CSC263: Problem Set 5

November 12, 2019

1 Problem 2

(a) The worst-case time for SEARCH will be $O((log(n))^2)$. This is because the worst-case run time of binary search for each array is $O(log(n))$. Additionally, given n elements we will have $\lfloor log(n) \rfloor + 1$ sorted arrays (this is from the notes for binomial heaps, since we can view the entire linked list as the heap and at each node we have a "binomial tree" i.e. 2^k elements at each "binomial tree" which in this case is a sorted array). So we get $log(n) \cdot log(n) + 1 = (log(n))^2 + log(n)$. But since $(log(n))^2$ is larger than $log(n)$ (for large n) we end up with a worst-case time complexity of $O((log(n))^2)$.

(b) To calculate the worst-case time of INSERT, we assess each part of the INSERT process. When creating an array of size 1 containing x, this will occur in constant time, $O(1)$, regardless of what x is. Similarly, inserting this array at the beginning of the linked list will also occur in constant time, $O(1)$, since we need to only update the head pointer once. Finally, if there are two arrays of the same size, we merge the two arrays into an array that is twice the size. In the worst case, there is already an array of size 1 in the linked list, so adding another array of size 1, will require us to create an array of size 2, and if an array of size 2 already exists, then we create one of size 4, and so on. Ultimately we could have all n elements in one sorted array. Using the merge function from merge sort, the worst-case time will be in $O(n)$.

(c) Consider a set of n elements. Based on the given parameters, we would have a linked list with at most $\lfloor log(n) \rfloor + 1$ sorted arrays. Now, we consider each array of length 2^k, where k is a different positive integer for each array. Note that since the creation of the array with 1 element and inserting at the beginning of the linked list occur in constant time, we are more interested in the merging of arrays.

We assign the following amortized cost to the operation:

Insert: \$$2^k$ where k is a positive integer. Where A[k] denotes the size of the corresponding array (i.e. A[0] is the array of size 1, A[1] is the array of size 2, A[2] is the array of size 4 and so on, A[k] has a size of 2^k, so this is similar to the binary counter).

We argue this in a similar way to the binary counter. A[0] gets ”flipped” every time for a cost of \$1. A[1] gets ”flipped” every other time for a cost of \$2. A[2] gets flipped every 4th time for a cost of \$4. This continues till A[$log(n)$] which would get flipped once for the cost of at most \$n. So then the total cost is given by $n(log(n) + 1)$, which is $O(log(n))$ amortized per increment.