CLRS Chapter 1 Edition 4 Solutions

Cameron Beck

Updated June 14, 2023

Section 1.1

- An ***algorithm*** is any well-defined computational procedure that takes some value, or a set of values, as **input** and produces some value, or set of values, as **output** in a finite amount of time.
- An algorithm is **correct** if, for every problem instance provided as input, it **halts** – finishes its computing in finite time – and outputs the correct solution to the problem instance.
- An incorrect algorithm might not halt at all on some input instances, or it might halt with an incorrect answer. However, incorrect algorithms can sometimes be useful if we can control their error rates.

Exercise 1.1-1

A real-world example that requires...

- Sorting: organizing files in a file system (for example, by name, date created, etc).
- Shortest distance: DoorDash figuring out which restaurants are closest to my house and then routing the driver.

Exercise 1.1-2

Another measure of efficiency might be the amount of resources something takes to complete. It could be space (memory), available workers, or anything of the sort.

Exercise 1.1-3

Arrays: very good when you know you want to iterate over the entire collection or only need to access things by index. Good for insertions towards the end, although really frequent insertions are bad due to the need for resizing. Less good for insertions towards the beginning, or for when you need to access particular entries of the array based on something other than index.

Exercise 1.1-4

One major similarity between the two is the idea of minimizing some quantity (distance). However, in the travelling salesperson problem, we are forced to consider many distances between nodes in a graph – and end where we started, whereas the shortest-path problem only requires us to consider the distance between two nodes.

Exercise 1.1-5

A real-world problem where our requirements are satisfied by...

- Only the best solution: an algorithm controlling space shuttle movements.
- An "approximately" best solution: A google search where the result of the query can be good enough for what we need, even if it's not perfect.

Exercise 1.1-6

A real-world example that applies to both of these categories is audio data.

- In some contexts, you may have access to the entire audio file before you need to worry about analyzing it, such as with a YouTube video.
- However, you may also deal with audio data in the context of a live stream, where you may not be able to wait to have the entire file before doing your processing.

Section 1.2

- Although the saying goes, "time is money," time is even more valuable than money: you can get back money after you spend it, but once time is spent, you can never get it back.
- Even if we don't care about time/space, the analysis of an algorithm can still tell us important information about its correctness.

Exercise 1.2-1

An example of an application that requires algorithmic content at the application level is YouTube. YouTube presumably makes extensive use of algorithms to determine what kinds of channels, videos, etc it should recommend to the current user, based on information it has about the user.

Exercise 1.2-2

To solve this, we must find an integer value of n such that

$8n^2 < 64n \log_2 n$

Simplifying to

$n < 8 \log_2 n$

We find that these functions intersect at $n \approx 43.559$, meaning that insertion sort is better for $n < 44$.

Exercise 1.2-3

To solve this, we must find the smallest integer n such that

$100n^2 < 2^n$

Which intersect at $n \approx 14.325$, meaning that the algorithm whose running time is $100n^2$ will be faster when the input size is 15 or greater.

Problems

Problem 1-1

Assumptions (calculated microseconds in parentheses):

- 1 million μs = 1 second (1.0×10^6)
- 60 seconds = 1 minute (6.0×10^7)
- 60 minutes = 1 hour (3.6×10^9)
- 24 hours = 1 day (8.64×10^{10})
- 30 days = 1 month (2.592×10^{12})
- 12 months = 1 year (3.1104×10^{13})
- 1 century = 100 years (3.1104×10^{15})

$f(n)$	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\log_2(n)$	$2^{1\times10^6}$	$2^{6\times10^7}$	$2^{3.6\times10^9}$	$2^{8.6\times10^{10}}$	$2^{2.5\times10^{12}}$	$2^{3.1\times10^{13}}$	$2^{3.1\times10^{15}}$
$\sqrt{n}$	1.0×10^{12}	3.6×10^{15}	1.3×10^{19}	7.5×10^{21}	6.7×10^{24}	9.9×10^{26}	9.9×10^{30}
n	1.0×10^6	6.0×10^7	3.6×10^9	8.6×10^{10}	2.5×10^{12}	3.1×10^{13}	3.1×10^{15}
$n \log_2(n)$	62746	2801417	1.3×10^8	2.8×10^9	7.2×10^{10}	7.9×10^{11}	6.8×10^{13}
n^2	1000	7745	60000	293938	1609968	5577096	5.6×10^7
n^3	100	391	1532	4420	13736	31448	145972
2^n	19	25	31	36	41	44	51
$n!$	9	11	12	13	15	16	17

Table 1: 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.