# Homework 1

Michael Morikawa

April 2, 2020

## Chapter 9

R-9.1 Which of the hash table collision-handling schemes could tolerate a load factor above 1 and which could not?

**Answer:** Seperate chaining can tolerate a load factor above 1 but none of the open addressing schemes can.

R-9.6 Describe how to use a skip-list map to implement the dictionary ADT, allowing the user to insert different entries with equal keys.

**Answer:** Have the entries be stored in a bucket and when a entry with a key already in the skip list is inserted, have it be added to the bucket. Of course you would have to update all levels of the skiplist. Then for findAll have a range class nested in the dicionary class and return a range that contains all entries that have that key.

R-9.7 Draw the 11-entry hash table that results from using the hash function, h(i) = (3i + 5) mod 11, to hash the keys 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, and 5, assuming collisions are handled by chaining.

**Answer:**

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Key | 13 | 94, 39 | | | | 44, 88, 11 | | | 12, 23 | 16, 5 | 20 |

R-9.8 What is the result of the previous exercise, assuming collisions are handled by linear probing?

**Answer:**

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Key | 13 | 94 | 39 | 16 | 5 | 44 | 88 | 11 | 12 | 23 | 20 |

R-9.10 What is the result of Exercise R-9.7 when collisions are handled by double hashing using the secondary hash function $h`(k) = 7 - (k \ mod \ 7)$?

**Answer:**

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Key | 13 | 94 | 23 | 11 | 39 | 44 | 5 | 88 | 12 | 16 | 20 |

R-9.14 Explain why a hash table is not suited to implement the ordered dictionary ADT.

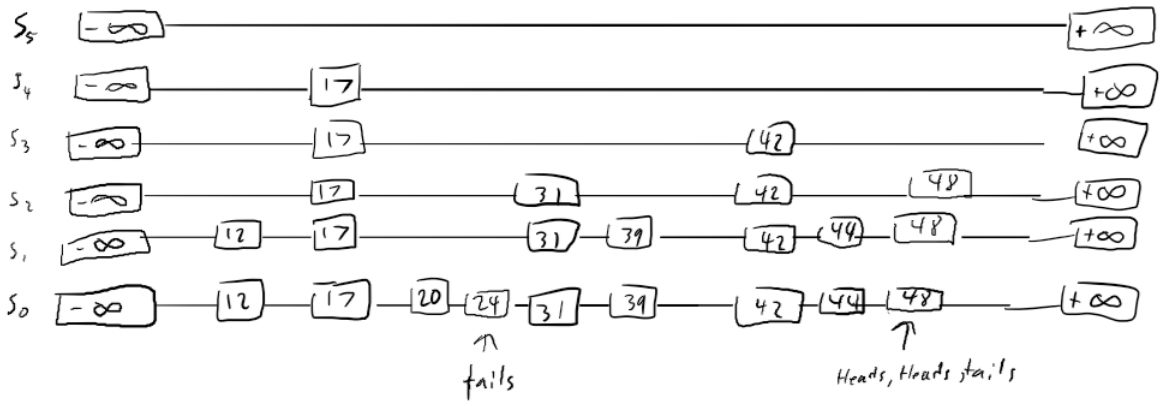
**Answer:** A hash table isn't suited to implement the ordered dictionary ADT because by putting keys through a hash function doesn't preserve order because it compresses the keys.

R-9.15 What is the worst-case running time for inserting n items into an initially empty hash table, where collisions are resolved by chaining? What is the best case?

**Answer:** The worst case is $\mathcal{O}(n^2)$ where each key is mapped to to the same bucket. It will take one probe for the first, two for the second and so on until the $n^{\text{th}}$ probe. The best case running time is $\mathcal{O}(n)$ where every key maps to a different index.

R-9.16 Draw an example skip list that results from performing the following series of operations on the skip list shown in Figure 9.12: erase(38), insert(48, x), insert(24, y), erase(55). Record your coin flips, as well.

**Answer:**


$S_5$
$S_4$
$S_3$
$S_2$
$S_1$
$S_0$
-∞
+∞
17
42
48
12
31
39
44
20
24
tails
Heads, Heads, tails

C-9.3 Suppose we are given two ordered dictionaries S and T , each with n items, and that S and T are implemented by means of array-based ordered sequences. Describe an $O(\log^2 n)$-time algorithm for finding the kth smallest key in the union of the keys from S and T (assuming no duplicates).

**Answer:** First we can ignore all indices greater than k in both arrays.

C-9.5 Design a variation of binary search for performing findAll(k) in an ordered dictionary implemented with an ordered array, and show that it runs in time $O(\log n + s)$, where n is the number of elements in the dictionary and s is the size of the iterator returned.

**Answer:** Do normal binary search until you find the index that holds the key you are searching for. To then get the range of indices to return make 2 loops, one that decreases the index until it reaches a new key at index i, and one that increases the index until it finds a new key at index j. The range would then be indices i+1 through j-1. This runs in $O(\log n + x)$ because the initial search is is $O(\log n)$ and finding the range is $O(s)$.

C-9.14 Describe an efficient data structure for implementing the bag ADT, which supports a function add(e), for adding an element e to the bag, and a function remove, which removes an arbitrary element in the bag. Show that both of these functions can be done in O(1) time.

**Answer:** We can use an array with 2 member variables, index and last. Index will keep track of of where we are inserting into the array. It will start at zero and increment after insertion and last will be the index of the last inputted item. This means that insertion will be $O(1)$. Now for removal we will accept some index to remove. We will copy last over to that index effectively erasing it and then decrement both last and index by one. This is also done in constant time.

# Chapter 11

R-11.5 In the merge-sort tree shown in Figures 11.2 through 11.4, some edges are drawn as arrows. What is the meaning of a downward arrow? How about an upward arrow?

**Answer:** The downward arrow represents the divide step in the divide and conquer algorithm where we split the lists into two. The upward arrow represents the merging of the separated lists.

R-11.8 Suppose we are given two n-element sorted sequences A and B that should not be viewed as sets (that is, A and B may contain duplicate entries). Describe an $O(n)$-time method for computing a sequence representing the set $A \cup B$ (with no duplicates).

**Answer:** Have two variables i and j keeping track of the indices of A and B respectively. Have i and j initialize to zero. Compare A[i] and B[j] and output the smaller value to a new sequence and increment the proper variable. There will also be another variable keeping track of the end of the new sequence. Before adding to the sequence we will check to see if the element we want to add is the same as the last one if so ignore it and don't add it to the sequence.

R-11.10 Suppose we modify the deterministic version of the quick-sort algorithm so that, instead of selecting the last element in an n-element sequence as the pivot, we choose the element at index $\lfloor n/2 \rfloor$. What is the running time of this version of quick-sort on a sequence that is already sorted?

**Answer:** Since the sequence is already sorted by choosing the pivot to be the middle of the sequence guarantees that the 2 subsequences will be the same size +/- 1. From the book it states that when the 2 subsequences have roughly the same size then the runtime will $O(n \log n)$.

R-11.11 Consider a modification of the deterministic version of the quick-sort al- gorithm where we choose the element at index $\lfloor n/2 \rfloor$ as our pivot. De- scribe the kind of sequence that would cause this version of quick-sort to run in $\Omega(n^2)$ time.

**Answer:** The sequene will start with the second smallest element, followed by the fourth smallest element and so on until the middle where the largest element would be. After the largest element it will be the 3rd largest element, the 5th largest element and so on until the smallest element. This makes it so that each time every element will be smaller than the pivot.

R-11.18 Is the merge-sort algorithm in Section 11.1 stable? Why or why not?

**Answer:** It is stable because it resolves the left subtree first so the first element will stay to the left of the second element.

R-11.21 Is the bucket-sort algorithm in-place? Why or why not?

**Answer:** No it is not in-place since you have to create the buckets to put the entries into and you empty the initial sequence into the buckets then back into the original sequence.

C-11.5 Describe and analyze an efficient function for removing all duplicates from a collection A of n elements.

**Answer:** First sort the collection using an appropriate sorting algorithm. Then go through the collection element by element and check if the element is equal to the element before it. If so remove it. This will run in $O(n \log n)$ for the sorting and then the removal of duplicates should be $O(n)$. So the algorithm is $O(n \log n + n)$ which is $O(n \log n)$.

C-11.23 Let A and B be two sequences of n integers each. Given an integer m, describe an $O(n \log n)$-time algorithm for determining if there is an integer a in A and an integer b in B such that m = a + b.

**Answer:** First sort the two sequences using a $O(n \log n)$ algorithm. Then start at the first index of A, i, and the last index of B, j. Then compare A[i]+B[j] and m. If the sum is less than the value then increase i by one, if the sum is larger decrease j by one, and if the sum is equal to m output true. This will run in $O(n \log n + 2n)$ which is $O(n \log n)$