Chapter 1

Interesting read, but you can skip it.

Chapter 2

2.1 Insertion Sort - To be honest you should probably know all major sorting algorithms, not just insertion sort. It's just basic knowledge and you never know when it can help.
2.2 Analysis of Algorithms - you can skip the small intro, but know the rest.
2.3 Designing algorithms - contains merge sort and its analysis as well as an overview of divide-and-conquer, very important stuff, so worth a read.

Chapter 3

All of it. You have to know big-O notation and time complexity analysis, period.

Chapter 4

4.1 Maximum subarray problem - Can kind of be worth your time. There are better solutions to this problem than divide and conquer but it's good practice and the flow of logic may help develop how you think.
4.2 Strassen's algorithm - I really love this algorithm and was astounded at how cool it was the first time I saw it, but you can skip it for the interviews. It won't come up.
4.3 Substitution method - you won't be using this method in an interview, but you should know it since it's a basic tool for finding the time complexity of a recursive algorithm.
4.4 Recurrence tree method - same as 4.3
4.5 Master method - essential knowledge. You should know it and practice with it and be able to use it in 3 seconds. This is the method you would use in an interview if analyzing a recursive algorithm that fits the form.
4.6 Proof of the master theorem - you can probably skip this, though it's good to read at least once so that you understand what you're doing with the master method.

Chapter 5

I've never read this chapter, to be honest, but what I know is that you need a basic grasp of probability in interviews because there's a good chance they may come up. That said, as long as you know basic probability concepts and practice on probability-related interview problems (there are such problems with solution explanations in Elements of Programming Interviews, the book I recommend for interview most), you can probably skip this chapter. From a cursory glance, it's more math than algorithms.

Chapter 6

6.1, 6.2, 6.3, 6.4, 6.5 - Heaps and heapsort. Check.

Chapter 7

7.1, 7.2, 7.3 - Quicksort and its randomized version. Need-to-know concepts. I also recommend 7.4 (I was once asked in an interview to high-level-analyze a randomized algorithm), though the probability you have to deal with something like 7.4 in an interview is pretty low, I'd guess.

Chapter 8

8.1 - Lower bounds on sorting - Yes. Basic knowledge. May be asked in a Google interview (though unlikely, I know of a case it happened in before).
8.2 - Counting sort - Need-to-know in detail. It comes up in disguised forms.
8.3 - Radix sort - Yup. It's an easy algorithm anyway.
8.4 - Bucket sort - can skip.

Chapter 9

9.1 - Small section, worth a read.
9.2 - Selection in expected linear time - Very important, as it's not common knowledge like quicksort and yet it comes up often in interviews. I had to code the entire thing in an interview once.
9.3 - Selection in worst-case linear time - Can skip. Just know that it's possible in worst-case linear time, because that might help somewhat.

Chapter 10

10.1 - Stacks and queues - basic knowledge, definitely very important.
10.2 - Linked lists - same as 10.1
10.3 - Implementing pointers and objects - If you use C++ or Java, skip this. Otherwise I'm not sure.
10.4 - Representing rooted trees - Small section, worth a quick read.

Chapter 11

For hashing, I'd say the implementation isn't as important to know as, for example, linked lists, but you should definitely have an idea about it and most importantly know the (expected and worst-case) time complexities of search/insert/delete etc. Also know that practically, they're very important data structures and, also practically, the expected time complexity is what matters in the real world.
11.1 - Direct addressing - Just understand the idea.
11.2 - Hash tables - important.
11.3 - Hash functions - it's worth having an idea about them, but I wouldn't go too in-depth here. Just know a couple examples of good and bad hash functions (and why they are good/bad).
11.4 - Open addressing - Worth having an idea about, but unlikely to come up.
11.5 - Perfect hashing - skip.

Chapter 12

12.1 - What is a binary search tree? - Yep.
12.2 - Querying a BST - Yep. All of it.
12.3 - Insertion/Deletion - Same as 12.2
12.4 - Randomly built BSTs - just know Theorem 12.4 (expected height of random BST is O(lgn)) and an idea of why it's true.

Chapter 13

This one is easy. Know what a Red-Black tree is, and what its worst-case height/insert/delete/find are. Read 13.1 and 13.2, and skip the rest. You will never be asked for RB-tree insert/delete unless the interviewer is "doing it wrong", or if the interviewer wants to see if you can re-derive the cases, in which case knowing them won't help much anyway (and I doubt this would happen anyway). Also know that RB-trees are pretty space-efficient and some C++ STL containers are built as RB-trees usually (e.g. map/set).

Chapter 14

Might be worth skimming 14.2 just to know that you can augment data structures and why it might be helpful. Otherwise do one or two simple problems on augmenting data structures and you're set here. I'd skip 14.1 and 14.3.

Chapter 15

DP! Must-know.
15.1 - Rod-cutting. Standard DP problem, must-know.
15.2 - Matrix-chain multiplication - same as 15.1, though I don't particularly like the way this section is written (it's rare for me to say that about CLRS).
15.3 - Elements of DP - worth a read so that you understand DP properly, but I'd say it's less important than knowing what DP is (via the chapter introduction) and practicing on it (via the problems in this book and in interview preparation books).
15.4 - LCS - same as 15.1
15.5 - Optimal binary search trees - I've never read this section, so I can't argue for its importance, but I did fine without it.

Chapter 16

You should definitely know what a greedy algorithm is, so read the introduction for this chapter.
16.1 - An activity selection problem - Haven't read this in detail, but I'd say check it out, if not in-depth.
16.2 - Elements of the greedy strategy - same as 16.1
16.3 - Huffman codes - I'd say read the problem and the algorithm, but that's enough. I've seen interview questions where the answer is Huffman coding (but the question will come up in a 'disguised form', so it won't be obvious.)
16.4 - Matroids and greedy methods - I've never read this section, but I've done a lot of greedy problems during interview prep and this stuff never came up, so I'd say this section is irrelevant for the interview.
16.5 - Task-scheduling problem as a matroid - Same as 16.4.

Chapter 17

Okay, you should definitely know what amortized analysis is, but I've never read it from the book and I feel it's a sufficiently simple concept that you can just Google it and check a few examples on what it is, or understand it just by reading section 17.1. So:
17.1 - Aggregate analysis - read this, it explains the important stuff.
17.2, 17.3, 17.4 - Skip.

Chapter 18

You should probably have an idea of what B-Trees (and B+ trees) are, I've heard of cases where candidates were asked about them in a general sense (high-level questions about what they are and why they're awesome). But other than that I'd skip this chapter.

Chapter 19

Fibonacci heaps - nope.

Chapter 20

van Emde Boas Trees - double, triple, and quadruple nope.

Chapter 21

Disjoint sets
Update: I originally recommended skipping this section, but on reconsideration, I've noticed that it's actually more important than I originally thought. Thus, I recommend reading sections 21.1 and 21.2, while skipping the rest.
Union-find is somewhat important and I've seen at least one problem which uses it, though that problem could also be solved using DFS and connected components. That said, I also believe that it's not strictly necessary because one can probably, for interview purposes, come up with a similar enough structure easily to solve a problem which requires union-find, without knowing the material in this chapter. However, I believe it's worth a read so that if a problem comes up whose intended solution is a union-find data structure, you don't spend time in an interview coming up with it, and rather know from before, which can be a good advantage. Still, I'd probably rank it as less important than most of t

Okay, now graph algorithms. First read the introduction. Now, there's a lot to know here, so hang on.

Chapter 22

22.1 - Representations of graphs - Yes.
22.2 - BFS - Yes. After you do that, solve this problem: ACM ICPC Live Archive - Kermit the Frog. The whole "state-space search using BFS" thing is an important concept that might be used to solve several interview problems.
22.3 - DFS - Yes.
22.4 - Topological sort - Yes.
22.5 - Strongly connected components - much less likely to come up than the above 4, but still possible, so: Yes.

Chapter 23

Minimum spanning trees - probably the least important graph algorithm, other than max flow (I mean for interview purposes, of course). I'd still say you should read it because it's such a well-known problem, but definitely give priority to the other things.
23.1 - Growing a MST - sort of, yes.
23.2 - Prim and Kruskal's algorithms - sort of, yes.

Chapter 24

Shortest path algorithms are important, though maybe less so than BFS/DFS.

Read the introduction. You should, in general, read all introductions anyway, but this one's important (and long), so it warranted a special note.
24.1 Bellman-Ford - Know the algorithm and its proof of correctness.
24.2 Shortest paths in DAGs - definitely worth knowing, may come up, even more so than Bellman-Ford I'd say.
24.3 Dijkstra's algorithm - Yes. Of course. I've seen this come up multiple times (with slight variations), and I've even seen A* come up.
24.4 Difference constraints and shortest paths - Skip.

Chapter 25

Read the intro as well.
25.1 - Matrix multiplication - I'd say skip. It might be possible for this to come up (very very slim chance that it does though), but the chances are so low in my view that it's probably not worth it. If you have some extra time, though, give it a read.
25.2 - Floyd-Warshall - Yep, worth knowing the algorithm and its time complexity and when it works (which is for all weighted graphs, except ones with negative weight cycles). Its code is something like 5 lines so there's no reason not to know it. The analysis might be a bit overkill though.
25.3 - Johnson's algorithm - Skip.

Chapter 26

Maximum flow - I've never heard of this coming up in an interview and I can't imagine why it would, so skip.

Chapters 27+

Most of this stuff is never going to come up, so it's easier for me to tell you what to actually read than what not to read, so here are a few selected topics from the Selected Topics in the book:

Chapter 31
Most of what you should learn from this chapter you can learn from practicing on interview problems from Elements of Programming Interviews (and your time is better spent doing that), so I'd say skip it all except Euclid's algorithm for the GCD, under section 31.2.

Chapter 32
32.1 - Naive method - just read it quickly.
32.2 - Rabin-Karp - I'd say you should know this, the rolling hash concept is very important and can be useful in many string- or search-related interview problems.

Appendices

A - Summations
Know the important summations for time complexity analysis.

C - Counting and Probability
Give C.4 a read if you don't know the material, Bernoulli trials may come up in problems (not explicitly, but you might use them, specifically for time analysis of questions that involve probability/coin flips).

And that's it! Once you've covered all that, I'd say pick up Elements of Programming Interviews and prepare a lot with its problems. I've outlined my own interview preparation process here - it might help.

Note: Keep in mind, though, that the above knowledge from CLRS isn't sufficient on its own. There are many topics which are not in CLRS but may be relevant in an interview, many of them practical (like language-specific questions) and many others which are theoretical but aren't covered explicitly in CLRS. For example, tries and tournament trees are important data structures (the former more important than the latter) which don't have a dedicated section in CLRS, and skip lists have also been known to be asked, etc... Furthermore, the above-included sections are exhaustive, in that they're all the topics that have a remote chance of coming up from CLRS. Some of them you can probably skip at