

Chapter 1

Algorithm Analysis

1.1 Logarithms

For $b > 1$ and $x > 0$, $L = \log_b x$ is a real number such that $b^L = x$.

Log has the following properties:

- If $x > y$ then $\log_b x > \log_b y$.
- If $\log_b x = \log_b y$ then $x = y$.
- $\log_b x^a = a \log_b x$.
- $\log_b (xy) = \log_b x + \log_b y$.
- $\log_c (x) = (\log_b x) / \log_b c$
- $x^{\log_b y} = y^{\log_b x}$
- We will use the notations: $\log_2 x \equiv \lg x$ for the binary log and $\log_e x \equiv \ln x$ for the natural log.

Exercise:

- (a) verify that $n \leq 2^{\lceil \lg n \rceil} \leq 2n$.
- (b) Establish bounds similar to (a) for $2^{\lfloor \lg n \rfloor}$.

(c) If the derivative of $\ln x$ is $\frac{1}{x}$, i.e. show that

$$\frac{d}{dx}(\lg x) = \frac{1}{x} \lg e.$$

1.2 Permutations

A permutation of n distinct objects is a sequence that contains each object once.

$$S = \{S_1, S_2, \dots, S_n\},$$

Permutation of S : $\{S_2, S_1, \dots, S_n\}$

$$\{S_n, S_1, S_2, \dots, S_{n-1}\}$$

Number of permutations is $n!$ (why?),

$S = \{2, 5, 9\}$, $3! = 6$ permutations: $\{2, 5, 9\}$, $\{2, 9, 5\}$, $\{5, 2, 9\}$, $\{5, 9, 2\}$, $\{9, 2, 5\}$ and $\{9, 5, 2\}$.

1.3 Sets and Relations

A set is a collection of distinct elements or numbers.

- $s_1 \in S$

Example:

$S_1 = \{u, v, w\}$, or $S_2 = \{x \mid x \text{ is a positive odd integer } < 10\}$, i.e.

S_2 consists of elements 1, 3, 5, 7, 9.

- $S_1 \subseteq S_2$

Example: $S_1 = \{a, b, c\}$ and $S_2 = \{a, b, c, d, e\}$.

- $S_1 \subset S_2$: S_1 is a proper subset of S_2 . Here S_1 and S_2 must be different.
- $S_1 = \emptyset$: an empty set.
- A set does not have an order, e.g. $\{a, b, c\}$ and $\{c, a, b\}$ are the same set, but different sequences.
- S : an n member set.

Number k element subset: $C(n, k)$.

Example: $S = \{a, b, c\}$ and $k = 2$, subsets: $\{a, b\}$, $\{a, c\}$, $\{b, c\}$.
 $C(3, 2) = 3$.

$$C(n, k) = \frac{n(n-1)\cdots(n-k+1)}{k!} = \frac{n!}{(n-k)!k!}$$

- **Cartesian Product** of two sets S_1 and S_2 is the set of pairs that is formed by choosing one element from S_1 and one element from S_2 , i.e.

$$S_1 \times S_2 = \{(x, y) \mid x \in S_1, y \in S_2\}$$

Example: $S_1 = \{a, b, c\}$, $S_2 = \{1, 2\}$

$$S_1 \times S_2 = \{(a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)\}$$

if S_1 and S_2 have n_1 and n_2 members, number of elements in the Cartesian product is $n_1 n_2$.

- A **relation** is some subset of the Cartesian product.

Example: Let $S_1 = S_2 = S$ where S denotes the set of real numbers, i.e. 1, 2, 3, ... Then the relation “greater than” can be defined as

$$\{(x, y) \mid x \in S, y \in S, x > y\}.$$

Example: Let M (men) and W (women) $M = \{\text{John, Adam, Robert}\}$ and $W = \{\text{Nancy, Mary, Suzy}\}$. Then the relation “married to” might be

$$\{(x, y) \mid x \in M, y \in W\} = \{(\text{John, Mary}), (\text{Adam, Suzy}), (\text{Robert, Mary})\}$$

- **Equivalence Relation R** is one that satisfies the following properties:

- **Reflexive:** for all $x \in S$, $(x, x) \in R$.

- **Symmetric:** if $(x, y) \in R$, then $(y, x) \in R$.

- **Transitive:** if $(x, y) \in R$ and $(y, z) \in R$ then $(x, z) \in R$.

Exercise: which of the following are equivalent relations, why?

- “Less than” ($<$) if S is the set of real numbers.
- “Married to” if S is the set of people.
- “Equal to” ($=$) if S is the set of real numbers.
- “Divisible” if S is the set of even integers.

The equivalent class of an element $x \in S$ is a subset of S ($S_1 \in S$) that contains all the elements that are related to x .

1.4 Proofs

(a) Proof by induction

It consists of two parts – basis case and induction.

Basis: Show that the proposition to be proved is valid for a few initial values.

Induction: Assume that the proposition is valid for $k-1$ steps starting from the initial values, show that it is valid for k .

Example: Show that $\sum_{i=1}^n i = \frac{n(n+1)}{2}$ for all positive n .

Basis: $n = 1$, $\sum_{i=1}^1 i = 1$, and $\frac{n(n+1)}{2} = \frac{1 \times 2}{2} = 1$.

Induction: Assume $\sum_{i=1}^{k-1} i = \frac{(k-1)(k)}{2}$ is true for any $k < n$,

show $\sum_{i=1}^k i = \frac{k(k+1)}{2}$.

$$\sum_{i=1}^k i = \left(\sum_{i=1}^{k-1} i \right) + k = \frac{(k-1)(k)}{2} + k = \frac{k(k+1)}{2}$$

Exercise: Prove by induction that:

(a) $\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$ for $n \geq 1$.

(b) $\sum_{i=1}^{n-2} F_i = F_n - 2$ where F_i is the set of Fibonacci numbers, i.e. $F_{i+1} = F_i + F_{i-1}$, with $F_0 = F_1 = 1$.

(c) Show that the Fibonacci numbers satisfy the following inequality:

$$F_{i+1} \leq (5/3)^{i+1}$$

(b) Proof by Contradiction

Assume that the proposition is not true, then show this will lead to a contradiction.

Example : Show that the in a tree $n = b+1$, where n is the number of nodes and b is the number of branches.

Proof: Suppose the above is not true in which case either (i) $n > b+1$ or (ii) $n < b+1$. If the first case (i), there must be more than one node without incoming branch. Since in a tree only the root has no incoming branch, then this is a contradiction. In the second case (ii), there either root must have one incoming branch to it, or some other nodes must have more than one incoming branches. In this case there must be loops (cycles). This is a contradiction, since a tree does not have a cycle.

(c) Disprove by Counter-Example

Give one counterexample to disprove a proposition.

Example: Prove or disprove the statement “The ratio of any two integers is another integer”

The statement is wrong since 5 and 3 are two integers but their ratio $5/3$ is not an integer.

Important Note: You cannot give one (or more, even many) example(s) to prove a proposition.

For example, you cannot prove the above statement “The ratio of any two integers is another integer” by giving a pair of integers such as 6 and 3 even though the ratio $6/3$ is an integer.

1.5 Series and Summations

- $\sum_{i=1}^n i = \frac{n(n+1)}{2}$ for any integer $n \geq 1$
- $\sum_{i=0}^n 2^i = 2^{n+1} - 1$ for any integer $n \geq 1$
- $\sum_{i=1}^n i 2^i = (n-1)2^{n+1} + 2$ for any integer $n \geq 1$
- $\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1}$ for any integer $n \geq 1$ (prove this by long division)
- $\sum_{i=0}^n \frac{1}{2^i} = 2 - \frac{1}{2^n}$ for any integer $n \geq 1$ (set $a = \frac{1}{2}$ in above summation)
- $\sum_{i=0}^n a^i \leq \frac{1}{1-a}$ for $0 < a < 1$ and any integer $n \geq 1$
- $\sum_{i=0}^{\infty} a^i = \frac{1}{1-a}$ for $0 < a < 1$

To prove the last summation:

$$S = \sum_{i=0}^{\infty} a^i = 1 + a + a^2 + \dots$$

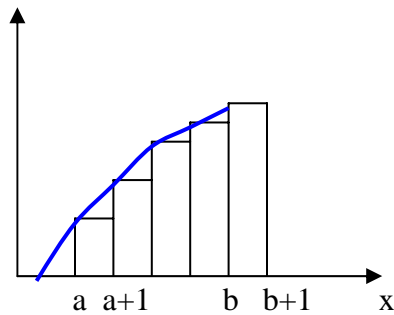
$$aS = a + a^2 + a^3 + \dots$$

$$S - aS = 1, \quad S = \frac{1}{1-a}.$$

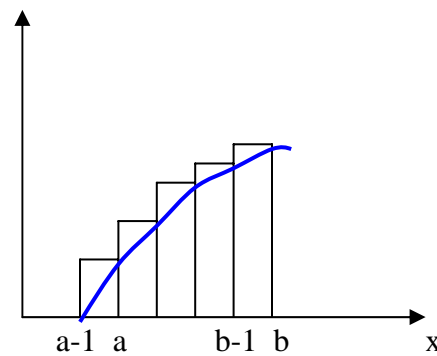
1.5.1 Summation using Integration

- $f(x)$ is a non-decreasing if $x \leq y$ implies $f(x) \leq f(y)$. Similarly $f(x)$ is non-increasing if $x \leq y$ implies $f(x) \geq f(y)$.

- If $f(x)$ is non-decreasing then
$$\int_{a-1}^b f(x) dx \leq \sum_{i=a}^b f(i) \leq \int_a^{b+1} f(x) dx.$$



$$\sum_{i=a}^b f(i) \leq \int_a^{b+1} f(x) dx$$



$$\int_{a-1}^b f(x) dx \leq \sum_{i=a}^b f(i)$$

- If $f(x)$ is non-increasing then
$$\int_a^{b+1} f(x) dx \leq \sum_{i=a}^b f(i) \leq \int_{a-1}^b f(x) dx.$$

- Few useful integration formulas

$$\int_0^n x^k dx = \frac{1}{k+1} n^{k+1} ; \quad \int_0^n e^{ax} dx = \frac{1}{a} (e^{an} - 1)$$

$$\int_0^n x^k \ln x dx = \frac{1}{k+1} n^{k+1} \ln x - \frac{1}{(k+1)^2} n^{k+1}$$

Example- Lower bound for $\sum_{i=1}^n \lg i$

$$\sum_{i=1}^n \lg i = 0 + \sum_{i=2}^n \lg i \geq \int_1^n \lg x \, dx.$$

$$\int_1^n \lg x \, dx = \int_1^n \lg e \ln x \, dx = \lg e (n \ln n - n + 1) = n \lg n - \lg e n - \lg e$$

Hence

$$\sum_{i=1}^n \lg i \geq n \lg n + 1.433(n - 1), \text{ where } \lg e = 1.433$$

- It can be shown that

$$\left(\frac{n}{e}\right)^n \sqrt{2\pi n} < n! < \left(\frac{n}{e}\right)^n \sqrt{2\pi n} \left(1 + \frac{1}{11n}\right)$$

1.6 Algorithm Analysis

Goal : To improve algorithm efficiency or to choose the most efficient algorithm among several algorithms.

Algorithm/program efficiency is a measure of :

- Amount of work needed by the computer: “**complexity** of the algorithm”.
- Amount of computer space needed
- Simplicity

- Optimality

Amount of work or complexity:

- Must be independent of particular computer used
- Must reflect execution time
- Must reflect the amount of basic or fundamental operations such as comparisons, multiplication, etc.
- Must be independent of implementation details such as initialization of a variable
- Must reflect the size of the input data (n)

We will express the algorithm complexity as a function of input size, i.e. $C(n)$.

Example: Find x in array A of size n .
Traversing an n node.

Amount of Space:

- The number of bytes of memory needed for the program is the amount of space.
- Some programs require making copies of data during execution, and thus need more space.
- In modern computers space is not a critical factor for most programs.

Simplicity:

- A simple and straight forward algorithm is not usually the most efficient one.
- It is desirable to have a simple algorithm, but efficiency is usually the deciding factor.

Optimality:

- A program is optimal for a given problem if there is no other algorithm that performs fewer basic operations.
- Each problem has an inherent least complexity, i.e. it needs some minimum amount of work to solve it.

1.61. Worst Case and average Case Complexity

- The **worst case complexity** $C_w(n)$ is the maximum number of basic operations performed on any input of size n .

Example: Sequential search in an unordered array A of size n for a key K in the array.

```
int SeqSearch (A, n, K)
{
    int i =0;
    while (A[i] !=K )
        i++;
    return A[i];
}
```

Worst case: K last array position, needs n comparisons,
 $C_w(n) = n$.

Example:

```

int summation(n)
{
    int sum = 0, i, j;
    for( i=0; i<n; i++)
        for(j=0; j<n; j++)
            if(j % i == 0)
                sum ++;
    return sum;
}

```

Basic operation: addition.

Worst case complexity: j is divisible by i every time the “if condition” is tested. For loops iterate n^2 times, $C_w(n) = n^2$.

- **Average Case Complexity** $C_a(n)$: Let E be the set of inputs of size n , and let e be a member of E . Also let $\text{Pr}(e)$ be the probability that the input e occurs and $C_e(n)$ be the number of basic operations performed by the algorithm on input e .

$$C_a(n) = \sum_{e \in E} \text{Pr}(e) C_e(n)$$

Example: Sequential Search

Assume K has the same probability of being in locations $0, 1, 2, \dots, n-1$ in the array, i.e. $\text{Pr}(e) = \frac{1}{n}$.

Number of basic operations (comparisons): 1 if $A[0] = K$, is 2 if $A[1] = K$, and is $(i+1)$ if $A[i] = K$. Thus

$$C_a(n) = \sum_{i=0}^{n-1} \frac{1}{n}(i+1) = \frac{1}{n} \frac{n(n+1)}{2} = \frac{n+1}{2}$$

Example - Binary Search

Given a key K , and a A such that $A[i] < A[i+1]$ for $i=0,1,2, \dots, n-2$, find i such that $A[i] = K$, or return $i = -1$ if K not in array.

```
int BinarySearch(int A[], int n, int K)
{
    int low=0, high = n-1, mid;
    while(low <= high)
    {
        mid = (low + high)/2;
        if( A[mid] < K)
            low = mid + 1;
        else if (A[mid] > K)
            high = mid -1;
        else
            return mid;
    }
    return -1;
}
```

Basic operation is comparison.

Worst case: the maximum number of times around the loop. The loop starts with $(high - low) = n-1$ and finishes when $(high - low) = -1$. Every time through the loop, $(high - low)$ must be at least halved,

$$C_w(n) = \lceil \lg(n-1) \rceil + 2$$

1.6.2 Asymptotic Growth and Order Notation

Suppose two algorithms solve the same problem with $C_1(n) = 100n$ and $C_2(n) = 2n^2$.

The rate of growth of $C_2(n) = 2n^2 \gg C_1(n) = 100n$ for large n (i.e. asymptotic growth)

The order notation: we say $C_1(n) = 100n$ is of the order n or $C_1(n)$ is $O(n)$, or $C_1(n) \in O(n)$, and $C_2(n)$ is $O(n^2)$.

$C(n)$ is $O(f(n))$ if $C(n)$ is at most a constant times $f(n)$ for large n .

Definition: $C(n)$ is $O(f(n))$ if there exist an integer $n_0 > 1$ and a constant $k > 0$ such that $C(n) \leq kf(n)$ for all $n \geq n_0$.

Example: Suppose $C(n) = (n + 1)^2$.

Choose $k = 4$, and $n_0 = 1$. $C(n) \leq 4n^2$ for all $n \geq 1$, then $C(n)$ is $O(n^2)$. Here $f(n) = n^2$, growth is quadratic.

- k and n_0 are not unique

Let $k = 2$, and $n_0 = 3$

$$C(n) = (n + 1)^2 \leq 2n^2 \text{ for all } n \geq 3.$$

$C(n)$ is $O(n^2)$.

- Note: $C(n) = (n + 1)^2$ is not $O(n)$ - there are no k and n_0 $C(n) = (n + 1)^2 \leq kf(n)$ for all $n \geq n_0$.

- When $C(n) = \text{constant}$, we say $C(n)$ is $O(1)$.

General rules for the order notation:

- **Constant factors don't matter:** if $C(n)$ is $O(f(n))$, then $C(n)$ is also $O(a f(n))$.

$$C(n) \leq k f(n) \text{ for all } n \geq n_0$$

or $C(n) \leq \frac{k}{a} \times a f(n) = k_1 \times a f(n)$ for all $n \geq n_0$. Thus $C(n)$ is $O(a f(n))$.

- **Low order terms don't matter:** Let

$$C(n) = a_p n^p + a_{p-1} n^{p-1} + \dots + a_1 n + a_0$$

Then $C(n)$ is $O(n^p)$ because

$$\begin{aligned} C(n) &\leq a_p n^p + a_{p-1} n^p + \dots + a_1 n^p + a_0 n^p \\ &\leq (a_p + a_{p-1} + \dots + a_1 + a_0) n^p \\ &\leq k n^p \text{ for all } n \geq 1 \end{aligned}$$

- **Only dominant terms matter:**

If $C(n) = C_1(n) + C_2(n)$ and $\lim_{n \rightarrow \infty} \frac{C_2(n)}{C_1(n)} = 0$, then

$C(n)$ is $O(C_1(n))$

Example: $C(n) = 2n + 10 \lg n$

then $C(n)$ is $O(n)$.

- **Transitive law holds:** If $C(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$ then $C(n)$ is $O(g(n))$.

$C(n) \leq k_1 f(n)$ for all $n \geq n_1$ since $C(n)$ is $O(f(n))$

$C(n) \leq k_2 g(n)$ for all $n \geq n_2$ since $C(n)$ is $O(g(n))$

$C(n) \leq k_2 g(n)$ for all $n \geq \max(n_1, n_2)$.

- **The Big Omega :** Big O provides an upper bound -the actual complexity can be better than that provided by the big O notation. Big omega gives a lower bound
- **Definition:** $C(n)$ is $\Omega(g(n))$ if there exist a positive constant k and an integer $n \geq n_0$ such that $C(n) \geq k g(n)$ for all $n \geq n_0$.
- **The Big Theta Θ :** The bounds provided by Ω and O are not necessarily tight. For example, if $C(n) = (n + 1)^2$ we can say that $C(n)$ is $O(n^2)$, or $O(n^3)$, $O(n^2 \lg n)$, $O(n^5)$, etc.

Similarly $C(n) = (n + 1)^2$ is $\Omega(n)$, or $\Omega(\lg n)$ or even $\Omega(n^{0.2})$, but none is a tight bound.

Definition: $C(n)$ is $\Theta(f(n))$ if $C(n)$ is both $O(f(n))$ and $\Omega(f(n))$.

Example: $C(n) = 3n^2 + \lg n$

$$C(n) \leq 4n^2 \text{ for all } n \geq 1$$

$$C(n) \geq 2n^2 \text{ for all } n \geq 1$$

$C(n)$ is both $O(n^2)$ and $\Omega(n^2)$, and thus it is $\Theta(n^2)$.

Example: Show $C(n) = \sum_{i=1}^n (i)^b$ is $\Theta(n^{b+1})$, where $b > 1$ is a constant integer.

$$\begin{aligned} C(n) &= 1^b + 2^b + 3^b + \dots + n^b \\ &\leq n^b + n^b + n^b + \dots + n^b = n^b \underbrace{(1 + 1 + 1 + \dots + 1)}_{n \text{ times}} = n^{b+1} \end{aligned}$$

Thus $C(n)$ is $O(n^{b+1})$.

$$\text{Also } C(n) = 1^b + 2^b + 3^b + \dots + \left(\frac{n}{2}\right)^b + \left(\frac{n+1}{2}\right)^b + \dots + n^b$$

$$C(n) \geq \left(\frac{n}{2}\right)^b + \left(\frac{n}{2}\right)^b + \left(\frac{n}{2}\right)^b + \dots + \left(\frac{n}{2}\right)^b + \left(\frac{n}{2}\right)^b \quad \text{there are } \frac{n}{2} \text{ terms}$$

$$\geq \left(\frac{n}{2}\right)^b \times \left(\frac{n}{2}\right) = \left(\frac{1}{2}\right)^{b+1} (n)^{b+1} \quad \text{Note: } \left(\frac{1}{2}\right)^{b+1} \text{ is a constant}$$

Therefore $C(n)$ is $\Omega(n^{b+1})$ and hence it is $\Theta(n^{b+1})$.

Exercise: Show that $C(n) = \sum_{i=1}^n \lg i$ is $\Theta(n \lg n)$