CIS 122 Summer 2019
Challenge Problems – Set 5

Challenge problems will not be guaranteed to follow our material exactly – meaning that solutions to these problems may involve concepts or techniques that we haven't seen yet, or may not formally cover at all. Additionally, these problems are neither required nor testable material.

This week's theme – complexity and search algorithms – **Binary Search.**

"Complexity" in computer science refers to the amount of work that an algorithm has to perform *in terms of the size of its input* before it arrives at an answer. If we consider the size of an input to be *n* (for example, n items in a list or a database), then we can refer to several complexity classes. We usually use "Big O" notation when speaking of complexity.
O(1) is constant time – the work performed by the algorithm is constant no matter what the size of the input. For example, taking the *first* item out of a list is independent of the size of the list, and so has constant time complexity.
O(n) is linear time – the work performed by the algorithm is linearly proportional to the size of the input. For example, if we are searching a list for a value, we might have to examine all *n* items in the list, so our complexity is linear with respect to the size of the input.
$O(n^2)$ is quadratic time – for example, many sorting algorithms have to run through an entire list *for every item in the list*, which is *n * n* total work, quadratic with respect to the size of the input.

Wikipedia has a lot more to say about this: https://en.wikipedia.org/wiki/Time_complexity

Searching is often a O(n) activity – we have to look at everything to know definitively if we've found our target or not. However, if our list or data structure is *sorted*, we can take advantage of this delicious information and perform a search more efficiently. Binary search is one such efficient search algorithm, which lowers the search complexity from O(n) to O(log n).

Read this wiki about binary search: https://en.wikipedia.org/wiki/Binary_search_algorithm

And then implement the algorithm in two ways – once iteratively and once recursively. The functions should have two parameters, a list and a target, and should return True or False depending on whether the target is in the list or not. They should run in O(log n).

Notice that binary search is inherently recursive – searching the upper or lower sublist around some median point is the exact same process as searching the original list.