Problems marked with **E** are graded on effort, which means that they are graded subjectively on the perceived effort you put into them, rather than on correctness. For bonus questions, we will not provide any insight during office hours or Piazza, and we do not guarantee anything about the difficulty of these questions. We strongly encourage you to typeset your solutions in LaTeX.

If you collaborated with someone, you must state their names. You must write your own solution and may not look at any other student's write-up.

0. If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

1. Show that the following languages are in the class P.

   (a) $L_{TuringEven} = \{\langle M = (Q, \Sigma, \Gamma, q_{start}, q_{accept}, q_{reject}, \delta)\rangle : M$ is a Turing machine with an even number of states$\}$

   Solution: We can define a polynomial-time decider for this language. The Turing machine is encoded as a 7-tuple, containing a list of states. We can count those states, then determine if that number is odd or even.

   $L_{TuringEven}$ = "On input M:
   1. $count \leftarrow 0$
   2. For q in Q:
   3. $\quad count \leftarrow count + 1$
   4. If $count$ mod $2 \equiv 0$, ACCEPT. Else REJECT."

   This is a decider; if the machine M is in the language, it will have an even number of states, and this machine will count them accurately and accept. If M is not, it will have an odd number of states, and the machine will reject. This is polynomial-time; we count the items in the state list in $O(|Q|)$ time, then take that number mod 2 (which is trivially O(1) on binary strings; check the last digit). As Q is bounded by the size of the input M, this is $O(M)$ time and therefore an efficient decider on the input.

   (b) $L_{LIS} = \{(S, k) : S$ is a sequence of numbers, and S has a strictly increasing subsequence of size greater than $k\}$

   We define a subsequence as a string composed of characters from another string, without repeats, in their original order (ex: "137" is a strictly increasing subsequence of "123456789012345", but "376" is not a subsequence and "370" is not strictly increasing).

   Recall: k is encoded in binary, therefore it takes $O(\log k)$ bits to encode k.

Solution: We can define a polynomial-time decider for this language. Recall the Longest Increasing Subsequence Dynamic Programming algorithm from Lecture 5. This algorithm operates on a sequence S with length n, memoizing to a memo with dimensions $1 \times n$. This is a bottom-up implementation of that algorithm:

L_{LIS} = "On input S, k:
1. We define a table M with dimensions 1xn
2. For i from 1 to n:
3. $\quad M[i] \leftarrow 1$
4. $\quad$ For j from 1 to $i-1$:
5. $\quad\quad$ If $S[i] > S[j]$ and $M[j] \geq M[i]$:
6. $\quad\quad\quad M[i] \leftarrow M[j] + 1$
7. $\quad$ If $M[n-1] > k$, ACCEPT.
8. REJECT."

This algorithm runs the Longest Increasing Subsequence DP algorithm to find the longest increasing subsequence of S. If S, k is in the language then there exists some such subsequence with length greater than k; the decider will return true once it finds the last element in such a subsequence. If S, k is not in the language, there cannot exist an increasing subsequence with length greater than k; the longest increasing subsequence will have length at most k and so the decider will reject. This decider runs in polynomial time on the input length. In both the inner and outer loops, it iterates over the length of S at most; this is $O(|S|^2)$, and polynomial in the size of the input. The comparisons are either between numbers in the input, or numbers that are at most $max(|S|, k)$ and therefore have size at most $O(\log|S| + \log k)$, so the comparisons are also efficient. Having constructed an efficient decider, we conclude this must be in P.

An apparent solution to this problem might be to generate all subsequences of S of size $k+1$ (as any longer strictly increasing subsequence contains at least one subsequence of this length), and test them all to see if they are strictly increasing. This is not efficient, however; there are $\binom{|S|}{k+1}$ ways to select k+1-length subsequences, or $\frac{|S|!}{(k+1)!(|S|-(k+1))!}$. This is superpolynomial in $|S|$ and in $|k| = O(\log k)$ and therefore not efficient.

(c) AFFORDABLE-MENU $= \{(\langle L \rangle, k) : L$ is a list of menu items with prices, and there are at least k distinct items on the menu that when added together have a total price $\leq 10k\}$

Consider the following menu:

Item	Price
Cinnamon raisin bread	\$6.00
Macarons	\$13.00
Challah	\$12.00

For this menu (which we will denote by L), $(\langle L \rangle, 2)$ is in AFFORDABLE-MENU because the total price of Cinnamon raisin bread and Challah is $\$6 + \$12 = \$18 < \$20$.

In contrast, $(\langle L \rangle, 3)$ is not in AFFORDABLE-MENU, as there are not three distinct items whose prices added together is less than or equal to \$30.

Solution: We can define a polynomial time decider for this using a greedy algorithm which sorts objects by their price, then takes the cheapest k and checks whether their price is less than or equal to $10k$. A pseudocode implementation is visible here:

AFFORDABLE-MENU = "On input $\langle L \rangle, k$:
1. If $k > |L|$:
2. REJECT
3. Sort L using an efficient sort (mergesort, for example)
4. $sum \leftarrow 0$
5. For i from 1 to k:
6. $sum \leftarrow sum + L[i]$
7. If $sum \leq 10k$, ACCEPT. Else REJECT."

This is a decider. Let us consider the behavior of the greedy algorithm. By sorting and taking the k items with lowest cost, we guarantee that their sum will be equal to or less than any other set of k items (if we switched one item in the greedy-selected set with one not in the set, the one not selected by the algorithm must have a cost greater than or equal to the one in the set, resulting in an overall cost greater than or equal to the greedy cost). If the L, k combination is in the language, then there exists some set of k items such that their cost is less than $10k$; this greedy algorithm will find the cheapest set of k items, which must have value equal to or less than that set; the algorithm accepts. If L, k is not in the language, even the cheapest k items will cost more than $10k$, and the decider will reject.

This decider is also efficient. If $|L| < k$, there are too few elements in $|L|$ and so we reject – there are not k distinct elements. Otherwise, $k \leq |L|$ and is bounded by its length. That means that line 3 occurs in $O(|L|\log|L|)$ time and line 5 in $O(|L|)$ time, making this algorithm polynomial in the length of $|L|$, which is also polynomial in the input size (the size of L in bits plus the size of k in bits), and therefore this is an efficient decider.

2. Show that the following languages are in NP.

   (a) AFFORDABLE-MENU as defined in Problem 1(c)

Solution: In order to show that AFFORDABLE-MENU $\in$ NP, we can construct an efficient verifier.

Our certificate will be a list of the k menu items whose prices added together have a total price that is less that or equal to $10k$.

D = "V = On input $(\langle L, k \rangle, C)$:
1. Check whether C is a valid subset of L of size k. If not, $REJECT$
2. $price \leftarrow 0$
3. Loop through every item of C and add its price to $price$. Verify $price$ does not exceed $10k$. If it exceeds $10k$, $REJECT$
4. $ACCEPT$"

$(\langle L, k\rangle, C) \in$ AFFORDABLE-MENU $\implies$ there is a list C of k menu items that is a subset of L whose total price is less than or equal to $10k$ $\implies$ V ACCEPTS given C
$(\langle L, k\rangle, C) \notin$ AFFORDABLE-MENU $\implies$ there does not exist a list of menu items C that is a subset of L of size k where the total price of the items is less than or equal to $10k$ $\implies$ V REJECTS for all C

We must verify that our solution is polynomial in runtime. Checking for a valid subset is polynomial in time as it simply involves an element-wise comparison. Checking whether the total price of our items is less than or equal to $10k$ involves k additions. We know k is less than or equal to the size of L, because if it were greater, it would not be possible for C to be a valid subset of L. Therefore, the number of additions is less than or equal to the size of L. Thus our verifier is polynomial in runtime.

Alternate solution: We can use the efficient decider we constructed in Problem 1c for AFFORDABLE-MENU to create an efficient verifier. Let the efficient decider that we constructed in Problem 1c be denoted D. Then we can construct an efficient verifier V for AFFORDABLE-MENU as follows.

$D =$ "V = On input $(\langle L, k\rangle, C)$:
1. Ignoring C, run D on $\langle L, k\rangle$ and output the same"

$(\langle L, k\rangle, C) \in$ AFFORDABLE-MENU $\implies$ D accepts when run on $\langle L, k\rangle$ $\implies$ V ACCEPTS given any C $\implies$ There exists some C where V ACCEPTS
$(\langle L, k\rangle, C) \notin$ AFFORDABLE-MENU $\implies$ D rejects when run on $\langle L, k\rangle$ $\implies$ V REJECTS for all C

Therefore, V is a verifier for AFFORDABLE-MENU. We also know that V is efficient because it only runs D which we know is an efficient decider. Therefore, V is an efficient verifier for AFFORDABLE-MENU and we can conclude that AFFORDABLE-MENU $\in$ NP.

Alternate solution: We proved in Problem 1c that AFFORDABLE-MENU $\in$ P, and we know that P $\subseteq$ NP, so we can conclude that AFFORDABLE-MENU $\in$ NP without explicitly writing a verifier.

(b) AFFORDABLE-EXACT-MENU $= \{(\langle L\rangle, k) : L$ is a list of menu items with prices, and there are at least k distinct items on the menu that added together have a total price that is *exactly equal* to $10k\}$

Solution: In order to show that AFFORDABLE-EXACT-MENU $\in$ NP, we must construct an efficient verifier.

Our certificate will be a list of atleast k menu items whose prices added together have a total price that is exactly equal to $10k$.

$D =$ "V = On input $(\langle L, k\rangle, C)$:

1. Check whether C is a valid subset of L of size $\geq k$. If not, $REJECT$
2. $price \leftarrow 0$
3. Loop through every item of C and add its price to $price$. Verify $price$ does not exceed $10k$. If it exceeds $10k$, $REJECT$
4. If $price == 10k$, $ACCEPT$, otherwise $REJECT$"

$(\langle L, k\rangle, C) \in$ Affordable-Exact-Menu $\implies$ there is a list C of atleast k menu items that is a subset of L whose total price exactly equals $10k$ $\implies$ V ACCEPTS given C
$(\langle L, k\rangle, C) \notin$ Affordable-Exact-Menu $\implies$ there does not exist a list of menu items C that is a subset of L of size atleast k where the total price of the items exactly equals $10k$ $\implies$ V REJECTS for all C

We must verify that our solution is polynomial in runtime. Checking for a valid subset is polynomial in time as it simply involves an element-wise comparison. Checking whether the total price of our items is equal to $10k$ involves at most $|L|$ additions because C is a subset of L. Thus our verifier is polynomial in runtime.

(c) Not-Prime $= \{n \in \mathbb{N} : \text{n is not a prime number}\}$

Solution: In order to show that Not-Prime $\in$ NP, we need to build an efficient verifier. The certificate for our verifier will be an integer C.

$D =$ "V = On input (n, C):
1. Check whether C is an integer greater than 1 and less than n. If not, $REJECT$
2. Check whether $n \bmod C \equiv 0$. If so, $ACCEPT$. Else, $REJECT$."

$n \in$ Not-Prime $\implies$ There exists some integer C where $1 < C < n$ that divides n $\implies$ V ACCEPTS given C
$n \notin$ Not-Prime $\implies$ There exists no integer C where $1 < C < n$ that divides n $\implies$ V REJECTS for all C

This algorithm runs in polynomial time as comparing the value of a number to 1 and n is polynomial in time with respect to the size of the integers. The mod operation is also polynomial time with respect to the size of n. Thus our verifier is efficient with respect to our inputs.

3. (a) Recall

$$L_{\text{ACC}} = \{(\langle M \rangle, x) : M \text{ is a TM that accepts } x\}$$

Does the following verifier V where C is an integer prove that $L_{\text{ACC}} \in \mathsf{NP}$? Show why or why not.

V = "On input $(\langle M \rangle, x, C)$:
1. Run M on x for C steps
2. If M accepts x in those C steps, ACCEPT. Else, REJECT."

Solution: V does not prove that $L_{\text{ACC}} \in \mathsf{NP}$. While V is a correct verifier for L_{ACC}, it may not run in polynomial time. The runtime of V is at least $O(C)$ because of step 1, but C is not necessarily a polynomial function of $|\langle M \rangle|$ and $|x|$, thus V fails the requirement for a verifier of an NP language to always run in polynomial time.

(b) Define

$\textsc{First-Digit-Factorial} = \{n \in \mathbb{N} : \text{the first digit of the decimal representation of } n! \text{ is } 2\}$

Does the following program F show that $\textsc{First-Digit-Factorial} \in \mathsf{P}$? Show why or why not.

F = "On input n:
1. $product \leftarrow 1$
2. For i from 2 to n
3. $\quad product \leftarrow product \cdot i$
4. $digit \leftarrow product$
5. While $digit \geq 10$
6. $\quad digit \leftarrow \lfloor digit/10 \rfloor$
7. If $digit = 2$, ACCEPT. Else REJECT."

Solution: F does not prove that $\textsc{First-Digit-Factorial} \in \mathsf{P}$. While F is a correct decider for $\textsc{First-Digit-Factorial}$, it does not run in polynomial time with respect to the size of the input n. F performs at least n steps, which is approximately equal to $2^{|n|}$. Therefore, F actually runs in exponential time with respect to the size of the input n, and thus does not show that $\textsc{First-Digit-Factorial} \in \mathsf{P}$.

E 4. Let EXP be the class of all languages that are decidable in exponential time, i.e., in time $O(2^{n^k})$ for some constant k (where n is the length of input). Formally,

$$\mathsf{EXP} = \bigcup_{k \in \mathbb{N}} \mathsf{DTIME}\left(2^{n^k}\right).$$

It remains unknown whether $\mathsf{NP} = \mathsf{EXP}$, but it is known that $\mathsf{P} \neq \mathsf{EXP}$. Prove that $\mathsf{NP} \subseteq \mathsf{EXP}$. In order words, show that any problem that can be efficiently verified can be decided in exponential time.

Solution: Consider an arbitrary language $L \in \mathsf{NP}$; we will show that $L \in \mathsf{EXP}$ (and thus $\mathsf{NP} \subseteq \mathsf{EXP}$).

By hypothesis, L has a verifier $V(\cdot, \cdot)$ which is polynomial time in the length of the first input, and where $x \in L$ if and only if there exists a certificate c such that $V(x, c)$ accepts. A key observation is that because V runs in at most some polynomial $|x|^k$ time (for some constant $k \geq 1$), it can *read at most* $|x|^k$ *bits* of any given certificate c before halting. Thus, the accept/reject behavior of $V(x, c)$ is determined by x and *the first* $|x|^k$ *bits of* c.

With the above observation, to decide L we can simply do a brute-force search, running the verifier with every possible certificate of appropriate length, and accepting if any of these runs accepts. More precisely, we define the following program M:

```
1: function M(x)
2:     for each possible certificate c of length at most |x|^k do
3:         if V(x, c) accepts then
4:             ACCEPT
5:     REJECT
```

We claim that M decides L. If $x \in L$, then by definition there exists some certificate c of length $\leq |x|^k$ such that $V(x, c)$ accepts. Thus, $M(x)$ will find such a c and accept. If $x \notin L$, then $V(x, c)$ rejects for every certificate c, so $M(x)$ rejects, as required.

We now analyze the running time of $M(x)$. There are fewer than 2^{t+1} bitstrings of length at most t, so the loop will iterate fewer than $2^{|x|^k+1}$ times. Each iteration runs the verifier, which takes at most $|x|^k$ time. Thus $M(x)$ takes time at most

$$2^{|x|^k+1} \cdot |x|^k = 2^{|x|^k+1+k\log|x|} = O(2^{|x|^k}).$$

Thus, M is an exponential-time decider for L, so $L \in \mathsf{EXP}$, as claimed.

Alternate solution: Consider the following decider D which shows that SAT $\in \mathsf{EXP}$.

```
1: function D(φ)
2:     for each possible assignment y of variables in φ do
3:         Compute φ(y)
4:         if φ(y) is true then
5:             ACCEPT
6:     REJECT
```

D always halts because ϕ has a finite number of possible assignments.

$\phi \in$ SAT $\implies$ There exists some satisfying assignment y for ϕ $\implies$ D ACCEPTS

$\phi \notin$ SAT $\implies$ There does not exist some satisfying assignment y for ϕ $\implies$ D REJECTS

Therefore, we can conclude that D is a decider for SAT. The decider runs in $O(|\phi| \cdot 2^{|\phi|})$ as there are $2^m = O(2^{|\phi|})$ possible assignments, where $m \leq |\phi|$ is the number of variables in ϕ, and it takes $O(|\phi|)$ to check if an assignment satisfies ϕ. Therefore, SAT $\in \mathsf{EXP}$. Using Cook-Levin, for any language L in NP, given an input of size n, we can construct a $O(n^{2k})$ Boolean formula that is satisfiable exactly when the input is in language L. Running the

SAT solver D on the formula takes time $O(|\phi| \cdot 2^{|\phi|}) = O(n^{2k} \cdot 2^{O(n^{2k})}) = O(2^{n^{k'}})$ for some constant k'. We can therefore construct a decider that runs in exponential time for every language in NP by first converting the input and efficient verifier to a Boolean formula and then running our decider D for SAT on the formula. Therefore, $\mathsf{NP} \subseteq \mathsf{EXP}$.

5. Consider the language L_D defined by $L_D = (L_A \cup L_B) \cap L_C$

   (a) Is L_D (always/sometimes/never) in P if L_A, L_B and L_C are? Prove your answer.

Solution: L_D is always in P.
First prove that $L_A \cup L_B \in \mathsf{P}$.
Let M_A, M_B respectively be associated efficient deciders for L_A, L_B. We define a decider $M_{A\cup B}$ for the language $L_A \cup L_B$ as follows:

```
1: function M_{A∪B}(x)
2:     if M_A(x) accepts then
3:         ACCEPT
4:     if M_B(x) accepts then
5:         ACCEPT
6:     REJECT
```

We show that $M_{A\cup B}$ satisfies the requirements of a decider for $L_A \cup L_B$. It is clear that $M_{A\cup B}$ is polynomial time in its input, because both M_A, M_B are.
If $x \in L_A \cup L_B$, then either $x \in L_A$, in which case $M_A(x)$ accepts; or $x \in L_B$, in which case $M_B(x)$ accepts. Either way, $M_{A\cup B}(x)$ accepts, as required.
If $x \notin L_A \cup L_B$, then $x \notin L_A$ and $x \notin L_B$, so $M_A(x)$ and $M_B(x)$ both reject, hence $M_{A\cup B}(x)$ rejects, as required.

Now we prove that $L_D = (L_A \cup L_B) \cap L_C \in P$
Let $M_{A\cup B}, M_C$ respectively be associated efficient deciders for $L_A \cup L_B, L_C$. We define a decider M_D for the language L_D as follows:

```
1: function M_D(x)
2:     if M_{A∪B}(x) rejects then
3:         REJECT
4:     if M_C(x) rejects then
5:         REJECT
6:     ACCEPT
```

We show that M_D satisfies the requirements of a decider for $(L_A \cup L_B) \cap L_C$. It is clear that M_D is polynomial time in its input, because both $M_{A\cup B}, M_C$ are.
If $x \in (L_A \cup L_B) \cap L_C$, then $x \in L_A \cup L_B$ and $x \in L_C$, so $M_{A\cup B}(x)$ and $M_C(x)$ both accept. Thus $M_D(x)$ accepts, as needed.
If $x \notin (L_A \cup L_B) \cap L_C$, then $x \notin L_A \cup L_B$ or $x \notin L_C$ (or both). In the former case, $M_{A\cup B}(x)$ rejects, and similarly for the latter case. Either way, $M_D(x)$ rejects, as required.

Because we were able to construct an efficient decider M_D for $L_D = (L_A \cup L_B) \cap L_C$, $L_D \in \mathsf{P}$.

(b) Is L_D (always/sometimes/never) in NP if L_A, L_B and L_C are? Prove your answer.

Solution: L_D is always in NP.

First prove that $L_A \cup L_B \in \mathsf{NP}$.

Let V_A, V_B respectively be associated efficient verifiers for L_A, L_B. We define a verifier $V_{A\cup B}$ for the language $L_A \cup L_B$ as follows:

```
1: function V_{A∪B}(x, c)
2:     parse c = (c_1, c_2) if possible (if not, REJECT)
3:     if V_A(x, c_1) accepts then
4:         ACCEPT
5:     if V_B(x, c_2) accepts then
6:         ACCEPT
7:     REJECT
```

We show that $V_{A\cup B}$ satisfies the requirements of a verifier for $L_A \cup L_B$. It is clear that $V_{A\cup B}$ is polynomial time in its first argument, because both V_A, V_B are.

If $x \in L_A \cup L_B$, then either $x \in L_A$, in which case there exists some c_1 such that $V_A(x, c_1)$ accepts; or $x \in B$, in which case there exist some c_2 such that $V_B(x, c_2)$ accepts. Either way, there exists some $c = (c_1, c_2)$ such that $V_{A\cup B}(x, c = (c_1, c_2))$ accepts, as required.

If $x \notin L_A \cup L_B$, then $x \notin L_A$ and $x \notin L_B$, so $V_A(x, c_1)$ and $V_B(x, c_2)$ both reject for every $c = (c_1, c_2)$, hence $V_{A\cup B}(x, c = (c_1, c_2))$ rejects for every c, as required.

Now we prove that $L_D = (L_A \cup L_B) \cap L_C \in \mathsf{NP}$.

Let $V_{A\cup B}, V_C$ respectively be associated efficient verifiers for $L_A \cup L_B, L_C$. We define a verifier V_D for the language $(L_A \cup L_B) \cap L_C$ as follows:

```
1: function V_D(x, c)
2:     parse c = (c_1, c_2) if possible (if not, REJECT)
3:     if V_{A∪B}(x, c_1) rejects then
4:         REJECT
5:     if V_C(x, c_2) rejects then
6:         REJECT
7:     ACCEPT
```

We show that V_D satisfies the requirements of a verifier for $(L_A \cup L_B) \cap L_C$. It is clear that V_D is polynomial time in its first argument, because both $V_{A\cup B}, V_C$ are.

If $x \in (L_A \cup L_B) \cap L_C$, then $x \in L_A \cup L_B$ and $x \in L_C$, so there exist certificates c_1 and c_2 such that $V_{A\cup B}(x, c_1)$ and $V_C(x, c_2)$ both accept. Thus there exists a certificate $c = (c_1, c_2)$ such that $V_D(x, c = (c_1, c_2))$ accepts, as needed.

If $x \notin (L_A \cup L_B) \cap L_C$, then $x \notin L_A \cup L_B$ or $x \notin L_C$ (or both). In the former case,

$V_{A\cup B}(x, c_1)$ rejects for all c_1, and similarly for the latter case. Either way, $V_D(x, c)$ rejects for all c, as required.

Because we were able to construct an efficient decider V_D for $L_D = (L_A \cup L_B) \cap L_C$, $L_D \in \mathsf{NP}$.