Lecture 3: Computational Complexity
Modeling Social Data, Spring 2017
Columbia University

Yunsi Zhang

February 3, 2017

NOTES

1.Computational Tractability

- The optimal value could be calculated in a reasonable amount of time.

2.Asymptotic order of growth — Big O notation

- It is used to understand approximation when variables go to infinite.
- In computer science, big O notation is usually used to evaluate algorithms on their running time and space as the input size grows.

3.Common running times

- It depends on data structure and algorithms.

4.Running time in size n input

- brute force algorithm : $2^n \Rightarrow$ exponential
- cn^d $(c > 0\ d > 0) \Rightarrow$ polynomial
- It is theoretical step here, but does not perform well in practice purpose.
  Constant matter: $\underbrace{20n^{100}}_{polynomial}$ vs $\underbrace{n^{1+0.02lgn}}_{notpolynominal,butstillbetter}$ $\Rightarrow$ So notice that constant cannot be ignored!
- Order of running time: $n < nlogn < n^2 < n^3 < 1.5n^3 < 2^n < n!$

5.Runtime analysis

1. Worst-case Complexity analysis(Our focus):
   Worst-case Complexity analysis considers the maximal complexity of the algorithm over all possible inputs.
2. Average-case Complexity:
   Average-case Complexity considers the average complexity over all possible inputs. (Making an assumption of the statical distribution of the inputs $\Rightarrow$ expected time, uniform distribution) It may be a more accurate measure of an algorithm's performance in practice.

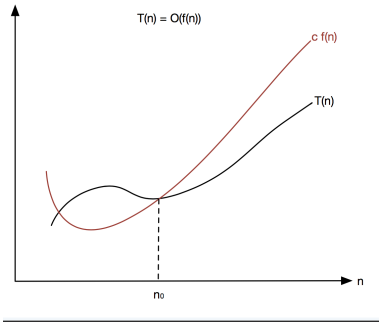

Figure 1: Big-) notation

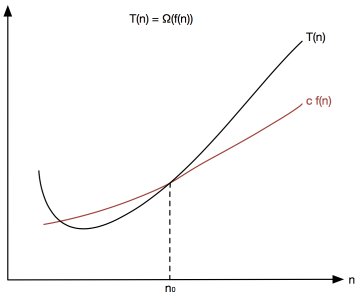

Figure 2: Big-Omega notation

3. Amortized Analysis:
   Amortized Analysis considers the running time for a sequence of operations rather than worst-case analysis focusing on each operation.

   - Eg. In stack, we push in for O(1), pop all for O(n) in worst case.
   For worst-case analysis, its running time should be O(n);
   For average-case analysis, calculate the expectation of it;
   For amortized analysis, O(1).

6.Asymptotic Analysis

1. - Upper Bound(Big-O notation):
   For a function T(n)(running time of y), f(n) is an upper bound if for "big enough n", $\exists c > 0$, $T(n) \leq cf(n)$.
   Here f dominates T.(Figure1)
   Eg. $T(n) = 32n^2 + 17n + 1 \Rightarrow T(n) = O(n^2)$, choose $c = 50, n = 1$

2. - Lower Bound(Big-Ω notation):
   For a function T(n)(running time of y), f(n) is a lower bound if for "big enough n", $\exists c > 0$, $T(n) \leq cf(n)$.
   Here f dominates g. (Figure2)
   Eg. For the same example $\Rightarrow T(n) = \Omega(n)$; $T(n) = \Omega(n^2)$, choose $c = 31, n \geq 1$

3. -Tight Bound(Big-Θ notation):
   If f(n) is both upper bound and lower bound of T(n) [with different c], we say f(n) is a tight bound for T(n). That is, $\exists c_1 > 0, c_2 > 0$, $c_1 f(n) \leq T(n) \leq c_2 f(n)$.(Figure3)
   Eg. For the same example $\Rightarrow T(n) = \Theta(n^2)$

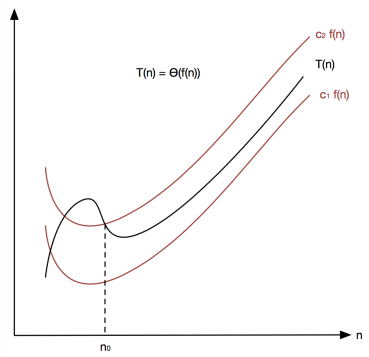

Figure 3: Big-Theta notation

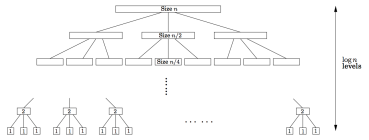

Figure 4: Divide and Conquer

7. Rules of Thumb

1. $T(n) = a_0 + a_1 n + \ldots + a_d n^d \Rightarrow T(n) = O(n^d)$
2. $O(log_a^n) = O(log_b^n)$, $a, b > 0$
3. For every $r > 1, d > 0 \Rightarrow n^d = O(r^n)$

8. Examples

1. O(n) - Linear
   Assumption: We can do constant time work for each number for i in the set $((O(1)))$
   Notion: Modern computers have parallel computation ability. for example, all log could be computed at once.
2. $O(nlog_2 n)$ - Divide and Conquer algorithms (figure4)
   Divide the problem into a number of subproblems that are smaller instances of the same problem.
   Conquer the subproblems by solving them recursively. Combine the solutions to the subproblems into the solution for the original problem. Eg.Sorting: each level: $log_2 n$, n levels $\Rightarrow n(log_2 n + 1) \Rightarrow O(nlog_2 n)$
3. $O(n^2)$ - Quadratic
   Eg. Bubble sort, quick sort $\Rightarrow$ Worst:$O(n^2)$; Average: $O(nlogn)$
4. $O(n^3)$ - Cubic time
   Eg. Count the number of triangles in graphs
5. $O(logn)$
   Eg. Searching sorted array; Binary Tree Search(Balance Tree)

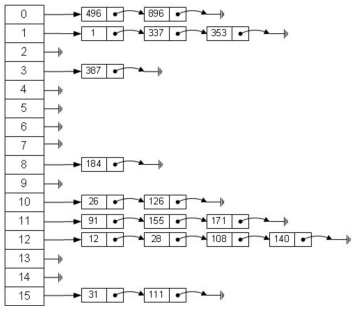

Figure 5: Hash Table.

9.Hash table

- It is a data structure used to implement an associative array, a structure that can map keys to values. It adopts hash function to compute an index into an array of buckets or slots with values. (Figure5)
- Hash collisions are unavoidable in practice because hashing a random subset of a large set of possible keys often occurs.
- Eg. Query problem:
  - Sorting $\Rightarrow O(nlogn)$
  - Hash Table $\Rightarrow O(n)$