Algorithm Design, Analysis & Complexity

Lecture 1 - Divide and Conquer

Koushik Pal

University of Toronto

May 4, 2021

What is an algorithm?

Informally, an **algorithm** is a finite sequence of computational steps that converts an input set to an output set.

Formally, an algorithm is defined via Turing Machines.

Two main issues to be considered when designing an algorithm are:

- Proof of correctness: An algorithm is correct if, for **every** input instance, it halts with the correct output.
- Efficiency/Complexity: How fast an algorithm runs and/or how much space an algorithm requires as a function of the input size.
  - Time Complexity
  - Space Complexity

Note: For this course, algorithm is synonymous with pseudocode.

Proof Techniques

- Direct Proof
- Proof by Contradiction
- Proof by Contraposition
- Proof by Cases
- Proof by Elimination
- Proof by Mathematical Induction

Mathematical Induction

Mathematical Induction

For any property P, if

1. $P(a)$ holds (base case), and
2. $P(n)$ holds $\implies P(n+1)$ holds (induction step),

then $P(n)$ holds for all $n \geq a$.

Strong Mathematical Induction

For any property P, if

1. $P(a)$ holds (base case), and
2. $P(m)$ holds for all $a \leq m < n \implies P(n)$ holds (induction step),

then $P(n)$ holds for all $n \geq a$.

Complexity Orders

- $\mathcal{O}(g(n)) := \{f(n) \mid \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$
- $\Omega(g(n)) := \{f(n) \mid \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}$
- $\Theta(g(n)) := \{f(n) \mid \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}$
- $o(g(n)) := \{f(n) \mid \text{for any positive constant } c\text{, there exists a positive constant } n_0 \text{ such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0\}$
- $\omega(g(n)) := \{f(n) \mid \text{for any positive constant } c\text{, there exists a positive constant } n_0 \text{ such that } 0 \leq cg(n) < f(n) \text{ for all } n \geq n_0\}$.

Complexity Orders

Examples

- $n = \mathcal{O}(n^2)$
- $n^2 = \Omega(n)$
- $n^2 = \Theta(5n^2 + 6n + 8)$
- $n = o(n^2)$
- $\lg n = o(n^\epsilon)$ for any $\epsilon > 0$

Divide and Conquer Approach

In divide-and-conquer, we solve a problem recursively, applying three steps at each level of the recursion:

- **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
- **Conquer** the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
- **Combine** the solutions to the subproblems into the solution for the original problem.

Binary Search

Goal: Search for an item v in a sorted array A of size n.

---

procedure $\textsc{BinarySearchWrapper}(A, v)$
    return $\textsc{BinarySearch}(A, v, 0, n-1)$

---

procedure $\textsc{BinarySearch}(A, v, l, r)$
    if $r < l$ **then**
        return -1
    $m := \left\lfloor \frac{l+r}{2} \right\rfloor$
    if $A[m] > v$ **then**
        return $\textsc{BinarySearch}(A, v, l, m-1)$
    else if $A[m] < v$ **then**
        return $\textsc{BinarySearch}(A, v, m+1, r)$
    else
        return m

Binary Search

Proof of correctness: Via (strong) mathematical induction on n.

Recurrence:

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

Complexity: $\Theta(\lg n)$.

Merge Sort

Goal: Sort an array A of size n.

procedure $\textsc{MergeSortWrapper}(A)$
  $\textsc{MergeSort}(A, 0, n-1)$

procedure $\textsc{MergeSort}(A, l, r)$
  if $r > l$ **then**
    $m := \left\lfloor \frac{l+r}{2} \right\rfloor$
    $\textsc{MergeSort}(A, l, m)$
    $\textsc{MergeSort}(A, m+1, r)$
    $\textsc{Merge}(A, l, m, r)$

Merge Sort

Exercise

Write a pseudocode for $\textsc{Merge}$.

Show that the complexity of $\textsc{Merge}$ is $\Theta(n)$.

Recurrence:

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

master theorem

$a=2,\ b=2$

Complexity: $\Theta(n \lg n)$.

case2

Powering a Number

Goal: Computing a^n for two integers a and n.

```
procedure RecursivePower(a, n)
    if n == 0 then
        return 1
    if n even then
        b = RecursivePower(a, n/2)
        return b × b
    if n odd then
        b = RecursivePower(a, (n-1)/2)
        return b × b × a
```

Recurrence:

$$T(n) = \begin{cases} T(\frac{n}{2}) + \Theta(1) & \text{if } n > 0 \\ \Theta(1) & \text{if } n = 0. \end{cases}$$

Complexity: $\Theta(\lg n)$.

Fibonacci Series

Definition (Fibonacci Series)

$F_0 = F_1 = 1$
$F_{n+2} = F_{n+1} + F_n$ for all $n \geq 0$.

Goal: Compute the n^{th} Fibonacci number F_n.

Bottom-up Approach: Compute $F_0, F_1, F_2, \ldots F_n$ in order.
Complexity: $\Theta(n)$.

Matrix Multiplication Approach:
Verify via mathematical induction that

$$\begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+2} \qquad \text{for all } n \geq 0.$$

Complexity: $\Theta(\lg n)$.

Integer Multiplication

Goal: Multiply two integers x and y of bit size n.

---

procedure $\textsc{RecursiveMultiply}(x, y)$
    if $x == 0$ or $y == 0$ **then**
        return 0
    if $x == 1$ **then**
        return y
    if $y == 1$ **then**
        return x
    Let $n =$ max number of bits in x or y
    Write $x = x_1 \cdot 2^{n/2} + x_0$, $y = y_1 \cdot 2^{n/2} + y_0$
    Compute $x_1 + x_0$ and $y_1 + y_0$
    $p = \textsc{RecursiveMultiply}(x_1 + x_0, y_1 + y_0)$
    $x_1 y_1 = \textsc{RecursiveMultiply}(x_1, y_1)$
    $x_0 y_0 = \textsc{RecursiveMultiply}(x_0, y_0)$
    return $x_1 y_1 \cdot 2^n + (p - x_1 y_1 - x_0 y_0) \cdot 2^{n/2} + x_0 y_0$

Integer Multiplication

Observe that $xy = x_1 y_1 \cdot 2^n + (x_1 y_0 + x_0 y_1) \cdot 2^{n/2} + x_0 y_0$.
If we want to compute xy using this formula, we would have to compute 4 smaller multiplications, namely $x_1 y_1$, $x_1 y_0$, $x_0 y_1$ and $x_0 y_0$. This requires 4 calls to RecursiveMultiply.

But $x_1 y_0 + x_0 y_1 = (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0$.
Because of this observation, we only need to compute 3 smaller multiplications to compute xy, namely $(x_1 + x_0)(y_1 + y_0)$, $x_1 y_1$ and $x_0 y_0$. This requires 3 calls to RecursiveMultiply.

Recurrence:

$$T(n) = \begin{cases} 3T(\frac{n}{2}) + \Theta(n) & \text{if } n > 1 \\ \Theta(1) & \text{if } n = 1. \end{cases}$$

Note that adding two n bit integers is $\Theta(n)$. So is multiplying any integer by 2^n, as this is simply left shifting the integer by n bits.

Complexity: $\Theta(n^{\log_2 3}) = \Theta(n^{1.59})$.

Solving Recurrences

- Substitution Method
- Recursion Tree Method
- Master Theorem

Master Theorem

Theorem (Master Theorem)

Let $T(n) = aT(\frac{n}{b}) + f(n)$ *be a recurrence relation where* $a \geq 1$ *and* $b > 1$ *are constants and* $f(n)$ *is an asymptotically positive function. Then* $T(n)$ *has the following asymptotic bounds:*

- *If* $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ *for some constant* $\epsilon > 0$*, then* $T(n) = \Theta(n^{\log_b a})$*.*
- *If* $f(n) = \Theta(n^{\log_b a})$*, then* $T(n) = \Theta(n^{\log_b a} \lg n)$*.*
- *If* $f(n) = \Omega(n^{\log_b a + \epsilon})$ *for some constant* $\epsilon > 0$*, and if* $af(\frac{n}{b}) \leq cf(n)$ *for some constant* $c < 1$ *and all sufficiently large* n*, then* $T(n) = \Theta(f(n))$*.*

Master Theorem

Examples

- $T(n) = T(\frac{n}{2}) + \Theta(1)$
  Here $a = 1, b = 2$, $f(n) = \Theta(1)$, and
  $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$.
  By Case 2 of Master Theorem, $T(n) = \Theta(\lg n)$.

- $T(n) = 2T(\frac{n}{2}) + \Theta(n)$
  Here $a = 2, b = 2$, $f(n) = \Theta(n)$, and
  $n^{\log_b a} = n^{\log_2 2} = n^1 = n$.
  By Case 2 of Master Theorem, $T(n) = \Theta(n \lg n)$.

Master Theorem

Examples

- $T(n) = 4T(\frac{n}{2}) + n$
  Here $a = 4, b = 2, f(n) = n$, and $n^{\log_b a} = n^{\log_2 4} = n^2$.
  Case 1 of Master Theorem applies with $\epsilon = 1$.
  Therefore, $T(n) = \Theta(n^2)$.
- $T(n) = 3T(\frac{n}{4}) + n \lg n$
  Here $a = 3, b = 4, f(n) = n \lg n$, and $n^{\log_4 3} = n^{0.793}$.
  Case 3 of Master Theorem applies with $\epsilon = 0.2$, since

  $$f(n) = n \lg n = \Omega(n^{0.993}) = \Omega(n^{0.793+0.2})$$

  Also,

  $$af\left(\frac{n}{b}\right) = 3\left(\frac{n}{4}\right) \lg \left(\frac{n}{4}\right) \leq \frac{3}{4} n \lg n = cf(n)$$

  for $c = \frac{3}{4}$ and sufficiently large n.
  Therefore, $T(n) = \Theta(n \lg n)$.

Master Theorem

Non-Example

- $T(n) = 2T(\frac{n}{2}) + \frac{n}{\lg n}$
  Here $a = 2, b = 2, f(n) = \frac{n}{\lg n}$, and $n^{\log_b a} = n^{\log_2 2} = n$.
  Observe that
  $$\frac{f(n)}{n^{\lg_b a}} = \frac{\frac{n}{\lg n}}{n} = \frac{1}{\lg n}$$
  is asymptotically greater than $n^{-\epsilon}$ for any $\epsilon > 0$, whereas we want to show that $\frac{f(n)}{n^{\log_b a}} = \mathcal{O}(n^{-\epsilon})$ for some $\epsilon > 0$.
  Thus, (Case 1 of) Master Theorem does not apply.

Exercise

Show that $T(n) = \Theta(n \lg \lg n)$ by using recursion tree method and the fact that $H_n = \Theta(\lg n)$, where H_n is the n^{th} Harmonic number.

$\sum_{k=1}^{n} \frac{1}{k}$

Theorem (Master Theorem)

Let $T(n) = aT(\frac{n}{b}) + f(n)$ be a recurrence relation where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function. Then $T(n)$ has the following asymptotic bounds:

- *If $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.*
- *If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.*
- *If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.*

$$T(n) = 2T(\frac{n}{2}) + \frac{n}{\lg n}$$

$$= 2\cdot(2\cdot T(\frac{n}{4})) + 2\cdot\frac{\frac{n}{2}}{\lg\frac{n}{2}} + \frac{n}{\lg n}$$

$$= 4\cdot T(\frac{n}{4}) + \frac{n}{\lg n} + \frac{n}{\lg\frac{n}{2}}$$

$$= 8\,T(\frac{n}{8}) + \frac{n}{\lg n} + \frac{n}{\lg\frac{n}{2}} + \frac{n}{\lg\frac{n}{4}}$$

$$= nT(1) + n(\frac{1}{\lg n} + \frac{1}{\lg\frac{n}{2}} + \frac{1}{\lg\frac{n}{4}} + \cdots + \frac{1}{\lg\frac{n}{2^{k-1}}})$$

$$\frac{1}{2}\sum_{i=1}^{\lg n}\frac{1}{i} \in \frac{1}{2}\Theta(\lg\lg n)$$

$$\Rightarrow T(n) \in \Theta(n\lg\lg n)$$

4.4-7

Draw the recursion tree for $T(n) = 4T(\lfloor n/2 \rfloor) + cn$, where c is a constant, and provide a tight asymptotic bound on its solution. Verify your bound by the substitution method.

n

$\frac{n}{2}$

$\frac{n}{4}$

$\vdots$

1

$\log(n)$ $\left\{ \begin{array}{l} cn \\ 2cn \\ 4cn \\ \vdots \end{array} \right.$

$cn(1+2+4+\dots n)$

$cn \cdot n(\underbrace{1+\frac{1}{2}+\frac{1}{4}+\dots}_{2})$

$2cn^2 = O(n^2)$

$= \Theta(n^2)$

$2^{\log n} cn = ncn$