Introduction to Time Complexity

Mark Greenstreet, CpSc 421, Term 1, 2008/09

- Time complexity
  - Reviewing Big-O
  - Defining Time Complexity
  - Complexity vs. models of computation
- P and NP
  - P: Polynomial time on a deterministic TM
  - NP: Polynomial time on a non-deterministic TM

Big-O Review

- Definition:
  - Let $f : \mathbb{N} \to \mathbb{R}$ and $g : \mathbb{N} \to \mathbb{R}$ be functions.
  - We say that $f(n) = O(g(n))$ iff there exists constants n_0 and c such that for all $n \geq n_0$, $f(n) \leq cg(n)$.
  - Intuitively, $f(n) = O(g(n))$ says that f grows no faster than g.
- True or false:
  - 1. $n + 2 = O(n^2)$?
  - 2. $1000n + 200 = O(n)$?
  - 3. $\sum_{i=1}^{n} n = O(n \log n)$?
  - 4. $\log(n!) = O(n \log n)$?
  - 5. $n^2 = O(2^n)$?

Big-O short cuts

- Big-O and sums and products:

$$\begin{aligned} O(f(n)+g(n)) &= O(\max(f(n),g(n))) = O(f(n))+O(g(n)) \\ O(f(n)g(n)) &= O(f(n))+O(g(n)) \end{aligned}$$

- Transitivity: If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.
- A few more rules:
  - If $f(n) = O(g(n))$, then $O(f(n)+g(n)) = O(f(n))$.
  - If $f(n) = 2^{O(g(n))}$, that means there is some constant c such that $2^{cg(n)}$ is an upper bound for f (for n sufficiently large.
  - If $f(n) = 2^{O(\log n)} = n^{O(1)}$, that means there is some constant c such that n^c is an upper bound for f. In other words, f is polynomial in n.
- Little-o:
  - If $lim_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$, then we say that $f(n) = o(g(n))$.
  - Sipser discusses little-o. Read.

Time Complexity

- A language, A has time-complexity $g(n)$ iff there is some TM that decides A and performs $O(g(n))$ steps to decide an input of length n.
- We write $TIME(g(n))$ to denote the class of all languages with a time complexity of $O(g(n))$.
- We can extend this to arbitrary functions. Function f has time complexity $O(g(n))$ if there is a TM that given an input of length n computes $f(n)$ and writes it on its tape using $O(g(n))$ steps.
- Big-O ignores constant factors. Therefore:

$$O(\log_2(n)) \quad = \quad O(\log_{10}(n)) \quad = \quad O(\ln(n))$$

Computation Models

- Whether or not a function is computable or a language is decidable is the same for TMs, multi-tape TMs, non-deterministic TMs, traditional computers, etc.
- The time complexity of a computation depends on the model.
- For example, if a language has a time complexity of $O(t(n))$ on a multi-tape TM, then it has a complexity of $O(t^2(n))$ on a single-tape TM.
  - If a k-tape TM decides a language in $O(t(n))$ time, it writes at most $O(kt(n)) = O(t(n))$ symbols on its tape.
  - A single-tape TM can simulate the k-tape machine using a tape of length $n + O(t(n)) = O(t(n))$. Here, I'm assuming that $t(n) \geq n$, i.e. the TM reads all of its input.
  - The single-tape TM can simulate the k-tape machine by making two-passes over its tape for each step of the k-tape machine. Thus, the single-tape TM takes at most time $O(t(n))$ to simulate a step of the k-tape machine.
  - Thus, the total time for the single-tape machine to simulate a k-tape machine, when the later takes $O(t(n))$ steps is $O(t(n)) \cdot O(t(n)) = O(t^2(n))$.
  - $\square$

Random Access Machines (RAMs)

- Most algorithm analysis is based on a machine that can perform one "operation" at each time step where an "operation" can be:
  - An arithmetic operation: add, subtract, mutiply, divide, etc.
  - A comparison
  - A branch
  - A memory access

  Typically, we assume that there is some fixed-sized "word" for these operations, where a "word" is large enough to hold the individual data values that occur in the problem.
- In principle, a TM can perform arithmetic and comparisons on fixed sized words in a single step.
- Branches and memory operations require going to arbitary memory locations.
  - If a RAM computes a function in $O(t(n))$ time, then it access at most $O(t(n))$ different memory locations.
  - By an argument like that on the previous slide, then a single-tape TM can perform the same computation in $O(t^2(n))$ time.
  - Conversely, if a single tape TM requires at least $T(n)$ time, then a RAM requires at least $\sqrt{T(n)}$ time.

Polynomial Time

- A language, A, is decidable in polynomial time iff there is a TM that decides it in $O(p(n))$ steps, where p is a polynomial.
- We write P to denote the set of all languages that can be decided in polynomial time.
- Examples:
  - Is there a path from vertex s to vertex t in graph G?
  - Are integers n and d relatively prime?
  - Is string w generated by CFG G?
  - Are there two people in this class with the same birthday?
  - Is m a factor of n?

Handy Properties of Polynomials

- Properties – let $f(n)$ and $g(n)$ be polynomials.
  - Let $s(n) = f(n) + g(n)$. Then $s(n)$ is a polynomial.
  - Let $p(n) = f(n) * g(n)$. Then $p(n)$ is a polynomial.
  - Let $h(n) = f(g(n))$. Then $h(n)$ is a polynomial.
  - Let $x(n) = c$ for some constant c. Then $x(n)$ is a polynomial.
- Uses
  - If an algorithm performs a fixed set of steps, and each step takes polynomial time, then the algorithm takes polynomial time.
  - If an algorithm performs a polynomial number of steps, and each step take polynomial time, then the algorithm takes polynomial time.
  - If an algorithm includes a loop that is performed a polynomial number of times, and each itereation takes polynomial time, then the algorithm talks polynomial time.
  - If algorithm X when run on input s produces data structure w in polynomial time, and then runs algorithm Y on w where Y takes polynomial time (in the length of w), then X takes polynomial time (in the length of s).

P, TMs and RAMs

- As shown above, if there is a RAM that computes function f in $O(t(n))$ time, then there is a TM that computes the same function in at most $O(t^2(n))$ time.
- Furthermore, a RAM can simulate a TM such that if there is a TM that computes f in $O(u(n))$ time, then there is a RAM that computes the same function in at most $O(u(n))$ time as well.
- Thus, the class of polynomial-time functions (including polynomial time languages) is the same for RAMs and TMs.
- In fact, P is the same for any "reasonable" model of computation.

Non-Deterministic Polynomial Time

- A language, A, is decidable in polynomial time iff there is a non-deterministic TM that decides it in $O(p(n))$ steps, where p is a polynomial.
- We write NP to denote the set of all languages that can be decided in non-deterministic polynomial time.

An NP example

$SubsetSum$

- Given a set S of integers, and an integer m, is there a set $B \subseteq S$ such that the sum of the elements of B is equal to m?
- We can describe S and m with strings. Thus, $SubsetSum$ is a language.
- A polynomial time algorithm on a non-deterministic TM:

```
Q = ∅;                    /* Elements we've selected from S */
q = 0;                    /* Sum of elements of Q */
T = S;                    /* T = S − Q: elements left to choose. */
while(q < m) {
    p = choose(T);        /* non-deterministically select an element from T */
    Q = q ∪ {p}; q = q + p;   /* update Q and q */
    T = T − {p};              /* update T */
}
return(q == m);
```

- Thus, $SubsetSum \in NP$.
- Is $SubsetSum \in P$?

Certificates

- We could also write a non-deterministic program that picks a subset of Q first and then (deterministically) verifies it:

  Choose $Q \subseteq S$; /* Non-deterministic choice */
  /* Now, verify Q */
  $k = 0$;
  for each q in Q {
  $k = k + p$;
  }
  return(k $==$ m);

- We call Q a certificate for the subset sum problem.
  - Intuitively, a certificate specifies the choices made by a non-deterministic TM for an accepting run (if the input is in the $SubsetSum$ language.
  - Because the non-deterministic TM ran in a polynomial number of steps, the certificate has a polynomial length.

Certificates and Class NP

- A language, A is in class NP iff there exists another languager $A' \in P$, and a polynomial, f such that

$$A = \{s \mid \exists c.\ (|c| \leq f(|s|)) \wedge (s\#c \in A')\}$$

  - c is the certificate for s.
  - The requirement that $|c| \leq f(|s|)$ says that the certificate must have a length that is at most a polynonial in the input size.
  - For example, our certificate for subset sum was the subset of the input set whose elements sum to the target value. Clearly, this certificate is smaller than the input and satisfies this requirement.
  - A non-deterministic TM (NTM) that runs in polynomial time for all inputs (whether or not the input is in A) can write c on its tape in polynomial time and then run a deterministic TM that decides A'.
    - If $sin A$, then the NTM writes a c that proves this and the decider for A' accepts.
    - Otherwise, the NTM can write anything, and the decider for A' rejects.

- This definition of NP is equivalent to the one given on slide 10.

This coming week (and beyond)

- Reading
  - Today: Sipser 5.3
  - Nov. 14 (Friday): Sipser 7.1
  - Nov. 17 (Monday): Sipser 7.2
  - Nov. 19 (A week from today): Tutorial by Brad Bingham
- Homework
  - Nov. 14 (Friday): HW 10 goes out.
  - Nov. 17 (Monday): HW 9 due.