HW 4

Daniel Sawyer

November 14, 2018

Q1.

The probability of $h(k) = h(l)$, $Pr\{h(k) = h(l)\} = \frac{1}{m}$ of n distinct keys with m slots.
Since there are n distinct keys and $k \neq l$, the number of combinations is $\binom{n}{2} \cdot \frac{1}{m}$
Therefore, since $\binom{n}{2} = \frac{n(n-1)}{2}$, the number of expected collisions is $\frac{n(n-1)}{2m}$

Q2.

With chained hash tables, unsorted, the running time for all of these operations except insertion (which is $O(1)$) would be $O(1+\alpha)$, where $\alpha = \frac{n}{m}$ and n = number of elements, m = slots in hash table. Also, we are assuming a linked list implementation:

Successful searches wouldnt change since you would have to check every element since you cannot run binary search on a linked list. Therefore the running time is unchanged at $O(1+\alpha)$

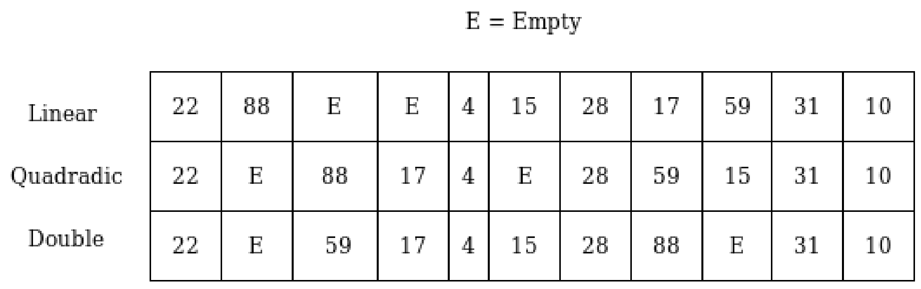
Unsuccessful searches are the same as above, you would have to search each list of the hash table since you cannot use binary search on linked lists. Hence, running time is unchanged with $O(1+\alpha)$

Insertions are normally $O(1)$ since they are just popped onto the beginning of the list. However, since this new implementation is sorted, we must first find its place in the list, which could take up to α time. Therefore, running time is increased for insertion to $O(1+\alpha)$

Deletion, like searches, maintains the original implementation's complexity of $O(1+\alpha)$ since you cannot run binary search on linked lists. Therefore, the complexity is $O(1+\alpha)$

Since we implemented the chains as linked lists, there is no improvement having the chains sorted. However, the insert function would be α slower than a non-sorted implementation. This could be improved by using a binary tree as the chains, but the problem said to use lists, which are slower sorted than non-sorted.

Q3.



E = Empty

Linear	22	88	E	E	4	15	28	17	59	31	10
Quadradic	22	E	88	17	4	E	28	59	15	31	10
Double	22	E	59	17	4	15	28	88	E	31	10

Q4.

Expected number of probes is $\frac{1}{1-\alpha}$ for all $\alpha < 0$.
Unsuccessful Search:

$\alpha = \frac{3}{4}$, probes $= 4$
$\alpha = \frac{7}{8}$, probes $= 8$
Expected number of probues is $\frac{ln(\frac{1}{1-\alpha})}{\alpha}$
Successful Search:
$\alpha = \frac{3}{4}$, probes $= \frac{4}{3} \cdot ln(4)$
$\alpha = \frac{7}{8}$, probes $= \frac{8}{7} \cdot ln(8)$

Q5.

With a min-heap, the node's value is less than or equal to the children. Where a binary tree's node value is greater or equal to left child and less than or equal to the right child.

Heaps properties dont allow it to be able to get sorted order in O(n) time like a binary tree, since the values arent in order whatsoever in a heap. The fastest if could do it is by sorting all the values first, which would take O(nlgn) via some kind of comparison sort.

Q6.

Algorithm 1 Tree Predecessor

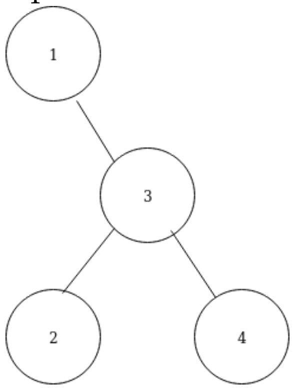
```
1: procedure TreePred(n)
2:
3:     if $n.left \neq NULL$ then
4:         return TreeMax(n.left)
5:     else $k = n.parent$
6:     while $k \neq NULL$ and $n == k.right$ do
7:         $n = k$
8:         $k = k.parent$
       return k
```

Algorithm 2 Tree Max

```
1: procedure TreeMax(n)
2:     while $n.right \neq NULL$ do
3:         $n = n.right$
       return n
```

Q7.

With the tree below, and searching for 4, you would get A = {2}, B = {1, 3, 4}, and C = Empty, since 2 > 1, the professor is wrong



Q8.

If the node has two children, that means there is one less than and one greater than the value of the node. So the successor would be the next number greater than the node, so if there were a left child, it would be the successor since children to the left are less, so the successor cannot have a left child.

The predecessor would be the next value less than the current node. So the predecessor couldnt have a right child, bc then that right child would be $pred \leq pred.right \leq node$, which would make it the right child the predecessor since it would be the next number below the node.

Q9.

Since insertion finds the place of where the key should be inserted, it iterates through the the tree comparing less or greater and going left or right. It is the same way you search the tree. So you would make the same number of comparisons/steps in order to get to the node/key, plus one extra comparison/step on the actual key. So you get the same amount of steps as insert plus one more comparison/step.

Q10.

Worst case, is that the n numbers are in ascending or descending order and the tree would not be balanced at all, it would be a straight line and take $\sum_{i=0}^{n}(i) = \frac{n^2-n}{2}$ which is $O(n^2)$ for a highly unbalanced tree.

Best case, when the tree is balanced, insert only takes $lg(n)$, so with n insertions, best case scenario is $\Omega(nlgn)$