

BBM402-Lecture 10: Measuring Complexity and the Class P

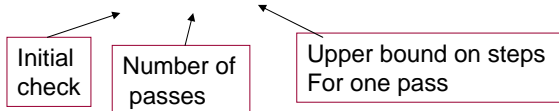
Lecturer: Lale Özkahya

Resources for the presentation:

<http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-045j-automata-computability-and-complexity-spring-2011/Syllabus/>

Examples of time complexity analysis

- Consider a basic 1-tape Turing machine M that decides membership in the language $L = \{0^k 1^k \mid k \geq 0\}$:
 - M first checks that its input is in $0^* 1^*$, using one left-to-right pass.
 - Returns to the beginning (left).
 - Then does repeated passes, each time crossing off one 0 and one 1, until it runs out of at least one of them.
 - If it runs out of both on the same pass, accepts, else rejects.
- Q:** How much time until M halts?
- Depends on the particular input.
- Example:** $0111\dots 1110$ (length n)
 - Approximately n steps to reject---not in $0^* 1^*$,
- Example:** $00\dots 011\dots 1$ ($n/2$ 0s and $n/2$ 1s)
 - Approximately (at most) $2n + (n/2) 2n = 2n + n^2$ steps to accept.



Time complexity analysis

- $L(M) = \{0^k 1^k \mid k \geq 0\}$.
- Time until M halts depends on the particular input.
- $0111\dots 1110$ (length n)
 - Approximately n steps to reject---not in 0^*1^* ,
- $00\dots 011\dots 1$ ($n/2$ 0s and $n/2$ 1s)
 - Approximately (at most) $2n + n^2$ steps to accept.
- It's too complicated to determine exactly how many steps are required for every input.
- So instead, we:
 - Get a close upper bound, not an exact step count.
 - Express the bound as a function of the input length n , thus grouping together all inputs of the same length and considering the max.
 - Often ignore constant factors and low-order terms.
- So, we describe the time complexity of M as $O(n^2)$.
 - At most some constant times n^2 .

Time complexity analysis

- $L(M) = \{0^k 1^k \mid k \geq 0\}$.
- Time complexity of machine $M = O(n^2)$.
- Q: Can we do better with a multitape machine?
- Yes, with 2 tapes:
 - After checking $0^* 1^*$, the machine copies the 0s to the second tape.
 - Then moves 2 heads together, one scanning the 0s on the second tape and one scanning the 1s on the first tape.
 - Check that all the symbols match.
 - Time $O(n)$, proportional to n .

Time complexity analysis

- $L(M) = \{0^k 1^k \mid k \geq 0\}$.
- 1-tape machine: $O(n^2)$, 2-tape machine: $O(n)$.
- **Q: Can we beat $O(n^2)$ with a 1-tape machine?**
- Yes, can get **$O(n \log n)$** :
 - First check $0^* 1^*$, as before, $O(n)$ steps.
 - Then perform **marking** phases, as long as some unmarked 0 and some unmarked 1 remain.
 - In each marking phase:
 - Scan to see whether $\#$ of unmarked 0s $\equiv \#$ of unmarked 1s, mod 2.
 - That is, see whether they have the same parity.
 - If not, then reject, else continue.
 - Scan again, marking every other 0 starting with the first and every other 1 starting with the first.
 - After all phases are complete:
 - If just 0s or just 1s remain, then reject
 - If no unmarked symbols remain, then accept.

Time complexity analysis

- $O(n \log n)$ algorithm:
 - Check 0^*1^* .
 - Perform **marking** phases, as long as some unmarked 0 and some unmarked 1 remain.
 - In each marking phase:
 - Scan to see if # of unmarked 0s \equiv # of unmarked 1s, mod 2; if not, then reject, else continue.
 - Scan again, marking every other 0 starting with the first and every other 1 starting with the first.
 - If just 0s or just 1s remain, then reject, else accept.
- **Example:** $00\dots011\dots1$ (25 0s and 25 1s)
 - Correct form, 0^*1^* .
 - Phase 1: Same parity (odd), marking leaves 12 0s and 12 1s.
 - Phase 2: Same parity (even), marking leaves 6, 6.
 - Phase 3: Same parity (even), marking leaves 3, 3.
 - Phase 4: Same parity (odd), marking leaves 1, 1.
 - Phase 5: Same parity (odd), marking leaves 0, 0
 - Accept

Time complexity analysis

- **Example: 00...011...1** (25 0s and 25 1s)
 - Correct form, 0^*1^* .
 - Phase 1: Same parity (odd), marking leaves 12 0s and 12 1s.
 - Phase 2: Same parity (even), marking leaves 6, 6.
 - Phase 3: Same parity (even), marking leaves 3, 3.
 - Phase 4: Same parity (odd), marking leaves 1, 1.
 - Phase 5: Same parity (odd), marking leaves 0, 0
 - Accept
- Odd parity leads to remainder 1 on division by 2, even parity leads to remainder 0.
- Can read off odd-even parity designations to get binary representations of the numbers, starting with final phase for high-order bit:
 - 5: odd; 4: odd; 3: even; 2: even; 1: odd
 - Yields 1 1 0 0 1, binary representation of 25
- If the algorithm accepts, it means the 2 numbers have the same binary representation, so they are equal.

Time complexity analysis

- Example: 00...011...1 (17 0s and 25 1s)
 - Correct form, 0^*1^* .
 - Phase 1: Same parity (odd), marking leaves 8 0s and 12 1s.
 - Phase 2: Same parity (even), marking leaves 4, 6.
 - Phase 3: Same parity (even), marking leaves 2, 3.
 - Phase 4: Different parity, reject
 - Don't complete this, so don't generate the complete binary representation of either number.

Time complexity analysis

- Algorithm
 - Check 0^*1^* .
 - Perform **marking** phases, as long as some unmarked 0 and some unmarked 1 remain.
 - In each marking phase:
 - Scan to see if # of unmarked 0s \equiv # of unmarked 1s, mod 2; if not, then reject, else continue.
 - Scan again, marking every other 0 starting with the first and every other 1 starting with the first.
 - If just 0s or just 1s remain, then reject, else accept.
- **Complexity analysis:**
 - Number of phases is $O(\log_2 n)$, since we (approximately) halve the number of unmarked 0s and unmarked 1s at each phase.
 - Time for each phase: $O(n)$.
 - Total: $O(n \log n)$.
- This analysis is informal; now define O , etc., more carefully and then revisit the example.

Asymptotic function notation:

O , o , Ω , Θ

Asymptotic function notation

- **Definition: O (big- O)**

- Let f, g be two functions: $N \rightarrow \mathbb{R}^{\geq 0}$.
- We write $f(n) = O(g(n))$, and say “ $f(n)$ is big- O of $g(n)$ ” if the following holds:
 - There is a positive real c , and a positive integer n_0 , such that $f(n) \leq c g(n)$ for every $n \geq n_0$.
 - That is, $f(n)$ is bounded from above by a constant times $g(n)$, for all sufficiently large n .
- Often used for complexity upper bounds.
- **Example:** $n + 2 = O(n)$; can use $c = 2$, $n_0 = 2$.
- **Example:** $3n^2 + n = O(n^2)$; can use $c = 4$, $n_0 = 1$.
- **Example:** Any degree- k polynomial with nonnegative coefficients, $p(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0 = O(n^k)$
 - Thus, $3n^4 + 6n^2 + 17 = O(n^4)$.

More big-O examples

- Definition:
 - Let $f, g: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$
 - $f(n) = O(g(n))$ means that there is a positive real c , and a positive integer n_0 , such that $f(n) \leq c g(n)$ for every $n \geq n_0$.
- Example: $3n^4 = O(n^7)$, though this is not the tightest possible statement.
- Example: $3n^7 \neq O(n^4)$.
- Example: $\log_2(n) = O(\log_e(n))$; $\log_a(n) = O(\log_b(n))$ for any a and b
 - Because logs to different bases differ by a constant factor.
- Example: $2^{3+n} = O(2^n)$, because $2^{3+n} = 8 \times 2^n$
- Example: $3^n \neq O(2^n)$

Other notation

- **Definition:** Ω (big-Omega)

- Let f, g be two functions: $N \rightarrow \mathbb{R}^{\geq 0}$
- We write $f(n) = \Omega(g(n))$, and say “ $f(n)$ is big-Omega of $g(n)$ ” if the following holds:
 - There is a positive real c , and a positive integer n_0 , such that $f(n) \geq c g(n)$ for every $n \geq n_0$.
 - That is, $f(n)$ is bounded from below by a positive constant times $g(n)$, for all sufficiently large n .

- Used for complexity lower bounds.

- **Example:** $3n^2 + 4n \log(n) = \Omega(n^2)$

- **Example:** $3n^7 = \Omega(n^4)$.

- **Example:** $\log_e(n) = \Omega(\log_2(n))$

- **Example:** $3^n = \Omega(2^n)$

Other notation

- **Definition: Θ (Theta)**

- Let f, g be two functions: $N \rightarrow \mathbb{R}^{\geq 0}$
- We write $f(n) = \Theta(g(n))$, and say “ $f(n)$ is Theta of $g(n)$ ” if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
- Equivalently, there exist positive reals c_1, c_2 , and positive integer n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for every $n \geq n_0$.

- **Example:** $3n^2 + 4n \log(n) = \Theta(n^2)$
- **Example:** $3n^4 = \Theta(n^4)$.
- **Example:** $3n^7 \neq \Theta(n^4)$.
- **Example:** $\log_e(n) = \Theta(\log_2(n))$
- **Example:** $3^n \neq \Theta(2^n)$

Plugging asymptotics into formulas

- Sometimes we write things like $2^{\Theta(\log_2 n)}$
- What does this mean?
- Means the exponent is some function $f(n)$ that is $\Theta(\log n)$, that is, $c_1 \log(n) \leq f(n) \leq c_2 \log(n)$ for every $n \geq n_0$.
- So $2^{c_1 \log(n)} \leq 2^{\Theta(\log_2 n)} \leq 2^{c_2 \log(n)}$
- In other words, $n^{c_1} \leq 2^{\Theta(\log_2 n)} \leq n^{c_2}$
- Same as $n^{\Theta(1)}$.

Other notation

- **Definition: o (Little-o)**

- Let f, g be two functions: $N \rightarrow \mathbb{R}^{\geq 0}$
- We write $f(n) = o(g(n))$, and say “ $f(n)$ is little-o of $g(n)$ ” if **for every positive real c , there is some positive integer n_0 , such that $f(n) < c g(n)$ for every $n \geq n_0$.**
- In other words, no matter what constant c we choose, for sufficiently large n , $f(n)$ is less than $g(n)$.
- In other words, $f(n)$ grows at a slower rate than any constant times $g(n)$.
- In other words, $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$.

- **Example:** $3n^4 = o(n^7)$

- **Example:** $\sqrt{n} = o(n)$

- **Example:** $n \log n = o(n^2)$

- **Example:** $2^n = o(3^n)$

Back to the TM running times...

- Running times (worst case over all inputs of the same length n) of the 3 TMs described earlier:
 - Simple 1-tape algorithm: $\Theta(n^2)$
 - 2-tape algorithm: $\Theta(n)$
 - More clever 1-tape algorithm: $\Theta(n \log n)$
- More precisely, consider any Turing machine M that decides a language.
- Define the running time function $t_M(n)$ to be:
 - $\max_{w \in \Sigma^n} t'_M(w)$, where
 - $t'_M(w)$ is the exact running time (number of steps) of M on input w .
- Then for these three machines, $t_M(n)$ is $\Theta(n^2)$, $\Theta(n)$, and $\Theta(n \log n)$, respectively.

Time Complexity Classes

Time Complexity Classes

- Classify decidable languages according to upper bounds on the running time for TMs that decide them.
- **Definition:** Let $t: \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$ be a (total) function. Then $\text{TIME}(t(n))$ is the set of languages:
 $\{ L \mid L \text{ is decided by some } O(t(n))\text{-time Turing machine} \}$
- Call this a “time-bounded complexity class”.
- **Notes:**
 - Notice the O ---allows some slack.
 - To be careful, we need to specify which kind of TM model we are talking about; assume basic 1-tape.
- **Complexity Theory studies:**
 - Which languages are in which complexity classes.
 - E.g., is the language PRIMES in $\text{TIME}(n^5)$?
 - How complexity classes are related to each other.
 - E.g., is $\text{TIME}(n^5) = \text{TIME}(n^6)$, or are there languages that can be decided in time $O(n^6)$ but not in time $O(n^5)$?

Time Complexity Classes

- A problem: Running times are model-dependent.
- E.g., $L = \{0^k 1^k \mid k \geq 0\}$:
 - On 1-tape TM, can decide in time $O(n \log n)$.
 - On 2-tape TM, can decide in time $O(n)$.
- To be definite, we'll define the complexity classes in terms of 1-tape TMs (as Sipser does); others use multi-tape, or other models like Random-Access Machines (RAMs).
- **Q:** Is this difference important?
- Only up to a point:
 - If $L \in \text{TIME}(f(n))$ based on any “standard” machine model, then also $L \in \text{TIME}(g(n))$, where $g(n) = O(p(f(n)))$ for some polynomial p , based on any other “standard” machine model.
 - Running times for L in any two standard models are polynomial-related.
- **Example:** Single-tape vs. multi-tape Turing machines

Time Complexity Classes

- If $L \in \text{TIME}(f(n))$ based on any “standard” machine model, then also $L \in \text{TIME}(g(n))$, where $g(n) = O(p(f(n)))$ for some polynomial p , based on any other “standard” machine model.
- **Example:** 1-tape vs. multi-tape Turing machines
 - 1-tape \rightarrow multi-tape with no increase in complexity.
 - Multi-tape \rightarrow 1-tape: If $t(n) \geq n$ then every $t(n)$ -time multi-tape TM has an equivalent $O(t^2(n))$ -time 1-tape TM.
 - **Proof idea:**
 - 1-tape TM simulates multi-tape TM.
 - Simulates each step of multi-tape TM using 2 scans over non-blank portion of tapes, visiting all heads, making all changes.
 - **Q:** What is the time complexity of the simulating 1-tape TM? That is, how many steps does the 1-tape TM use to simulate the $t(n)$ steps of the multi-tape machine?

Time Complexity Classes

- **Example:** 1-tape vs. multi-tape Turing machines
 - Multi-tape \rightarrow 1-tape: If $t(n) \geq n$ then every $t(n)$ -time multi-tape TM has an equivalent $O(t^2(n))$ -time 1-tape TM.
 - 1-tape TM simulates multi-tape TM; simulates each step using 2 scans over non-blank portion of tapes, visiting all heads, making all changes.
 - Q: What is the time complexity of the 1-tape TM?
 - Q: How big can the non-blank portion of the multi-tape TM's tapes become?
 - Initially n , for the input.
 - In $t(n)$ steps, no bigger than $t(n)$, because that's how far the heads can travel (starts at left).
 - So the number of steps by the 1-tape TM is at most:

$$t(n) \times c t(n), \text{ hence } O(t^2(n)).$$

Number of steps of
multi-tape machine

Steps taken by the scans,
to emulate one step of the
multi-tape machine.

Time Complexity Classes

- If $L \in \text{TIME}(f(n))$ based on any “standard” machine model, then also $L \in \text{TIME}(g(n))$, where $g(n) = O(p(f(n)))$ for some polynomial p , based on any other “standard” machine model.
- Slightly-idealized versions of real computers, programs in standard languages, other “reasonable” machine models, can be emulated by basic TMs with only polynomial increase in running time.
- Important exception: Nondeterministic Turing machines (or other nondeterministic computing models)
 - For nondeterministic TMs, running time is usually measured by max number of steps on any branch.
 - A bound of $t(n)$ on the maximum number of steps on any branch translates into $2^{O(t(n))}$ steps for basic deterministic TMs.

P, Polynomial Time

P, Polynomial Time

- A formal way to define **fast computability**.
- Because of simulation results, polynomial differences are considered to be unimportant for (deterministic) TMs.
- So our definition of fast computability ignores polynomial differences.
- **Definition:** The class P of languages that are decidable in polynomial time is defined by:

$$P = \cup_{p \text{ a poly}} \text{TIME}(p(n)) = \cup_{k \geq 0} \text{TIME}(n^k)$$

- Notes:
 - These time-bounded language classes are defined with respect to basic (1-tape, 1-head) Turing machines.
 - Simulation results imply that we could have used any “reasonable” deterministic computing model and get the same language class.
 - Robust notion.

P, Polynomial Time

- Definition: The class P of languages that are decidable in polynomial time is defined by:
$$P = \cup_{p \text{ a poly}} \text{TIME}(p(n)) = \cup_{k \geq 0} \text{TIME}(n^k)$$
- P plays a role in complexity theory loosely analogous to that of decidable languages in computability.
- Recall Church-Turing thesis:
 - If L is decidable using some reasonable model of computation, then it is decidable using any reasonable model of computation.
- **Modified Church-Turing thesis:**
 - If L is decidable **in polynomial time** using some reasonable **deterministic** model of computation, then it is decidable **in polynomial time** using any reasonable **deterministic** model of computation.
- This is not a theorem---rather, a philosophical statement.
- Can think of this as defining what a reasonable model is.
- We'll focus on the class P for much of our work on complexity theory.

P, Polynomial Time

- We'll focus on the class P for much of our work on complexity theory.
- **Q:** Why is P a good language class to study?
- It's **model-independent** (for reasonable models).
- It's **scalable**:
 - Constant-factor dependence on input size.
 - E.g., an input that's twice as long requires only c times as much time, for some constant c (depends on degree of the polynomial).
 - E.g., consider time bound n^3 .
 - Input of length n takes time n^3 .
 - Input of length $2n$ takes time $(2n)^3 = 8n^3$, $c = 8$.
 - Works for all polynomials, any degree.

P, Polynomial Time

- Q: Why is P a good language class to study?
- It's model-independent (for reasonable models).
- It's scalable.
- It has nice composition properties:
 - Composing two polynomials yields another polynomial.
 - This property will be useful later, when we define polynomial-time reducibilities.
 - Preview: $A \leq_p B$ means that there exists a polynomial-time computable function f such that $x \in A$ if and only if $f(x) \in B$.
 - Desirable theorem: $A \leq_p B$ and $B \in P$ imply $A \in P$.
 - Proof:
 - Suppose B is decidable in time $O(n^k)$.
 - Suppose the reducibility function f is computable in time $O(n^l)$.

P, Polynomial Time

- P has nice composition properties:
 - $A \leq_p B$ means that there's a polynomial-time computable function f such that $x \in A$ if and only if $f(x) \in B$.
 - Desirable theorem: $A \leq_p B$ and $B \in P$ imply $A \in P$.
 - Proof:
 - Suppose B is decidable in time $O(n^k)$, and f is computable in time $O(n^l)$.
 - How much time does it take to decide membership in A by reduction to B ?
 - Given x of length n , time to compute $f(x)$ is $O(n^l)$.
 - Moreover, $|f(x)| = O(n^l)$, since there's not enough time to generate a bigger result.
 - Now run B 's decision procedure on $f(x)$.
 - Takes time $O(|f(x)|^k) = O((n^l)^k) = O(n^{lk})$.
 - Another polynomial, so A is decidable in poly time, so $A \in P$

P, Polynomial Time

- Q: Why is P a good language class to study?
 - It's model-independent (for reasonable models).
 - It's scalable.
 - It has nice composition properties.
- Q: What are some limitations?
 - Includes too much:
 - Allows polynomials with arbitrarily large exponents and coefficients.
 - Time $10,000,000 n^{10,000,000}$ isn't really feasible.
 - In practice, running times are usually low degree polynomials, up to about $O(n^4)$.
 - On the other hand, proving a non-polynomial lower bound is likely to be meaningful.

P, Polynomial Time

- Q: Why is P a good language class to study?
 - It's model-independent (for reasonable models).
 - It's scalable.
 - It has nice composition properties.
- Q: What are some limitations?
 - Includes too much.
 - Excludes some things:
 - Considers **worst case time complexity** only.
 - Some algorithms may work well enough in most cases, or in common cases, even though the worst case is exponential.
 - **Random choices**, with membership being decided with high probability rather than with certainty.
 - **Quantum computing**.

P, Polynomial Time

- **Example:** A language in P.
 - $\text{PATH} = \{ \langle G, s, t \rangle \mid G = (V, E) \text{ is a digraph that has a directed path from } s \text{ to } t \}$
 - Represent G by adjacency matrix ($|V|$ rows and $|V|$ columns, 1 indicates an edge, 0 indicates no edge).
 - **Brute-force algorithm:** Try all paths of length $\leq |V|$.
 - Exponential running time in input size, not polynomial.
 - **Better algorithm:** BFS of G starting from s .
 - Mark new nodes accessible from already-marked nodes, until no new nodes are found.
 - Then see if t is marked.
 - Complexity analysis:
 - At most $|V|$ phases are executed.
 - Each phase takes polynomial time to explore marked nodes and their outgoing edges.

A Language Not in P

A Language Not in P

- **Q:** Is every language in P?
- No, because $P \subseteq$ decidable languages, and not every language is decidable.
- **Q:** Is every decidable language in P?
- No again, but it takes some work to show this.
- **Theorem:** For any computable function t , there is a language that is decidable, but cannot be decided by any basic Turing machine in time $t(n)$.
- **Proof:**
 - Fix computable function t .
 - Define language $\text{Acc}(t)$
 $= \{ \langle M \rangle \mid M \text{ is a basic TM and } M \text{ accepts } \langle M \rangle \text{ in } \leq t(|\langle M \rangle|) \text{ steps} \}.$
 - **Claim 1:** $\text{Acc}(t)$ is decidable.
 - **Claim 2:** $\text{Acc}(t)$ is not decided by any basic TM in $\leq t(n)$ steps.

A Language Not in P

- **Theorem:** For any computable function t , there is a language that is decidable, but cannot be decided by any basic Turing machine in time $t(n)$.
- **Proof:**
 - $\text{Acc}(t) = \{ \langle M \rangle \mid M \text{ is a basic TM that accepts } \langle M \rangle \text{ in } \leq t(|\langle M \rangle|) \text{ steps} \}$.
 - **Claim 1:** $\text{Acc}(t)$ is decidable.
 - Given $\langle M \rangle$, simulate M on $\langle M \rangle$ for $t(|\langle M \rangle|)$ simulated steps and see if it accepts.
 - **Claim 2:** $\text{Acc}(t)$ is not decided by any basic TM in $\leq t(n)$ steps.
 - Use a diagonalization proof, like that for Acc_{TM} .
 - Assume $\text{Acc}(t)$ is decided in time $\leq t(n)$ by some basic TM.
 - Here, $n = |\langle M \rangle|$ for input $\langle M \rangle$.

A Language Not in P

- **Theorem:** For any computable function t , there is a language that is decidable, but cannot be decided by any basic Turing machine in time $t(n)$.
- $\text{Acc}(t) = \{ \langle M \rangle \mid M \text{ is a basic TM that accepts } \langle M \rangle \text{ in } \leq t(|\langle M \rangle|) \text{ steps} \}$.
- **Claim 2:** $\text{Acc}(t)$ is not decided by any basic TM in $\leq t(n)$ steps.
- **Proof:**
 - Assume $\text{Acc}(t)$ is decided in time $\leq t(n)$ by some basic TM.
 - Then $\text{Acc}(t)^c$ is decided in time $\leq t(n)$, by another basic TM.
 - Interchange q_{acc} and q_{rej} states.
 - Let M_0 be a basic TM that decides $\text{Acc}(t)^c$ in time $\leq t(n)$.
 - That means $t(n)$ steps of M_0 , not $t(n)$ simulated steps.
 - Thus, for every basic Turing machine M :
 - If $\langle M \rangle \in \text{Acc}(t)^c$, then M_0 accepts $\langle M \rangle$ in time $\leq t(|\langle M \rangle|)$.
 - If $\langle M \rangle \in \text{Acc}(t)$, then M_0 rejects $\langle M \rangle$ in time $\leq t(|\langle M \rangle|)$.

A Language Not in P

- **Theorem:** For any computable function t , there is a language that is decidable, but cannot be decided by any basic Turing machine in time $t(n)$.
- $\text{Acc}(t) = \{ \langle M \rangle \mid M \text{ is a basic TM that accepts } \langle M \rangle \text{ in } \leq t(|\langle M \rangle|) \text{ steps} \}$.
- **Claim 2:** $\text{Acc}(t)$ is not decided by any basic TM in $\leq t(n)$ steps.
- **Proof:**
 - Assume $\text{Acc}(t)$ is decided in time $\leq t(n)$ by some basic TM.
 - $\text{Acc}(t)^c$ is decided in time $\leq t(n)$, by basic TM M_0 .
 - Thus, for every basic Turing machine M :
 - If $\langle M \rangle \in \text{Acc}(t)^c$, then M_0 accepts $\langle M \rangle$ in time $\leq t(|\langle M \rangle|)$.
 - If $\langle M \rangle \in \text{Acc}(t)$, then M_0 rejects $\langle M \rangle$ in time $\leq t(|\langle M \rangle|)$.
 - Thus, for every basic Turing machine M :
 - $\langle M \rangle \in \text{Acc}(t)^c$ iff M_0 accepts $\langle M \rangle$ in time $\leq t(|\langle M \rangle|)$.

A Language Not in P

- **Theorem:** For any computable function t , there is a language that is decidable, but cannot be decided by any basic Turing machine in time $t(n)$.
- $\text{Acc}(t) = \{ \langle M \rangle \mid M \text{ is a basic TM that accepts } \langle M \rangle \text{ in } \leq t(|\langle M \rangle|) \text{ steps} \}$.
- **Claim 2:** $\text{Acc}(t)$ is not decided by any basic TM in $\leq t(n)$ steps.
- **Proof:**
 - Assume $\text{Acc}(t)$ is decided in time $\leq t(n)$ by some basic TM.
 - $\text{Acc}(t)^c$ is decided in time $\leq t(n)$, by basic TM M_0 .
 - For every basic Turing machine M :
 $\langle M \rangle \in \text{Acc}(t)^c$ iff M_0 accepts $\langle M \rangle$ in time $\leq t(|\langle M \rangle|)$.
 - However, by definition of $\text{Acc}(t)$, for every basic TM M :
 $\langle M \rangle \in \text{Acc}(t)^c$ iff M does not accept $\langle M \rangle$ in time $\leq t(|\langle M \rangle|)$.

A Language Not in P

- **Claim 2:** $\text{Acc}(t)$ is not decided by any basic TM in $\leq t(n)$ steps.
- **Proof:**
 - Assume $\text{Acc}(t)$ is decided in time $\leq t(n)$ by some basic TM.
 - $\text{Acc}(t)^c$ is decided in time $\leq t(n)$, by basic TM M_0 .
 - For every basic Turing machine M :
 - $\langle M \rangle \in \text{Acc}(t)^c$ iff M_0 accepts $\langle M \rangle$ in time $\leq t(|\langle M \rangle|)$.
 - $\langle M \rangle \in \text{Acc}(t)^c$ iff M does not accept $\langle M \rangle$ in time $\leq t(|\langle M \rangle|)$.
 - Now plug in M_0 for M in both statements:
 - $\langle M_0 \rangle \in \text{Acc}(t)^c$ iff M_0 accepts $\langle M_0 \rangle$ in time $\leq t(|\langle M_0 \rangle|)$.
 - $\langle M_0 \rangle \in \text{Acc}(t)^c$ iff M_0 does not accept $\langle M_0 \rangle$ in time $\leq t(|\langle M_0 \rangle|)$.
 - Contradiction!

A Language Not in P

- $\text{Acc}(t) = \{ \langle M \rangle \mid M \text{ is a basic TM that accepts } \langle M \rangle \text{ in } \leq t(|\langle M \rangle|) \text{ steps} \}$.
- We have proved:
- Theorem: For any computable function t , there is a language that is decidable, but cannot be decided by any basic Turing machine in time $t(n)$.
- Proof:
 - Claim 1: $\text{Acc}(t)$ is decidable.
 - Claim 2: $\text{Acc}(t)$ is not decided by any basic TM in $\leq t(n)$ steps.
- Thus, for every computable function $t(n)$, no matter how large (exponential, double-exponential,...), there are decidable languages not decidable in time $t(n)$.
- In particular, there are decidable languages not in P.