Lecture 21

The Time and Space Hierarchy Theorems

In this lecture, we will prove that $\mathsf{P} \neq \mathsf{EXP}$ and that $\mathsf{PSPACE} \neq \mathsf{EXPSPACE}$. These will follow from the so-called "Hierarchy theorems": the Time Hierarchy Theorem says that you can always decide strictly more languages when given a strictly larger time budget (assuming some technical conditions), and similarly, the Space Hierarchy Theorem says that you can always decide strictly more languages when given a strictly larger space budget.

So while we do not know how to separate pairs of classes such as P and PSPACE, we do know how to separate two classes that are both defined purely in terms of time (like P and EXP) or purely in terms of space (like PSPACE and $\mathsf{EXPSPACE}$).

21.1 Time-constructible functions

To get to the statement of the Time Hierarchy Theorems, we will need to define the slightly technical notion of a time-constructible function.

Definition 21.1. *A function $f\colon \mathbb{N} \to \mathbb{N}$ is called* time constructible *if it satisfies $f(n) = \Omega(n \log n)$, and if, in addition, there exists a Turing machine M such that*

1. *$M(0^n)$ outputs $\langle f(n) \rangle$ (in binary notation) for each $n \in \mathbb{N}$*
2. *The running time t of M satisfies $t(n) = O(f(n))$.*

In other words, a function is time-constructible if we can compute $f(n)$ it time which is itself at most $f(n)$. Actually, it is even easier to be time constructible than that: we just need to be able to compute $f(n)$ when given 0^n, rather than when given $\langle n \rangle$ as input (recall that the latter would only take $O(\log n)$ bits to specify, so the length of the string is n for the string 0^n but it's $O(\log n)$ for the string $\langle n \rangle$). We also require that time-constructible functions grow at least as fast as $n \log n$.

Why are we introducing such a weird definition? The answer is the time-constructible functions are those that can serve as upper bounds on computation by a Turing machine. That is, suppose we want to simulate a Turing machine M on an input w, but we want to cut it off if it takes more than $f(|w|)$ steps. Can we do this? Well, to do so we'd have to first compute $f(|w|)$, which itself might take a long time. Then we would simulate M while counting its steps, and if the number of steps exceeds the number $f(|w|)$ that we wrote down, we'd terminate. To ensure that this simulation can be done using only something like $O(f(|w|))$ steps, we need as a requirement that computing the

function f itself doesn't take too long. This is exactly where the definition of a time-constructible function comes from.

As it turns out, almost any function you might encounter which grows faster than $n \log n$ is going to be time-constructible. This includes

1. The functions $f(n) = n^k$ for any integer $k \geq 2$
2. The functions $f(n) = k^n$ for any integer $k \geq 2$
3. The function $f(n) = \lceil n\sqrt{n} \rceil$ and functions similar to it.

Also, combining time constructible functions gives rise to new time constructible functions. If f_1 and f_2 are time constructible, then

1. the function $f_1(n) + f_2(n)$ is time-constructible
2. the function $f_1(n) \cdot f_2(n)$ is time-constructible
3. the function $f_1(f_2(n))$ is time-constructible.

We won't go through the details of these proofs. The point is just that it is very easy to satisfy the conditions of a time-constructible function, and only very strange functions (or functions growing slower than $n \log n$) do not satisfy the conditions. That's because our time budget for computing $f(n)$ is pretty large: for instance, if $f(n) = n^2$, we just need to compute $\langle n^2 \rangle$ given 0^n in $O(n^2)$ time. But we can convert 0^n to the binary notation $\langle n \rangle$ in at most $O(n \log n)$ time, and then we can use a simple algorithm like long multiplication to compute $\langle n^2 \rangle$ in around $|\langle n \rangle|^2$ time, which is only $O(\log^2 n)$ time. In other words, computing n^2 from n is so easy we can do it in $O(\log^2 n)$ time, let alone $O(n^2)$ time (though we will need a time budget of at least $O(n \log n)$ to convert the unary notation 0^n we are given into binary notation).

21.2 The Time Hierarchy Theorem

We are now ready to state the Time Hierarchy theorem.

Theorem 21.2 (Time Hierarchy Theorem)**.** *Let f and g be time constructible functions such that $f(n) = o(g(n)/\log g(n))$. Then*

$$\mathsf{TIME}(f) \subsetneq \mathsf{TIME}(g).$$

This theorem says that if f and g are time-constructible and if f grows substantially slower than g (indeed, slower than g by more than a factor of $\log g$), then there is at least one language decidable in $O(g(n))$ time which is not decidable in $O(f(n))$ time.

How might we prove this theorem? We want to construct a language decidable in $O(g(n))$ time but not in $O(f(n))$ time, but how do we rule out all algorithms that run in $O(f(n))$ time?

The idea essentially uses a diagonalization argument. Let $h\colon \mathbb{N} \to \mathbb{N}$ be the function $h(n) = \lceil g(n)/\log g(n) \rceil$. Note that $f(n) = o(h(n))$ by the conditions of the theorem statement. Our language A will be

$$A = \{\langle M \rangle 01^k : k \in \mathbb{N}, M \text{ is a TM, } M(\langle M \rangle 01^k) \text{ rejects within } h(|\langle M \rangle 01^k|) \text{ steps}\}.$$

Observe that this language cannot be decided by any Turing machine with running time $O(f(n))$. Why? Because if we had such a Turing machine M deciding A, then we could run it on inputs $w_k = \langle M \rangle 01^k$. Now, let t be the running time of M. Then $t(n) = O(f(n))$, and hence $t(n) =$

$o(h(n))$. This means that for some sufficiently large k, we will have $t(|w_k|) < h(|w_k|)$. If we run $M(w_k)$ for this k, the machine M will halt within $t(|w_k|) < h(|w_k|)$ steps. Now, if M rejects the string $w_k = \langle M \rangle 01^k$, then by the definition of A, w_k must be in A, because it is of the form $\langle M \rangle 01^k$ and $M(\langle M \rangle 01^k)$ rejects within $h(|\langle M \rangle 01^k|)$ steps; but since M decides A, this means that M must accept w_k. Conversely, if M accepts w_k, then by the definition of A, $w_k = \langle M \rangle 01^k$ cannot be in A; but this means that M must reject it. Either way, we get a contradiction.

This construction of A basically "diagonalizes" over the possible Turing machines M that run faster than $h(n)$; the definition itself is set up to ensure that no such TM can decide A. This allows us to conclude that $A \notin \mathsf{TIME}(f)$. To complete the proof, we need to show that A is in $\mathsf{TIME}(g)$.

To do so, we will construct a Turing machine M' that decides A in $O(g(n))$ time. This TM will work as follows. Given input w, it will first check the formatting: it will find the last 0 in the string, and check that everything before the last 0 can be viewed as an encoding of a Turing machine $\langle M \rangle$. Finding the last 0 of a string can be done in linear time. Checking the encoding depends on exactly how a Turing machine is encoded as a string, but we will claim (without proof, as the proof is technical) that this can be done in $O(n \log n)$ time, where n is the length of the string we are checking.

Next, the Turing machine M' will simulate M on input w for $h(|w|)$ steps. If $M(w)$ rejects within $h(|w|)$ steps, M' will accept this input. Otherwise, it will reject.

It should be clear that this causes M' to correctly decide the language A (it is just following the definition of A when deciding whether to accept or reject a string). The tricky part is to show that the running time of M' is no more than $O(g(|w|))$.

The idea here is that the function $h(n) \log h(n)$ is at most $O(g(n))$, which follows from our definition of $h(n) = \lceil g(n)/\log g(n) \rceil$ (this requires proof, but we will omit it). In other words, we just need to show how to simulate a given Turing machine $\langle M \rangle$ on a given string w for $T = h(|w|)$ steps, using at most $O(T \log T)$ steps for our simulation. For this, we essentially need to analyze the running time of the universal Turing machine U (the one that takes in a TM and an input, and simulates that TM on that input). This analysis is rather technical. Naively, it would take a number of steps that is polynomial (perhaps quadratic) in the number of steps of the simulated program M. However, there are some technical tricks that can bring this down from a quadratic overhead to an $O(T \log T)$ overhead. We will not go through the technical details of how to do this.

This completes the proof sketch of the Time Hierarchy Theorem. Note that a lot of the technicalities we skipped have to do with the implementation details of a Turing machine. Such technicalities are necessary if we want to show $\mathsf{TIME}(f) \subsetneq \mathsf{TIME}(g)$ where f grows only a little slower than g (say, $f(n) = o(g(n)/\log g(n))$).

If we merely wanted to show $\mathsf{TIME}(f) \subsetneq \mathsf{TIME}(g)$ for f and g satisfying $f(n) = o(g(n)^{1/100})$, this would be much easier: the simulation overhead we would need to achieve would be something like T^{100} (i.e. we would need to show that we can simulate a Turing machine given M on a given string w for T steps using at most $O(T^{100})$ steps). This is pretty easy to argue.

In fact, this easier version of the Time Hierarchy Theorem is already enough to conclude the following.

Corollary 21.3. $\mathsf{P} \subsetneq \mathsf{EXP}$.

Proof. Recall that $\mathsf{P} = \bigcup_{k \in \mathbb{N}} \mathsf{TIME}(n^k)$. Consider the function $2^{\sqrt{n}}$. We have $n^k = O(2^{\sqrt{n}})$ for every constant $k \in \mathbb{N}$. Hence $\mathsf{TIME}(n^k) \subseteq \mathsf{TIME}(2^{\sqrt{n}})$ for all $k \in \mathbb{N}$. It follows that $\mathsf{P} \subseteq \mathsf{TIME}(2^{\sqrt{n}})$. It's not hard to show that $\lceil 2^{\sqrt{n}} \rceil$ is time-constructible, and so is 2^n. We also have $2^{\sqrt{n}} = O((2^n)^{1/100}) = O(2^{n/100})$, and certainly $2^{\sqrt{n}} = O(2^n/\log 2^n) = O(2^n/n)$. From the Time Hierarchy Theorem, it follows that $\mathsf{TIME}(2^{\sqrt{n}}) \subsetneq \mathsf{TIME}(2^n)$, and hence $\mathsf{P} \subsetneq \mathsf{TIME}(2^n)$. Since $\mathsf{TIME}(2^n) \subseteq \mathsf{EXP}$, we conclude that $\mathsf{P} \subsetneq \mathsf{EXP}$. □

Note that this proof of the corollary also proves that $\mathsf{P} \subsetneq \mathsf{E}$. It is also not hard to see (via a similar application of the Time Hierarchy Theorem) that $\mathsf{E} \subsetneq \mathsf{EXP}$.

21.3 The Space Hierarchy Theorem

The Space Hierarchy Theorem says something very similar to the Time Hierarchy Theorem, just for space instead of time.

Theorem 21.4 (Space Hierarchy Theorem)**.** *Let f and g be time-constructible functions, and suppose that $f(n) = o(g(n))$. Then*

$$\mathsf{SPACE}(f) \subsetneq \mathsf{SPACE}(g).$$

Note that the statement of the Space Hierarchy theorem is slightly cleaner than that of the Time Hierarchy Theorem: we simply require that $f(n) = o(g(n))$, instead of the more complicated condition $f(n) = o(g(n)/\log g(n))$.

Proof. The proof of the Space Hierarchy Theorem is pretty much the same as that of the Time Hierarchy Theorem. We need to find a language B which is in $\mathsf{SPACE}(g)$ but not in $\mathsf{SPACE}(f)$. This language b will "diagonalize" over the Turing machines that run in at most $g(n)$ space. We will set

$$B = \{\langle M \rangle 01^k : k \in \mathbb{N}, M \text{ is a TM, } M(\langle M \rangle 01^k) \text{ rejects and uses at most } g(|\langle M \rangle 01^k|) \text{ space}\}.$$

If M is any Turing machine that with space function s satisfying $s(n) = O(f(n))$, then we have $s(n) = o(g(n))$. In this case, we claim M cannot decide B. To see this, suppose that M decided B. For each $k \in \mathbb{N}$, let w_k be the string $\langle M \rangle 01^k$. Then for sufficiently large k, we will have $s(|w_k|) < g(|w_k|)$. Pick such k, and consider the behavior of $M(w_k)$. Since M decides a language, it always halts. If M accepts w_k, then w_k must be in B, but since $w_k = \langle M \rangle 01^k$, the definition of B says that $M(w_k)$ must reject. Conversely, suppose M rejects w_k. We know M uses at most $s(|w_k|) < g(|w_k|)$ space when run on w_k. so $M(\langle M \rangle 01^k)$ rejects and uses at most $g(|\langle M \rangle 01^k|)$ space, so by the definition of B, w_k must be in B. This means M must accept w_k, which is a contradiction.

We've concluded that B cannot be decided by a TM that uses only $f(n)$ space, so $B \notin \mathsf{SPACE}(f)$. It remains to show that $B \in \mathsf{SPACE}(g)$. To do so, we must provide a Turing machine M' that decides B and whose space function s satisfies $s(n) = O(g(n))$.

The Turing machine M' will work as follows. On input w, it will first check that w is formatted like $\langle M \rangle 01^k$ for some Turing machine M and integer k; if not, M' will reject w. If w is correctly formatted, M' will compute the value of $g(|w|)$ (in binary) and store it. Next, M' will simulate M on input w, keeping track of the amount of space S used by M and also of the number of steps T used by M. If the space S used by M ever exceeds $g(|w|)$, M' will stop the simulation and reject. If M accepts w, M' rejects, and if M rejects w, M' accepts.

This seems like a reasonable definition of M', which just decides if a string w is in B by following the definition of B. However, there is a slight wrinkle: what if the simulation of M on w never halts? We need to ensure M' always halts, so that it decides the language B.

To ensure M' halts, we make M' keep track of the number of steps taken by its simulation of M on w. M' will set an upper bound on the allowed number of steps; this upper bound will be of the form $2^{Cg(|w|)}$, where C is a constant. The idea is that since $M(w)$ only uses at most $g(|w|)$ space (or else M' will halt the simulation), it can only have at most $|\Gamma|^{|g(w)|} \cdot |g(w)| \cdot |Q|$ different unique configurations, where Γ is the tape alphabet of M and where Q is the set of states of M. This is because there are only $|g(w)|$ cells of the tape reachable by the run of $M(w)$, and each can contain

only $|\Gamma|$ different symbols, for a total of $|\Gamma|^{g(|w|)}$ different combinations for the tape; there are also $g(|w|)$ different positions the tape head of M can be in, and $|Q|$ different internal states M can be in. Now, if we pick C large enough (larger than around $\log|\Gamma|$ plus a constant depending on $|Q|$), we can ensure that $2^{Cg(|w|)}$ is greater than $|\Gamma|^{g(|w|)} \cdot |g(w)| \cdot |Q|$. This means that if $M(w)$ runs for more than $2^{Cg(|w|)}$ steps without exceeding the space bound of $g(|w|)$, then $M(w)$ will run forever, since it must have entered some configuration twice.

OK, so the machine M' checks the formatting of the input w, calculates $g(|w|)$, and simulates M on input w until M either (1) accepts, (2) rejects, (3) exceeds the space bound $g(|w|)$, or (4) exceeds the time bound $2^{Cg(|w|)}$. This allows M' to decide B. How much space does M' need in order to do all of this?

The format check can be done in linear space in $|w|$; in fact, the machine M' barely needs any extra memory except for the input w itself, since it is only checking that the formatting of w is correct (and since we have no constraints on the running time of this check). Calculating $g(|w|)$ takes $O(g(|w|))$ time since g is time-constructible; but any computation that takes only T time can use only T space, since you can access at most one cell of the tape per time step. Hence calculating $g(|w|)$ takes at most $O(g(|w|))$ space. Next, the simulation of M on w. Since the space budget of M will be $g(|w|)$, this simulation will not need more than $O(g(|w|))$ space to store the tape of M, and we only need a small amount of additional overhead; thus we can simulate M on w using $O(g(|w|))$ space. There is still one catch, however: we need to keep track of the number of steps taken by the simulation of $M(w)$. If we store this as a number on the side, written in binary, then storing the number T will take around $O(\log T)$ space. Since we are allowing $M(w)$ to run for at most $2^{Cg(|w|)}$ steps, the number of steps T that we are writing down will never exceed $O(\log 2^{Cg(|w|)}) = O(g(|w|))$.

Finally, how do we check if the number of steps T that $M(w)$ has taken so far exceeds the bound $2^{Cg(|w|)}$? To do this, all we need to do is examine the length of the string $\langle T\rangle$ written in binary. If this length is greater than $Cg(|w|)$, it means T exceeded $2^{Cg(|w|)}$. So we just need to calculate the length of this string $\langle T\rangle$, and compare it to $Cg(|w|)$. This can be done in $O(g(|w|))$ space.

This completes the proof: M' decides B, and uses at most $O(g(n))$ space to do so, which means that $B \in \mathsf{SPACE}(g)$, as desired. □

As in the case of the Time Hierarchy Theorem, we get a corollary regarding the cleaner complexity classes.

Corollary 21.5. $\mathsf{PSPACE} \subsetneq \mathsf{EXPSPACE}$.

Proof. We have $\mathsf{PSPACE} = \bigcup_{k\in\mathbb{N}} \mathsf{SPACE}(n^k)$. Since $n^k = o(2^{\sqrt{n}})$ for all $k \in \mathbb{N}$, we have $\mathsf{SPACE}(n^k) \subseteq \mathsf{SPACE}(2^{\sqrt{n}})$ for all k. This implies that $\mathsf{PSPACE} \subseteq \mathsf{SPACE}(2^{\sqrt{n}})$. Since $2^{\sqrt{n}}$ is time constructible and since $2^{\sqrt{n}} = o(2^n)$, the Space Hierarchy Theorem implies that $\mathsf{SPACE}(2^{\sqrt{n}}) \subsetneq \mathsf{SPACE}(2^n)$. Since $\mathsf{SPACE}(2^n) \subseteq \mathsf{EXPSPACE}$, we conclude that $\mathsf{PSPACE} \subsetneq \mathsf{EXPSPACE}$. □

21.4 Further Remarks

We've now seen how to separate P from EXP and PSPACE from $\mathsf{EXPSPACE}$. We do not know whether $\mathsf{P} = \mathsf{PSPACE}$ or whether $\mathsf{PSPACE} = \mathsf{EXP}$, though they clearly cannot both be true. We conjecture that both are false, but no one can prove this.

What we've seen so far is close to the end of our knowledge regarding the separations between these time and space classes. In general, we do not know how to separate the time classes from the space classes, only the time classes from each other or the space classes from each other. Of course, we know that $\mathsf{P} \subsetneq \mathsf{EXPSPACE}$, but only because $\mathsf{P} \subsetneq \mathsf{EXP}$ and $\mathsf{EXP} \subseteq \mathsf{EXPSPACE}$.

We've also seen that if $\mathsf{P} = \mathsf{PSPACE}$, then $\mathsf{EXP} = \mathsf{EXPSPACE}$ (we do not know the reverse implication). There is one more interesting property we can prove regarding the classes we've seen so far.

Theorem 21.6. $\mathsf{PSPACE} \neq \mathsf{E}$.

Proof. We prove this by a padding argument. Suppose by contradiction that $\mathsf{PSPACE} = \mathsf{E}$. We will show that $\mathsf{EXP} = \mathsf{E}$. This will give us a contradiction, since the Time Hierarchy Theorem says there are languages in EXP that are not in E (since $\mathsf{E} \subseteq \mathsf{TIME}(2^{n^2})$ but EXP contains $\mathsf{TIME}(2^{n^3})$, for example).

To this end, let A be a language in EXP. Since $\mathsf{EXP} = \bigcup_{k \in \mathbb{N}} \mathsf{TIME}(2^{n^k})$, it follows that $A \in \mathsf{TIME}(2^{n^k})$ for some $k \in \mathbb{N}$. Consider the padded language

$$B = \{w01^{|w|^{k+1}} : w \in A\}.$$

Let M_A be a TM deciding A with running time $O(2^{n^k})$. We can decide B in at most $O(2^n)$ time using a Turing machine M, as follows. On input x, M will first find the last 0 of x and count the number of 1s following it, to check whether the input has the form $x = w01^{|w|^{k+1}}$. If so, M will simulate M_A on w, and accept if it accept and reject if it rejects. Note that the running time of $M_A(w)$ is $O(2^{|w|^k})$, and so the running time of $M(x)$ is a bit more than $O(2^{|w|^k})$ (due to the simulation overhead), say $O(2^{|w|^{k+1}})$. What is this running time in terms of $|x|$? Well, $|x|$ is a little over $|w|^{k+1}$, which means that $|w|$ is a little under $|x|^{1/(k+1)}$. Hence this running time is $O(2^{|x|})$. This means that M decides B in time $O(2^n)$ (in terms of its input size), as desired.

We conclude that $B \in \mathsf{TIME}(2^n)$, and hence $B \in \mathsf{E}$. Since we are assuming that $\mathsf{E} = \mathsf{PSPACE}$, it follows that $B \in \mathsf{PSPACE}$, so $B \in \mathsf{SPACE}(n^c)$ for some $c \in \mathbb{N}$. This means there is a Turing machine M' which decides B and has space function s satisfying $s(n) = O(n^c)$. We now construct a Turing machine M'_A which decides A in polynomial space.

On input w, the machine M'_A will append $01^{|w|^{k+1}}$ to w to get the string $w01^{|w|^{k+1}}$. Then M'_A will simulate $M'(w01^{|w|^{k+1}})$, and accept if it accept and reject if it rejects. Now, since the latter uses at most $O(n^c)$ space when run on inputs of size n, it must use at most $O(|w01^{|w|^{k+1}}|^c) = O(|w|^{c(k+1)})$ space. The simulation overhead is linear in terms of space, so we conclude that M'_A uses at most $O(|w|^{c(k+1)})$ space. In other words, its space function s' satisfies $s'(n) = O(n^{c(k+1)})$. It is also not hard to see that M'_A correctly decides A, since M' decides B and the string $w01^{|w|^{k+1}}$ is in B if and only if w is in A.

We conclude that $A \in \mathsf{SPACE}(n^{c(k+1)})$, and hence $A \in \mathsf{PSPACE}$. Since we are assuming that $\mathsf{PSPACE} = \mathsf{E}$, it follows that $A \in \mathsf{E}$. Since A was an arbitrary language in EXP, we conclude that $\mathsf{EXP} \subseteq \mathsf{E}$, and so these two classes are equal. This is a contradiction, because the Time Hierarchy Theorem implies they are not equal. □

Although this proves that E is not equal to PSPACE, we do not know whether one of these classes is a superset of the other. We conjecture that this is not the case: that there are languages in E that are not in PSPACE, and also languages in PSPACE that are not in E. However, proving this is beyond our current abilities.