Question 1

Part 1

Required: Show that INT-FACT is in NP and in co-NP. (You may use any result stated or proved in class.)

NOTE: We assume that INT-FACT contains pairs of non-negative integers.

First we will show that INT-FACT $\in$ NP.

Consider the following verifier V_1 for INT-FACT.

V_1 = "on input $\langle\langle m, n\rangle, c\rangle$:

1. Test that $c = \langle q\rangle$ for some integer q such that $1 < q < n$.

2. Test that q divides m.

3. If all tests pass, then accept. Otherwise, reject.

Clearly V_1 is a decider, as it always halts. And V is a verifier for INT-FACT, since if $\langle m, n\rangle \in$ INT-FACT, then m has a factor q less than n and greater than 1. Hence, $c = \langle q\rangle$ is a certificate and V_1 will accept $\langle\langle m, n\rangle, c\rangle$. And if $\langle m, n\rangle \notin$ INT-FACT, then m has no factor less than n and greater than 1. Hence, no such certificate exists.

And if a certificate c exists, then clearly $|c| < |\langle n\rangle| < |\langle m, n\rangle|$. And all the tests in V_1 take polynomial steps (checking $<$ and divisibility).

So V_1 is a poly-time verifier for INT-FACT. Therefore, INT-FACT $\in$ NP.

Now we will show that $\overline{\text{INT-FACT}} \in$ NP. Consider the following verifier V_2 for $\overline{\text{INT-FACT}}$.

V_2 = "on input $\langle\langle m, n\rangle, c\rangle$:

1. Test that c is an encoding such that $c = \langle p_1^{\alpha_1}, p_2^{\alpha_2}, ..., p_k^{\alpha_k}\rangle$ where each p_i is prime and each $\alpha_i \in \mathbb{N}$. i.e. Use AKS primality test to check that each p_i is prime.

2. Test that $m = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots p_k^{\alpha_k}$.

3. Test that each $p_i > n$.

4. If all tests pass, then accept. Otherwise, reject."

Clearly V_2 is a decider, as it always halts. And V_2 is a verifier for $\overline{\text{INT-FACT}}$, since if $\langle m, n\rangle \in \overline{\text{INT-FACT}}$, then m has no factor less than n and greater than 1. Hence, $c = \langle p_1^{\alpha_1}, p_2^{\alpha_2}, ..., p_k^{\alpha_k}\rangle$ where $m = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ is a prime factorization is a certificate, and V_2 will verify that each prime p_i is such that $p_i > n$ and accept. And if $\langle m, n\rangle \notin \overline{\text{INT-FACT}}$, then m does have a factor less than n and greater than 1. So m will have a prime factor less than n and greater than 1. Hence, no such accepting certificate would exist.

And $\overline{\text{INT-FACT}}$ contains ill-formed strings as well. So if V_2 is given a string that does not include a pair of non-negative integers (m, n), then also accept.

And V_2 is a poly-time verifier for $\overline{\text{INT-FACT}}$ since each step is clearly polynomial, including the step for the AKS primality test which we know is polynomial from lecture. And a valid certificate c would also be polynomial length in terms of $|\langle m\rangle|$, as c is a prime factorization. Hence, c would have polynomial length in terms of $|\langle m, n\rangle|$.

Therefore, $\overline{\text{INT-FACT}} \in$ NP. Therefore, INT-FACT $\in$ co-NP.

Therefore, we have shown that INT-FACT $\in$ NP and INT-FACT $\in$ co-NP.

Part 2

Required: Show that if INT-FACT $\in$ P, then there is a polynomial time algorithm for factoring binary integers.

Proof. Assume INT-FACT $\in$ P. Let M be the TM that decides INT-FACT in polynomial time.

Note: As indicated by Prof Saraf on Piazza, it is sufficient to construct an algorithm N that factorizes integers x such that $x = ab$ for $1 < a < x$ and $1 < b < x$. To get complete prime factorization, we would repeatedly apply this algorithm N recursively.

So WLOG, we will construct an algorithm to factor an integer into two non-trivial factors.

Consider the following algorithm N for factoring binary integers. We will use binary search.

Note, the notation $\leftarrow$ is just to indicate an assignment. i.e. instead of writing $x = x + 5$, we can write $x \leftarrow x + 5$.

Please see the next page.

$N =$ "on input $\langle x \rangle$:

1. If $x = 1$, output $\langle x \rangle$ on the tape, and stop.

2. Test if x is prime by AKS primality test. If x is prime, then output $\langle x \rangle$ on the tape, and stop.

3. Otherwise, let $m = x$ and let $n = x$ and let $p = 0$. Consider the following variation of binary search.

Repeat so long as $p + 1 \neq n$:
i) Run M on $\langle m, \lceil (n + p)/2 \rceil \rangle$.
ii) If M accepts, let $n \leftarrow \lceil (n + p)/2 \rceil$
iii) If M rejects, let $p \leftarrow \lceil (n + p)/2 \rceil$

4. After Step 3, from the binary search we get that p is the smallest positive, non-trivial factor of x. Since p is the smallest, positive, non-trivial factor of x, we get that p is prime. Let $a = p$, and let $b = \frac{x}{p}$. Output $\langle a, b \rangle$ on the tape."

To see that the above algorithm N is polynomial, notice that Step 1 is polynomial as it just checks if $x = 1$. And Step 2 is polynomial since the AKS primality test is polynomial.

Step 3 has $O(\log(x))$ iterations as it is simply a variation of binary search. And $|\langle x \rangle| = \lceil \log(x) \rceil$. So the number of iterations is polynomial in terms of $|\langle x \rangle|$. Each iteration of Step 3 runs in polynomial time since we assumed that M (which decides INT-FACT) is polynomial. Hence, Step 3 is polynomial with respect to the length of the input.

Finally, Step 4 is polynomial as it is just involves division which is polynomial.

Hence, N is a polynomial algorithm.

Note, if N were to factor an integer x such that $x = ab$ for $1 < a < x$ and $1 < b < x$, we know that a would be a prime factor from our earlier discussion. Hence, we can just reapply the algorithm on b. Hence, we can get the complete prime factorization of x by recursion. And this would be polynomial since N is a polynomial algorithm, and we would require at most $O(log(x))$ recursive calls to N as each prime factor is greater than or equal to 2.

In fact, we can modify N above to explicitly include this recursion as a final step. We could modify the final step of N to say,

"Run N on b, and then output the factor a and all the factors that come from running N with input b. These factors are a complete prime factorization of x".

But for simplicity, we keep N as originally stated above.

This completes the proof, as required. $\square$

Question 2

Required: Show that MODEXP $\in P$

Note: From Prof Saraf, we are given the hint that we can use the fact that addition, multiplication and mod are all polynomial. We are also given the hint of squaring.

Consider the following TM M. Note, the notation $\leftarrow$ is just to indicate an assignment. i.e. instead of writing $x = x + 5$, we can write $x \leftarrow x + 5$.

Note: We implicitly assume ill-formed strings are rejected.

$M =$ "on input $\langle a, b, c, p\rangle$:

1. Consider the binary integer representation of b to be $\langle b\rangle = (b)_2 = b_1 b_2 ... b_j$.

2. Let $k = 1$. Repeat the following for $i = 1$ to j:
   i) Let $k \leftarrow k^2$
   ii) If $w_i = 1$, let $k \leftarrow ak$
   ii) Let $k \leftarrow k \pmod p$ within the the residue system $\{0, 1, ..., p-1\}$

3. After Step 2, we get that $k = a^b \pmod p$. Now find $c \pmod p$ within the residue system $\{0, 1, ..., p-1\}$.

4. If $a^b \pmod p$ equals $c \pmod p$ from step 3, then accept. Otherwise, reject."

Clearly M is a decider since M always halts. To show that it decides MODEXP, consider $\langle a, b, c, p\rangle \in$ MODEXP. Hence, $a^b \equiv c \pmod p$. We know Step 2 will compute $a^b \pmod p$. And Step 3 will find $c \pmod p$. If these two are equal in Step 4, then we know from elementary number theory that $a^b \equiv c \pmod p$. Hence, we will accept. If $\langle a, b, c, p\rangle \notin$ MODEXP, then $a^b \not\equiv c \pmod p$, and on Step 4 equality cannot possibly hold, so we will reject.

Also, we implicitly assume that ill-formed strings are rejected too.

To show that we have a polynomial algorithm, note that Step 1 is just considering the binary representation of b.

And Step 2 has j iterations, where j is the length of the binary representation of b. i.e. we have $j = |\langle b\rangle|$ many iterations in Step 2. And each iteration clearly takes polynomial time since sub-steps i) and ii) are just multiplication and sub-step iii) is taking mod which we assume is polynomial. So Step 2 is polynomial with respect to the length of the input.

And Step 3 is also polynomial by assumption since we are just taking mod of c. And Step 4 is polynomial since we are just checking equality. Therefore, MODEXP $\in$ P.

Question 3

Required: Show that the set of incompressible strings contains no infinite subset that is Turing recognizable.

Proof. Assume for the sake of contradiction that the set of incompressible strings contains an infinite subset A that is recognizable.

Since A is recognizable, there exists an enumerator E that enumerates A.

Consider the following TM M.

$M =$ "on input w:

1. Obtain, via the recursion theorem, own description $\langle M \rangle$.
2. Run E.
3. Consider the string $\langle M, w \rangle$. Consider the first string x that E prints out such that $|x| > |\langle M, w \rangle|$.
4. Output the string x on the tape."

To see that M is well defined, notice that we know there are only finitely many strings with length less than or equal to $|\langle M, w \rangle|$. Since A is infinite, we know that E will eventually print a string x such that $|x| > |\langle M, w \rangle|$.

And M is defined so that it leaves x on its tape. Hence, $\langle M, w \rangle$ is a description of x.

Hence, by definition of Kolmogorov complexity, we get $K(x) \leq |\langle M, w \rangle|$.

And we know from M that $|x| > |\langle M, w \rangle|$.

Combining our two inequalities, we get $|x| > |\langle M, w \rangle| \geq K(x)$.

Since $|x| > K(x)$, we get that x must be compressible. Hence, $x \notin A$.

But x was printed by E which is an enumeration of A. Hence, $x \in A$.

So we have $x \notin A$ and $x \in A$. This is a contradiction.

Therefore, our initial assumption was wrong. Therefore, the set of incompressible strings contains no infinite subset that is Turing recognizable.

□

Question 4

Required: Show that any infinite subset of MinTM is not Turing-recognizable.

Proof. Assume for the sake of contradiction that $MinTM$ had an infinite recognizable subset A.

Hence, there exists an enumerator E that enumerates A.

Consider the following TM M.

$M =$ "on input w:

1. Obtain, via the recursion theorem, own description $\langle M \rangle$.
2. Run E.
3. Consider the first string $\langle N \rangle$ that E prints out such that $|\langle N \rangle| > |\langle M \rangle|$.
4. Run N on w. If N accepts, then accept. If N rejects, then reject."

To see that M is well defined, notice that we know that there are only finitely many strings with length less than or equal to $|\langle M \rangle|$. Since A is infinite, we know that E will eventually print a string $\langle N \rangle$ such that $|\langle N \rangle| > |\langle M \rangle|$.

And we have that M is defined in such a way that it accepts/rejects/loops on exactly the same strings that N accepts/rejects/loops on. i.e. M simulates N. Hence, M is equivalent to N.

But $|\langle N \rangle| > |\langle M \rangle|$. Hence, N is not minimal. Hence, $\langle N \rangle \notin A$. But $\langle N \rangle$ was printed by E which enumerates A. So $\langle N \rangle \in A$.

So we have $\langle N \rangle \notin A$ and $\langle N \rangle \in A$. This is a contradiction.

Therefore, our initial assumption was wrong. Therefore, $MinTM$ has no infinite recognizable subset.

□