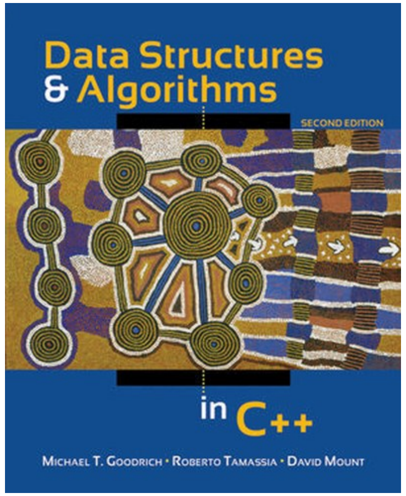
Data Structure

Analysis of Algorithms

Shin Hong

21 Mar 2023



DS&A. Chapter 4. Analysis Tools

Merging sorted array

- Which algorithm is faster?
- To what extent an algorithm is faster than the other?

```
merge1 (a1, a2) {
  a = duplicate(a1) ;
  foreach e in a2 {
    for (i = len(a) - 1 ; e < a[i] ; i--) ;

    for (j = len(a) + 1 ; i < j ; j--) {
      a[j] = a[j - 1] ;
    }
    a[i] = e ;
  }
  return a ;
}
```

```
merge2 (a1, a2) {
  i1 = 0, i2 = 0 ;
  while (i1 + i2 < len(a1) + len(a2)) {
    if (i2 == len(a2) ||
          (i1 < len(a1) && a1[i1] < a2[i2])) {
      a[i1 + i2] = a1[i1] ;
      i1++ ;
  }
  else {
      a[i1 + i2] = a2[i2] ;
      i2++ ;
  }
 }
 return a ;
}
```

Comparing Running Time of Algorithms (1/2)

- Experiment studies: use wall clock
  - Procedure
    - Implement the two algorithms as programs
    - Run the two programs with given inputs
    - Measure the physical running times, and compare them
  - Issues
    - Only limited inputs are used
    - The running time is affected by the hardware and software environment
    - An algorithm may be implemented in various ways
- Analytic approach is needed to evaluate algorithms, not their implementations

Comparing Running Time of Algorithms (2/2)

- Analytic comparison: asymptotic time complexity
  - count the number of computation steps, rather than wall clock time
  - consider all inputs by modeling running time as a function of input size
  - among many inputs of the same size, consider the worst case
  - compare the growth rate of the functions, rather than the function values at a specific input size

Primitive Operations

- Count how many primitive operations in a pseudo-code are executed
- Examples of primitive operations
  - Assigning a value to a variable
  - Calling a function
  - Performing an arithmetic operation (e.g., adding two numbers)
  - Comparing two numbers
  - Indexing into an array
  - Following an object reference
  - Returning from a function

Cost Function

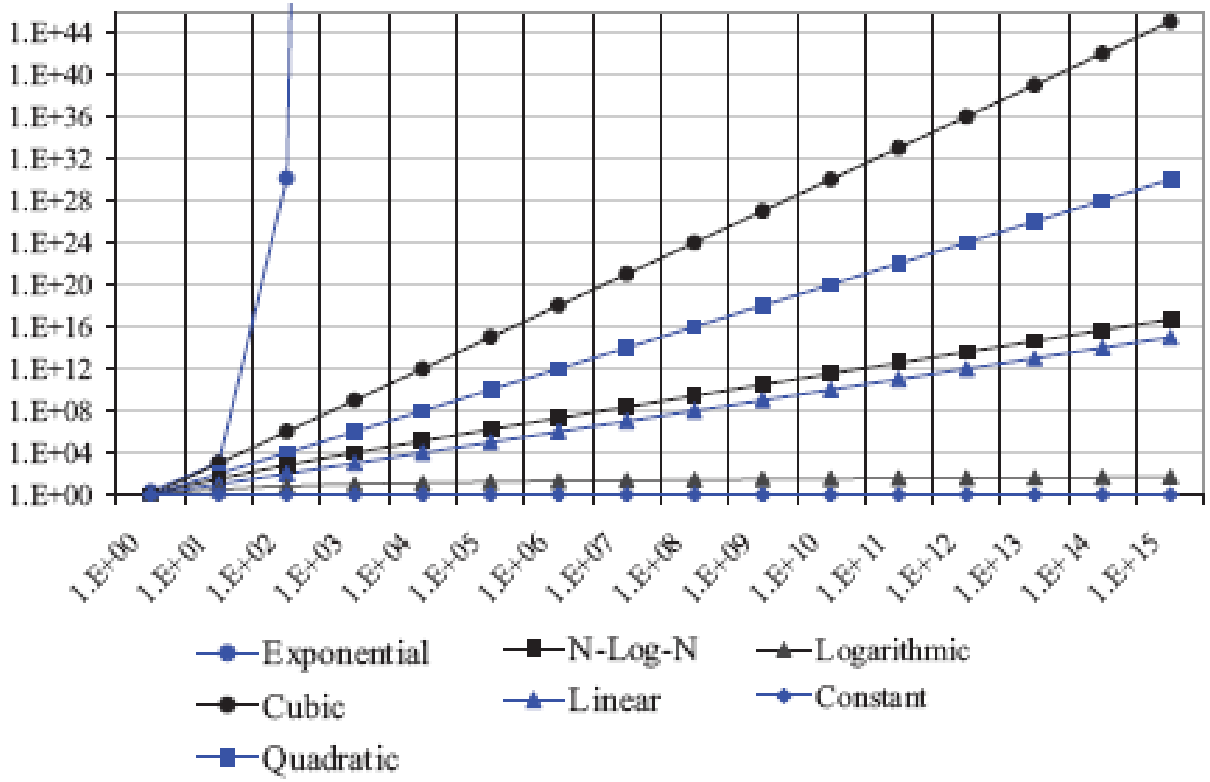
- Find a function $f(n)$ that determines how much primitive operations are required for processing an input of size n

- Focus on the worst case among inputs of the same size
  - intuitive abstraction of an input set

Asymptotic Notation (1/2)

- Compare the growth rate of the running time function

- Six popular kinds of functions
  - constant function $f(n) = c$
  - logarithm function $f(n) = \log n$
  - linear function $f(n) = n$
  - N-log N function $f(n) = n \log n$
  - quadratic function $f(n) = n^2$
  - exponential function $f(n) = a^n$



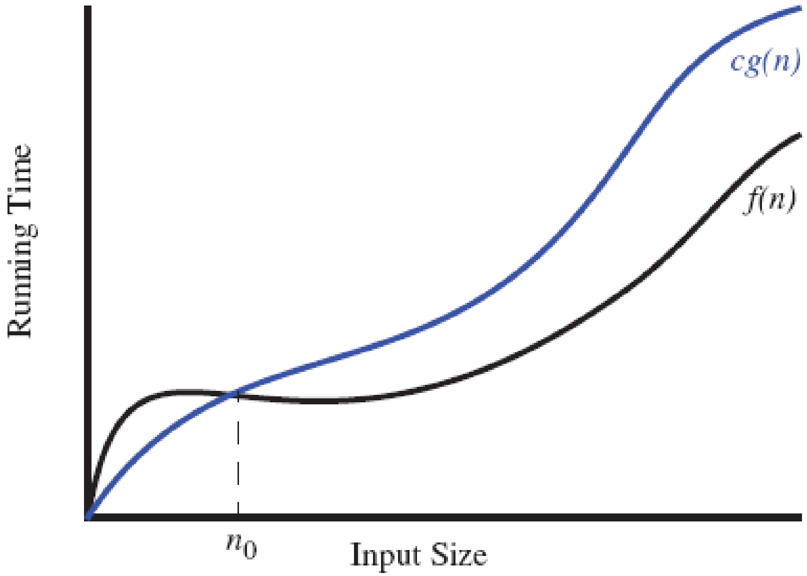
Asymptotic Notation (2/2)

- Big-O notation
  - let $f(n)$ and $g(n)$ be functions mapping nonnegative integers to real numbers
  - we say that "$f(n)$ is O($g(n)$)", "$f(n) \in$ O($g(n)$)" or "$f(n)$ is big-O of $g(n)$" if there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that

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

  - $g(n)$ eventually exceeds $f(n)$ indefinitely, thus it can be said $g(n)$ is greater than $f(n)$

  - e.g., $8n - 2$ is O(n)



Example

```
merge1 (a1, a2) {
  a = duplicate(a1) ;
  foreach e in a2 {
    for (i = len(a) - 1 ; e < a[i] ; i--) ;

    for (j = len(a) + 1 ; i < j ; j--) {
      a[j] = a[j - 1] ;
    }
    a[i] = e ;
  }
  return a ;
}
```

```
merge2 (a1, a2) {
  i1 = 0, i2 = 0 ;
  while (i < len(a1) + len(a2)) {
    if (i2 == len(a2) ||
        (i1 < len(a1) && a1[i1] < a2[i2])) {
      a[i1 + i2] = a1[i1] ;
      i1++ ;
  }
  else {
      a[i1 + i2] = a2[i2] ;
      i2++ ;
  }
 }
 return a ;
}
```

Characterizing Running Time using Big-O

- The big-O notation allows us to focus on the greatest order term and ignore the lower order terms
  - examples
    - $5n^4 + 3n^3 + 2n^2 + 4n + 1$ is $O(n^4)$
      - $5n^4 + 3n^3 + 2n^2 + 4n + 1 \leq (5 + 3 + 2 + 4 + 1)\ n^4 = cn^4$, for $c = 15$, when $n \geq n_0 = 1$.
    - $5n^2 + 3n\log n + 2n + 5$ is $O(n^2)$
    - 2^{n+2} is $O(2^n)$

Big-Omega and Big-Theta

- The big-O notation provides an asymptotic way of saying that a function is “less than equal to” another function
  - gives an upper bound of a function

- The big-Omega is to give a lower bound of a function
  - $f(n)$ is $\Omega(g(n))$ if $f(n) \geq cg(n)$ for all $n \geq n_0$

- The big-Theta declares that two functions grow at the same rate
  - $f(n)$ is $\Theta(g(n))$ if $c'g(\mathrm{n}) \leq f(n) \leq c''g(n)$ for all $n \geq n_0$