Jose Marquez Jaramillo
EN.605.621 Foundations of Algorithms - Spring 2023
February 6, 2023

Homework 1

Question 1. Describe the time complexity of the linear search algorithm. Choose the tightest asymptotic representation, from Θ, O, or Ω and argue why that is the tightest bound.

Algorithm 1. Linear Search

Input: sorted array A (indexed from 1), search item x
Output: index into A of item x if found, zero otherwise

```
1:  function LINEAR-SEARCH(x, A)
2:      i = 1
3:      n = len(A)
4:      while i <= n and x ≠ A[i] do
5:          i = i + 1
6:          if i ≤ n then
7:              loc = i
8:          else
9:              loc = 0
10:         return loc
```

Considering the algorithm as depicted in Algorithm 1, we can divide its time complexity between the best and worst time complexities:

- **Best case:** As the while loop traverses the array, the best case is given when the first item encountered is the first item being searched for. This causes the best case of the linear search to have constant or $O(1)$ time complexity.
- **Worst case:** The worst case time complexity is given when the while loop needs to traverse the entire array without finding a solution. This will depend on the length of the array or n, therefore the worst time complexity is given by $O(n)$.

Now, in order to find the asymptotically tight bound, we can direct our attention toward Θ notation and how it can be derived based on Theorem 3.1 from [Cormen et al., 2022]. Denote $f(n)$ the function of Linear Search in Algorithm 1. Let's see if $g(n) = n$ satisfies $f(n) = O(n)$ and $f(n) = \Omega(n)$:

- $f(n) = O(n)$: This was already casually established but for completeness, we will follow the method of the book. In the worst case, we need to have a positive constant c and n_0 such that $n \leq cn$. Considering that $n > 1$ in general, we have that $c \geq 1$.
- $f(n) = \Omega(n)$: At the lower bound, we would have $n_0 = 1$ and the best case is given when $n_0 = n = 1$. Let's consider if there is a value for c such that $0 \leq 1 \leq cn$. Just as before, solving for c we get that the value $c = 1$ satisfies the condition.
- **Conclusion**: Because $g(n) = n$ satisfies $f(n) = O(n)$ and $f(n) = \Omega(n)$ we can say that $f(n) = \Theta(n)$.

Question 2. Analyze the following binary search algorithm. Assume the input A is a sorted list of elements; x may or may not be in A.

Algorithm 2. My Binary Search Algorithm

Input: sorted array A (indexed from 1), search item x
Output: index into A of item x if found, zero otherwise

```
1:  function BINARY-SEARCH(x, A)
2:      i = 1
3:      j = len(A)
4:      while i < j do
5:          m = ⌊(i + j)/2⌋
6:          if x > A[m] then
7:              i = m + 1
8:          else
9:              j = m
10:     if x = A[i] then
11:         loc = i
12:     else
13:         loc = 0
14:     return loc
```

1) Describe the time complexity of the following binary search algorithm in terms of number of comparisons used ignore the time required to compute m = b(i+j)=2c). Choose the tightest asymptotic representation, from Θ, O, or Ω and argue why that is the tightest bound.

   In this case it is beneficial to consider an example. Denote the ordered array A as
   |-4|-3|-2|-1|0|1|2|3|4| and consider the following cases:
   - Let us try to find the object right at the middle of the array, that is $x = 0$. As we go into the while loop, the array gets subset to |-4|-3|-2|-1|0|. Subsequently, it gets subset to |-1|0| and finally it gets subset to |0|. At this point, the pointer 5 is returned. This means that the array is traversed a total of 3 times. Each time the array is divided in half.
   - Let us try to find the object right at the end of the array, that is $x = 4$. As we go into the while loop, the array gets subset to |0|1|2|3|4|. Subsequently, it gets subset to |3|4| and finally it gets subset to |4|. At this point, the pointer 9 is returned. This means that the array is traversed a total of 3 times. Each time the array is divided in half.
   - The second case is repeated when we look for the first item in the array, or $x = -4$. As we go into the while loop, the array gets subset to |-4|-3|-2|-1|0|. Subsequently, it gets subset to |-4|-3| and finally it gets subset to |-3|. At this point, the pointer 1 is returned. This means that the array is traversed a total of 3 times. Each time the array is divided in half.

   By these three cases we can see that the algorithm has best and worst time complexity of $O(\log n)$. Because the algorithm always runs with complexity of $O(\log n)$, then the algorithm is $f(x) = O(\log n) = \Omega(\log n) = \Theta(\log n)$.

2) Make one small change to the algorithm above to improve its runtime, and give the revised tightest asymptotic representation, from from Θ, O, or Ω. Show your change using proper pseudocode.

Algorithm 3. Modified binary search algorithm

Input: sorted array A (indexed from 1), search item x
Output: index into A of item x if found, zero otherwise

```
1:  function MOD-BINARY-SEARCH(x, A)
2:      i = 1
3:      j = len(A)
4:      while i < j do
5:          m = ⌊(i + j)/2⌋
6:          if x == A[m] then                    ▷ Check if midpoint is x
7:              return m                         ▷ Return the pointer of the midpoint
8:          if x > A[m] then
9:              i = m + 1
10:         else
11:             j = m
12:     if x = A[i] then
13:         loc = i
14:     else
15:         loc = 0
16:     return loc
```

Algorithm 3 corresponds to a modified version of Algorithm 2. In lines 6 and 7 of Algorithm 3 you will find a step that checks if once initialized, the midpoint of the array contains the value that is being searched. If so, the index for the mid-point m is returned. This modification changes the best-case running time of the algorithm. Now, if the item that is being searched for is in the middle of the array, the algorithm runs in constant time. This means that $f(n) = \Omega(1)$.

Question 3. Use the Master Theorem to find the asymptotic bounds of $T(n) = 4T(n/4) + n^2$.

Based on the Master Theorem (Theorem 4.1) on [Cormen et al., 2022] we can denote:

$$\begin{aligned} T(n) &= aT(n/b) + f(n) \\ &= 4T(n/4) + n^2 \Rightarrow a = 4, b = 4, f(n) = n^2 \end{aligned} \tag{1}$$

We know that $f(n) = O(n^2) = \Omega(n^2) = \Theta(n^2)$. Therefore by the third characterization of the Master Theorem we should find a value for ϵ such that $f(n) = \Omega(n^{log_b a+\epsilon})$ which gives:

$$\begin{aligned} f(n) &= \Omega(n^{log_b a+\epsilon}) \\ &= \Omega(n^{log_4 4+\epsilon}) \\ &= \Omega(n^{1\times log_4 \epsilon}) \Rightarrow \epsilon = 2 \end{aligned} \tag{2}$$

Next we need to satisfy the regularity condition $af(n/b) \leq cf(n)$ for some constant $c < 1$:

$$\begin{aligned} af(n/b) \leq cf(n) &= 4(n/4)^2 \leq cn^2 \\ &= \frac{4}{16}n^2 \leq cn^2 \\ &= \frac{1}{4} \leq c \Rightarrow c < 1 \end{aligned} \tag{3}$$

By the third condition of the Master Theorem, we can conclude that $T(n) = \Theta(f(n)) = \Theta(n^2)$.

Question 4. Use the Master Theorem to find the asymptotic bounds of $T(n) = 3T(\frac{n}{3} + 1) + n$. *Hint:* use a substitution to handle the "+1" term.

Consider the following

$$\begin{aligned} T(n) &= 3T\left(\frac{n}{3} + 1\right) + n \\ &= 3T\left(\frac{n+3}{3}\right) + n \end{aligned} \tag{4}$$

Where for large quantities of n (asymptotically) we can indicate that adding 3 to the value does not impact the running time complexity. We, therefore, proceed by dropping the three and leaving the recurrence as $T(n) = 3T(\frac{n}{3}) + n$. From this recurrence, we have $a = 3$ and $b = 3$, which implies that $n^{\log_3 3} = \Theta(n)$. This takes us to case 2 of the master theorem. We need to find $k \geq 0$ such that $f(n) = n = \Theta(n^{\log_3 3} lg^k n)$ which implies that $k = 0$. Therefore $T(n) = \Theta(n^{\log_3 3} \lg^{k+1} n) = \Theta(nlgn)$

Question 5. Collaborative Problem –CLRS 2-1: Although merge sort runs in $\Theta(nlgn)$ worst-case time and insertion sort runs in $\Theta(n^2)$ worst-case time, the constant factors in insertion sort can make it faster in practice for small problem sizes on many machines. Thus, it makes sense to coarsen the leaves of the recursion by using insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which n=k sublists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

1) Prove that insertion sort can sort the n/k sublists, each of length k, in $\Theta(nk)$ worst-case time.

   Considering that insertion sort runs at $\Theta(n^2)$ as specified above, sorting n/k lists should take $\frac{n}{k}\Theta(n^2) = \Theta(\frac{n}{k}n^2) = \Theta(nk)$.

2) Prove how to merge the sublists in $\Theta(nlg(n = k))$ worst-case time.

   Considering that we will have n elements distributed in n/k sorted sub-lists each with k elements, we would need to merge these lists by two at a time to get a single list of length n. This operation will take $\Theta(lg(n/k))$. On every step we compare n, It is therefore the case that the whole procedure will run at $n \times \Theta(lg(n/k)) = \Theta(n \times lg(n/k)$

3) Given that the modified algorithm runs in $\Theta(nk + nlg(n/k))$ worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as standard merge sort, in terms of Θ-notation?

   From the problem statement we know that standard merge sort runs at $\Theta(nlgn)$ while the modified algorithm runs at $\Theta(nk + nlg(n/k))$. For both to be the same we would need to find values of k and n for which these two magnitudes are the same. Let us consider the time complexity of the modified algorithm closer:

$$\begin{aligned} \Theta(nk + nlg(n/k)) &= \Theta(nk + nlgn - nlgk) \\ &= \Theta(nlgn + nlgn - nlg(lgn)) \rightarrow k = lgn \\ &= \Theta(2nlgn - nlg(lgn)) \end{aligned} \tag{5}$$

   Considering only the highest order coefficient and ignoring the constant, these two expressions should be asymptotically equivalent and therefore k as a function of n is $k = lg(n)$.

4) Why would we ever do this—why not just use merge sort? Argue it the other way, too—what are some problems with using our modified method?

   In previous sections of this question, we have shown that there are efficiencies to be made by running the modified algorithm. Therefore, in practice, k should be the largest list length for which insertion sort runs faster than merge sort.

References

[Cormen et al., 2022] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2022). *Introduction to Algorithms, Fourth Edition*. MIT Press.