Algorithms, Logic, and Proofs - Detailed Notes

1 Divide and Conquer: The Master Method

The Master Method is a powerful tool for analyzing the asymptotic running time of recursive algorithms. For an algorithm with the recurrence relation $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$, the Master Method gives $T(n) = \Theta\left(n^{\log_b a}\right)$.
Where $log_b(a) = log(a)/log(b)$

Example

Consider the recurrence relation $T(n) = 9 \cdot T\left(\frac{n}{3}\right) + n^2$. Here, $a = 9$, $b = 3$, and $f(n) = n^2$. The Master Method yields $T(n) = \Theta(n^2)$.

Example

Consider this algorithm

```
    FastPower(a,b) :
  if b = 1
    return a
  else
    c := a*a
    ans := FastPower(c,[b/2])
  if b is odd
    return a*ans
  else return ans
end
```

The algorithm has $log_2(b)$ recursive calls because it keeps recursively calling itself until b=1, while b/2.
The running time is $T(b) = log_2(b)$

2 Big O-Notation

Definition

The Big O-notation ($f(n) = O(g(n))$) denotes that there exist constants c and n_0 such that $0 < f(n) < c \cdot g(n)$ for all $n > n_0$.

Omega and Theta Notation

Ω represents a lower bound, and Θ provides a precise estimate. $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

3 proof by using Big-O

Proof by Contradiction: Goal: proof $2^{2n} \neq O(2^n)$ Assume, for the sake of contradiction, that $2^{2n} = O(2^n)$. This implies the existence of constants c and n_0 such that for all $n > n_0$, $0 \leq 2^{2n} \leq c \cdot 2^n$.
Now, let's simplify this assumption by dividing both sides by 2^n (assuming $2^n \neq 0$):

$$0 \leq 2^n \leq c$$

This implies that 2^n is bounded by a constant c for all $n > n_0$. However, as n approaches infinity, 2^n grows without bound, and no constant c can limit its growth.

Therefore, the assumption that $2^{2n} = O(2^n)$ leads to a contradiction. Hence, we conclude that 2^{2n} is not $O(2^n)$.

Proof two

To prove that $n^2 + n\log_2(n)$ is $O(n^2)$, we use the definition of big-O:

$$f(n) = n^2 + n\log_2(n), \quad g(n) = n^2$$

We want to show that there exist constants c and n_0 such that:

$$0 \leq n^2 + n\log_2(n) \leq c \cdot n^2 \quad \forall n \geq n_0$$

We divide both sides by n^2:
We get:

$$\frac{n^2 + n\log_2(n)}{n^2} \leq c$$

Then, after using the following rule:

$$\frac{a+b}{a} = \frac{a}{a} + \frac{b}{a} = 1 + \frac{b}{a}$$

We get:

$$1 + \frac{n\log_2(n)}{n^2} \leq c$$

Now, as n approaches infinity, the term $\frac{n\log_2(n)}{n^2}$ tends to zero. Therefore, the entire expression tends to approach $1 + 0$:
It can be expressed as follows:

$$\lim_{n\to\infty} \frac{n\log_2(n)}{n^2} = 0$$

We can now say that:

$$1 \leq c \quad n$$

For all sufficiently large n
Thus, we have shown that $n^2 + n\log_2(n)$ is $O(n^2)$.

Proof 3 Claim: $g(x) = x^2 - x$ is not $O(f(x))$ where $f(x) = x$.

Proof by Contradiction:

Assume, for the sake of contradiction, that $g(x) = O(f(x))$. This implies the existence of constants $c > 0$ and $x_0 > 0$ such that $0 \leq g(x) \leq cf(x)$ for all $x \geq x_0$.

Now, let's consider the expression for $g(x)$:

$$g(x) = x^2 - x$$

Assume there exist constants $c > 0$ and $x_0 > 0$ such that:

$$0 \leq x^2 - x \leq cx$$

Simplify the inequality:

$$0 \leq x(x-1) \leq cx$$

Now, consider the roots of the quadratic $x(x-1)$. The roots are $x = 0$ and $x = 1$. Let's analyze the behavior around these roots:

- For $0 \leq x < 1$, the expression $x(x-1)$ is non-negative, but cx is always non-negative. So, the inequality holds.

- For $x = 1$, the expression $x(x-1)$ becomes 0, and cx is also 0. So, the inequality holds.

- For $x > 1$, the expression $x(x-1)$ becomes positive, but cx grows without bound. Thus, the inequality does not hold.

This contradicts the assumption that $0 \leq x(x-1) \leq cx$ for all $x \geq x_0$. Therefore, our initial assumption that $g(x) = O(f(x))$ is false, and we conclude that $g(x)$ is not $O(f(x))$.

4 Running Time Analysis

Functions Analysis

Consider $T_a(n) = 2n^2 \log_2 n$ and $T_b(n) = n^3$. To determine the fastest running time, evaluate them for large n or use formal proofs.

Answer to Questions:

For $T_a(n)$:

$$T_a(n) = 2n^2 \log_2 n$$

For $T_b(n)$:

$$T_b(n) = n^3$$

To find which has the fastest running time, observe that the term n^3 will dominate $2n^2 \log_2 n$ for sufficiently large n. Therefore, $T_b(n)$ has the fastest running time.

Additional Proof by Formal Evaluation:

Let's consider $T_a(n) = 2n^2 \log_2 n$:

$$\lim_{n \to \infty} \frac{T_a(n)}{T_b(n)} = \lim_{n \to \infty} \frac{2n^2 \log_2 n}{n^3} = 0$$

This confirms that $T_b(n)$ has the fastest running time.

Determine a problem size $\sim n$ such that for all $n >\sim n$, the algorithm from your answer above is the fastest:

Given that $T_b(n)$ has the fastest running time, we can choose $\sim n$ such that n is sufficiently large to make $T_b(n)$ more efficient than $T_a(n)$.
Just set f(x) = g(x) and find the intersection

5 Proof by Contrapositive

Examples

Example 1

Prove the statement: *If k^3 is odd, then k is odd.* Use contrapositive: *If k is not odd, then k^3 is not odd.*

Proof:

Assume k is not odd ($k = 2m$ for some integer m). Then $k^3 = (2m)^3 = 8m^3 = 2(4m^3)$, which is even. Thus, if k^3 is not odd, then k is not odd.

Proof by Contrapositive:

Goal: Prove that if k is not odd, then k^3 is not odd.

Example Statement: If k^3 is odd, then k is odd.

Example 2

Prove that if r is irrational, then $r^{1/5}$ is irrational as well.

Proof:

Assume by contrapositive that $r^{1/5}$ is rational, then r is rational. Then:
$r^{1/5} = \frac{p}{q}$, where p and q are integers with no common factors (other than 1) and $q \neq 0$.

Raising both sides to the power of 5, we get $r = \left(\frac{p}{q}\right)^5$, which implies that r is rational. This contradicts our initial assumption that r is irrational. Therefore, if $r^{1/5}$ is rational, then r is rational.

6 Proof by Induction

Examples

Example 1

Prove by induction that the sum of the first n natural numbers, $p(n) = \frac{n(n+1)}{2}$, holds for all positive integers.

Proof:

Base case: For $n = 1$, $p(1) = \frac{1(1+1)}{2} = 1$, which holds.

Inductive step: Assume $p(k)$ is true for some arbitrary k. Now, we prove that $p(k + 1)$ is true.

$$\begin{aligned} p(k+1) &= \frac{(k+1)((k+1)+1)}{2} \\ &= \frac{(k+1)(k+2)}{2} \\ &= \frac{k(k+1)}{2} + (k+1) \end{aligned}$$

By the inductive assumption, $\frac{k(k+1)}{2}$ is the sum of the first k natural numbers. Therefore, $p(k + 1)$ is the sum of the first $k + 1$ natural numbers.

Example 2

Prove that $4 \mid (3^n + 6n - 1)$ for all positive integers n.

Proof:

Base case: For $n = 1$, $3^1 + 6 - 1 = 8$, which is divisible by 4.

Inductive step: Assume $4 \mid (3^k + 6k - 1)$ for some arbitrary k. Now, we prove that $4 \mid (3^{k+1} + 6(k + 1) - 1)$.

$$\begin{aligned} b_k &= 3^k + 6k - 1 \\ b_{k+1} &= 3^{k+1} + 6(k+1) - 1 = 3 \cdot 3^k + 6k + 5 \\ &\qquad = 3(b_k - 6k + 1) + 6k + 5 \\ &\qquad = 3b_k - 18k + 3 + 6k + 5 = 3b_k - 12k + 8 \end{aligned}$$

Example 4

Claim: For all positive integers n, the n-th term of the Fibonacci sequence (F_n) is given by $F_n = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}$, where ϕ is the golden ratio.

Base Cases:

- For $n = 1$, $F_1 = 1$, which satisfies the formula.
- For $n = 2$, $F_2 = 1$, which also satisfies the formula.

Inductive Hypothesis: Assume that the formula holds for k and $k + 1$, i.e., $F_k = \frac{\phi^k - (1-\phi)^k}{\sqrt{5}}$ and $F_{k+1} = \frac{\phi^{k+1} - (1-\phi)^{k+1}}{\sqrt{5}}$ for some positive integer k.

Inductive Step: We want to show that the formula holds for $k + 2$. Consider $F_{k+2} = F_{k+1} + F_k$. Substitute the inductive hypotheses, simplify, and factor out ϕ^{k+2}, yielding the formula.

By the principle of strong induction, the formula holds for all positive integers n.

7 Loop Invariant

A loop invariant is a condition that holds before and after each iteration of a loop, crucial for proving the correctness of an algorithm.

Example

Consider the pseudocode for the MUL algorithm. Prove its correctness using a loop invariant.

Loop Invariant: $x_n + by_n = a$

Proof:

Initialization: Before the loop, $x_0 = a$ and $y_0 = 0$. The loop invariant holds.

Maintenance: In each iteration, $x_{n+1} = x_n - b$ and $y_{n+1} = y_n + 1$. We need to show that $x_{n+1} + by_{n+1} = a$.

$$\begin{aligned} x_{n+1} + by_{n+1} &= (x_n - b) + b(y_n + 1) \\ &= x_n + by_n - b + b \\ &= x_n + by_n \end{aligned}$$

The loop invariant is maintained.

Termination: The loop terminates when $x = 0$. At this point, $x = 0$ and $y = \frac{a}{b}$, satisfying the loop invariant.

Algorithm to Find the Maximum Element:

```
max_element = A[0]
for i = 1 to n-1 do
    if A[i] > max_element then
        max_element = A[i]
```

Loop Invariant: At the start of each iteration of the loop, `max_element` is the maximum element among the first `i` elements of the array.

Proof 2:

1. *Initialization:* Before the loop starts, $i = 0$, and `max_element` is the maximum element among the first 0 elements, which is just the first element. The loop invariant holds.
2. *Maintenance:* Assume the loop invariant is true for some $i = k$, i.e., `max_element` is the maximum among the first k elements. In the next iteration ($i = k + 1$), if $A[k + 1]$ is greater than the current `max_element`, then `max_element` is updated to be $A[k + 1]$, which is the maximum among the first $k + 1$ elements. The loop invariant holds.
3. *Termination:* The loop terminates when $i = n - 1$, and `max_element` is the maximum among all elements, satisfying the loop invariant.

Algorithm to Compute Factorial:

```
result = 1
for i = 1 to n do
    result = result * i
```

Loop Invariant: At the start of each iteration of the loop, `result` is $i!$, where i is the loop variable.

Proof:

1. *Initialization:* Before the loop starts, $i = 1$, and `result` is 1, which is 1!. The loop invariant holds.
2. *Maintenance:* Assume the loop invariant is true for some $i = k$, i.e., `result` is $k!$. In the next iteration ($i = k + 1$), `result` is multiplied by $k + 1$, so it becomes $(k + 1)!$. The loop invariant holds.
3. *Termination:* The loop terminates when $i = n$, and `result` is $n!$, satisfying the loop invariant.

8 Proof by Contrapositive

Additional Example

Proof:

Proof by contrapositive: If r is irrational, then $r^{1/5}$ is irrational

Contrapositive expression:

Assume by contrapositive that $r^{1/5}$ is rational. Then $r^{1/5} = \frac{p}{q}$, where p and q are integers with no common factors (other than 1) and $q \neq 0$.

Raising both sides to the power of 5, we get $r = \left(\frac{p}{q}\right)^5$, which implies that r is rational. This contradicts our initial assumption that r is irrational. Therefore, if $r^{1/5}$ is rational, then r is rational.

The proof establishes that if r is irrational, then $r^{1/5}$ is also irrational, as the contrapositive of the statement holds.

Proof by Contrapositive example 2:

Goal: Prove that if k is not odd, then k^3 is not odd.

Example Statement: If k^3 is odd, then k is odd.

Assume k is not odd, i.e., k is even. Then, k can be expressed as $k = 2k_1$ for some integer k_1.

Now, consider k^3:

$$\begin{aligned} k^3 &= (2k_1)^3 \\ &= 8k_1^3 \\ &= 2(4k_1^3). \end{aligned}$$

Let $k_2 = 4k_1^3$. Then, $k^3 = 2k_2$, which implies k^3 is even.

Therefore, if k is not odd, then k^3 is not odd, proving the contrapositive.

As a consequence, the original statement is also true: If k^3 is odd, then k is odd.

Is this conversation helpful so far?

9 Truth Tables

Truth tables are a valuable tool for analyzing logical propositions and determining their validity.

Examples

Example 1

Construct a truth table for $p \wedge (q \vee \neg p)$.

Truth Table:

p	q	$\neg p$	$q \vee \neg p$	$p \wedge (q \vee \neg p)$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	F
F	F	T	T	F

Example 2

Determine the validity of the logical implication: $p \to (q \wedge \neg r)$.

Truth Table:

p	q	r	$\neg r$	$q \wedge \neg r$	$p \to (q \wedge \neg r)$
T	T	T	F	F	F
T	T	F	T	T	T
T	F	T	F	F	F
T	F	F	T	F	F
F	T	T	F	F	T
F	T	F	T	T	T
F	F	T	F	F	T
F	F	F	T	F	T

10 Combinatorics and Probability

In this section, we explore fundamental concepts in combinatorics and probability, specifically focusing on permutations and combinations. These concepts are crucial for solving problems involving arrangements and selections.

10.1 Permutations

Permutations involve arranging objects in a specific order. The formula for permutations is given by:

$$P(n, r) = \frac{n!}{(n-r)!}$$

where n is the total number of objects, r is the number of objects to be arranged, and ! denotes the factorial.

10.1.1 Example: Arranging Letters

Consider arranging the letters A, B, and C. The number of permutations is calculated as:

$$P(3, 3) = \frac{3!}{(3-3)!} = 6$$

The permutations are ABC, ACB, BAC, BCA, CAB, CBA.

10.2 Combinations

Combinations involve selecting objects without considering the order. The formula for combinations is given by:

$$C(n, r) = \frac{n!}{r!(n-r)!}$$

where n is the total number of objects, r is the number of objects to be selected, and ! denotes the factorial.

10.2.1 Example: Choosing Cards

Consider choosing 2 cards from a standard deck of 52 cards. The number of combinations is calculated as:

$$C(52, 2) = \frac{52!}{2!(52-2)!} = 1326$$

10.3 Probability

Probability measures the likelihood of an event occurring. For permutations and combinations, it is often useful to consider the probability of specific outcomes.

10.3.1 Birthday Paradox

In a group of n people, the probability of at least two sharing the same birthday is given by:

$$P(n) = 1 - \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \ldots \times \frac{365-n+1}{365}$$

10.3.2 Drawing Certain Cards

The probability of drawing specific cards (Ace of Hearts, King of Diamonds, 2 of Spades, 10 of Clubs) from a shuffled deck is given by:

$$P(\text{drawing specific cards}) = \frac{1}{(52)(51)(50)(49)}$$

These formulas and examples serve as a foundation for solving a wide range of problems in combinatorics and probability.