Introduction to Algorithms: A Solutions Manual

Peter Hayes

January 2019

Preface

This book is meant to be a solutions manual for the text *Introduction to Algorithms* by CLRS, and this preface is meant to introduce how the material will be structured and what it will cover. With this text I plan to thoroughly cover each section and problem not marked with a star (★). Perhaps these will be covered in a later edition, but this is my current mission. I'm doing this not out of generosity but because I wish to develop myself as a software engineer and as an academic. However, this should hopefully serve other students in their pursuits of the same. I'm not looking for profit in my endeavors either, as I feel this is more of a personal passion. I hope you don't use this as a homework dodging mechanicism but as a way to deepen your understanding of algorithms.

I know I said this was a solutions manual, but it is, in fact, much more. If I feel the authors did not fully explain a concept, I will provide extra notes. These will normally cover background mathematics or "fill in the gaps" as they say. After that, we will create any data structures mentioned in my practical language of choice – C++. Of course, just printing the code isn't going to help many students, so I will provide the code itself in separate files. After, I've defined our data structures, I will move on to actually providing solutions. Each solution will be detailed and thorough. They will walk you through thought processes, outline solutions, and then solve the problems. Some problems have been solved by CLRS and are publicly availble on their website (https://mitpress.mit.edu/algorithms). If a solution is publicly availble, I will mark it with a diamond (♦).

I chose C++ for no shortage of reasons. Firstly, it's well known. Secondly, it provides classes. Thirdly, it provides pointers. Fourthly, it provides basic functional programming. And lastly, it provides operator overloading. I might not use all of these features, but them existing has convinced me that this is the right choice. C++ afficionados, please don't crucify me. I don't use the language professionally nor do I have much experience with it.

I don't claim to have perfect answers. If you would like to submit your own and believe it to be better than mine, please send it to me and I will credit you if I choose to include it (and if you want to be credited). However, I ask that you provide a few things lest it will not be added. Firstly, provide explanations as detailed or more detailed than my own. Secondly, provide a LaTeX file I can use to reproduce your result without much effort. Lastly, provide me a way to contact you along with a short description of who you are and if you'd like to be credited. Thank you for your support.

Chapter 1

1.1 Algorithms

Solutions

Exercises

1.1-1 Give a real-world example that requires sorting or a real-world example that requires computing a convex hull.

Answer:

An obvious case for sorting is any kind of data visualization. Suppose you're looking for an author's earlier works, then sorting by date would be useful. The convex hull problem has less obvious uses, but topography and mapping come to mind.

1.1-2 Other than speed, what other measures of efficiency might one use in a real-world setting?

Answer:

Space is the common alternative to speed. Suppose you have a sorting Algorithm that works in n time but takes 5 petabytes of storage. It's so incredibly inefficient that you'll be unable

to run the algorithm at all unless you're using possibly more than one whole storage server. The real life costs of such an algorithm are prohibitive.

1.1-3 Select a data structure that you have seen previously, and discuss its strengths and limitations.

Answer:

I will discuss two. Arrays are perhaps the most primitive data structure and allow fast access and replacement. However, arrays are very costly to keep in order. If you are inserting a value into a sorted array, to keep the order, you will have to perform move operations on every node with a higher or lower value than the one inserted depending on the original order. This starts getting costly fast. Now consider a linked list. A more complex data structure, but one that can handle insertions easily while maintaining order. But a linked list is slow to access randomly and is more complex to sort initially.

1.1-4 How are the shortest-path and traveling-salesman problems given above similar? How are they different?

Answer:

The traveling salesman problem is essentially performing the shortest path problem over and over again. Furthermore, the order of the nodes visited is undecided; this is where the complexity for the traveling salesman problem arises.

1.1-5 Come up with a real-world problem in which only the best solution will do. Then come up with one in which a solution that is approximately the best is good enough.

Answer:

A literal search is something that must be satisfied exactly. If I'm looking for the word "book," I will not be happy when "look" turns up after I run the algorithm. On the other hand, a search engine does more and often will not return with an exact match, but rather a "close enough" match. This is useful when you don't know exactly what you're looking for, but have a general idea. Interestingly, the editor I'm using right now (Visual Studio Code) uses both. CTRL-F gives me an in-document literal search and CTRL-SHIFT-P gives me a command palette which uses a search engine to find what commands might be useful.

1.2 Algorithms as a Technology

Solutions

Exercises

1.2-1 Give an example of an application that requires algorithmic content at the application level, and discuss the function of the algorithms involved.

Answer:

If you haven't learned about networking layers yet, or if you need a refresher, refer to https://en.wikipedia.org/wiki/Application_layer. A clear example would be a security algorithm that will deny access to certain parts of a website if your credentials don't match with a role that would have those options.

1.2-2 Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in $64n \lg n$ steps. For which values of n does insertion sort beat merge sort?

Answer:

Assume $n \in \mathbb{Z}^+$. We are trying to find when insertion sort uses less steps than merge sort. So we are solving the inequality $8n^2 < 64n \lg n$. We are allowed to divide by n since $n > 0$. Along with some manipulation, we get $n < \lg n^8$. Since the inverse function of $\lg n$ is monotone increasing, we will apply it. Thus $2^n < n^8$. Furthermore, $n^8 > 0 \ \forall \ n$ and so $\frac{2^n}{n^8} < 1$. Clearly, $n = 1$ is not a solution, but $n = 2$. Now we must find where the upper limit is. We know there is an upper limit because 2^n increases faster than n^8. Using a calculator (or writing a simple bit of code) will tell us that $\frac{2^n}{n^8} \geq 1$ when $n \geq 44$. So we have our answer as $\boxed{1 < n < 44, n \in \mathbb{Z}^+.}$ You might ask why I didn't just use a calculator in the first place, and that's becasue computers can compute exponents signficantly faster than logarithms. It's more efficient to write algortithms such that they are both easier to program and faster to run.

1.2-3 What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?

Answer:

Assume $n \in \mathbb{Z}^+$. We are solving the inequality $100n^2 < 2^n$. So $\frac{100n^2}{2^n} < 1$ since $2^n > 0$. Using a calculator or a simple program, we find that $\boxed{n > 14.}$

Problems

1-1 For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes $f(n)$ microseconds.

Answer:

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$	2^{10^6}	$2^{6\cdot 10^7}$	$2^{3.6\cdot 10^9}$	$2^{8.64\cdot 10^{10}}$	$2^{2.592\cdot 10^{12}}$	$2^{3.1536\cdot 10^{13}}$	$2^{3.1536\cdot 10^{15}}$
$\sqrt{n}$	10^{12}	$3.6 \cdot 10^{15}$	$1.30 \cdot 10^{19}$	$7.46 \cdot 10^{21}$	$6.72 \cdot 10^{24}$	$9.95 \cdot 10^{26}$	$9.95 \cdot 10^{30}$
n	10^6	$6 \cdot 10^7$	$3.6 \cdot 10^9$	$8.64 \cdot 10^{10}$	$2.59 \cdot 10^{12}$	$3.15 \cdot 10^{13}$	$3.15 \cdot 10^{15}$
$n \lg n$							
n^2	1000	7746	60000	293939	1609969	5615693	56156923
n^3	100	392	1533	4421	13737	31594	146646
2^n	6	26	32	37	42	45	52
$n!$							

Unfortunately, these numbers are so large that the answers for the functions $\sqrt{n}$, n, and $n \lg n$ must be rounded. Remember that these must be integer solutions if you're required to write them out. But this table is "close enough" to the actual answer that I don't care to provide a separate file enumerating the numbers in full.

In general, this problem can be stated as "Solve for n given $f(n) = t$ where t is in microseconds." For all but two cases my approach will be applying the inverse function to both sides for an easy solution (given one has a calculator). The factorial function along with $n \lg n$ are the only functions here that don't have a trivial inverse. I will provide a solution in C++ for these problems. Keep in mind that 1 second = 10^6 microseconds. 1 minute = 60 seconds or (1000000)(60) microseconds – and so on. In the "1 month" column, I will assume 1 month = 30 days. The longest time is going to be (1000000)(60)(60)(24)(365)(100) (or

3,153,600,000,000,000) microseconds. This number is too large for 32 bit machines (which someone might still have), so the solution will have to be a little creative.
