COS 485 — Homework 6

Samuel Barton Benjamin Montgomery

April 18th, 2017

Problem 1

In this problem we are asked to make a decision tree argument about the lower limit on the worst case number of comparisons needed to find a key in a sorted list.

Let us assume our list has N items. We assume that these items are in sorted order. There are N possible solutions, namely keys, from which we must choose one. Thus the height of the decision tree must be large enough to cover the search space of N items. Since we have sorted the items we have a binary decision, either an item is greater than its predecessor or it is less than its predecessor. Thus we have a binary tree where each node can have at most two children, the height of the tree can be calculated by solving the following equation.

$$N \leq 2^h \implies \lg(N) \leq h \implies h \geq \lg(N)$$

Since the height of the decision tree is equal to the number of needed comparisions, we have that there can be a minimum of $\lg N$ comparisions in the worst case to find a particular key in a sorted list of N elements.

Problem 2

In this problem we consider the case of one miserly king who has demanded a weighing of N coins with the understanding that one coin is counterfeit and lighter than the others.

The algorithm to efficiently solve this is as follows:

1. Divide coins into three parts A, B, and C where the size each is $\lfloor \frac{n}{3} \rfloor$. If N is not divisible by 3, then we will put the 1 or two additional coins in D and set them aside.
2. Weigh two of A, B, or C.

   If $A = B$, then throw out A and B and keep C

   If $A < B$, then throw out B and C and keep A

   If $B < A$, then throw out A and C and keep B
3. Repeat until only one of A, B, and C remain
4. If N is divisible by 3, then we are done. Otherwise, we have either 1 or 2 elements in D to consider. Compare whichever of A, B, or C remains with 1 of the elements in D using the above logic.

In the worst case, N is not divisible by 3 and has a remainder of 2. Thus there are 2 elements in D and we do $\lceil \log_3(n) \rceil$ comparisons to end up with 1 element from A, B, or C and two elements from D to consider. We make one additional comparision with 1 element from D and the remaining element from the prior comparisons, and can then determine which coin is the counterfeit. Thus in the worst case our algorithm will have a time complexity equal to the number of comparisons which is

$$T_n \;=\; \lfloor \log_3(n) \rfloor + 1 \;=\; \lceil \log_3(n) \rceil$$

Problem 3

In this problem we are asked to make a decision tree argument to determine the absolute lower bound for any algorithm to solve the *Miserly King Coin Counterfeit Checking Problem.*

In the problem we are given N coins, and so there are N possible solutions to the problem. Each decision we make has three possible outcomes, so we have a ternary tree. Thus in order to determine the minimum possible height of a decision tree with N leaves we must solve the following equation:

$$N \leq 3^h \implies \log_3 N \leq \log_3 3^h \implies \log_3 N \leq h$$

Thus we see that there must be at least $\log_3 N$ decisions for any algorithm to solve the *Miserly King Coin Counterfeit Checking Problem.*

Problem 4

In order to have a worst-case linear algorithm for finding the mode in an array, consider the following algorithm:

1. Create an empty HashMap in the form of K, V. The type of K is the same as that of the array values, and the type of V is an integer, used to count the number of occurrences of that key.

2. Perform a linear pass on our array. For each element, check to see if it is in our HashMap as a key.

   If it is not in our HashMap, put in the element as the key, and set the HashMap's value to 1 to indicate 1 occurrence.

   If it is in our HashMap, increase the associated value in the HashMap by one, to indicate one more occurrence than before.

3. Perform a linear pass across the set of entries in the HashMap, keeping track of the entry with the largest value. This entry holds the element that is the mode as the key, and the number of times it occurred as the value.

Our analysis is as follows:

1. Creation of the **empty** HashMap is $O(1)$.

2. Our linear pass across the array is, by construction, $T_n = n$. At each element, we perform an $O(1)$ *get* operation. We then perform another $O(1)$ insertion, whether it be an entry that gets inserted, or an existing value that gets updated. This is a total of $T_n = 3n$

3. Our linear pass across the set of entries in the HashMap is also $T_n = n$. In the worst case, there are N entries, so we check n times in total if we have a new max. This is a total of $T_n = 2n$.

Therefore, we can express our time complexity as the following:

$$T_n = O(1) + 3n + 2n \implies T_n \in \Theta(n)$$