2 Relations Between Time and Space Complexity Classes

Recall that our motivation behind introducing time and space constructible functions was to avoid anomalies like the gap theorem and to regain our intuition that, for any two functions f_1 and f_2 where $f_1 \ll f_2$, $\mathsf{DTIME}(f_1(n)) \subset \mathsf{DTIME}(f_2(n))$. (Likewise for space complexity, or for nondeterministic measures.) In this section, we will begin to develop inclusion relationships between time and space complexity classes depending on the amount of resources we allocate, in turn producing a hierarchy of these classes.

To begin, though, we should address one possible criticism: how can we be sure that restricting our resource bounds to use time constructible functions gives an accurate picture of the hierarchy, when the set of computable functions is so much more general? To address this criticism, we will prove one small result which shows that we can find a time constructible function that is capable of growing as fast as any given computable function.

Lemma 11. *For every computable function f, there exists a time constructible function g such that, for all $n \in \mathbb{N}$, $g(n) > f(n)$.*

Proof. Let $\mathcal{M}_f$ be a deterministic Turing machine that receives as input the word $\mathtt{1}^n$ and produces as output the word $\mathtt{1}^{f(n)}$; that is, $\mathcal{M}_f$ computes the function f.

Similarly, let $\mathcal{M}_g$ be a deterministic Turing machine that, given an input word of length n, writes $\mathtt{1}^n$ to a work tape and then simulates the computation of $\mathcal{M}_f$ on the input $\mathtt{1}^n$.

Denote by $g(n)$ the number of computation steps used by $\mathcal{M}_g$ on its input of length n. Then g is time constructible, since it is the running time of $\mathcal{M}_g$, and since $n + f(n)$ time is required to write $\mathtt{1}^n$ to a work tape and simulate the computation of $\mathcal{M}_f$, we have that $g(n) > f(n)$. □

As a consequence of Lemma 11, we obtain our main result: time constructible functions are enough for our purposes.

Theorem 12. *For any computable function f, there exists a time constructible function g such that*

$$\mathsf{DTIME}(f(n)) \subseteq \mathsf{DTIME}(g(n)).$$

Proof. Follows from the existence of the Turing machines constructed in the proof of Lemma 11. □

Since any computable resource bound is included in some time constructible resource bound, we are not missing anything from our picture, and so we can move on with producing our hierarchy.

2.1 Time and Space Hierarchy Theorems

The first step in our hierarchy will be to establish strict inclusion relationships between complexity classes involving the same model when we give that model differing amounts of resources. In the deterministic case, these relationships are characterized by two theorems: the *time hierarchy theorem* and the *space hierarchy theorem.* As we did with our constructibility results, we will first consider the space hierarchy theorem, as its formulation is simpler and its proof will motivate our proof of the time hierarchy theorem.

If we give a machine more space, then we would expect it to do more with that additional space as per our intuition. However, if this additional space, say s', only exceeds the original amount of space, say s, by a small amount, then we may not gain any advantage—indeed, we may even need more space than we're given just to compute the small difference itself! Thus, if we restrict ourselves to space constructible functions as we've been doing all along, the space hierarchy theorem tells us that languages exist that can be decided in some amount of space s', but not in space s.

Theorem 13 (Space hierarchy theorem)**.** *For any space constructible function $s(n) \geq \log(n)$, there exists a language $L \in \mathsf{DSPACE}(s(n))$ that cannot be recognized by any deterministic Turing machine using space $o(s(n))$.*

Proof. We want to show that there exists some language L that we can recognize in space $O(s(n))$, but not in space $o(s(n))$.

Consider two functions s_1 and s_2, where $s_1(n) \geq \log(n)$ and $s_1 \in o(s_2(n))$. We construct a deterministic Turing machine $\mathcal{M}$ which takes some input $w \in \{\mathtt{0}, \mathtt{1}\}^*$ and uses $s_2(n)$ space to perform its computation. Specifically, $\mathcal{M}$ will do the following:

1. Mark $s_2(|w|)$ work tape cells. If the computation tries to use more space, then $\mathcal{M}$ rejects.
2. Write w to the work tape in the form $w = w_1 w_2$, where $w_1 = \mathtt{1}^*\mathtt{0}$.
3. Check that w_2 is a correct encoding of a Turing machine $\mathcal{M}_2$, and simulate the computation of $\mathcal{M}_2$ on the input word w. If the simulation halts and rejects, then $\mathcal{M}$ accepts. Otherwise, $\mathcal{M}$ rejects.

Now, assume that the language $L(\mathcal{M})$ is recognized by some deterministic Turing machine $\mathcal{M}'$ using space $s_1(n)$. Let r denote the number of work tape symbols used by $\mathcal{M}'$, and let $\text{bin}(\mathcal{M}')$ denote the binary representation of the Turing machine $\mathcal{M}'$. Choose some $n \geq |\text{bin}(\mathcal{M}')|$ such that

$$\lceil \log(r) \rceil \cdot s_1(n) < s_2(n). \tag{1}$$

Finally, choose some word $w = \mathtt{1}^j\mathtt{0}\text{bin}(\mathcal{M}')$, where j is such that $|w| = n$.

By Equation 1, we know that $\mathcal{M}$ can simulate $\mathcal{M}'$ in space $s_2(n)$. Thus, running $\mathcal{M}$ on the input word w is equivalent to simulating the computation of $\mathcal{M}'$ on the input word w. However, if $\mathcal{M}'$ accepts w, then $\mathcal{M}$ will reject w, and vice versa! Therefore, we see that $L(\mathcal{M}) \neq L(\mathcal{M}')$. □

The proof of the space hierarchy theorem uses the familiar technique of diagonalization that we've seen before. Furthermore, note the use of little-o notation in the statement of the theorem. This notation effectively says that any deterministic Turing machine using strictly less than $s(n)$ space cannot recognize the language L.

As a consequence of the space hierarchy theorem, we get the following corollary.

Corollary 14. *For any space constructible function s,*

$$\mathsf{DSPACE}(o(s(n)) \subset \mathsf{DSPACE}(s(n)).$$

We can also adapt the space hierarchy theorem to work for nondeterministic Turing machines with not much more effort.

We now turn to the time hierarchy theorem, where the diagonalization argument we'll use will be quite similar to that used in the proof of the space hierarchy theorem, but with one additional snag we must account for: the diagonalizing Turing machine will have a fixed number of work tapes k_1, but it must be capable of simulating Turing machines having any number of work tapes k_2 where $k_1 < k_2$. This simulation can't be done without suffering an impact to the time bound, so the diagonalizing Turing machine will incur a logarithmic slowdown in its computation, which in turn leads to a weaker statement for the time hierarchy theorem. Namely, we require that the time bound be increased by a logarithmic factor to account for the slowdown.

Theorem 15 (Time hierarchy theorem)**.** *For any time constructible function $t(n) \geq n\log(n)$, there exists a language $L \in \mathsf{DTIME}(t(n))$ that cannot be recognized by any deterministic Turing machine using time $o(t(n)/\log(t(n)))$.*

Proof. We want to show that there exists some language L that we can recognize in time $O(t(n))$, but not in time $o(t(n)/\log(t(n)))$.

The proof of this theorem goes through in much the same way as in the proof of Theorem 13. We construct a deterministic Turing machine $\mathcal{M}$ which takes some input $w \in \{\mathtt{0}, \mathtt{1}\}^*$ and uses $t(n)$ time to perform its computation. Specifically, $\mathcal{M}$ will do the following:

1. Compute $t(n)$ and store the binary representation of the value $\lceil t(n)/\log(t(n)) \rceil$ on a work tape. For each computation step, decrement this value by one. If the value stored on this work tape ever reaches 0, then $\mathcal{M}$ rejects.
2. Write w to the work tape in the form $w = w_1 w_2$, where $w_1 = \mathtt{1}^*\mathtt{0}$.
3. Check that w_2 is a correct encoding of a Turing machine $\mathcal{M}_2$, and simulate the computation of $\mathcal{M}_2$ on the input word w. If the simulation halts and rejects, then $\mathcal{M}$ accepts. Otherwise, $\mathcal{M}$ rejects.

Regardless of how long the simulation of $\mathcal{M}_2$ takes to compute, it will be time-bounded by the counter on the work tape, so $\mathcal{M}$ will perform at most $\lceil t(n)/\log(t(n)) \rceil$ computation steps on the simulation alone. Since the counter value is stored in binary, it has a length of $\log(t(n)/\log(t(n)))$, which is $O(\log(t(n)))$. This means that the process of updating the counter adds a factor of $\log(t(n))$ to the computation time, and so $\mathcal{M}$ recognizes its language in time $O(t(n))$.

Now, assume that the language $L(\mathcal{M})$ is recognized by some deterministic Turing machine $\mathcal{M}'$ using time $f(n) \in o(t(n)/\log(t(n)))$, and consider what happens when we simulate the computation of $\mathcal{M}'$ on $\mathcal{M}$. Recall that the simulation component of $\mathcal{M}$ takes at most $\lceil t(n)/\log(t(n)) \rceil$ computation steps to complete. Since $f(n) \in o(t(n)/\log(t(n)))$, there exists some value n_0 where $c \cdot f(n) < t(n)/\log(t(n))$ for all $n \geq n_0$. Therefore, $\mathcal{M}$ will only be able to complete the simulation of $\mathcal{M}'$ if its input word has length at least n_0.

Choose some word $w = \mathtt{1}^{n_0}\mathtt{0}\text{bin}(\mathcal{M}')$, where $\text{bin}(\mathcal{M}')$ is as defined in the proof of Theorem 13. Since this input has sufficient length, $\mathcal{M}$ will complete its simulation. However, if $\mathcal{M}'$ accepts w, then $\mathcal{M}$ will reject w, and vice versa! Therefore, we see that $L(\mathcal{M}) \neq L(\mathcal{M}')$. □

Again, we get the following corollary as a consequence of the time hierarchy theorem.

Corollary 16. *For any time constructible function t,*

$$\mathsf{DTIME}\left(o\left(\frac{t(n)}{\log(t(n))}\right)\right) \subset \mathsf{DTIME}(t(n)).$$

Interestingly, the nondeterministic analogue to the time hierarchy theorem doesn't require the extra logarithmic term! However, we will not consider the nondeterministic version here.

2.2 Savitch's Theorem

Up to now, the relationships we've seen between time and space complexity classes has been either among like classes—such as comparing DTIME and DTIME, or DSPACE and DSPACE—or going from deterministic to nondeterministic classes—such as comparing DTIME and NTIME, or DSPACE and NSPACE. However, it is possible for us to establish further relationships between these classes to get a better picture of how they interact.

For instance, we know that $\mathsf{DTIME}(f(n)) \subseteq \mathsf{NTIME}(f(n))$ for any function $f(n)$, but what about the other way around? Well, if we could prove the same inclusion but in reverse, then we would also have proved that $\mathsf{P} = \mathsf{NP}$. Since nobody has been able to successfully do this (yet...), we will have to settle for a weaker relationship in the other direction. Namely, we can show that any nondeterministic computation taking $f(n)$ time can also be done by a deterministic machine in time exponential in $f(n)$.

Theorem 17. *For any time constructible function f,*

$$\mathsf{NTIME}(f(n)) \subseteq \mathsf{DTIME}(2^{O(f(n))}).$$

Proof. To prove this result, we must first prove that if f is a space constructible function, then $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{DTIME}(2^{O(f(n))})$. Since we took f to be time constructible, it is also space constructible.

Let $\mathcal{M}$ be a nondeterministic Turing machine that accepts input words of length n in space $f(n)$. We can modify $\mathcal{M}$ so that, if it accepts its input word, then it erases all of its work tapes and moves all input heads to their leftmost tape cells before accepting. This does not affect the language recognized by $\mathcal{M}$, but it does result in $\mathcal{M}$ having exactly one accepting configuration. Moreover, we know that an $f(n)$-space-bounded nondeterministic Turing machine can be modified not to repeat configurations as a corollary of Theorem 10. Therefore, the number of configurations of the $f(n)$-space-bounded version of $\mathcal{M}$ is $2^{O(f(n))}$.

Next, let $\mathcal{N}$ be a deterministic Turing machine that, given an input word w of length n, constructs a graph where the vertices correspond to configurations of $\mathcal{M}$ on input w and where there exists a directed edge from vertex v_i to vertex v_j if and only if configuration C_j is reachable from configuration C_i in one computation step. The machine $\mathcal{N}$ checks whether there exists a path of length at most n from the vertex corresponding to the initial configuration C_0 to the vertex corresponding to the unique accepting configuration C_{acc}, and if such a path exists, then $\mathcal{N}$ accepts. In other words, $\mathcal{N}$ solves a specific instance of the problem S-T-Connectivity on the configuration graph of $\mathcal{M}$.

Because $\mathcal{N}$ accepts only those words for which an accepting computation of $\mathcal{M}$ exists, $L(\mathcal{N}) = L(\mathcal{M})$, and $\mathcal{N}$ recognizes its language in time $2^{O(f(n))}$.

Finally, recall that $\mathsf{NTIME}(f(n)) \subseteq \mathsf{NSPACE}(f(n))$ for all functions $f(n)$. Altogether, we have that

$$\mathsf{NTIME}(f(n)) \subseteq \mathsf{NSPACE}(f(n)) \subseteq \mathsf{DTIME}(2^{O(f(n))}). \qquad \square$$

Turning to space complexity, we might reasonably expect that like the relationship between nondeterministic and deterministic time, there is an exponential gap between nondeterministic and deterministic space. After all, we know that $\mathsf{DTIME}(f(n)) \subseteq \mathsf{DSPACE}(f(n))$ for all functions f, so Theorem 17 also tells us that $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{DSPACE}(2^{O(f(n))})$.

However, an astonishing result published by the American computer scientist Walter Savitch in 1970 shows that we can do *much* better than an exponential gap—we only need a quadratic difference between nondeterministic and deterministic space!

Theorem 18 (Savitch's theorem)**.** *For any space constructible function $f(n) \geq \log(n)$,*

$$\mathsf{NSPACE}(f(n)) \subseteq \mathsf{DSPACE}(f(n)^2).$$

Proof. Let $\mathcal{M}$ be a nondeterministic Turing machine that accepts input words of length n in space $f(n) \geq \log(n)$. We construct a deterministic Turing machine $\mathcal{N}$ that takes as input two vertices v_i and v_j from some graph $G = (V, E)$ together with some $k \in \mathbb{N}$ and accepts if vertex v_j is reachable from vertex v_i after following at most k edges. In other words, $\mathcal{N}$ solves the general problem S-T-Connectivity, but unlike in the proof of Theorem 17, here $\mathcal{N}$ takes the graph as input instead of constructing it.

The sticking point here is that we cannot solve this connectivity problem by simply running a breadth-first search or depth-first search on the input graph, since this would require $\Omega(n)$ time where $n = |V|$ and would therefore rule out sublinear functions $f(n)$ from consideration. In order to make our connectivity procedure more space-efficient, we must make use of recursion. Our Turing machine $\mathcal{N}$ will therefore run the following steps:

1. If $k = 1$:
   (a) If $v_i = v_j$ or if $(v_i, v_j) \in E$, then accept. Otherwise, reject.
2. If $k > 1$, then for each vertex $u \in V \setminus \{v_i, v_j\}$:
   (a) Run $\mathcal{N}$ on the input v_i, u, and $\lfloor k/2 \rfloor$.
   (b) Run $\mathcal{N}$ on the input u, v_j, and $\lceil k/2 \rceil$.
   (c) If both computations are accepting, then accept. Otherwise, reject.

Before proceeding, we modify $\mathcal{M}$ as in the proof of Theorem 17, where if $\mathcal{M}$ accepts its input word, then it erases all of its work tapes and moves all input heads to their leftmost tape cells before accepting. In doing so, $\mathcal{M}$ has exactly one accepting configuration C_{acc}. Recall also that, since $\mathcal{M}$ is an $f(n)$-space-bounded Turing machine, it has at most $2^{O(f(n))}$ configurations.

Now, we run $\mathcal{N}$ on the configuration graph of $\mathcal{M}$, where the configuration graph is again defined as in the proof of Theorem 17. We give $\mathcal{N}$ the following input: C_0, the initial configuration of the modified $\mathcal{M}$; C_{acc}, the accepting configuration of the modified $\mathcal{M}$; and the value $2^{c \cdot f(n)}$ for some constant c. If $\mathcal{N}$ accepts, then $\mathcal{M}$ accepts as there must exist some path through the configuration graph from C_0 to C_{acc}, and so $\mathcal{N}$ simulates the computation of $\mathcal{M}$.

It remains to show how much space is used by $\mathcal{N}$ over the course of its computation. Each time the computation of $\mathcal{N}$ recurses, it stores the input values v_i, v_j, and k for later retrieval, and storing these values as their binary representations uses logarithmic space, which is $O(f(n))$. Moreover, each recursive call divides the space used by half. Since we start the computation with the value $k = 2^{c \cdot f(n)}$, the depth of the recursion is $O(\log(2^{c \cdot f(n)}))$, or $O(f(n))$. In total, the amount of space used across all levels of the recursive computation is $O(f(n) \cdot f(n))$, or $O(f(n)^2)$ as required. □

So, what makes Savitch's theorem work? It essentially comes down to one key idea that we mentioned previously: unlike time, space is a resource that can be reused. In our proof, we are reusing space by way of recursion, and testing subgraphs of the configuration graph instead of performing a more intensive search. Savitch's theorem is among the earliest results in complexity theory, and the question of whether the quadratic gap it gives can be improved is long-standing.

2.3 The Fundamental Complexity Hierarchy

We now have enough machinery to prove all of the other known relations between the basic complexity classes we defined in the previous lecture and, in doing so, to establish the fundamental complexity hierarchy. We begin by proving the most elementary relations between complexity classes, which follow immediately from what we know about DTIME and NTIME and about DSPACE and NSPACE.

Theorem 19. *L ⊆ NL.*

Proof. Follows immediately from the fact that DSPACE$(\log(n)) \subseteq$ NSPACE$(\log(n))$. □

Theorem 20. *The following inclusions hold:*

1. P ⊆ NP.
2. EXP ⊆ NEXP.

Proof. Follows immediately from the fact that DTIME$(f(n)) \subseteq$ NTIME$(f(n))$. □

With just these relations, we can begin to draw the fundamental complexity hierarchy as a diagram (which will become much clearer as we continue to add further relations):

L ⊆ NL   P ⊆ NP   PSPACE   NPSPACE   EXP ⊆ NEXP   EXPSPACE   NEXPSPACE

Next, we will draw connections between nondeterministic time complexity classes and deterministic space complexity classes. To obtain these particular inclusions, we must prove an intermediate result relating the classes NTIME and DSPACE.

We already know that NTIME$(f(n)) \subseteq$ NSPACE$(f(n))$, and we get as a consequence of Theorem 17 that NTIME$(f(n)) \subseteq$ DSPACE$(2^{O(f(n))})$. However, we can greatly narrow this exponential gap between NTIME and DSPACE to obtain a much-improved relationship.

Theorem 21. *The following inclusions hold:*

1. NP ⊆ PSPACE.
2. NEXP ⊆ EXPSPACE.

Proof. Let f be a time constructible function and let $\mathcal{M}$ be a nondeterministic Turing machine that accepts words of length n in time $f(n)$. Since f is time constructible, we can define a deterministic Turing machine $\mathcal{N}$ that simulates $f(n)$ steps of each computation path of $\mathcal{M}$ and accepts if $\mathcal{M}$ accepts. If no accepting computation path exists, then $\mathcal{N}$ rejects. Each of these simulated computations is performed using the same space, and we can keep track of which computation path we're simulating by maintaining an $f(n)$-bit counter on a work tape. Therefore, $\mathcal{N}$ simulates the computation of $\mathcal{M}$ in space $f(n)$, and so $\mathsf{NTIME}(f(n)) \subseteq \mathsf{DSPACE}(f(n))$. Both of the inclusions listed in the statement of the theorem follow directly. $\square$

Adding these relations to our diagram, we get the following:

$$\mathsf{L} \subseteq \mathsf{NL} \qquad \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \qquad \mathsf{NPSPACE} \qquad \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXPSPACE} \qquad \mathsf{NEXPSPACE}$$

We turn now to an important consequence of Savitch's theorem. Recall that, when we defined space complexity classes like $\mathsf{NPSPACE}$ and $\mathsf{NEXPSPACE}$, we mentioned that we wouldn't pay too much mind to such classes because of an "interesting result" that meant we didn't need to define such classes independently. That interesting result, as we now know, is none other than Savitch's theorem.

Since we know that the class DSPACE is contained within the class NSPACE, and since Savitch's theorem tells us that NSPACE is contained within "DSPACE squared", we obtain equality between deterministic and nondeterministic space complexity classes that use at least polynomial space!

Theorem 22. *The following equalities hold:*

1. $\mathsf{PSPACE} = \mathsf{NPSPACE}$.
2. $\mathsf{EXPSPACE} = \mathsf{NEXPSPACE}$.

Proof. Follows immediately from Savitch's theorem and the fact that $\mathsf{DSPACE}(f(n)) \subseteq \mathsf{NSPACE}(f(n))$. $\square$

These results are effectively the answers to the space complexity equivalent of the P vs. NP problem. As a consequence, we will speak no further of the classes $\mathsf{NPSPACE}$ or $\mathsf{NEXPSPACE}$ on their own.

Note, however, that Savitch's theorem does not also allow us to cast away the class NL, since it does not establish that $\mathsf{L} = \mathsf{NL}$. This is because if f is at least a polynomial function, then f^2 is also at least polynomial, but the square of a logarithmic function is not polynomial. Thus, the only thing Savitch's theorem can tell us is that $\mathsf{NL} \subseteq \mathsf{DSPACE}(\log(n)^2)$, and clearly we have that $\mathsf{L} \neq \mathsf{DSPACE}(\log(n)^2)$. Indeed, the question of whether $\mathsf{L} = \mathsf{NL}$ is another long-standing problem in complexity theory.

The two equalities we do obtain from Savitch's theorem allow us to simplify our diagram somewhat:

$$\mathsf{L} \subseteq \mathsf{NL} \qquad \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE} \qquad \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXPSPACE} = \mathsf{NEXPSPACE}$$

It remains for us to prove two more inclusions individually, but fortunately, both of these relationships follow as a consequence of a prior result we investigated.

Theorem 23. $\mathsf{NL} \subseteq \mathsf{P}$.

Proof. Follows from the proof of Theorem 17 where we showed that $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{DTIME}(2^{O(f(n))})$ for any space constructible function f. Namely, if we take $f(n) = \log(n)$, then we have that

$$\mathsf{NSPACE}(\log(n)) \subseteq \mathsf{DTIME}(2^{\log(n)}) = \mathsf{DTIME}(n) \in \mathsf{P}. \qquad \square$$

Theorem 24. $\mathsf{PSPACE} \subseteq \mathsf{EXP}$.

Proof. Follows from the proof of Theorem 17 where we showed that $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{DTIME}(2^{O(f(n))})$ for any space constructible function f, together with the fact that $\mathsf{DSPACE}(f(n)) \subseteq \mathsf{NSPACE}(f(n))$ for any function f. Namely, if we take $f(n) = n^k$ for any $k \geq 0$, then we have that

$$\mathsf{DSPACE}(n^k) \subseteq \mathsf{NSPACE}(n^k) \subseteq \mathsf{DTIME}(2^{n^k}) \in \mathsf{EXP}. \qquad \square$$

These results, in turn, fill in the remaining gaps of our diagram:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXPSPACE} = \mathsf{NEXPSPACE}$$

Going one step further, we can establish a handful of strict inclusion results between certain complexity classes, and the strictness of each of these results is due to the time and space hierarchy theorems we studied in an earlier section.

Theorem 25. $\mathsf{P} \subset \mathsf{EXP}$.

Proof. Follows immediately from the time hierarchy theorem. □

Theorem 26. $\mathsf{NL} \subset \mathsf{PSPACE} \subset \mathsf{EXPSPACE}$.

Proof. Follows immediately from the space hierarchy theorem. □

All told, our fundamental complexity hierarchy looks like the following:

$$\begin{tikzcd}
\mathsf{L} \arrow[r, phantom, "\subseteq"] & \mathsf{NL} \arrow[r, phantom, "\subseteq"] \arrow[rrr, bend left, dash, "\subset"] & \mathsf{P} \arrow[r, phantom, "\subseteq"] \arrow[rrr, bend right, dash, "\subset"'] & \mathsf{NP} \arrow[r, phantom, "\subseteq"] & \mathsf{PSPACE} = \mathsf{NPSPACE} \arrow[r, phantom, "\subseteq"] \arrow[rrr, bend left, dash, "\subset"] & \mathsf{EXP} \arrow[r, phantom, "\subseteq"] & \mathsf{NEXP} \arrow[r, phantom, "\subseteq"] & \mathsf{EXPSPACE} = \mathsf{NEXPSPACE}
\end{tikzcd}$$

As a consequence of Theorems 25 and 26, we know that in the chains of inclusions $\mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$ and $\mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXPSPACE}$, *at least* one of the inclusions in each chain must be strict. However, nobody knows which one—or indeed, which ones—should be! The general consensus among complexity theorists is that *all* of the inclusions in these two chains are strict.