Machine Learning Assignment 51

Elijah Tarr

February 24, 2021

Problem 1

(a) *Suppose you have a sorted list of 16 elements. What is the greatest number of iterations of binary search that would be needed to find the index of any particular element in the list? Justify your answer.*

We have list [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] Imagine we need to find the index of the item 0. First, the search would go to the middle of the list and compare with 0. 8 ¿ 0, so we do this for the lower half of the list.

Go to the middle of the list and compare with 0. 4 ¿ 0, so we do this for the lower half of the list.

Go to the middle of the list and compare with 0. 2 ¿ 0, so we do this for the lower half of the list.

Go to the middle of the list and compare with 0. 1 ¿ 0, so we do this for the lower half of the list.

The lower half of the list only contains 0, and that's what we want to find, so return the index of that element. We get 0.

This took 4 iterations to get to 0. Items are found quicker if they are a midpoint of the whole list, or a half of the list, or a quarter of the list, etc. Therefore it would take the most time to get to an element which is the midpoint of the smallest fraction of the list, which could be the leftmost, rightmost, one-away-from-middlemost item, etc. Since 0 is a worst-case scenario, the maximum amount of iterations it can take is 4.

(b) *State and justify a recurrence equation for the time complexity of binary search on a list of n elements. Then, use it to derive the time complexity of binary search on a list of n elements.*

To start, we have $f(n) = O(1) + f(\frac{n}{2})$ $f(\frac{n}{2}) = O(1) + f(\frac{n}{4})$ $f(\frac{n}{4}) = O(1) + f(\frac{n}{8})$ And so on.

This can be modeled as: $f(n) = kO(1) + f(\frac{n}{2^k})$ The only value of f we know for sure is $f(1)$ because it only takes 1 operation to search in a list of length 1. Therefore we have $f(1) = 1$, and if we set $\frac{n}{2^k} = 1$, then we can solve for k. We have $k = \log_2 n$. Plugging this in we have:

$$f(n) = O(1)\log_2 n + f(1) = O(\log_2 n + 1) \qquad = O(\log n)$$

The reason we can just remove the constant and log base 2 is because big O time is proportional, meaning constants don't matter.

```
from otest import do_assert

def binary_search(x, l):
    m = len(l)//2
    return m if x == l[m] else binary_search(x, l[m:] if x > l[m] else l[:m])

do_assert("Binary Search Test", binary_search(
    7, [2, 3, 5, 7, 8, 9, 10, 11, 13, 14, 15, 16]), 3)
```