CS303 Algorithms

1 Introduction

Reading: MAW: 1.1-1.3

1.1 Opening remarks

• Algorithm:
  – An informal definition of a program (step-by-step procedure).
  – Output, correctness, termination, unambiguity, and effectiveness.
  – Written in pseudocode. (Human language + math language + computer language).
  – Running time/efficiency/time complexity.

• Why are algorithm design and analysis important:
  The selection problem: Find the \( k \)th largest among \( n \) numbers.
  Algorithm 1 takes several days to find 500,000th largest among 1,000,000 numbers.
  Algorithm 2 takes about one second for the same input.

• Ways to achieve high algorithmic efficiency:
  – By data structures/implementations: For example, to insert an item after a known item in a list of size \( n \), it takes \( n \) steps in the worst case if the list is maintained in a linear array and only 1 step if the list is maintained in a linked list.
  – By algorithm design techniques: For example, to search a member in a sorted array of size \( n \), it takes \( n \) steps in the worst case if one uses linear search and only \( \log n \) steps if binary search is used.

1.2 Math review

• \([x]\) and \([x]\).

• Polynomials: \( p(x) = \sum_{i=0}^{n} a_i x^i \). (Note: Coefficients \( a_i \) and degree \( n \) are constants.)

• Exponents: \( a^0 = 1, a^{-1} = \frac{1}{a}, a^m \cdot a^n = a^{m+n}, a^m / a^n = a^{m-n} \).

• Logarithms: \( \log_a b = x \iff a^x = b \).

\[
\log(ab) = \log a + \log b, \quad \log\left(\frac{a}{b}\right) = \log a - \log b.
\]

\[
\log_a b = \frac{\log_c b}{\log_c a}, \quad \log_a b^n = n \log_a b \neq (\log_a b)^n, \quad a^{\log_a n} = n, \quad a^{\log_c b} = b^{\log_c a}.
\]

\[
\log n = \log_2 n \text{ or } \log_c n \text{ for some } c \text{ we don’t care about.}
\]

• Series/Sums:
  Arithmetic: \( 1 + 2 + \cdots + n = \frac{1}{2} n(n + 1) \).
  Geometric: \( 1 + r + r^2 + \cdots + r^n = \frac{r^{n+1} - 1}{r - 1} \text{ for } r \neq 1 \).

• Proof techniques:
  Proof by induction: \( P(n) \) for \( n \geq c \) follows from
  1. Inductive base: \( P(c) \),
  2. Inductive hypothesi: Assume \( P(i-1) \) or \( P(j) \) for \( c < j < i \), and
  3. Inductive step: \( P(i) \).
Proof by contradiction: The following three statements are equivalent:

- If \( A \) then \( B \),
- If not \( B \) then not \( A \), and
- If not \( B \) and \( A \) then not \( C \), where \( C \) is a proven fact or an axiom.

Examples/Exercises:
1. \( 1 + 2 + 3 + \cdots + n = \frac{1}{2}n(n+1) \).
2. There is an infinite number of primes.

1.3 Recursive functions

- A function defined in terms of itself. The definition includes base cases + general cases (making progress).

For example, factorial \( f(n) = n! = 1 \times 2 \times \cdots \times n \) can be defined recursively as follows:

\[
f(n) = \begin{cases} 
1 & \text{if } n = 0 \\
 nf(n-1) & \text{if } n \geq 1 
\end{cases}
\]

For another example, \( f(n) = 1 + 2 + \cdots + n \) can be defined recursively as follows:

\[
f(n) = \begin{cases} 
1 & \text{if } n = 1 \\
f(n-1) + n & \text{if } n \geq 2 
\end{cases}
\]

- How to solve a recursive equation (to give a direct definition for a recursively defined function):

Example: By iterating.

\[
f(n) = \begin{cases} 
1 & \text{if } n = 1 \\
 2f(\frac{n}{2}) + n & \text{if } n \geq 2 
\end{cases}
\]

For simplicity, assume that \( n \) is a power of 2, i.e., \( n = 2^k \) for some \( k \).

\[
f(n) = 2f(\frac{n}{2}) + n
= 2^2 f(\frac{n}{2^2}) + 2n
= 2^3 f(\frac{n}{2^3}) + 3n
= \cdots
= 2^k f(\frac{n}{2^k}) + kn
= nf(1) + n \log n
= n(1 + \log n)
\]

Example: Still by iterating.

\[
f(n) = \begin{cases} 
1 & \text{if } n = 1 \\
 2f(n-1) + 1 & \text{if } n \geq 2 
\end{cases}
\]

\[
f(n) = 2f(n-1) + 1
= 2^2 f(n-2) + 2 + 1
= 2^3 f(n-3) + 2^2 + 2 + 1
= \cdots
= 2^{n-1} f(1) + 2^{n-2} + \cdots + 2 + 1
= 2^{n-1} + 2^{n-2} + \cdots + 2^2 + 2 + 1
= 2^n - 1
\]

- Design, analyze, and implement recursive algorithms: Divide and conquer (later).