Homework 4, additional problems, solution set

1. We begin by sorting array A use merge sort. This takes $O(m \log m)$ time. Now, we have a sorted array where we can search for an element using binary search in time $O(\log m)$. We search for each element in B in the sorted A, and if any element is found the arrays are clearly not disjoint. The search process takes a total of $O(n \log m)$ time.

Therefore, the total runtime is $O(m \log m) + O(n \log m) = O(n \log m)$.

2. Let $D(n)$ denote the average total depth for an n item binary search tree.

Observe that each item has a $1/n$ probability of being first and consequently of being placed at the root of the binary search tree. Suppose the ith item in the sorted order, e_i, is the one at the root. Then the smallest $i-1$ items will be in the root's left subtree, and the largest $n-i$ will be in its right subtree. The left subtree will have average total depth $D(i-1)$ (not counting the edge from the root of the whole tree) and the right subtree will have average total depth $D(n-i)$.

Every path includes the root node, thus this adds n to the total depth.

This yields the following recurrence equation for D:

$$D(n) = \frac{1}{n} \sum_{i=1}^{n} [D(i-1) + D(n-i)] + n \quad n \geq 1$$
$$D(0) = 0$$

This is very similar to the recurrence equation for the number of comparisons in Quicksort (in the Quicksort recurrence, the n is replaced by $n-1$).

3. The algorithm proceeds as follows.
Step 1. Merge sort the subarrays $A[(i-1)k+1 : ik]$ for $i = 1, 2, \ldots n/k$ (possibly the last subarray has fewer than k items). This takes $O(k \log k \cdot n/k) = O(n \log k)$ time.
Step 2. Create up to k buckets for the items in each sorted subarray, with one bucket for each distinct item. Put all the copies of each value in the corresponding bucket, and for each subarray keep the buckets in sorted order based on the value they are storing. It is helpful to allow the start node of the bucket to have a value field in which this value is stored. This takes $O(k)$ time per subarray, or $O(n)$ time in total.
Step 3. In turn, for $i, 1, 2, \ldots, n/k - 1$, combine the up to k buckets for the first i subarrays with the buckets for the $i+1$-st subarray, as follows. Merge the two lists of buckets in value order, combining the lists for equal buckets in $O(1)$ time (the list for the second bucket is simply appended to the list for the first bucket; to do this in $O(1)$ time we need pointers to the start and end of each list). This takes $O(k)$ time per subarray, or $O(n)$ time in total.

Overall, our algorithm runs in $O(n \log k)$ time.