CS 1501 Summer 2017: Quiz 1 Solution

1) **Fill in the Blanks and True/False (22 points -- 2 points each).**
Complete the statements below with the MOST APPROPRIATE words/phrases.

a) Order the following growth rates from smallest (best) to greatest (worst): n^2 nlgn lgn n! 2^n
_____**nlgn < n^2 < n^3 < 2^n < n!** _____

b) We can reduce the run-time of a brute-force algorithm by _____**pruning**_____ branches from its execution tree.

c) Given a **multiway radix search trie** in which **32-bit keys** are compared **4 bits at a time**, the **maximum height** of the tree is _____**8**_____ and **interior nodes** will each have up to __**2^4=16**____ children.

d) Consider an empty **separate chaining hash table** of size M = **100**. If we hash **500 keys** into this table, the average chain length will be _____**5**____ and the worst case chain length will be ___**500**_____.

e) Given an empty hash table of size M and a good hash function, the probability that a random key will be hashed to an arbitrary location k is _____**1/M**_____.

f) An example **text string** and **pattern string** that will produce the **worst case** for the **brute force string matching** algorithm are:

A (text) = ___**AAAAAAAAAAAAAAAAAB**_____

P (pattern) = ___**AAAAB**_____

Indicate whether each of the following is TRUE or FALSE, **explaining why in an informative way for false answers.**

g) In the 8-Queens problem solution, if a queen cannot be placed onto column k, we skip forward to column k+1.
False. In this case we backtrack to column k-1.

h) If we consider the **first level only** of a trie storing words in an English language dictionary, the **memory** required for a **DLB** will be **greater** than the **memory** required for a **multi-way trie.** **True**

i) Due to the **Pigeonhole Principle**, I cannot avoid collisions in hashing if the size of my **key space** is larger than the size of my **hash table**.
True

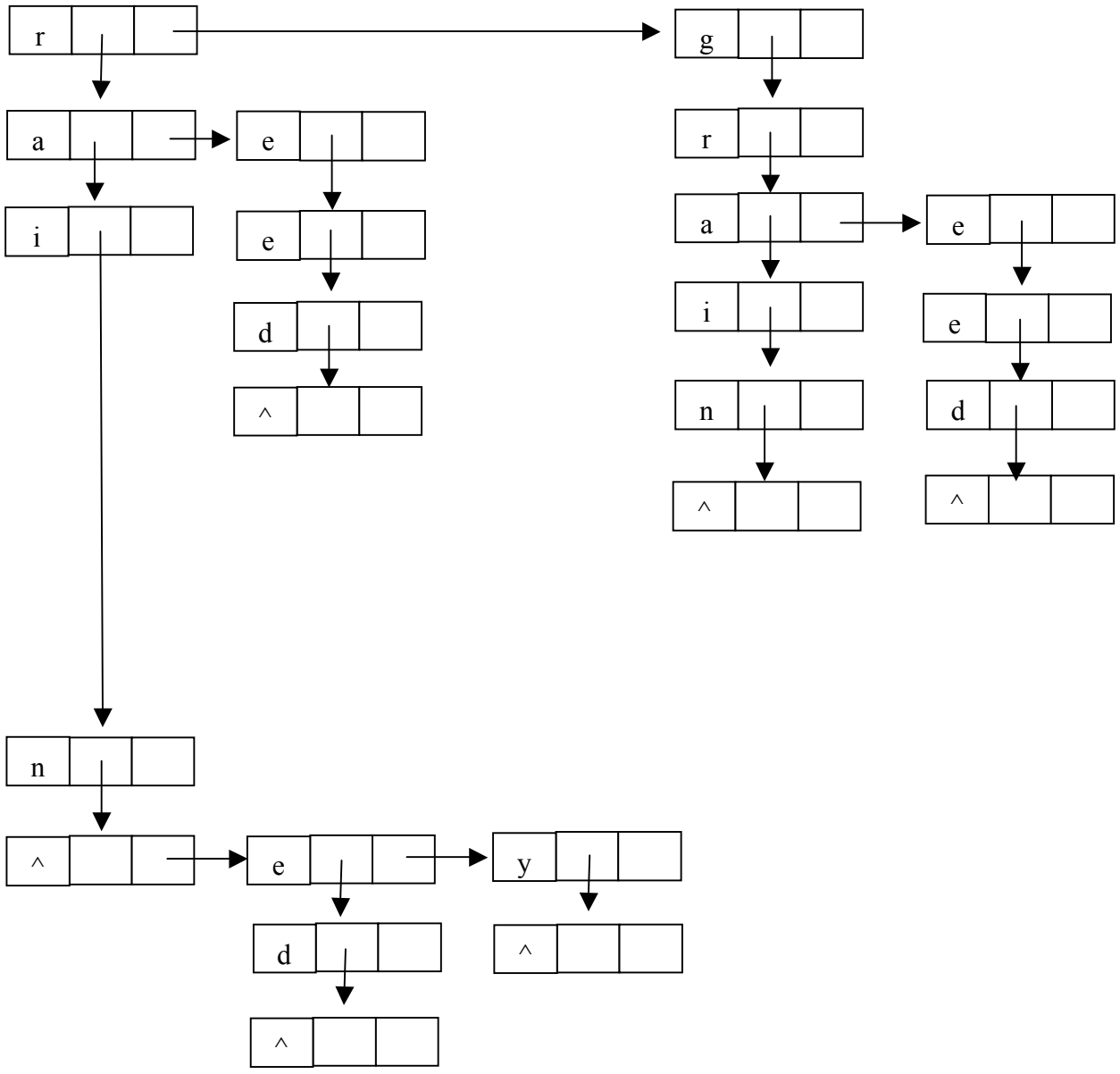
j) Hashing a string by **adding the ASCII values** of its characters results in a **good hash function**, since it utilizes the entire key.
False. It does not take position of the characters into account (among other things).

k) To fix the problem of **clustering** in a **linear probing** hash table, we can change the increment from 1 to a large fixed prime number such as 17 (mod the table size). **False – any fixed increment will cause clustering.**

2) **(8 points)** Consider an **emtpy de la Briandais Tree**, which uses the **lower case letters** (plus a string termination character) as its alphabet, using the implementation that we discussed in lecture. Also consider the following strings: `rain grain rained greed rainy reed`

Draw the **de la Briandais tree** that results after inserting the strings shown **in the order shown above**. Use the format as discussed in lecture for your tree.

Note: Empty squares indicate null references and the ^ is the terminator character



3) **(12 points – 6 + 6)** Consider the two **open addressing** hash tables, with h(x) = x mod 13, shown below. Also consider the following keys (in order): `28, 15, 29, 42, 17, 32`

Linear Probing	
Index	key
0	
1	
2	**28**
3	**15**
4	**29**
5	**42**
6	**17**
7	**32**
8	
9	
10	
11	
12	

Double Hashing	
Index	key
0	**42**
1	
2	**28**
3	**29**
4	**17**
5	
6	**32**
7	**15**
8	
9	
10	
11	
12	

h2(15) = 5

h2(42) = 10

a) Assume that **linear probing** is being used for collision resolution. Show the table after the keys shown above are inserted in the order shown above.

b) Assume that **double hashing** is being used for collision resolution, with increment $h_2(x) = (x \bmod 11) + 1$. Show the table after the keys shown above are inserted in the order shown above.

4) **(8 points)** Consider the algorithm for **finding all single word anagrams of a string (**i.e. all letters must be consumed by a single word). In Assignment 1 you built words one letter at a time, and you pruned your execution tree to stop recursing (and to backtrack) if the current string was not a valid prefix within your dictionary. Let's see if this will actually save any processing with the following test: Assume that you have (by chance) an initial string that happens to be "XQZU". Further, assume that (also by chance) **no words** in your dictionary **begin with X, Q, Z or U**. However, your program does not know either of these things in advance.

   a) Given the pruning implementation stated in Assignment 1, how many strings (prefixes) total will you have to look up in your MyDictionary before determining that no valid anagrams exist? Justify your answer.

   Answer: Each letter will, in turn, be tested as a prefix of a word in the MyDictionary and each test will return "false". Thus, no recursive calls are made and 4 total lookups will be needed.

   b) Now assume that **no pruning** is done, so that you **do not test for prefixes**, just **whole words** after they have been generated. How many strings total will you have to look up in your dictionary before determining that no valid anagrams exist? Thoroughly justify your answer.

   Answer: Since no prefixes are tested, we will not know if a word is in the dictionary until the word itself is generated and tested. Thus, we must try all 4 letter permutations of the letters before we are finished. This is equal to 4! = 4 * 3 * 2 = 24 total word lookups in the dictionary.