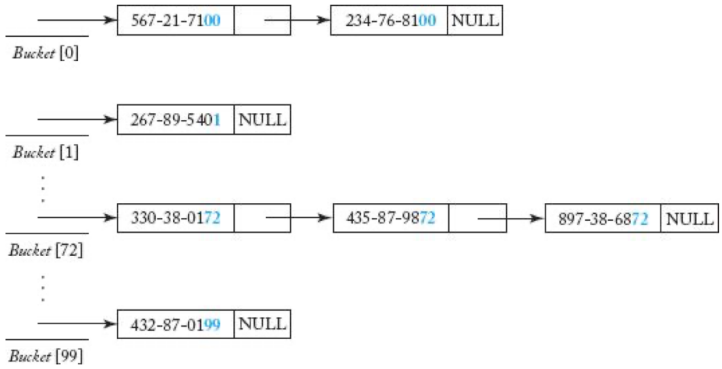
Algorithms Assignment 9

Clare Minnerath

Heapsort implementation assign9.py

26. Implementation: assign9.py
For a heap/complete binary tree of size n, the algorithm does a comparison $\lfloor \frac{n}{2} \rfloor$ times.
So, $T(n) = \frac{n}{2} \in \Theta(n)$

15.
By using Linear Probing (ie. Closed hashing) to solve the instance of the problem shown below with open hashing, we'll show how collisions can be avoided. This type of hashing requires the number of keys to be equal or less than the number of buckets.



Closed Hashing Solution:
(Since there is no order specified above, we'll assume 234-76-8100 arrived prior to 267-89-5401).

Bucket [0]: [567-21-7100]
Bucket [1]: [234-76-8100]
Bucket [2]: [267-89-5401]
...
Bucket [72]: [330-38-0172]
Bucket [73]: [435-87-9872]
Bucket [74]: [897-38-6873]
...
Bucket [99]: [432-87-0199]

This resolves clashing, and rather than searching through a linked list at a clashed bucket, it will search linearly through the buckets.

17. Implementation assign9.py
Say we have a table of size n. In the worst case scenario where the table is full and we attempt to delete a value that hashes to the first bucket, but isn't in the table, we will do n comparisons.
$\implies W(n) \in \Theta(n)$
When the element is not in the table, this algorithm performs the worst.

If the element to delete has the same likelihood of hashing to any of the places in the table, then the average number of comparisons done for deleting an element not in the table is also n because there is no way of knowing the element isn't in a bucket farther along in the table (And i have the table wrap around so that larger numbers that clash will go to smaller hash buckets.
So, $T(n) = n \in \Theta(n)$ for deleting an element not in the table.
(Moral of the story don't try to delete something not in the table with closed hashing)

If the element is in the table our performance improves dramatically. We can expect a slightly worse performance than the open hashing in terms of collisions. This is because when there is a collision, the element isn't added to a linked list but rather put in another bucket increasing the odds for more collision. As shown in the book, for large instances of n, open hashing performs dramatically better than binary search in terms of search time. So we can assume it is not much worse for this case when deleting. This case will also perform better when the table is more sparse. In comparison to open hashing, this table will not use any extra space. All elements will be stored in the table and not linked to from it.