Mathematical Foundation

CSE 6331 Algorithms

Steve Lai

Complexity of Algorithms

- Analysis of algorithm: to predict the running time required by an algorithm.
- Elementary operations:
  - arithmetic & boolean operations: +, –, ×, /, mod, div, and, or
  - comparison: if $a < b$, if $a = b$, etc.
  - branching: go to
  - assignment: $a \leftarrow b$
  - and so on

- The running time of an algorithm is the number of elementary operations required for the algorithm.
- It depends on the size of the input and the data themselves.
- The worst-case time complexity of an algorithm is a function of the input size n:

  $T(n)$ = the worst case running time over all instances of size n.
- The worst-case asymptotic time complexity is the worst case time complexity expressed in O, Ω, or Θ.
- The word asymptotic is often omitted.

O-Notation

Note: Unless otherwise stated, all functions considered in this class are assumed to be nonnegative.

- Conventional Definition: We say $f(n) = O(g(n))$ or $f(n)$ *is* $O(g(n))$ if there exist positive constants c and n_0 such that $f(n) \le cg(n)$ for all $n \ge n_0$.
- More Abstract Definition:

$$O(g(n)) = \left\{ \begin{array}{r} f(n):\ f(n) = O(g(n)) \text{ in the} \\ \text{conventional meaning} \end{array} \right\},$$

i.e., the set of all functions that *are* $O(g(n))$ in the conventional meaning.

- These expressions all mean the same thing:
  - $4n^2 + 3n = O(n^2)$
  - $4n^2 + 3n$ is $O(n^2)$
  - $4n^2 + 3n \in O(n^2)$
  - $4n^2 + 3n$ is in $O(n^2)$.
- Sometimes $O(g(n))$ is used to mean some function in the set $O(g(n))$ which we don't care to specify.
- Example: we may write:

  Let $f(n) = 3n^2 + O(\log n)$.

Theorem 1

If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then

1. $f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$
2. $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$
3. $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$

Proof. There exist positive constants c_1, c_2 such that $f_1(n) \le c_1 g_1(n)$ and $f_2(n) \le c_2 g_2(n)$ for sufficiently large n.

Thus,

$$\begin{aligned} f_1(n) + f_2(n) &\le c_1 g_1(n) + c_2 g_2(n) \\ &\le (c_1 + c_2)(g_1(n) + g_2(n)) \\ &\le 2(c_1 + c_2)\max(g_1(n), g_2(n)). \end{aligned}$$

By definition, 1 and 2 hold. 3 can be proved similarly.

Ω-Notation

- Conventional Definition: We say $f(n) = \Omega(g(n))$ if there exist positive constants c and n_0 such that $f(n) \geq cg(n)$ for all $n \geq n_0$.

- Or define $\Omega(g(n))$ as a set:

$$\Omega(g(n)) = \left\{ \begin{array}{r} f(n):\ f(n) = \Omega(g(n)) \text{ in the} \\ \text{conventional meaning} \end{array} \right\}$$

- Theorem 2: If $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n))$, then
  1. $f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n))$
  2. $f_1(n) + f_2(n) = \Omega(\max(g_1(n), g_2(n)))$
  3. $f_1(n) \cdot f_2(n) = \Omega(g_1(n) \cdot g_2(n))$

Θ-Notation

- Conventional Definition: We say $f(n) = \Theta(g(n))$ if there exist positive constants c_1, c_2, and n_0 such that $c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0$. That is, $f(n) = \Theta(g(n)) \equiv \left[f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n)) \right]$.

- In terms of sets: $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

- Theorem 3: If $f_1(n) = \Theta(g_1(n))$ and $f_2(n) = \Theta(g_2(n))$, then
  1. $f_1(n) + f_2(n) = \Theta(g_1(n) + g_2(n))$
  2. $f_1(n) + f_2(n) = \Theta(\max(g_1(n), g_2(n)))$
  3. $f_1(n) \cdot f_2(n) = \Theta(g_1(n) \cdot g_2(n))$

o-Notation, ω-Notation

- Definition of o

$$f(n) = o(g(n)) \text{ iff } \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

- Definition of ω

$$f(n) = \omega(g(n)) \text{ iff } \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty.$$

Some Properties of Asymptotic Notation

- Transitive property: If $f(n) = O(g(n))$ and $g(n) = O(h(n))$,

  then $f(n) = O(h(n))$.
- The transitive property also holds for Ω and Θ.
- $f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$.
- See the textbook for many others.

Asymptotic Notation with Multiple Parameters

- Definition: We say that $f(m, n) = O(g(m, n))$ iff

  there are positive constants c, m_0, n_0 such that

  $f(m, n) \le cg(m, n)$ for all $m \ge m_0$ and $n \ge n_0$.

- Again, we can define $O(g(m, n))$ as a set.

- $\Omega(g(m, n))$ and $\Theta(g(m, n))$ can be similarly defined.

Conditional Asymptotic Notation

- Let $P(n)$ be a predicate. We write $T(n) = O(f(n) \mid P(n))$ iff there exist positive constants $c,\ n_0$ such that $T(n) \le cf(n)$ for all $n \ge n_0$ for which $P(n)$ is true.
- Can similarly define $\Omega(f(n) \mid P(n))$ and $\Theta(f(n) \mid P(n))$.
- Example: Suppose for $n \ge 0$,

$$T(n) = \begin{cases} 4n^2 + 2n & \text{if } n \text{ is even} \\ 3n & \text{if } n \text{ is odd} \end{cases}$$

Then, $T(n) = \Theta(n^2 \mid n \text{ is even})$

Smooth Functions

- A function $f(n)$ is **smooth** iff $f(n)$ is asymptotically nondecreasing and $f(2n) = O(f(n))$.
- Thus, a smooth function does not grow very fast.
- Example: $\log n,\ n\log n,\ n^2$ are all smooth.
- What about 2^n?

- Theorem 4. If $f(n)$ is smooth, then $f(bn) = O(f(n))$ for any fixed positive integer b.

Proof. By induction on b.

Induction base: For $b = 1,\ 2,$ obviously $f(bn) = O(f(n))$.

Induction hypothesis: Assume $f\left((b-1)n\right) = O(f(n))$, where $b > 2$.

Induction step: Need to show $f(bn) = O(f(n))$. We have:

$$f(bn) \le f\left(2(b-1)n\right) = O\left(f\left((b-1)n\right)\right) \subseteq O(f(n))$$

(i.e., $f(bn) \le f\left(2(b-1)n\right) \le c_1 f\left((b-1)n\right) \le c_1 c_2 f(n)$ for some constants $c_1,\ c_2$ and sufficiently large n).

The theorem is proved.

- Theorem 5. If $T(n) = O(f(n) \mid n \text{ a power of } b)$, where $b \geq 2$ is a constant, $T(n)$ is asymptotically nondecreasing and $f(n)$ is smooth, then $T(n) = O(f(n))$.

Proof. From the given conditions, we know:

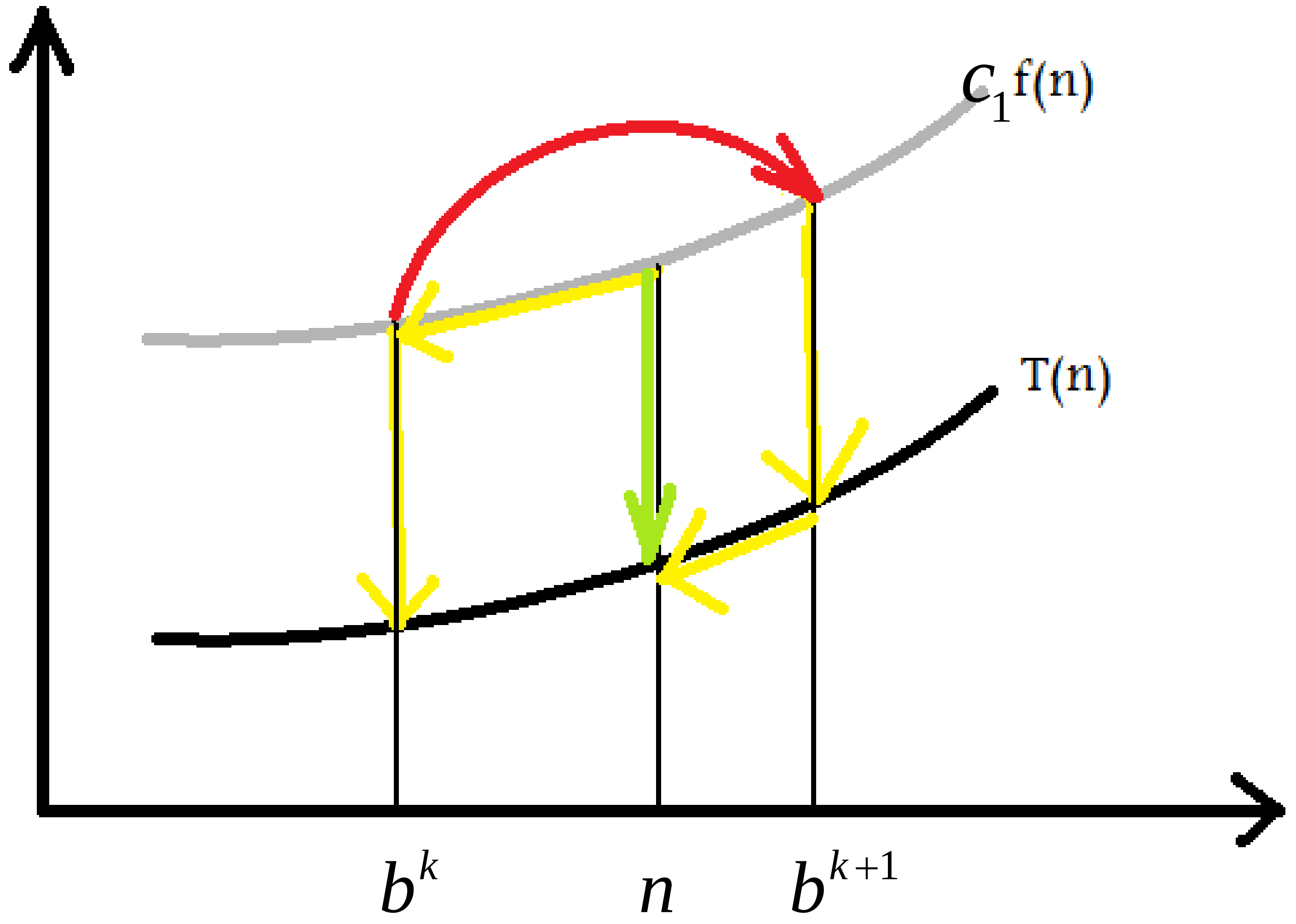
1. $T(n)$ is asymptotically nondecreasing.
2. $T(n) \leq c_1 f(n)$ for n sufficiently large and a power of b.
3. $f(bn) \leq c_2 f(n)$ for sufficiently large n.

For any n, there is a k such that $b^k \leq n < b^{k+1}$.

When n is sufficiently large, we have

$$T(n) \leq T(b^{k+1}) \leq c_1 f(b^{k+1}) \leq c_1 c_2 f(b^k) \leq c_1 c_2 f(n).$$

Be definition, $T(n) = O(f(n))$.



$$T(n) \le T(b^{k+1}) \le c_1 f(b^{k+1}) \le c_1 c_2 f(b^k) \le c_1 c_2 f(n).$$

- Theorem 6. If $T(n) = \Omega(f(n) \mid n \text{ a power of } b)$, where $b \geq 2$ is a constant, $T(n)$ is asymptotically nondecreasing and $f(n)$ is smooth, then $T(n) = \Omega(f(n))$.
- Theorem 7. If $T(n) = \Theta(f(n) \mid n \text{ a power of } b)$, where $b \geq 2$ is a constant, $T(n)$ is asymptotically nondecreasing and $f(n)$ is smooth, then $T(n) = \Theta(f(n))$.
- Application. In order to show $T(n) = O(n \log n)$, we only have to establish $T(n) \leq O(n \log n \mid n \text{ a power of } 2)$, provided that $T(n)$ is asymptotically nondecreasing.

Some Notations, Functions, Formulas

- $\lfloor x \rfloor$ = the floor of x.
- $\lceil x \rceil$ = the ceiling of x.
- $\log n = \log_2 n$. (Or $\lg n$)
- $1 + 2 + \cdots + n = n(n+1)/2 = \Theta(n^2)$.
- For constants $k > 0$, $1 + 2^k + 3^k + \cdots + n^k = \Theta(n^{k+1})$.
- If $a \neq 1$, then $1 + a + a^2 + \cdots + a^n = \dfrac{1 - a^{n+1}}{1 - a} = \dfrac{a^{n+1} - 1}{a - 1}$.
- If $a > 1$, then $f(n) = 1 + a + a^2 + \cdots + a^n = \Theta(a^n)$.
- If $a < 1$, then $f(n) = 1 + a + a^2 + \cdots + a^n = \Theta(1)$.

Approximating summations by integration

- Suppose function f is increasing or decreasing.
- $\int_m^n f(x)dx \le \sum_{i=m}^{n} f(i) \le \int_m^n f(x)dx + \begin{cases} f(n) & f \text{ increasing} \\ f(m) & f \text{ dereasing} \end{cases}$
- So, if f is increasing and $\int_m^n f(x)dx = \Omega(f(n))$, then

$$\sum_{i=m}^{n} f(i) = \Theta\left(\int_m^n f(x)dx\right)$$

- Similarly, if f is decreasing and $\int_m^n f(x)dx = \Omega(f(m))$, then

$$\sum_{i=m}^{n} f(i) = \Theta\left(\int_m^n f(x)dx\right)$$

- Example: $\sum_{i=m}^{n} \frac{1}{i} = \Theta\left(\int_m^n \frac{1}{x} dx\right) = \Theta(\ln n - \ln m) = \Theta(\lg n - \lg m)$

Analysis of Algorithm: Example

Procedure BinarySearch(A, x, i, j)

if $i > j$ then return(0)

$m \leftarrow \lfloor (i+j)/2 \rfloor$

case

$A[m] = x$: return(m)

$A[m] < x$: return$\left(\text{BinarySearch}(A,\ x,\ m+1,\ j)\right)$

$A[m] > x$: return$\left(\text{BinarySearch}(A,\ x,\ i,\ m-1)\right)$

end

Analysis of Binary Search

Let $T(n)$ denote the worst-case running time.

$T(n)$ satisfies the recurrence:

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + c.$$

Solving the recurrence yields:

$$T(n) = \Theta(\log n)$$

Euclid's Algorithm

- Find $\gcd(a,b)$ for integers $a, b \geq 0$, not both zero.
- Theorem: If $b = 0,\ \gcd(a,b) = a$.

  If $b > 0,\ \gcd(a,b) = \gcd(b, a \bmod b)$
- function Euclid(a,b)

  if $b = 0$

  then return(a)

  else return$\left(\text{Euclid}(b, a \bmod b)\right)$
- The running time is proportional to the number of recursive calls.

Analysis of Euclid's Algorithm

$$\begin{array}{lll} a_0 & b_0 & c_0 = a_0 \bmod b_0 \\ a_1 & b_1 & c_1 = a_1 \bmod b_1 \\ a_2 & b_2 & c_2 \\ \vdots & \vdots & \vdots \\ a_n & b_n & \end{array}$$

- Observe that $a_k = b_{k-1} = c_{k-2}$.
- W.l.o.g., assume $a_0 \geq b_0$. The values $a_0,\ a_2,\ a_4,\ \ldots$ decrease by at least one half with each recursive call.
- Reason: If $c := a \mod b$, then $c < a/2$.
- So, there are most $\mathrm{O}(\log a_0)$ recursive calls.

Solution to Q4 of example analysis

$$\sum_{i=2^0,\ 2^1,\ 2^2,\ \ldots,\ 2^{\log n^2}} i^2$$

$$= \sum_{k=0}^{\log n^2} \left(2^k\right)^2 = \sum_{k=0}^{\log n^2} \left(2^2\right)^k = \Theta\left(\left(2^2\right)^{\log n^2}\right)$$

$$= \Theta\left(\left(2^{\log n^2}\right)^2\right) = \Theta\left(n^4\right)$$