Problem 1. Write a program for a Random access machine which sorts a given list of elements. You can start by implementing any sorting algorithm. Can you sort n integers in time $\mathcal{O}(n \log n)$ using additional $\mathcal{O}(1)$ memory cells?

Problem 2. Construct a Turing machine which recognizes language

$$L = \{w;\ w \text{ contains twice as many 0s as 1s}\}.$$

Write exact definition of your Turing machine!

Problem 3. Show that every Turing machine can be modified to a Turing machine which can perform only two of three operations (i.e. move head and change a state, or write on tape and move head, or write on tape and change a state) in every step. Write exact definition of the modified Turing machine.

Problem 4. Construct a Turing machine which recognizes language $L = \left\{0^{(2^n)};\ n \in \mathbb{N}\right\}$, i.e. words containing only zeros whose length is a power of two. Write the transition function exactly!

Problem 5. An always-moving Turing machine is a Turing machine which must move its head in every step; i.e. the transition function is of the form $\delta : Q \times \Sigma \to Q \times \Sigma \times \{R, L\} \cup \{\perp\}$. Prove that for (standard) Turing machines there exists an equivalent always-moving Turing machine. Describe exactly the construction of a transition function.

Problem 6. Construct a Turing machine which recognizes language $L = \{a^n b^n c^n;\ n \in \mathbb{N}\}$. Write the transition function exactly!

Problem 7. Consider a variant of Turing machine which has one-directional tape and its head can perform only two movents: move right and reset which moves the head on the first cell. Show how a (standard) Turing machine can be transformed to the described variant of Turing machine.

Problem 8. Prove that one-tape Turing machines which are allowed to write on every cell at most once are equivalent to (standard) Turing machines.

Problem 9. Prove that a one-tape Turing machine which cannot overwrite their input are equivalent to a Finite automata.

Problem 10. Prove that the concatenation $L_1 \cdot L_2 = \{ab;\ a \in L_1, b \in L_2\}$ of (partially) decidable languages is (partially) decidable.

Problem 11. Consider a partially decidable language L. Prove that language

$$\{w_1 w_2 \cdots w_k;\ k \in \mathbb{N}, w_1, w_2, \ldots, w_k \in L\}$$

is partially decidable where $w_1 w_2 \cdots w_k$ is a concatenation of k words of L.

Problem 12. Is language $\{(M_1, M_2);\ M_1, M_2 \text{ are Turing machines and } L(M_1) = L(M_2)\}$ decidable?

Problem 13. Let EMPTY be the set of all Turing machines which does not accept any input. Prove that the complement $\overline{\mathrm{EMPTY}}$ is partially decidable.

Problem 14. Let M_1, M_2 be Turing machines. Is language $\{(M_1, M_2);\ L(M_1) \cap L(M_2) = \emptyset\}$ decidable?

Problem 15. Is language $\{M;\ M \notin L(M)\}$ is (partially) decidable? Note that we encoded every Turing machine by a number and also words are encoded as a number. Therefore, a number M encodes a word (input) and also a Turing machine. So, $M \notin L(M)$ meas that a word M is not accepted by a Turing machine M.

Problem 16. A deterministic queue automaton (DQA) is defined the same way as a deterministic pushdown automata (DPDA), except that it has a queue instead of a stack. In other words, a DQA is a deterministic finite automaton augmented with an unbounded queue, together with the operations of (a) pushing a symbol onto the "back" of the queue, and (b) popping the symbol at the "front" of the queue. Show that DQAs are equivalent in power to Turing machines: that is, any given language L is decidable by a DQA if and only if it's decidable by a Turing machine.