Topic list for revision

General techniques

1. Mathematical induction: ordinary, course of values, nested.
2. The countable/uncountable distinction is not required.

Languages and automata

1. Regular expression
2. Deterministic finite automaton, error state
3. Nondeterministic finite automaton, nondeterministic finite automaton with ε-moves
4. You should know how to algorithmically convert a regex to an DFA. You should know that a DFA can be converted to a regex, but don't need to know an algorithm for this.
5. Equivalence of DFAs
6. Minimization of total DFAs
7. DFAs for complements and intersections.
8. You should be able to prove that a language isn't regular. (The Pumping Lemma wasn't taught and isn't needed.)
9. Context free languages (not pushdown automata)
10. Chomsky normal form.
11. Ambiguity of context free grammars.

Complexity

1. Complexity of algorithms vs complexity of problems
2. Lower and upper bounds
3. O, Ω, θ notation

4. Polynomial and exponential complexity
5. $\mathcal{NP}$ problems, the two definitions, the question of whether $\mathcal{P} = \mathcal{NP}$.
6. $\mathcal{NP}$-completeness and SAT.

Turing machines

1. Execute Turing machines
2. Design Turing machines for simple tasks
3. Extended alphabet (2 tape, etc.) Turing machines look more powerful than a Turing machine, but it isn't because a Turing machine can simulate it
4. Macros and expansion
5. Church's thesis: any function on words that can be computed algorithmically can be computed by a Turing machine
6. Nondeterministic Turing machines.

Decidability

1. Problem, decision problem (a problem whose answer is yes/no)
2. Decidable and semidecidable problems
3. Primitive recursive functions: you need to be able to show that a function is primitive recursive by implementing it in Primitive Java. (You would be given the definition of Primitive Java.)
4. The Halting Theorem: whether a given program halts on given inputs is undecidable. (Knowing the proof is not required.)
5. Rice's Theorem: any semantic property of code that sometimes hold and sometimes fails to hold is undecidable. (Knowing the proof is not required, but may help you to understand other problems.)
6. If you're asked to show a problem is decidable, write a program to decide it. Or at least sketch how you would write a program.
7. If you're asked to show a problem is undecidable, you might reduce the halting problem to it, or you might appeal to Rice's theorem.

Note that at each point you should know practical aspects, e.g. uses of regular and context free languages, consequences of non-computability, etc.