Ally Smith

Homework 3

1. Design a dictionary data structure in which search, insertion, and deletion can all be processed in $O(1)$ time in the worst case. You may assume the set elements are integers drawn from a finite set $1, 2, .., n$, and initialization can take $O(n)$ time.

   We can implement this dictionary using a hash table, which will allow us to have constant time for our search, insert, and delete operations. The search is done using a simple hash table lookup, and the insert/delete operations rely on this table lookup also. In the delete, it is used to find the value to be deleted, and in the insert it is used to check if the value already exists in the set. With this definition, our initialization would take $O(n)$ since it would be $O(1)$ for each of the n elements.

2. Describe how to modify any balanced tree data structure such that search, insert, delete, minimum, and maximum still take $O(\log n)$ time each, but successor and predecessor now take $O(1)$ time each. Which operations have to be modified to support this?

   A balanced binary tree can be modified by adding additional pointers within each node to its respective successor and predecessor. By storing this data, we can access these nodes in constant time. However, this would require us to modify the insert and delete operations so that these pointers stay updated as the tree grows. When inserting a node, we need to adjust the nodes that are the successor and predecessor of the inserted node. When we delete a node, we need to adjust the successor and predecessor to go over the deleted node, effectively removing it.

3. In the *bin-packing problem*, we are given n metal objects, each weighing between zero and one kilogram. Our goal is to find the smallest number of bins that will hold the n objects, with each bin holding one kilogram at most.

   The *best-fit heuristic* for bin packing is as follows. Consider the objects in the order in which they are given. For each object, place it into the partially filled bin with the smallest amount of extra room after the object is inserted.. If no such bin exists, start a new bin. Design an algorithm that implements the best-fit heuristic (taking as input the n weights $w_1, w_2,$ ... , w_n and outputting the number of bins used) in $O(n \log n)$ time.

```
bins = []
for object in objects:
    minSpaceRemaining = $\infty$
    binIndex = -1
    for bin in existingBins:
        currentSpaceRemaining = bin.spaceRemaining - object.mass
        if currentSpaceRemaining < minSpaceRemaining:
            minSpaceRemaining = currentSpaceRemaining
            binIndex = bin.index
    if minSpaceRemaining == $\infty$:
        // case for items not fitting into an existing bin
        create new bin
        add current item to bin
        add bin to bins
    else :
        // general case for adding to best-fit bin
        add item to bins[binIndex]
return bins.size
```

4. Determine whether a linked list contains a loop as quickly as possible without using any extra storage. Also, identify the location of the loop.

   This can be implemented without using any additional storage by simulatneously looping through the list at two different speeds. If the values in these fast and slow loops intersect, then there is a loop in the list. If either of them reach *null* then we know that there isn't a loop. Pseudocode below:

```
boolean hasLoop(Node head):
    slow = fast = head
    while True :
        slow = slow.next
        if fast.next $\neq$ null:
            fast = fast.next.next;
        else return False

        if slow == null or fast == null:
            return False
        if slow == fast:
            return True
```