Homework 1

Joni Vrapi

09/10/2022

Statement of Integrity: I, Joni Vrapi, attempted to answer each question honestly and to the best of my abilities. I cited any, and all, help that I received in completing this assignment.

Problem 1. For this problem we are asked to describe the time complexity of the stated linear search algorithm, showing the tightest asymptotic representation from Θ, O, or Ω. My first inkling on this was to heed the name of the algorithm and assume that it is, in fact, linear in time. This algorithm accepts a parameter x, which is the thing that is being searched for, and another parameter A, which is the array that is to be searched. If $x \in A$, the index of x is returned.

It begins by iterating through the array, checking if the indexing variable $i \leq A.length$ and that $x \neq A[i]$. This, to me, implies that in the *worst case*, where the item that you are searching for is not in input the array, this algorithm will loop through every item in the array for a total of n times for a time complexity of $O(n)$. Finally, it checks if $i \leq A.length$ and assigns to the *location* variable either the indexing variable i or 0 if $i > A.length$. This implies that the *best case* scenario for this algorithm, where the item that you are looking for is in the first position of the array, is $\Omega(1)$. For the average case, we have two cases:

1. The item can be in any of n possible positions in the array, from index 1 to index $n + 1$.
2. The item is not in the array.

So, there are n possibilities in the first case, and 1 possibility in the second case, for a total of $n+1$ possibilities. If the item you are looking for is at index k, linear search will do $k + 1$ comparisons. Summing those up, you get the known sum of [1]:

$$\frac{n(n+1)}{2} \tag{1}$$

In the second case, there are n possible positions in the array, and all of them are searched, with no result being found. Therefore, the total amount of cases for both situations are:

$$\frac{n(n+1)}{2} + n \tag{2}$$
$$= n(\frac{n+1}{2} + 1) \tag{3}$$

For the average number of comparisons we have

$$\frac{n(\frac{n+1}{2} + 1)}{n+1} \tag{4}$$
$$= \frac{n}{2} + \frac{n}{n+1} \tag{5}$$

The dominant term here is the $\frac{n}{2}$, so the average case complexity here is also $O(n)$. What is the tightest asymptotic representation? Because of the best case complexity here being $\Omega(1)$ and the worst case being $O(n)$ as well as:

$$\Theta(f(n)) \Leftrightarrow (\Omega(f(n)) \wedge O(f(n))) \tag{6}$$

Which was pulled from [2], we can not have an asymptotically tight bound. I would therefore argue that the tightest bound here is $O(n)$.

Problem 2a. For this problem we are asked to describe the time complexity of the stated binary search algorithm. This is a wicked interesting searching algorithm because it begins with an item to be found, x, and a *sorted* input array A. It works by comparing x to the value in the middle of A, and then narrowing the search to only the left/right half of A (depending on if the value of $x > A[i] \vee x < A[i]$), and then repeating itself. This continues until either the value is found, or the interval of the array that you are looking at is empty, at which point it returns either the index of the found item, or zero if it's not found. This is an excellent strategy for finding things, because whenever you are able to reduce your problem size on each iteration, you are talking about logarithmic time (which is faster than linear time!). The question, however, begets a precise asymptotic bound, but knowing that the problem size is being reduced gives me a hint that my answer should have a $log(n)$ in it.

So, let's think about this. The *best case* scenario would be if $x = A[\frac{A.length}{2}]$, making the algorithm exit after only 1 comparison, leading to an $\Omega(1)$. For the *worst case*, letting $B =$ the interval of the array we are searching, we have

- On the first iteration: $B.length = n$.
- On the second iteration: $B.length = \frac{n}{2}$.
- On the third iteration: $B.length = \frac{n}{2^2}$.
- On the k^{th} iteration: $B.length = \frac{n}{2^k}$.

We also know that after k iterations, $B.length = 1$,

$$\therefore \frac{n}{2^k} = 1 \tag{7}$$

Applying the log function to both sides yields

$$log(n) = log(2^k) \implies log(n) = klog(2) \tag{8}$$

$$\therefore k = log(n) \tag{9}$$

This will therefore yield a worst case time complexity of $O(logn)$ for binary search. Due to (6), and considering the best and worst case complexities being different, I would argue the tightest bound is $O(logn)$.

Problem 2b. This algorithm makes use of the $\lfloor floor \rfloor$ function to determine the middle index of the array interval that it is currently looking at. While the floor function runs in constant time, its use can be eliminated by a clever manipulation of the middle index. From [3] we see that we can

```
BINARY-SEARCH(x, A)
 1  i = 1
 2  j = A.length
 3  while i < j
 4        m = (i + j)/2
 5        if x > A[m]
 6              i = m + 1
 7        else j = m − 1
 8        if x = A[i]
 9              location = i
10        else location = 0
11  return location
```

...change line 7 in the algorithm to subtract 1 from m instead of setting $j = m$, resulting in the same algorithm as before, but without use of the $\lfloor floor \rfloor$ function. This does not change the overall time complexity of the algorithm, keeping it at $O(logn)$, but it does make it run slightly faster.

Problem 3. For this problem we are asked to solve the recurrence relation

$$T(n) = 4T(\frac{n}{4}) + n^2 \tag{10}$$

The first thing that pops into my mind looking at this problem is that the Master Theorem applies, and will likely be the easiest way of proving this. From the master theorem

$$T(n) = aT(\frac{n}{b}) + f(n) \tag{11}$$

We can see that $a = 4$, and $b = 4$. Since $a > 1$, $b > 1$, and f is asymptotically positive, we can see that $n^{\log_a b} \implies n^{\log_4 4} \implies n$ and $f(n) = n^2$. The master theorem tells us to then check whether $4f(\frac{n}{4}) \leq cf(n)$ for some $c < 1$ and $\forall n$ sufficiently large which is indeed the case here.

$$\therefore T(n) \implies \Theta(f(n)) = \Theta(n^2) \tag{12}$$

References

[1] "List of mathematical series." https://en.wikipedia.org/wiki/List_of_mathematical_series. Accessed on 2022-09-11.

[2] G. D. Luca, "The difference between lower bound and tight bound." https://www.baeldung.com/cs/lower-bound-vs-tight-bound. Accessed on 2022-09-11.

[3] https://stackoverflow.com/questions/27655955/why-does-binary-search-algorithm-use-floor-and-not-ceiling-not-in-an-half-open. Accessed on 2022-09-11.