Advanced Exploration 3: Time Complexity of Algorithms

MATH2603: Discrete Mathematics

Overview: In this Advanced Exploration, you'll learn how to analyze the time complexity of algorithms. In computer science, a common problem is that time is limited. If you write a program to sort a list (for example), but the program takes 5 years to finish, that isn't very helpful. A more sophisticated sorting method might be the basis of a program that finishes in mere seconds. In this exploration, you'll read about asymptotic time complexity, then analyze the sorting algorithm I've made below.

Instructions: Section 6.1, 6.9, 6.12, 6.13 of the text are prerequisite material for this assignment. If you haven't read those sections (which we will discuss during class), then you should do so before continuing. Read Chapter 16 in the textbook (Growth of Functions & Time Complexity); you may have seen some of this material in other classes. Next, complete the exercises below. Your solutions should be complete, clear, and correct. Your solutions should also be accompanied by explanations written in complete sentences that justify your work.

Instructions for submitting your work, and information on how the EMRN rubric will be applied to evaluate this exploration, are at the end of the assignment.

Background: A *sorting algorithm* is an algorithm that takes as input a list of items and returns the same list sorted in ascending order. On input $(12, 4, 7, 9)$, for example, the correct output is $(4, 7, 9, 12)$. The items to be sorted must have a well-defined ordering. For example, integers can be sorted by value and names can be sorted according to alphabetical order.

If L is the list $L = (12, 4, 7, 9)$, then $length(L) = 4$, and we retrieve items from L as follows: $L[1]$ is the first item in L, so $L[1] = 12$. Similarly, $L[4] = 9$. All lists have indices beginning at 1.

A *recursive algorithm* is an algorithm that calls itself (albeit with a different input). In this exploration, you will study the best-case and worst-case time complexity of a recursive algorithm.

The Problem: Jasper is taking their first computer science class and has just learned a new sorting algorithm (pseudo-code is given below). Help Jasper understand the algorithm by answering the questions that follow.

```
1  ARecursiveSort(L)
2  %%%%% Input: list L
3  %%%%% Output: sorted version of the list L
4  n:=length(L)
5  if n=1
6      return L
7  for i:=1 to n-1
8      if L[i]>L[i+1]
9          temp:=L[i]
10         L[i]:=L[i+1]
11         L[i+1]:=temp
12 %%%%% apply algorithm to the sublist that omits the last entry of L
13 SortedSublist:=ARecursiveSort(L[1:n-1])
14 SortedList:=concatenate(SortedSublist,L[n])
15 return SortedList
```

1. First, help walk Jasper through the recursive sorting algorithm by applying it to the list $L = (12, 7, 4, 9)$. During each pass through the for loop, explain what the value of L is on line 12? What is the list given as an argument to the recursive call on line 13?

2. After working through the algorithm on a couple inputs, write a paragraph or two describing how the algorithm works.

Jasper needs to sort a rather large list, and is considering using an implementation of the above algorithm. Help Jasper determine whether this algorithm will be a good candidate for large lists by finding its asymptotic time complexity. Let $T(n)$ denote the maximum number of atomic operations used by `ARecursiveSort` on any list of size n.

3. If `ARecursiveSort` is called on a list of size n, how many recursive calls will be made? Explain your reasoning.

4. Find a recursive relation satisfied by $T(n)$. Then solve your recurrence relation using the methods of §6.12 and 6.13. Explain your answer.

5. What is the asymptotic time complexity of `ARecursiveSort`? Based on an internet search of commonly implemented sorting algorithms, is this a good candidate for large inputs? Explain how you arrived at your answer and what you learned from your internet search.

Submitting your work: Your work must be neatly typed up using a system that supports mathematical notation. For example, you can use MS Word and its equation editor; or you can write your work in a Jupyter notebook using Markdown and LaTeX. Once it is written up, the work must be saved as a PDF file and then uploaded as a PDF to the area on Blackboard where the original assignment is located. Remember that the work is not actually submitted until you upload the file and click the "Submit" button. Grading and feedback will take place entirely on Blackboard. The following are not allowed: Submissions outside Blackboard (for example through email); files that are not in PDF form; and work that contains any handwriting, though you may *draw a diagram* neatly by hand, scan it, and include it in your submission.

Evaluation: Like all Advanced Explorations, your work will be evaluated using the EMRN rubric. Please see the statement of this rubric in the syllabus for an explanation of how it is used. When applied to this Advanced Exploration, the following criteria help to assign the grade:

- **E:** The solution consists of a clear, correct, and complete solution. The solution contains no major errors (computation, logic, syntax, or semantic); it is also exceptionally clear and the writeup is professional in its look and style. The solution would be at home in a professional lecture or publication.

- **M:** The solution consists of a clear, correct, and complete solution. The solution contains no major errors (computation, logic, syntax, or semantic) and is neatly and professionally written up.

- **R:** The solution contains at least one, but not several, major errors (computation, logic, syntax, and/or semantic) that require revision. An "R" may also be given for writeups that do not expend sufficient effort to produce a good-looking writeup.

- **N:** The solution has several significant errors; or the submission is missing large portions of the solution; or the solution is for a significantly altered version of the problem; or the submission is excessively cluttered, messy, difficult to read, or handwritten.