CS 5114 Theory of Algorithms, Spring 2020
Homework 3: Due on 23 Feb. 2020, 11:59pm

I pledge that this test/assignment has been completed in compliance with the Graduate Honor Code and that I have neither given nor received any unauthorized aid on this test/assignment.

Name (Print): _____

Signed: _____

1. **(20%)** Consider a hash table of size $m = 1000$ and a corresponding hash function $h(k) = \lfloor m(kA \mod 1) \rfloor$ for $A = (\sqrt{5} - 1)/2$. Compute the locations to which the keys 32, 45, 56, 62, 78, and 90 are mapped (fill the below table).

k	kA	$kA \mod 1$	$m(kA \mod 1)$	$h(k) = \lfloor m(kA \mod 1) \rfloor$
32				
45				
56				
62				
78				
90				

2. **(20%)** Write the TREE-PREDECESSOR procedure in BST (refer to pp. 40-41 in L5).

3. **(20%)** Generate a Huffman tree using the variable-length codeword for the following table for a set of numbers and their frequency in a file (as Figure 16.4b). Also show a tree using the fixed-length codeword. Compare the numbers of bits required to encode the file of these two cases (Note: use a binary number to represent codewords). The solutions should show the same structure of trees and the same number of bits required for both variable length and fixed length codewords, but can show different codewords as far as they are unique.

Frequency	Value
5	1
7	2
10	3
15	4
20	5
45	6

(a) **(10%)** Show the tree with the variable length codewords and the tree with the fixed length codewords; and (b) **(10%)** Fill the below table (check L5, P. 71).

	1	2	3	4	5	6	Total # of bits
Frequency	5	7	10	15	20	45	N/A
Variable							
Fixed							

4. **(20%)** Demonstrate what happens when we insert the keys 5, 28, 19, 15, 20, 33, 12, 17, 10 into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be $h(k) = k \mod 9$.

5. **(20%)** Binary search of a sorted array takes logarithmic search time, but the time to insert a new element is linear in the size of the array. We can improve the time for insertion by keeping several sorted arrays. Specifically, suppose that we wish to support SEARCH and INSERT on a set of n elements. Let $k = \lceil \lg(n+1) \rceil$, and let the binary representation of n be $\langle n_{k-1}, n_{k-2}, \ldots, n_0 \rangle$. We have k sorted arrays $A_0, A_1, \ldots, A_{k-1}$ where for $i = 0, 1, \ldots, k-1$, the length of array A_i is 2^i. Each array is either full or empty depending on whether $n_i = 1$ or $n_i = 0$, respectively. The total number of elements held in all k arrays is therefore $\sum_{i=0}^{k-1} n_i 2^i = n$. Although each individual array is sorted, elements in different arrays bear no particular relationship to each other.

   (a) **(10%)** Describe how to perform the SEARCH operation for this data structure. Analyze worst-case running time.

   (b) **(10%)** Describe how to perform the INSERT operation. Analyze its worst-case running time.