Data Structure - Spring 2022

4. Introduction to DS and Algorithm

Walid Abdullah Al
Computer and Electronic Systems Engineering
Hankuk University of Foreign Studies

Based on:
Goodrich, Chapter 3-4
Karumanchi, Chapter 1-2
Slides by Prof. Yung Yi, KAIST

Computer Vision Lab
Hankuk University of Foreign Studies

Variables and Data Types

- **Variable**
  - Name that hold value (data)
- **Data type**
  - Set of data with predefined values
  - Ex) integer, floating point, character, string, etc.
- **System-defined data types**
  - Also known as primitive data types
  - int, float, str, etc.
- **User-defined data types**

```
class Student:
    def __init__(self):
        self.id = 20212345
        self.name = "Mr"
        self.cgpa = 4.20
        self.year = 2
```

Data Structure

- **Data structure**
  - Particular way of storing and organizing data
  - so that it can be used efficiently
- **Linear data structures**
  - Elements are accessed sequentially
  - Linked lists, stacks, queues, etc.
- **Non-linear data structures**
  - Stored/accessed in non-sequential order
  - Trees, graphs, etc.
- **Abstract data type (ADT)**
  - Data structure + its operation

Algorithm and its Performance

- **Algorithm**
  - Step-by-step unambiguous instructions to solve a problem
- **Program**
  - Data structure + algorithm
- **Performance analysis**
  - Running time
  - (also, memory, etc.)

Algorithm. Sum of Array
sum $\leftarrow$ 0
for i = 1 to N
    sum $\leftarrow$ sum + Array[i]

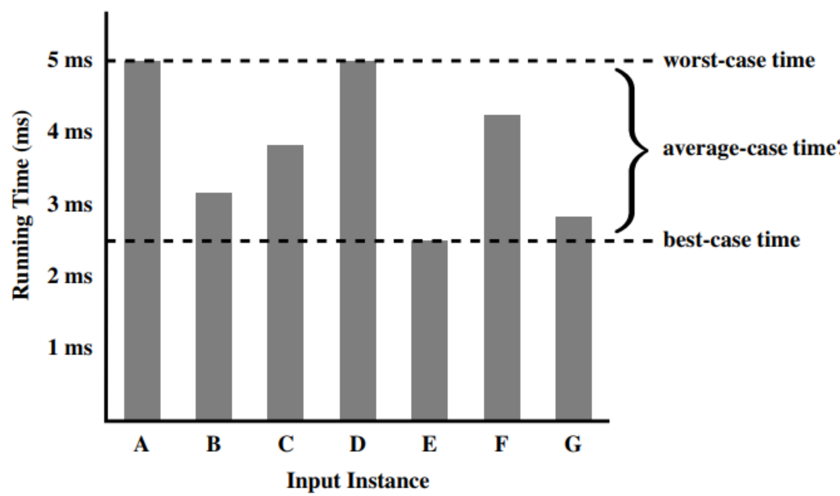
Running Time Analysis

- **Goal of analysis**
  - How processing time increases as the size of problem (input size) increases
  - Input size: size of an array, elements in a matrix, degree of polynomial, etc.
- **Experimental analysis**
  - Execution time
  - Specific to a computer
- **Number of statements**
  - Specific to a language
- **Number of primitive operations**
  - Generally applicable
  - Assignment, arithmetic-ops, function-call, return, etc.

```
from time import time
start_time = time( )
run algorithm
end_time = time( )
elapsed = end_time − start_time
```

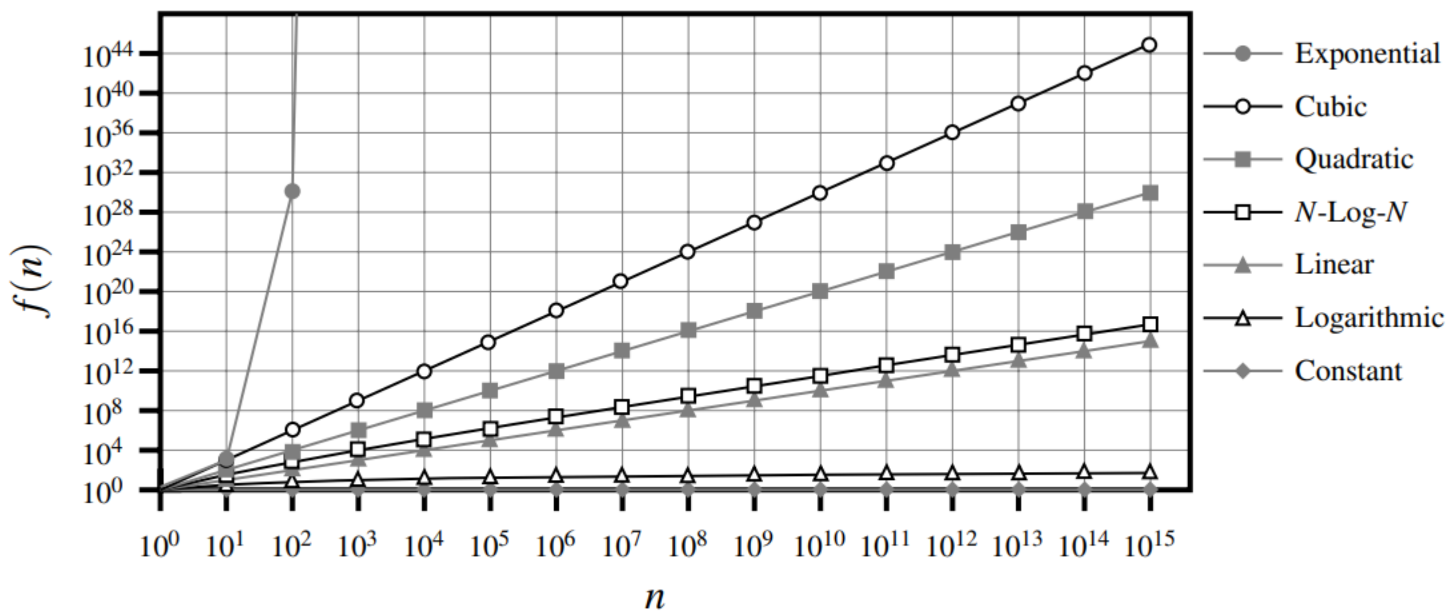
Theoretical Analysis

- **Number of operations as a function of input size**
  - f(n): n is the input size
  - Captures the growth rate of running time
- **Focuses on the worst-case input**
  - Problems with average-case analysis?



Common Functions

constant	logarithm	linear	n-log-n	quadratic	cubic	exponential
1	$\log n$	n	$n \log n$	n^2	n^3	a^n



Running Time Analysis: Example

```
def unique1(S):
  """Return True if there are no duplicate elements in sequence S."""
  for j in range(len(S)):
    for k in range(j+1, len(S)):
      if S[j] == S[k]:
        return False                 # found duplicate pair
  return True                        # if we reach this, elements were unique
```

Some useful summation series

Arithmetic series

$$\sum_{K=1}^{n} k = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

Geometric series

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 \ldots + x^n = \frac{x^{n+1} - 1}{x - 1} (x \neq 1)$$

Harmonic series

$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \ \ldots + \frac{1}{n} \approx \log n$$

Other important formulae

$$\sum_{k=1}^{n} \log k \approx n \log n$$

$$\sum_{k=1}^{n} k^p = 1^p + 2^p + \cdots + n^p \approx \frac{1}{p+1} n^{p+1}$$

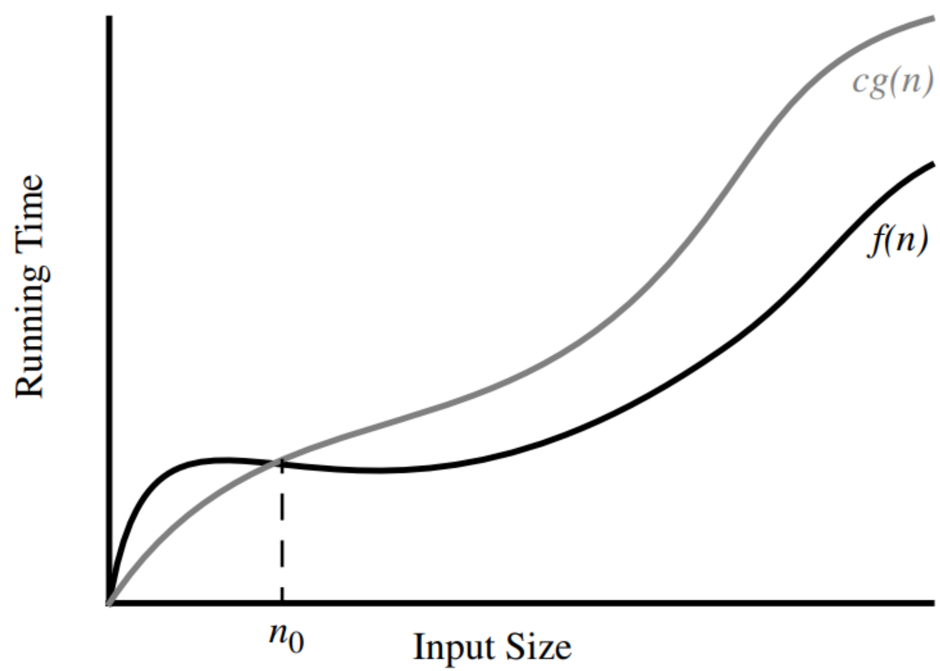
Asymptotic Analysis

- **The “big-picture” approach**
  - Only focuses on the main factor determining the growth
  - $2n^2 \approx n^2, 2n + 8 \approx n$
- **Running time of the following function**
  - Is proportional to n

```
def find_max(data):
  """Return the maximum element from a nonempty Python list."""
  biggest = data[0]                  # The initial value to beat
  for val in data:                   # For each value:
    if val > biggest                 # if it is greater than the best so far,
      biggest = val                  # we have found a new best (so far)
  return biggest                     # When loop ends, biggest is the max
```

“Big-Oh” Notation

- **f(n) is O(g(n))**
  - if $f(n) \leq c.g(n)$, for $n \geq n_0$
  - (c>0)



Some properties of Big-Oh

$5n^4 + 3n^3 + 2n^2 + 4n + 1$ is $O(n^4)$.   How can we justify?

If $f(n)$ is a polynomial of degree d, that is,

$$f(n) = a_0 + a_1 n + \cdots + a_d n^d,$$

and $a_d > 0$, then $f(n)$ is $O(n^d)$.

Big-Oh representation practice

$5n^2 + 3n\log n + 2n + 5$

$3\log n + 2$

2^{n+2}

$2n + 100\log n$

Example

```
def unique1(S):
  """Return True if there are no duplicate elements in sequence S."""
  for j in range(len(S)):
    for k in range(j+1, len(S)):
      if S[j] == S[k]:
        return False                # found duplicate pair
  return True                       # if we reach this, elements were unique
```

Example

```
def unique2(S):
  """Return True if there are no duplicate elements in sequence S."""
  temp = sorted(S)                    # create a sorted copy of S
  for j in range(1, len(temp)):
    if S[j−1] == S[j]:
      return False                    # found duplicate pair
  return True                         # if we reach this, elements were unique
```

Big-Oh comparison

- **Algorithm 1. unique1**
  - $O(n^2)$
- **Algorithm2. unique2**
  - $O(nlogn)$
- **Which algorithm is better/faster?**