Review: Time Hierarchy Theorems

**Time Hierarchy Theorem 12.12** If $f, g : \mathbb{N} \to \mathbb{N}$ are such that $f$ is time-constructible, and $g : \log g \in o(f)$, then

$$\text{DTime}_*(g) \subseteq \text{DTime}_*(f)$$

**Nondeterministic Time Hierarchy Theorem 12.14** If $f, g : \mathbb{N} \to \mathbb{N}$ are such that $f$ is time-constructible, and $g(n+1) \in o(f(n))$, then

$$\text{NTime}_*(g) \subseteq \text{NTime}_*(f)$$

In particular, we find that $P \neq \text{ExpTime}$ and $NP \neq \text{NExpTime}$:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq \text{PSpace} \subseteq \text{ExpTime} \subseteq \text{NExpTime} \subseteq \text{ExpSpace}$$
For space, we can always assume a single working tape:
- Tape reduction leads to a constant-factor increase in space
- Constant factors can be eliminated by space compression

Therefore, $\text{DSpace}_1(f) = \text{DSpace}_1(f)$.

Space turns out to be easier to separate – we get:

**Space Hierarchy Theorem 13.1:** If $f, g : \mathbb{N} \to \mathbb{N}$ are such that $f$ is space-constructible, and $g \in o(f)$, then $\text{DSpace}(g) \subseteq \text{DSpace}(f)$

**Challenge:** TMs can run forever even within bounded space.

**Proof (continued):** It remains to show that $D$ implements diagonalisation:

$L(D) \in \text{DSpace}(f)$:
- $f$ is space-constructible, so both the marking of tape symbols and the initialisation of the counter are possible in $\text{DSpace}(f)$
- The simulation is performed so that the marked $O(f)$-space is not left

There is $w$ such that $(M, w) \in L(D)$ iff $(M, w) \notin L(M)$:
- As for time, we argue that some $w$ is long enough to ensure that $f$ is sufficiently larger than $g$, so $D$’s simulation can finish.
- The countdown measures $2^{f(n)}$ steps. The number of possible distinct configurations of $M$ on $w$ is $|Q| \cdot n \cdot g(n) \cdot |\Gamma|^{|\Gamma|} \leq 2^{O(f(n) + \log g(n))}$, and due to $f(n) \geq \log n$ and $g \in o(f)$, this number is smaller than $2^{f(n)}$ for large enough $n$.
- If $M$ has $d$ tape symbols, then $D$ can encode each in $\log d$ space, and due to $M$’s space bound $D$’s simulation needs at most $\log d \cdot g(n) \in o(f(n))$ cells.

Therefore, there is $w$ for which $D$ simulates $M$ long enough to obtain (and flip) its output, or to detect that it is not terminating (and to accept, flipping again). □
The Gap Theorem

Why Constructibility?

The hierarchy theorems require that resource limits are given by constructible functions. Do we really need this?

Yes. The following theorem shows why (for time):

**Special Gap Theorem 13.5:** There is a computable function $f : \mathbb{N} \to \mathbb{N}$ such that $\text{DTime}(f(n)) = \text{DTime}(2^{f(n)})$.

This has been shown independently by Boris Trakhtenbrot (1964) and Allan Borodin (1972).

Reminder: For this we continue to use the strict definition of $\text{DTime}(f)$ where no constant factors are included (no hidden $O(f)$). This simplifies proofs; the factors are easy to add back.

Proving the Gap Theorem

Proof idea: We divide time into exponentially long intervals of the form:

- $[0, n]$,
- $[n + 1, 2^n]$,
- $[2^n + 1, 2^{2^n}]$,
- ...  

(for some appropriate starting value $n$)

We are looking for gaps of time where no TM halts, since:

- for every finite set of TMs,
- and every finite set of inputs to these TMs,
- there is some interval of the above form $[m + 1, 2^m]$ such none of the TMs halts in between $m + 1$ and $2^m$ steps on any of the inputs.

The task of $f$ is to find the start $m$ of such a gap for a suitable set of TMs and words.

Gaps in Time

We consider an (effectively computable) enumeration of all Turing machines:

$M_0, M_1, M_2, \ldots$

**Definition 13.6:** For arbitrary numbers $i, a, b \in \mathbb{N}$ with $a \leq b$, we say that $\text{Gap}(a, b)$ is true if:

- Given any TM $M_j$ with $0 \leq j \leq i$,
- and any input string $w$ for $M_j$ of length $|w| = i$,
- $M_j$ on input $w$ will halt in less than $a$ steps, in more than $b$ steps, or not at all.

**Lemma 13.7:** Given $i, a, b \geq 0$ with $a \leq b$, it is decidable if $\text{Gap}(a, b)$ holds.

Proof: We just need to ensure that none of the finitely many TMs $M_0, \ldots, M_i$ will halt after $a$ to $b$ steps on any of the finitely many inputs of length $i$. This can be checked by simulating TM runs for at most $b$ steps. □
We can now define the value \( f(n) \) of \( f \) for some \( n \geq 0 \):

Let \( \text{in}(n) \) denote the number of runs of TMs \( M_0, \ldots, M_i \) on words of length \( n \), i.e.,

\[
\text{in}(n) = |\Sigma_1^n| + \cdots + |\Sigma_i^n|
\]

where \( \Sigma_i \) is the input alphabet of \( M_i \).

We recursively define a series of numbers \( k_0, k_1, k_2, \ldots \) by setting \( k_0 = 2n \) and \( k_{i+1} = 2^k \) for \( i \geq 0 \), and we consider the following list of intervals:

\[
[k_0 + 1, k_1], [k_1 + 1, k_2], \ldots, [k_{\text{in}(n)} + 1, k_{\text{in}(n)+1}]
\]

\[
[2n + 1, 2^{2n}], [2^{2n} + 1, 2^{2^2}], \ldots, [2^{2^n} + 1, 2^{2^{2^n}}]
\]

Let \( f(n) \) be the least number \( k_i \) with \( 0 \leq i \leq \text{in}(n) \) such that \( \text{Gap}_k(k_i + 1, k_{i+1}) \) is true.

We first establish some basic properties of our definition of \( f \):

**Claim:** The function \( f \) is well-defined.

**Proof:** For finding \( f(n) \), we consider \( \text{in}(n) + 1 \) intervals. Since there are only \( \text{in}(n) \) runs of TMs \( M_0, \ldots, M_i \), at least one interval remains a “gap” where no TM run halts.

**Claim:** The function \( f \) is computable.

**Proof:** We can compute \( \text{in}(n) \) and \( k_i \) for any \( i \), and we can decide \( \text{Gap}_k(k_i + 1, k_{i+1}) \).

Papadimitriou: “notice the fantastically fast growth, as well as the decidedly unnatural definition of this function.”

---

Let us now complete the proof of the theorem:

**Claim:** \( \text{DTIME}(f(n)) = \text{DTIME}(2^{f(n)}) \).

Consider any \( L \in \text{DTIME}(2^{f(n)}) \).

Then there is an \( 2^{f(n)} \)-time bounded TM \( M \) with \( L = L(M) \).

For any input \( w \) with \( |w| \geq j \):

- The definition of \( f(|w|) \) took the run of \( M \) on \( w \) into account
- \( M \) on \( w \) halts after less than \( f(|w|) \) steps, or not until after \( 2^{f(n)} \) steps (maybe never)
- Since \( M \) runs in time \( 2^{f(n)} \), it must halt in \( \text{DTIME}(f(n)) \) on \( w \)

For the finitely many inputs \( w \) with \( |w| < j \):

- We can augment the state space of \( M \) to run a finite automaton to decide these cases
- This will work in \( \text{DTIME}(f(n)) \)

Therefore we have \( L \in \text{DTIME}(f(n)) \).

---

**Finishing the Proof**

We can now complete the proof of the theorem:

**Claim:** \( \text{DTIME}(f(n)) = \text{DTIME}(2^{f(n)}) \).

Consider any \( L \in \text{DTIME}(2^{f(n)}) \).

Then there is an \( 2^{f(n)} \)-time bounded TM \( M \) with \( L = L(M) \).

For any input \( w \) with \( |w| \geq j \):

- The definition of \( f(|w|) \) took the run of \( M \) on \( w \) into account
- \( M \) on \( w \) halts after less than \( f(|w|) \) steps, or not until after \( 2^{f(n)} \) steps (maybe never)
- Since \( M \) runs in time \( 2^{f(n)} \), it must halt in \( \text{DTIME}(f(n)) \) on \( w \)

For the finitely many inputs \( w \) with \( |w| < j \):

- We can augment the state space of \( M \) to run a finite automaton to decide these cases
- This will work in \( \text{DTIME}(f(n)) \)

Therefore we have \( L \in \text{DTIME}(f(n)) \). 

---

**Discussion: The case** \(|w| < j\)

**Borodin says:** It is meaningful to state complexity results if they hold for “almost every” input (i.e., for all but a finite number)

**Papadimitriou says:** These words can be handled since we can check the length and then recognise the word in less than \( 2j \) steps

**Really?**

- If we do these \( < 2j \) steps before running \( M \), the modified TM runs in \( \text{DTIME}(f(n) + 2j) \)
- This does not show \( L \in \text{DTIME}(f(n)) \)

**A more detailed argument:**

- Make the intervals larger: \([k_i + 1, 2^{k_i+2^n} + 2n]\), that is \( k_{i+1} = 2^{k_i+2^n} + 2n \).
- Select \( f(n) \) to be \( k_i + 2n + 1 \) if the least gap starts at \( k_i + 1 \).

The same pigeon hole argument as before ensures that an empty interval is found.

But now the \( f(n) \) time bounded machine \( M \) from the proof will be sure to stop after \( f(n) - 2n - 1 \) steps, so a shift of \( 2j \leq 2n \) to account for the finitely many cases will not make it use more than \( f(n) \) steps either
Discussion: Generalising the Gap Theorem

• Our proof uses the function \( n \mapsto 2^n \) to define intervals
• Any other computable function could be used without affecting the argument

This leads to a generalised Gap Theorem:

**Gap Theorem 13.8:** For every computable function \( g : \mathbb{N} \rightarrow \mathbb{N} \) with \( g(n) \geq n \), there is a computable function \( f : \mathbb{N} \rightarrow \mathbb{N} \) such that \( \text{DTime}(f(n)) = \text{DTime}(g(f(n))) \).

**Example 13.9:** There is a function \( f \) such that

\[
\text{DTime}(f(n)) = \text{DTime}\left(2^{2^{2^{\ldots^{2}}}}\text{ times}\right)
\]

Moreover, the Gap Theorem can also be shown for space (and for other resources) in a similar fashion (space is a bit easier since the case of short words \(|w| < j\) is easy to handle in very little space).

What have we learned?

• More time (or space) does not always increase computational power
• However, this only works for extremely fast-growing, very unnatural functions

"Fortunately, the gap phenomenon cannot happen for time bounds \( r \) that anyone would ever be interested in\(^1\)

Main insight: better stick to constructible functions

---

Summary and Outlook

Hierarchy theorems tell us that more time/space leads to more power:

\[
L \subset NL \subset P \subset NP \subset \text{PSpace} \subset \text{ExpTime} \subset \text{NExpTime} \subset \text{ExpSpace}
\]

However, they don’t help us in comparing different resources and machine types (P vs. NP, or PSpace vs. ExpTime)

With non-constructible functions as time/space bounds, arbitrary (constructible or not) boosts in resources do not lead to more power

**What’s next?**

• The inner structure of NP revisited
• Computing with oracles (reprise)
• The limits of diagonalisation, proved by diagonalisation

---