Foundations of Algorithms, Spring 2023
Homework #1

All members of the collaboration group are expected to participate fully in solving collaborative problems. Note, however, that each student is required to write up their solutions individually. Common solution descriptions from a collaboration group will not be accepted. Furthermore, to receive credit for a collaboration problem, each student in the collaboration group must actively and substantially contribute to the collaboration. This implies that no single student should post a complete solution to any problem at the beginning of the collaboration process.

1. [15 points] 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                                   ▷ Arrays typically start at index 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
```

2. [20 points total] 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. Binary Search

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                                   ▷ lower search bound
3:      j = len(A)                              ▷ upper search bound
4:      while i < j do                          ▷ while places remain to be checked
5:          m = ⌊(i + j)/2⌋                     ▷ assign the midpoint
6:          if x > A[m] then
7:              i = m + 1                       ▷ increase lower to mid
8:          else
9:              j = m                           ▷ decrease upper to mid
10:     if x = A[i] then
11:         loc = i
12:     else
13:         loc = 0
14:     return loc
```

   (a) [10 points] Describe the time complexity of the following binary search algorithm in terms of number of comparisons used (ignore the time required to compute $m = \lfloor (i+j)/2 \rfloor$). Choose the tightest asymptotic representation, from Θ, O, or Ω, and argue why that is the tightest bound.

   (b) [10 points] Make one small change to the algorithm above to improve its runtime, and give the revised tightest asymptotic representation, from Θ, O, or Ω. Show your change using proper pseudocode. (If you

like, the LaTeX code to this algorithm can be found on the Overleaf website[1]. You may find this useful for writing algorithms in future assignments.) If the asymptotic representation changed from your answer to **??**, argue why it is different.

3. [15 points] Use the Master Theorem to find the asymptotic bounds of $T(n) = 4T(n/4) + n^2$.

4. [15 points] Use the Master Theorem to find the asymptotic bounds of $T(n) = 3T\left(\frac{n}{3} + 1\right) + n$. *Hint:* use a substitution to handle the "+1" term.

5. [35 points] ***Collaborative Problem*** –CLRS 2-1: Although merge sort runs in $\Theta(n \lg n)$ 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.

   (a) [10 points] Prove that insertion sort can sort the n/k sublists, each of length k, in $\Theta(nk)$ worst-case time.

   (b) [10 points] Prove how to merge the sublists in $\Theta(n \lg (n/k))$ worst-case time.

   (c) [10 points] Given that the modified algorithm runs in $\Theta(nk + n \lg (n/k))$ 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?

   (d) [5 points] 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?

[1] Full URL: https://www.overleaf.com/read/gsjhctdjbbsw