CS 341: Algorithms

Michael Noukhovitch

Winter 2015, University of Waterloo

Notes written from Timothy Chan's lectures.

Contents

1 Introduction

1.1 Algorithm Analysis

analysis: determine if algorithm is correct and efficient
correct: formal proof using loop invariant
efficient: solving in polynomial time *(usually)*

1.2 Maximum Problem

Find the largest element in an array

```
max = list[0]
for i in list[1:]:
    if i > max:
        max = i
return max
```

1.3 3SUM Problem

2 Math Review

2.1 Asymptotic Notation

Definition (O)**.** upper bound ($\leq$)
$f(n) = O(g(n))$ if $\exists$ constants $c > 0, n_0 > 0$ such that $f(n) \leq c \cdot g(n)\ \forall n \geq n_0$

Definition (Ω)**.** lower bound ($\geq$)
$f(n) = \Omega(g(n))$ if $\exists$ constants $c > 0, n_0 > 0$ such that $f(n) \geq c \cdot g(n)\ \forall n \geq n_0$

Definition (Θ)**.** tight bound ($=$)
$f(n) = \Theta(g(n))$ if $\exists$ constants $c_1, c_2 > 0, n_0 > 0$ such that $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\ \forall n \geq n_0$

Definition (o)**.** loose upper bound ($<$)
$f(n) = o(g(n))$ if $\forall$ constants $c > 0,\ \exists$ constant $n_0 > 0$ such that $f(n) < c \cdot g(n)\ \forall n \geq n_0$

Definition (ω)**.** loose lower bound ($>$)
$f(n) = \omega(g(n))$ if $\forall$ constants $c > 0$ $\exists$ constant $n_0 > 0$ such that $f(n) > c \cdot g(n)\ \forall n \geq n_0$

2.2 Asymptotic Tricks

$f(n) \in o(g(n)) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
$f(n) \in \omega(g(n)) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$
$f(n) \in O(g(n)) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$
$f(n) \in \Omega(g(n)) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$

2.3 Summations

$\sum_{i=1}^{n} i^d = \Theta(n^{d+1})$ for any constant $d > -1$

$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1} = \begin{cases} \Theta(c^n) & \text{for all constants } c > 1 \\ \Theta(1) & \text{for all constants } c < 1 \end{cases}$

$\sum_{i=1}^{n} \frac{1}{i} = \Theta(\log(n))$

$\sum_{i=1}^{n} \log(i) = n\log(n) = \Theta(n)$

2.4 Divide and Conquer

2.5 Recurrences

2.5.1 Recursion Tree Method

Break down our recursion into a tree with each child node being a recursion. Find the cost of each child and sum across the level

Example 2.1. $T(n) = \begin{cases} 2T(\frac{n}{2}) + n^2 & \text{if } n > 1 \\ 7 & \text{if } n = 1 \end{cases}$

level 0	n^2
level 1	$2 \cdot \frac{n^2}{4} = \frac{n^2}{2}$
level 2	$4 \cdot \frac{n^2}{16} = \frac{n^2}{4}$
...	...
level k	$\frac{n^2}{2^k}$

level k must match our base case, so:

$$\frac{n^2}{2^k} = 7$$
$$k = \log_2 \frac{n^2}{7}$$

So we get

$$\begin{aligned} T(n) &= n^2 \sum_{i=0}^{k} (\frac{1}{2^i}) \\ &= n^2 + 7n \\ &= O(n^2) \end{aligned}$$

2.5.2 Master Method

Lookup the answer knowing that your algorithm is expressed as: $T(n) = \begin{cases} aT(\frac{n}{b}) + f(n) & \text{if } n > n_0 \\ c & \text{else} \end{cases}$

Set $d = \log_b a$ and we end up with three cases:

1. $f(n) \in O(n^{d-\epsilon}) \Rightarrow T(n) \in \Theta(n^d)$

2. $f(n) \in \Theta(n^d) \Rightarrow T(n) \in \Theta(n^d \log n)$

3. $f(n) \in \Omega(n^{d+\epsilon}) \Rightarrow T(n) \in \Theta(f(n))$

Example 2.2. $a = 2,\ b = 2,\ f(n) = n^2$
$d = 1$
$n^2 \in \Omega(n^{1+\epsilon}) \therefore T(n) \in \Theta(n^2)$

2.6 Design Strategy

1. **divide**: subdivide the problem into sub-problems

2. **conquer**: recursively solve the sub-problem

3. **combine**: combine the solutions of the sub-problem to solve the problem