Assignment 10 – Search Trees and Hashing

Write pseudo-code not Java for problems requiring code. You are responsible for the appropriate level of detail.

For questions 1 – 4, compare the efficiency of using sequential search on an ordered table of size n and an unordered table of the same size for the key *Item*:

1. **If no record with the key *Item* is present**

2. **If one record with the key *Item* is present and only one is sought.**

3. **If more than one record with the key *Item* is present and it is desired to find only the first**

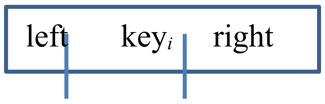
4. **If more than one record with the key *Item* is present and it is desired to find them all.**

5. Show that if k is the smallest integer greater than or equal to n+(log$_2$n)-2, k comparisons are necessary and sufficient to find the largest and second largest elements of a set of n distinct elements. (k comparisons are required to find what you are looking for, but no more than that are needed)

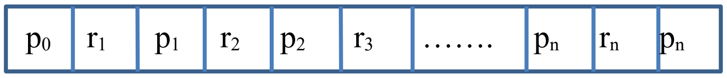
6. How many comparisons are necessary to find the largest and smallest of a set of n distinct elements?

7. Show that every n-node binary search tree is not equally likely (assuming items are inserted in random order), and that balanced trees are more probable than straight-line trees. (This question is asking you to look at the shape of the trees and show that some shapes are more probable than others.**)**

8. Write a method delete(key1, key2) to delete all records with keys between key1 and key2 (inclusive) from a binary search tree whose nodes look like this:



9. Write a method to delete a record from a B-tree of order n.



10. If a hash table contains *tablesize* positions and *n* records currently occupy the table, the load factor *lf* is defined as *n/tablesize.* Suppose a hash function uniformly distributes *n* keys over the *tablesize* positions of the table and the table has load factor *lf*. Show that of new keys inserted into the table, (n-1)lf*/2 of them will collide with a previously entered key. Think about the accumulated collisions over a series of collisions.**

11. Assume that *n* random positions of a *tablesize*-element hash table are occupied, using hash and rehash functions that are equally likely to produce any index in the table. The hash and rehash functions themselves are not important. The only thing that is important is they will produce any

index in the table with equal probability. Start by counting the number of insertions for each item as you go along. Use that to show that the average number of comparisons needed to insert a new element is *(tablesize* + 1*)*/*(tablesize*-n+1). Explain why linear probing does not satisfy this condition.