

Analysis of Algorithms
CS 376, Spring 2023

Problem Set 1

Due **AT THE BEGINNING OF CLASS** Monday, March 6

- From your textbook (CLRS), please read Chapters 2 and 3.
- The style guidelines for writing algorithms given on PS0 continue to apply. In general, unless explicitly instructed otherwise, these guidelines apply to all exercises in CS376 (Problem Sets, Smaller Assignments, ...) whether or not they explicitly appear as part of the assignment itself.
- *A general note for CS376:* When writing up your homework, please write neatly and **explain your answers clearly**, giving all details needed to make your answers easy to understand. Clear communication *is* the point, on every assignment.

In general in CS376, unless explicitly specified otherwise, answers should be accompanied by explanations; as part of this, in general, when you present an algorithm, give an accompanying correctness argument (i.e., proof). Answers without explanations may not receive full credit. Please feel free to ask me any questions about explanations that might come up!

Exercises

1. **Recursive Insertion Sort!** In this exercise, you'll write a pseudocode algorithm for a recursive version of Insertion sort, a different way of expressing the same underlying algorithmic idea as the iterative version from class.

We'll do this in two parts, ending up with an algorithm to sort LLists of numbers (i.e., using the LList data structure from class for the sequence to be sorted). For this exercise, sorting is taken to mean in non-decreasing order.

- (a) Write a recursive *LLInsert* algorithm that inserts a number x in the proper location in a sorted LList L .

```
# Input: Number  $x$  and sorted LList  $L = [a_0, a_1, \dots, a_n]$ ,  
#           where  $a_0 \leq a_1 \leq \dots \leq a_n$   
# Output: List  $L' = [b_0, b_1, \dots, b_{n+1}]$  containing input  $x$  and the  
#            $n + 1$  elements of  $L$ , in sorted order  
#            $b_0 \leq b_1 \leq \dots \leq b_{n+1}$ 
```

- (b) Using the *LLInsert* function, write a recursive *LLInsertionSort* algorithm that takes an LList L of numbers, possibly unsorted, and returns a sorted LList L' with the same elements as L in sorted order, consistent with the specification of the sorting problem.

```
# Input: LList  $L = [a_0, a_1, \dots, a_n]$   
# Output: List  $L' = [b_0, b_1, \dots, b_n]$  containing exactly the  
#           elements of  $L$ , in sorted order  $b_0 \leq b_1 \leq \dots \leq b_n$ 
```

As usual, explain your algorithms and give correctness arguments.

2. Consider this pseudocode algorithm for the sorting method *Selection Sort*. (The specification for this sorting algorithm is the same given in class for Insertion Sort, and for Bubble Sort.)

```

SELECTIONSORT(A[1...n])
  for i = 1 to length[A] - 1
    min = i
    for j = i + 1 to length[A]
      if A[j] < A[min]
        min = j
    // the next 3 lines swap A[i] and A[min], using a temporary variable
    temp = A[i]
    A[i] = A[min]
    A[min] = temp

```

Given the following proposed loop invariant for the outer for loop of SELECTIONSORT, give a correctness proof for the algorithm (including a proof of the validity of the invariant).

Proposed loop invariant for SELECTIONSORT:

Subarray $A[1..i-1]$ contains the $i-1$ smallest elements of A in sorted order, and $A[i..n]$ consists of the remaining values of A (no constraint on order).

3. List the following functions of n according to their order of growth—that is, how fast each function grows as n gets big—from lowest to highest:

$(n-2)!$, $5\lg(n+100)^{10}$, 2^{2n} , $0.001n^4 + 3n^3 + 1$, $\ln^2 n$, $\sqrt[3]{n}$, 3^n .

(As is conventional, the $\lg$ function is logarithm base 2; the $\ln$ function is the *natural logarithm*, logarithm base e ; and $\ln^2 n$ is common notation for $(\ln n)^2$.) Although you don't need to explain every part of the ordering for this exercise, please give short explanations (1–2 sentences) for the following:

- (a) how you know the second-smallest comes before the third-smallest; and
 - (b) how you know the second-largest comes after the third-largest.
4. Prof. E. Nigma of the Portland Institute of Technology (which does not actually exist; nonetheless, its motto is “Our CS Department is the PIT’s!”) hired you to analyze the algorithm given here in pseudocode, but as usual, Prof. Nigma neglected to explain what the algorithm does.

```

// Input: A matrix A[0..n-1, 0..n-1] of integers
for i = 0 to n-2 do
  for j = i+1 to n-1 do
    if A[i,j] != A[j,i]
      return False
return True

```

In the above, recall that a matrix is essentially just a two-dimensional array, so $A[i, j]$ might in some languages be written as $A[i][j]$.

- (a) What does this algorithm do? Give an English description of what inputs lead to it returning True and what inputs lead to it returning False. (You do not need to give examples as part of your answer, but you are welcome to include example 2D arrays along with the English description, if it would make your answer clearer.)
- (b) Give an *exact* count of the number of array accesses (count $A[x, y]$ as a single array access) done by this algorithm in the worst case on input of size n . Show all work that you did to count those operations, including a Σ summation expression and its solution to a simple form in terms of n .
- (c) Based on your answer to exercise 4b above, give the most informative *worst-case* asymptotic time complexity bound you can for this algorithm, using big-O, Θ , or Ω notation. Explain how your answer to exercise 4b was used to get an asymptotic bound for worst-case running time (1–2 sentences should suffice for that explanation), and then explain how you got from there to your particular expression using big-O, Θ , or Ω notation, giving the relevant threshold n_0 and leading constant (or constants, in the case of Θ) used to establish that asymptotic complexity bound.
- (d) Give the most informative *best-case* asymptotic time complexity bound you can for this algorithm. (Note that the summation you solved in exercise 4b was under *worst-case* assumptions—please think about what would make the *best-case* time complexity!) Once again, give the relevant threshold n_0 and leading constant (or constants, in the case of Θ) used to establish that asymptotic complexity bound.
- (e) Give the most informative asymptotic bound you can on the space complexity for this algorithm, for the best case (i.e., using the least space other than that needed for the input) and the worst case (i.e., using the most space other than that needed for the input); you do not need to give values for a threshold n_0 and leading coefficients, just the asymptotic complexity bound and a high-level explanation. Your explanation should include your reasoning about whether the best case and worst case for space complexity are the same or different from each other.

5. CLRS Exercise 3-4, parts b and f (page 62).

For each part, explain your answer fully. As part of that, if the assertion is true, prove it, making sure to give *witness* values for every existential quantifier; and if not, provide a specific *counterexample* that demonstrates the assertion is not true.

As always in CS376, for full credit, please make sure your explanations demonstrate command of the relevant concepts—merely saying that a statement is true because it is given as true in the textbook is not a sufficient explanation for CS376, because it does not demonstrate understanding of the reasons *why* the statement is true, based on the relevant definitions. Please feel free to ask your Prof. any questions about this, or about explanations in CS376 more generally!