Foundations of Algorithms Homework 1 FA25

Copyright © 2025 Johns Hopkins University. Homework assignments are for use only by students enrolled in this section/course. Making assigned problems or solutions available, in whole or in part, by posting, uploading, or distributing online, to anyone else by any means, and obtaining solutions by similar means are prohibited. You may not represent the work of others as your own, including copying/paraphrasing answers, in whole or part, from tutors, generative artificial intelligence (AI), former students and the like. Such acts constitute plagiarism and violate the graduate academic conduct policies.

Strategies for solving ***Collaborative Problem*** problems are to be formed by your assigned group, with each member *participating and contributing fully*. All answers must be your own work, in your own words, even on collaborative problems—collaborate on strategies only. Individual participation will be evaluated through collaborative assessments, submitted when the homework is due.

1. [15 points total] Describe the time complexity of the following 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. [25 points total] This problem was posed by Professor Hamid Sarbazi-Azad, who invented an algorithm that he called *stupid sort.* For this problem, we assume it sorts integers in order from least to greatest. Here's a version of it:

Algorithm 2. Stupid Sort

Input:array A of n integers (indexed from $0..n-1$)

```
1:  function STUPID-SORT(A)
2:      i = 0
3:      while i < n − 1 do
4:          if A[i] > A[i + 1] then
5:              swap A[i], A[i + 1]
6:              if i > 0 then
7:                  i = i − 1                   ▷ move backward
8:          else
9:              i = i + 1                       ▷ move forward
```

The running time of this algorithm depends on its input. Keep that fact in mind for your answers.

(a) [10 points] In terms of input length n, use **asymptotic notation** to express the best case running time of stupid sort, **and** give an example input of 5 integers that would cause this case to happen.

(b) [10 points] Give the *worst* case running time of stupid sort in asymptotic notation, **and** give an example input of 5 integers that would cause this case to happen.

(c) [5 points] Using Θ, Ω, or big-O notation, how would you express the general running time of this algorithm? Briefly explain why. (HINT: if it takes the same time to run everywhere, then it's a Θ relationship. Otherwise, it's either O if there only is a worst case that bounds the behavior from above, or Ω if there only is a best case that bounds the behavior from below.)[1]

[1] We try to sneak a little learning into these homework problems. Asymptotic notation is the most important thing to take away from this course.

3. [10 points total] This problem was posed by a student in a group discussion. Often when writing code, we see a double-nested loop pattern. In some cases, the inner loop runs a constant number of times. For example:

Algorithm 3. Double Loop

Input: array A of n items, some constant k

```
1: function DOUBLE-LOOP(A)
2:     for i = 1 to n do
3:         for j = 1 to k do
4:             some costly calculation involving only A_i
5:             some calculation involving A_i and A_j
6:     return something
```

   (a) [5 points] Using the tightest asymptotic representation, from Θ, O, or Ω, derive the asymptotic time complexity of the loop above in terms of k and/or n. HINT: consider whether you need k in your answer.

   (b) [5 points] Suppose you are running this loop on a low-powered system, e.g., an Arduino board. What change could you make to the logic above to speed up the operation, and what overall improvement would you gain in running time?

4. [10 points total]

Accepted Master Method

For all problems on the Master Theorem, you are to use the method as given in the textbook, and you must show all justifications for the case (1, 2, or 3) that you've chosen; if something says "for any constant ϵ," show us actual values. If something says "$f(n) < O(n^2)$," show us the definition of $O(n)$ and give us values for each constant that satisfy the definition.

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

5. [10 points total] The Fibonacci sequence is one where the ith value F_i is computed as the sum of the values F_{i-1} and F_{i-2}, where the initial two values F_1 and F_2 are 1 and 1, respectively. Suppose someone used different initial values F_1 and F_2, and told you the 5th number is 20. They also told you that the two values are both positive integers, and that $F_2 \geq F_1$. What initial values did they use, and what is the next number in this *Fink-onacci*[2] series? Remember to show your work.

6. [30 points total] ***Collaborative Problem:*** CLRS 2-1 (4th ed): 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) [9 points] Prove that insertion sort can sort the n/k sublists, each of length k, in $\Theta(nk)$ worst-case time.

   (b) [9 points] Prove how to merge the sublists in $\Theta(n \lg (n/k))$ worst-case time.

   (c) [9 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) [3 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?

[2] Prof. Fink wrote this one.