1 Exercises

1. *Consider the language* $\{a^n \mid n \text{ is composite}\}$.

   - *Describe a simple mechanism for accepting this language on a normal two-tape Turing machine, with an input tape and an oracle tape.*

     The genie just writes a proper factor on the oracle tape and then we check it (see previous solutions).
   - *Do the same, but with a "non-deterministic transitions" two-tape machine.*

     There's an initial phase where we write some block of a's on the second tape. At some point we non-deterministically transition to a phase where we check that block represents a proper factor.

     Any accepting computation demonstrates that n has a proper factor so we can't be forced to accept a prime (and every composite number has at least one accepting computation).

2. *Consider the following problem: given two input tapes, one containing a string* b^n*, the other containing input of the form* $a^{m_1}\#a^{m_2}\#\cdots\#a^{m_k}$ *(i.e. a sequence of blocks of* a*'s), determine whether some subset of the blocks of* a*'s is of total length* n *(this is called the* SUBSET-SUM *problem). Show that a three tape TM with oracle tape makes this problem almost trivial. Likewise model it in a TM with non deterministic transitions. Finally, consider the difficulties in dealing with it deterministically.*

   On the oracle tape the genie just writes out a binary string $w_1w_2\ldots w_k$ indicating which of the a's to use (i.e., if $w_i = 1$ we should use a^{m_i} as one of the chosen blocks and not otherwise. We can easily check that this is correct by computing the indicated sum (by concatenation on another tape) and then comparing the length to the block of b's (or, even more easily, must by moving m_i places along the block of b's when we're told to use a^{m_i} and checking that we get exactly to the end.

   Non-deterministically is similar - we move along the a tape and each time we hit a new block we decide non-deterministically whether to use it or not. If so, we advance along the b tape at the same time, if not, we just move to the next a block.

   The deterministic issue is that it seems difficult to guess whether we should be using any particular block or not. So the basic search procedure "try every pos-

sibility" requires 2^k trials. There is some improvement possible but, at least if we represent the numbers in binary (rather than unary as here) the problem is known to be NP-complete.

3. *(Review) Remember that regular languages form a subset of the context free languages, which in turn form a subset of the recursive languages, and those are a subset of the recursively enumerable languages (though, at the moment, we can't be sure that there are any recursively enumerable languages that are not recursive). To prove that a language is regular, we can either design a finite state automaton that accepts it, or show that it is generated by a regular language. To prove that a language is not regular, we usually use the pumping lemma for regular languages or the Myhill-Nerode theorem. Similarly, to prove that a language is context free, we either show that it is accepted by a push down automaton, or give a context free grammar for it. Again, to prove a language is not context free, we use the pumping lemma for a context free languages. To prove that a language is recursive, we design a TM which halts on all inputs and accepts it. So far, we have no mechanisms for proving that a language is not recursive. For each of the following languages determine exactly which type it is, and prove it (so, for instance if you are claiming it is context-free, you should both give a grammar/PDA and an argument that it is not regular.)*

   (a) $L = \{a^{3k}ba^{2l} \mid 0 \leq k, l\}$
   This is regular. We can have a cycle of three states that we move around using a's. From the initial state (which we visit when there are $0, 3, 6, \ldots$ a's) there is a b transition to a new cycle of two states (also on a's alone). The only accepting state is the one we move to along that b.

   (b) $L = \{a^{3k}ba^{2k} \mid 0 \leq k\}$
   This is context free but not regular. To see that it's not regular observe that for each k, $ba^{2k} \in \text{Suff}(w, L)$ if and only if $w = a^{3k}$. So the suffix languages for the words a^3, a^6, a^9 ...are all distinct and by the Myhill-Nerode theorem the language is not regular.
   To see that it's context free, note the following grammar works:

   $$S \to b|aaaSaa.$$

   A PDA also isn't too hard to describe - in an initial "pushing" phase, push one thing onto the stack for every third a (use state transitions to allow for

"reading a pushing nothing" in the first two a's of a block of three). Then transfer (reading a b leaving the stack alone) to a popping state where we pop on A for every pair of a's (or we could have pushed for two out of three in the first phase and now pop for every one.)

(c) $L = \{a^{k^2}ba^k \mid 0 \leq k\}$

This is not context free but it is recursive.

To see that it's not context free suppose that it were and consider the pumping lemma. For some p and all words z in the language of length $\geqslant p$ we can write $z = uvwxy$ where $|vwx| \leqslant p$, $|vx| > 0$ and for all i, $uv^iwx^iy \in L$.

Take $z = a^{p^2}ba^p$. The only factorisation that has a hope of succeeding would have v consisting of some non-empty segment of a's before the b, and x consisting of some non-empty segment of a's after the b. Say $|u| = s$ and $|x| = t$. But then, for the pumping lemmat to apply would require:

$$p^2 + (i-1)s = (p + (i-1)t)^2$$

for all i. That is:

$$\begin{aligned}(i-1)s &= 2(i-1)t + (i-1)^2t^2 \\ s &= 2t + (i-1)t^2,\end{aligned}$$

for all i. But the LHS is constant and the RHS is not (unless $t = 0$ - but then $s = 0$ as well which is not permitted). So this is a contradiction and the language is not context free.

It's clearly recursive since we can actually do arithmetic in Turing machines (I'll omit the details!)