Problem A

Let m be the number of elements when doing the previous contraction/ expansion, and let $(1 + 0.5\epsilon)m$ be the size of array after the contraction/ expansion, which is actually the current size. Let the number of elements is n after the insertion/ deletion. Define the contraction policy as, after the deletion of element, if the size of array is not smaller than $(1 + \epsilon)n$, contract the array to size $(1 + 0.5\epsilon)n$. Define the expansion policy as, after the insertion of element, if the size of array is the same as n, expand the array to size of $(1 + 0.5\epsilon)n$.

Define potential be number of insertion and deletion has been done so far after the previous expand/contract times $1/\epsilon$.

1. Insertion:
   Actual cost: $O(1)$
   Potential change: $O(1/\epsilon)$
   Amortized cost = actual + potential change = $O(1/\epsilon)$

   Trivial.

2. Expansion:
   Actual cost: $O(n)$
   Potential change: $-\Omega(n)$
   Amortized cost = actual + potential change ≤ 0 by choosing sufficiently large constant.

   When we are required to do expansion, $(1+0.5\epsilon)m = n \implies n - m = 0.5\epsilon m = \frac{\epsilon}{2+\epsilon}n \geq \epsilon n/3$ insertions have been done at least. Then the original potential $\Phi_{i-1} = \Omega(\epsilon n/3/\epsilon) = \Omega(n)$. Since the new potential Φ_i will be 0, then the potential difference $\Delta\Phi = \Phi_i - \Phi_{i-1} = -\Omega(n)$ The cost for reconstructing is linear to the current number of elements $= O(n)$. By setting sufficiently large constant, since actual cost is upper bounded by $O(n)$ and potential change is arbitaryly negative with respect to $-\Omega(n)$, we can have the amortized cost less than zero.

3. Deletion:
   Actual cost: $O(1)$
   Potential change: $O(1/\epsilon)$
   Amortized cost = actual + potential change = $O(1/\epsilon)$

   Trivial.

4. Contraction: Actual cost: $O(n)$
   Potential change: $-\Omega(n)$
   Amortized cost = actual + potential change ≤ 0 by choosing sufficiently large constant.

When we are required to do contraction,

$$\begin{aligned}(1+0.5\epsilon)m &= (1+\epsilon)n \\ (1+\epsilon-0.5\epsilon)m &= (1+\epsilon)n \\ m-n = \frac{0.5\epsilon}{1+\epsilon}m &= \frac{0.5\epsilon}{1+\epsilon}\frac{1+\epsilon}{1+0.5\epsilon}n \geq \frac{\epsilon}{3}n \\ \Phi_{i-1} &= \Omega(n)\end{aligned}$$

Since the new potential Φ_i will be 0, then the potential difference $\Delta\Phi = \Phi_i - \Phi_{i-1} = -\Omega(n)$. The cost for reconstructing is linear to the current number of elements $= O(n)$. By setting sufficiently large constant, since actual cost is upper bounded by $O(n)$ and potential change is arbitaryly negative with respect to $-\Omega(n)$, we can have the amortized cost less than zero.