Theory of Computation
Chapter 7

Introduction to Complexity Theory
&
Complexity Classes

School of Engineering | Computer Science

Manuel Blum

- Studies the theoretical limits of what computers can, and cannot, do
- Awarded 1995 Turing Award for "contributions to the foundations of computational complexity theory and its application to cryptography and program checking"
- Professor at UC Berkeley and, later, Carnegie Mellon

History Lesson

- In 1971-72, the Cook-Levin Theory was introduced. The theorem said that:
  - "There exists decidable problems which are believed impossible to solve efficiently."
  - This brought about the idea of NP-Completeness
  - Even if a problem is solvable in principle, it may not be solvable in practice, especially if the solution requires an inordinate about of time or memory.
- In 1991-92, the Probabilistically Checkable Proof (PCP) Theorem was proposed. It says that:
  - "Sometimes solving a problem approximately is still very hard."

P and NP Languages

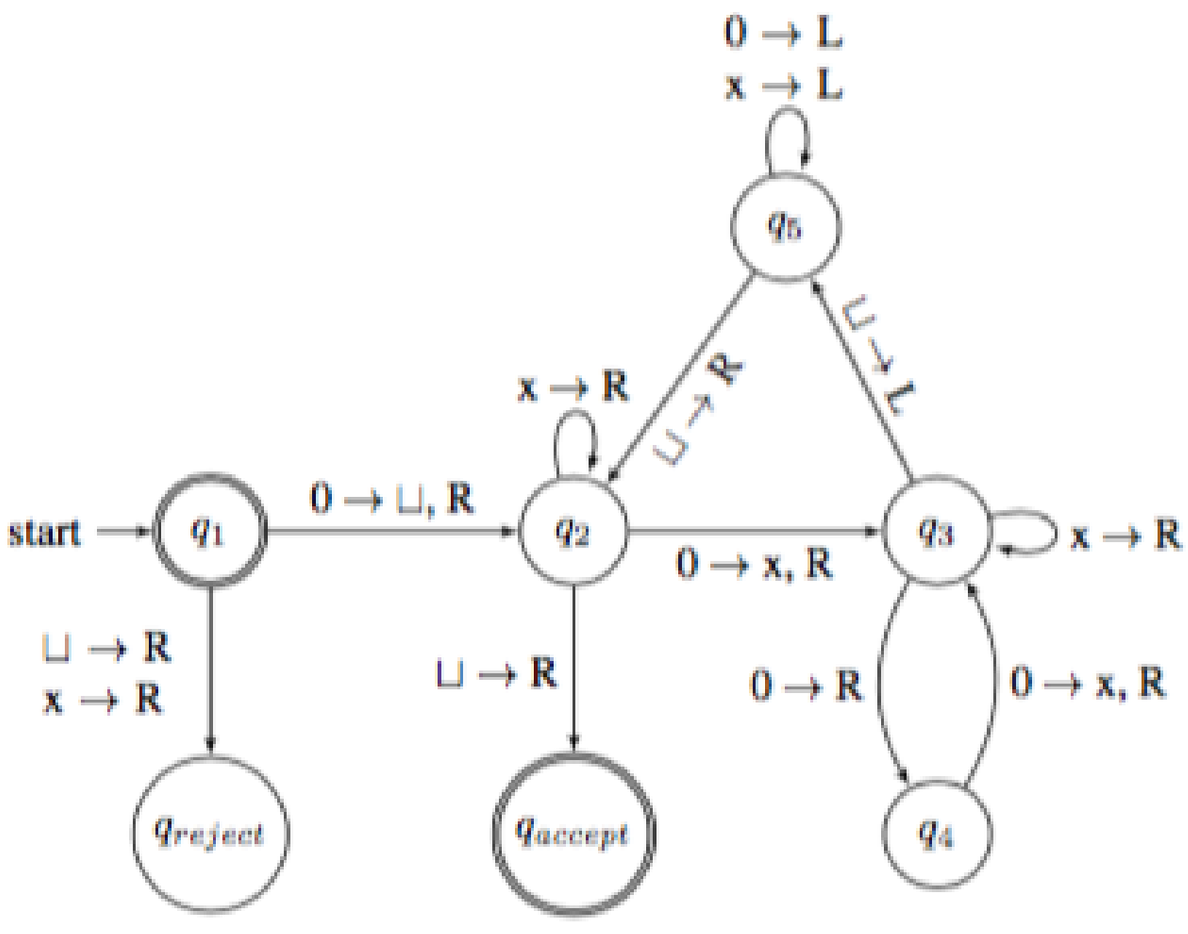
- We will now focus on two main language:
  - P: The class of languages that can be solved efficiently on a deterministic TM
  - NP: The class of languages that can be solved efficiently on a non-deterministic TM
- The big question right now is: Is P equal to NP?
  - There is actually a one million dollar prize for the solution to this question (http://www.claymath.org/millennium-problems/millennium-prize-problems)

Time Complexity

- How do we measure if a TM M decides a problem "efficiently"?
  - Even if a problem is decidable, computationally solvable, it may **not** be solvable in practice.
  - How do we measure time complexity?
- "Worst-Case" Running Time
  - Let M be a deterministic TM that halts on all inputs. The running time of this machine is a function $f: \mathbb{N} \rightarrow \mathbb{N}$ (natural numbers to natural numbers) where f(n) is the maximum number of steps M uses on any input of size n.

Time Complexity

- "Worst-Case" Running Time Example:
  - Machine to decide L = $\{0^{2^n} \mid n \geq 0\}$



How many steps will this machine take when n = 1?

8

How many steps when n = 2?

22

What about when n becomes large?

Big-O Notation

- To measure how fast a function grows, we use big-O notation or asymptotic analysis.
- For example, say a function:

  $f(n) = n^3 + 96n^2 + 1000n + 64$.

- All we care about is the value that will grow the fastest in size as n becomes large. Thus, all that matters is the biggest term, n^3 in this case.
  - We can say that $f(n) \in O(n^3)$

Big-O Notation

- Formal Definition 7.2:
  - Let f and g be functions f, g: $\mathbb{N} \rightarrow \mathbb{R}^+$, where $\mathbb{N}$ is the set of natural numbers and $\mathbb{R}^+$ is the set of non-negative real numbers. We say $f(n) = O(g(n))$ if there exists integers c, $n_0 > 0$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$
  - This means $f(n)$ "$\leq$" $g(n)$
  - Why the "n_0"?
    - We need a starting value for n when this condition will be true.

Big-O Notation

- Formal Definition 7.2:
  - Ex1: Prove $f(n) = 5n^3 + 2n^2 - 22n + 6 = O(n^3)$
    - We can immediately upper bound it by removing any terms that are subtracted as:
      $f(n) = 5n^3 + 2n^2 - 22n + 6 \leq 5n^3 + 2n^2 + 6$
    - Further we can promote each term to the largest term, $f(n) = 5n^3 + 2n^2 - 22n + 6 \leq 5n^3 + 2n^3 + 6n^3$
    - Add those terms together and we get an upper bound and a value for c:
      $f(n) = 5n^3 + 2n^2 - 22n + 6 \leq 13n^3$
    - This is true when $n \geq 1$, so, $c = 13$ and $n_0 = 1$
    - Therefore, $f(n) \leq 13n^3$ for all $n \geq 1$, so $f(n) = O(n^3)$

Big-O Notation

- Example 7.3:
  - Let $f(n) = 5n^3 + 2n^2 - 22n + 6 \in O(n^3)$
    - If we select $5n^3$ as our highest order term and say $f(n) = O(n^3)$. Is this correct?
    - Using the definition 7.2, we can say c = 6 and $n_0 = 10$, so $f(n) = 5n^3 + 2n^2 - 22n + 6 \leq 6n^3$ for every $n \geq 10$
    - We can test this by plugging in 10 for n,
      $f(n) = 5*10^3 + 2*10^2 - 22*10 + 6 \leq 6*10^3$
      $f(n) = 5000 + 200 - 220 + 6 \leq 6000$
      $f(n) = 5426 \leq 6000$, so yes, it is correct.
    - We could have also used $f(n) = O(n^4)$ since n^4 is larger that n^3, but we cannot use any term smaller than n^3 since there are no values for c and n_0 that will make a smaller term work.

Big-O Notation

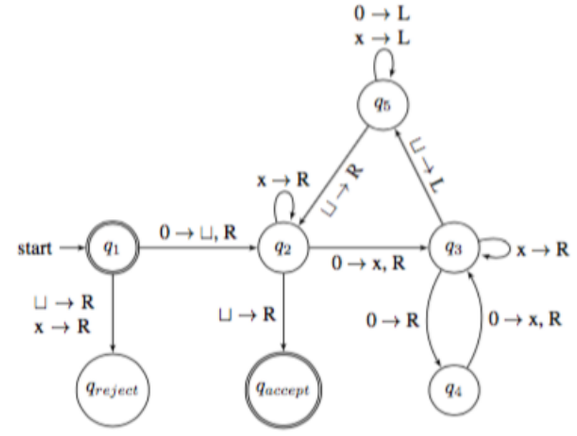
- Example 7.4:
  - Does the base of the log matter in Big-O Notation?
    - Suppose we have $\log_2 n$ verses $\log_3 n$. Which is larger?
    - Answer: They are the same.
      - Here is the change of base formula:

        $\log_b n = \frac{\log_2 n}{\log_2 b}$. So for a, b = $\mathbb{N}$, we have $\log_a n = O(\log_b n)$ and $\log_b n = O(\log_a n)$

        We can simply use $f(n) = O(\log n)$.

Bounds

- In this course we are focused on polynomial bounds
- Polynomial Bound: Dealing with efficient runtimes where $O(n^c)$ for all c>0 and c is a constant that is a Real Number
- Exponential Bound: Dealing with runtimes where $O(2^{n^c})$ for c>0, and c is a constant and a Real Number
  - What does this tell you about the TM to decide $L = \{0^{2^n} \mid n \geq 0\}$ on a previous slide?
    - It is exponential bound?



Big-O Notation

- Properties of Big-O to keep in mind:
  - If $f_1(n) = O(g_1)$ and $f_2(n) = O(g_2)$ then:
    - $f_1 + f_2 = \max(O(g_1), O(g_2))$
    - $f_1 * f_2 = \max(O(g_1 g_2))$
  - Knowing these properties, we can now say that $f(n) = O(g_n)$ means $f(n)$ "$\leq$" $g(n)$.

Little-o Notation

- If we want to say f(n) "<" g(n), we can use Little-o Notation.
- Little-o Notation:
  - Let f and g be functions f, g: $\mathbb{N} \rightarrow \mathbb{R}^+$
  - Say f(n) = o(g(n)) if for all c>0, there exists an n_0 such that f(n) < cg(n) for all n ≥ n_0 or $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$ (f(n) can never be o(g(n)).)
    - In this course, we will use the limit method.

Little-o Notation

- If we want to say f(n) "<" g(n), we can use Little-o Notation.
- Little-o Notation Example:
  - Prove $\sqrt{n}$ = o(n)
    - $\lim_{n \to \infty} \frac{\sqrt{n}}{\text{n}}$ = $\lim_{n \to \infty} \frac{\sqrt{n}}{\text{n}} * \frac{\sqrt{n}}{\sqrt{n}}$ (multiply by $\frac{\sqrt{n}}{\sqrt{n}}$)

      = $\lim_{n \to \infty} \frac{\text{n}}{\text{n}\sqrt{n}}$ (can eliminate n)

      = $\lim_{n \to \infty} \frac{1}{\sqrt{n}}$ = 0 (Since as n reaches infinity, $\frac{1}{\sqrt{n}}$ goes to 0.)

TIME(t(n))

- Definition 7.7
  - Let t: $\mathbb{N} \rightarrow \mathbb{R}^+$ ($\mathbb{N}$ = Natural numbers and $\mathbb{R}^+$ = non-negative real numbers) be a function. TIME (t(n)) is the set of all languages decidable by deterministic Turing Machines running in worst case O(t(n)) on all inputs of size n.
    - Ex: Consider the TM M for deciding A = $\{0^p1^p \mid p \geq 0\}$
      - What is the runtime for M?
        - Must create the implementation-level description of the TM
        - Then decide the worst-case runtime for each step
        - Continued…

TIME(t(n))

- Definition 7.7
  - Ex: Consider the TM M for deciding $A = \{0^p1^p \mid p \geq 0\}$ (Implementation-level description)
    - M = "On input w:
      1. Scan across the tape and reject if a 0 is found to the right of a 1
      2. Repeat if both 0s and 1s remain on the tape:
         a. Scan across the tape, crossing off a single 0, then a single 1
      3. If 0s remain after all 1s are gone, or if 1s remain after all 0s are gone, reject
      4. If neither 0s or 1s remain, accept."

TIME(t(n))

- Definition 7.7
  - Ex: Consider the TM M for deciding A = $\{0^p1^p \mid p \geq 0\}$ (Runtime of each step)
    - M = "On input w:
    1. Scan across the tape and reject if a 0 is found to the right of a 1 (2n steps ∈ O(n))
    2. Repeat if both 0s and 1s remain on the tape:
       a. Scan across the tape, crossing off a single 0, then a single 1 (n/2 * 2n = n^2 ∈ $O(n^2)$ – loop, so multiply where 2 symbols are crossed off each time, so n/2)
    3. If 0s remain after all 1s are gone, or if 1s remain after all 0s are gone, reject (O(n))
    4. If neither 0s or 1s remain, accept." (O(1))
  - Runtime is **$O(n^2)$** since that is the highest complexity

NTIME(t(n))

- This class of languages is the same as TIME(t(n)) except with a non-deterministic Turing Machine (NTM)
- What does it mean for a non-deterministic TM to be a decider?
  - All computational branches of the NTM halt.
- What does it mean for a non-deterministic TM to run in time O(t(n))?
  - All branches have length at most O(t(n))

Review of P and NP

- Recall that:
  - P is the class of languages that can be solved efficiently on a deterministic TM
  - NP is the class of languages that can be solved efficiently on a non-deterministic TM
- Let's define these formally with our new definitions:
  - P = $\mathrm{U}_{c \in \mathbb{N}+} TIME(n^c)$
  - NP = $\mathrm{U}_{c \in \mathbb{N}+} NTIME(n^c)$
- This brings back the question is P = NP?

Review of P and NP

- Question is P = NP?
  - This answer is not obvious,
  - Given a non-deterministic TM T_1, what overhead is required to simulate T_1 with a deterministic TM T_2?
- <u>Theorem 7.11</u>
  - Let $t(n) \geq n$ be a function. Then for all $O(t(n))$-time non-deterministic TM T_1 there exists a $2^{O(t(n))}$-time TM T_2 simulating T_1 (where T_1 and T_2 are both single-tape)

Conversion of P and NP

- Theorem 7.11
  - Let $t(n) \geq n$ be a function. Then for all O(t(n))-time non-deterministic TM T_1 there exists a $2^{O(t(n))}$-time TM T_2 simulating T_1 (where T_1 and T_2 are both single-tape)
  - Proof: Recall Theorem 3.16 from Chapter 3 where we showed every non-deterministic TM has an equivalent deterministic TM. We constructed a tree to see if there existed a leaf node marked accept. Regardless of our search (breadth-first or depth-first), in the worst-case scenario we must visit all the nodes of the tree to find an accept node.

Conversion of P and NP

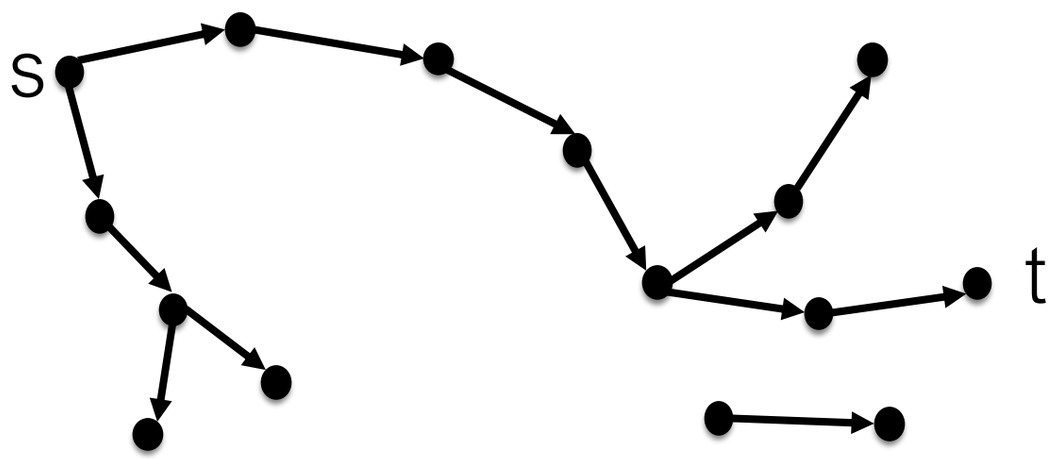
- Theorem 7.11
  - Question: How many nodes are there? This will give us a better idea of the worst possible runtime.
    - Suppose at each node in the tree we have at most b children (b is a constant $b \in O(1)$).
    - So, the number of nodes $\leq 1 + b + b^2 + \ldots + b^{(t(n))}$ (where t(n) is the length of the longest branches)
    - The total number of nodes in the tree is $O(b^{t(n)})$. The time it takes to start from the root and travel to a node is O(t(n)), so the running time = $O(t(n)b^{t(n))}) = 2^{O(t(n))}$ (power set)
    - This is exponential and not in P. Is there a better algorithm?

Problems in P

- Let's look at some problems in P:
  - The Path Problem
  - Context-free Languages

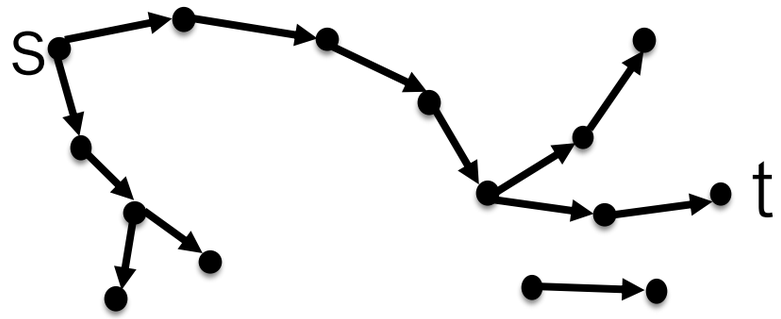
Path Problem

- Theorem 7.14
  - PATH ∈ P
  - PATH = {<G, s, t> | G is a directed graph that has a directed path from s to t}
  - Question: Is there a path from s to t?



Path Problem

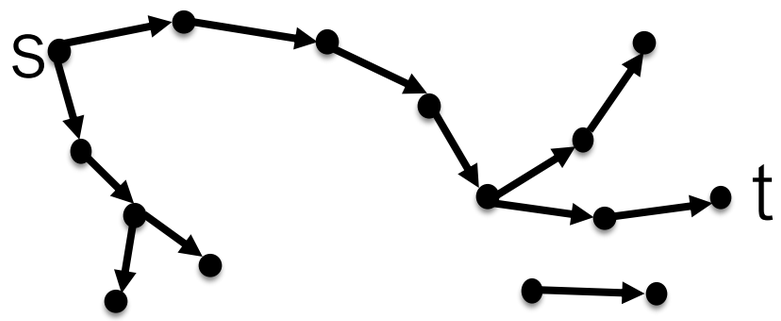
- Theorem 7.14
  - Question: Is there a path from s to t?



  - If we performed a brute-force algorithm for PATH and examined all the potential paths in G and determined whether any of them went from s to t, we would potentially examine all nodes in G and many of them many times.
  - If m is the length of the longest sequence of nodes in G, we could potentially travel m^m paths. This is exponential and not in P.

Path Problem

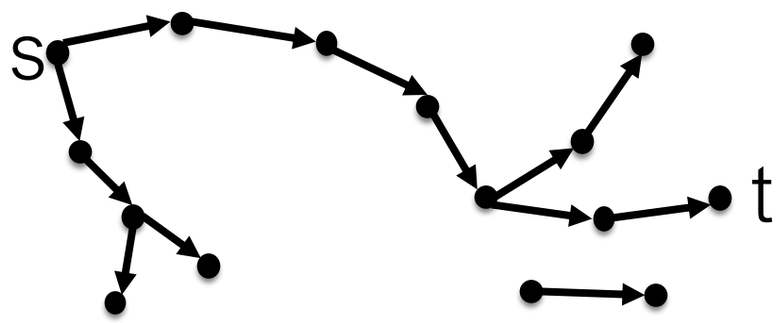
- Theorem 7.14
  - Question: Is there a path from s to t?



  - Instead, we can do a breadth-first algorithm for PATH

Path Problem

- Theorem 7.14 – Breadth-first algorithm for PATH
  - M = "On input <G, s, t>, where G is a directed graph with nodes s and t:
    1. Place a mark on node s
    2. Repeat the following until no additional nodes are marked:
    3. Scan all the edges of G. If an edge (a, b) is found going from a marked node a to an unmarked node b, mark node b.
    4. If t is marked, accept. Otherwise reject."
- Steps 1 and 4 are executed once. Step 3 runs at most m times. The run time is 1 + 1 + m * m which is $O(n^2)$, polynomial time.



Context-free Languages Property

- Theorem 7.16: Every context-free language is a member of P
- Proof Idea: Every CFL is decidable (proved in Theorem 4.9). If the algorithm that decides it runs in polynomial time, then context-free languages are members of P
  - Let L be a CFL generated by a CFG G that is in Chomsky Normal Form
  - Any derivation of a string w has 2n-1 steps where n is the length of w because G is in Chomsky Normal Form
  - The decider for L tries all possible derivations with 2n-1 steps when its input is a string of length n. If any of these is a derivation of w, the decider accepts; if not, it rejects.
  - This algorithm does not run in polynomial time. The number of derivations with k steps may be exponential in k.

Context-free Languages Property, cont.

- Theorem 7.16: Every context-free language is a member of P
- Proof Idea: Every CFL is decidable (proved in Theorem 4.9). If the algorithm that decides it runs in polynomial time, then context-free languages are members of P
  - How do we make this algorithm run in polynomial time?
    - Use dynamic programming where we accumulate information about smaller sub-problems to solve larger problems, so we solve the sub-problems just once
    - We make a table of the solutions of the sub-problems so we can reference them later.
    - The algorithm fills in the table entries for each substring of w starting at length 1, then of length 2 and so on, using the entries for the shorter lengths for solving the longer lengths.

Context-free Languages Property, cont.

- Theorem 7.16 Proof: Every context-free language is a member of P
  - D = " On input w = $w_1 \ldots w_n$:

1. For w = ε, if S → ε is a rule, accept; else reject. (w = ε case)
2. For i = 1 to n: (examine each substring of length 1) O(n)
3. For each variable A:
4. Test whether A → b is a rule, where b = w_i.
5. If so, place A in table(i, i). O(n)
6. For l = 2 to n: (l is the length of the substring) O(n)
7. For i = 1 to n-l+1: (i is the start position of the substring)
8. Let j = i+l-1: (j is the end position of the substring)
9. Let k = i to j-1: (k is the split position) $O(n^3)$
10. For each rule A → BC: $O(n^3)$
11. If table(i, k) contains B and table(k+1, j) contains C, put A in table(i, j)
12. If S is in table(1, n) accept; else, reject." O(n) Total: $O(n^3)$

Context-free Languages Property

- Theorem 7.16: Applied
- Consider the TM for deciding L = {#x_1#x_2#x_3 … #x_i | each x ∈ {0, 1}* and i is distinct (no two strings are the same)}
  1. If the first symbol is ⊔, accept.
  2. Place a mark on top of the left-most tape symbol. If the first symbol is not #, reject.
  3. Scan right to next # and place a mark on it. If no # is encountered before ⊔, accept.
  4. Compare two strings to the right of the marked #'s zigzagging back and forth. If they are equal, the machine rejects.
  5. Move to the right-most of the two marks to the next # to the right. If no # is found before ⊔, move the first mark to the next # to its right and the second mark to the next # after that. If no # is found for the second mark, accept.
  6. Scan left and go to step 4.

What is the runtime of M?

Context-free Languages Property

- Theorem 7.16: Applied
- Consider the TM for deciding L = {#x_1#x_2#x_3 … #x_i | each x ∈ {0, 1}* and i is distinct (no two strings are the same)}

O(1) 1. If the first symbol is ⊔, accept.

O(1) 2. Place a mark on top of the left-most tape symbol. If the first symbol is not #, reject.

O(n) 3. Scan right to next # and place a mark on it. If no # is encountered before ⊔, accept.

O(2n) 4. Compare two strings to the right of the marked #'s zigzagging back and forth. If they are equal, the machine rejects.

O(3n) 5. Move to the right-most of the two marks to the next # to the right. If no # is found before ⊔, move the first mark to the next # to its right and the second mark to the next # after that. If no # is found for the second mark, accept.

O(n/2) 6. Scan left and go to step 4.

What is the runtime of M?

$O(1) + O(1) + O(n) + (O(2n) + O(3n)) \times O(n) = O(n^2)$

(loop, so multiply steps in loop times number of times loop executes)

Try It

1. Find Big-O for $f(n) = 9n^5 - 6n^4 + 22n^3 + 68n^2$.
2. Here is a description of a TM to decide membership in language B = {w#w | w ∈ {0, 1}*}. What is the runtime of B?
   - On input x ∈ {0, 1}*
     1. If no # found, reject.
     2. Scan right to the first unmarked symbol before the # symbol, mark it, scan right to the # symbol and scan right to the first unmarked symbol. If it is not the same as the first symbol, reject. Else, mark it.
     3. Scan left, if unmarked symbols remain, go to Step 2.
     4. When all symbols to the left of the # symbol are crossed off, if any non-blank symbols remain to the right of the # symbol, reject.

Try It

1. Find Big-O for $f(n) = 9n^5 - 6n^4 + 22n^3 + 68n^2$.
   - 1 – Combine positive terms to get c = 9+22+68 = 99, and $n_0 = 1$, so $9n^5 - 6n^4 + 2n^3 + 8n^2 \leq 99n^5$, or
   - 2 – Let c = 10 and $n_0 = 10$, $g(n) = 10n^5$, so $9n^5 - 6n^4 + 22n^3 + 68n^2 \leq 10n^5$, since 9*100,000 + 6*10,000 + 22*1,000 + 68*100 ≤ 10*100000, or 988,800 ≤ 1,000,000

Try It

- Here is a description of a TM to decide membership in language B = {w#w | w ∈ {0, 1}*}. What is the runtime of B?
  - On input x ∈ {0, 1}*
    1. If no # found, reject. (O(n))
    2. Scan right to the first unmarked symbol before the # symbol, mark it, scan right to the # symbol and scan right to the first unmarked symbol. If it is not the same as the first symbol, reject. Else, mark it. (O(n))
    3. Scan left, if unmarked symbols remain, go to Step 2. (O(n))
    4. When all symbols to the left of the # symbol are crossed off, if any non-blank symbols remain to the right of the # symbol, reject. (O(n))

$O(n) + O(n)*O(n) + O(n) = O(n^2)$. So, the runtime is $O(n^2)$.