Introduction to Algorithms

As part of Unit 1

CC4 Data Structures and Algorithms

Christine T. Gonzales

Table of Contents

- Priori and posteriori estimates
- Asymptotic notation
- Frequency count and Big-O notation

Priori and Posteriori Estimates

Priori analysis | Posteriori testing

Priori and Posteriori Estimates

- Originally used for statistics and partial differential equations
- In computer science, they are used to analyze algorithms and test their effectiveness
  - Priori estimate / priori analysis – analysis of algorithms
  - Posteriori estimate / posteriori testing – testing of program codes

Priori Analysis

- Focuses on analyzing an algorithm
  - Determine the efficiency of the algorithm
  - Determine the time and space
  - Done before it is coded into a program
- Independent of language
  - Usually expressed in pseudocode
- Independent on hardware
  - Worst case scenario is usually measured

Posteriori Testing

- Done on a program
  - Usually focuses on testing the efficiency of the program
  - Also looks into watch time and bytes
  - There are other factors that can change the efficiency
- Dependent on the:
  - Language
  - Hardware (usually space)
  - Operating system

Asymptotic Notation

What is asymptotic notation? | Types of asymptotic notations

What is Asymptotic Notation?

- Languages that analyzes an algorithm’s running time
- Done by identifying its behavior as the input size for the algorithm increases
- Usually used to measure time, but can also measure space

Types of Asymptotic Notations

- **Big-O notation (O)**
  - Identifies the **worst case** scenario complexity of an algorithm
- **Omega notation (Ω)**
  - Identifies the **best case** scenario complexity of an algorithm
- **Theta notation (Θ)**
  - Identifies the **average case** scenario complexity of an algorithm

Frequency Count and Big-O Notation

What is the Frequency Count method? | Examples of frequency count method | What is the Big-O Notation? | Complexity chart | Examples of Big-O Notation

What is the Frequency Count Method?

- Method to determine the time (usually) and space of an algorithm
- Can be known by **assigning one unit of time / space for each statement**
- If any statement is repeating, then the frequency is calculated and the time taken is computed

Example – Frequency Count Method

```
                                              Frequency Count
K = 500;                                            1
for (j=1; j<=K; j++)                                1 + k + 1 + k
     x= x+1;                                        k
n=200;                                              1

Substituting actual value for k.
Freq. count. = 1504
             = O(1)
```

What is the Big O Notation?

- Type of asymptotic notation
- Used to determine the efficiency and **complexity** of an algorithm:
    - Average
    - Best
    - Worst case – usually used
- Looks into the complexity in terms of the input size
- Used for priori analysis
- Frequency count method must be done first to properly identify the complexity

Big O Notation

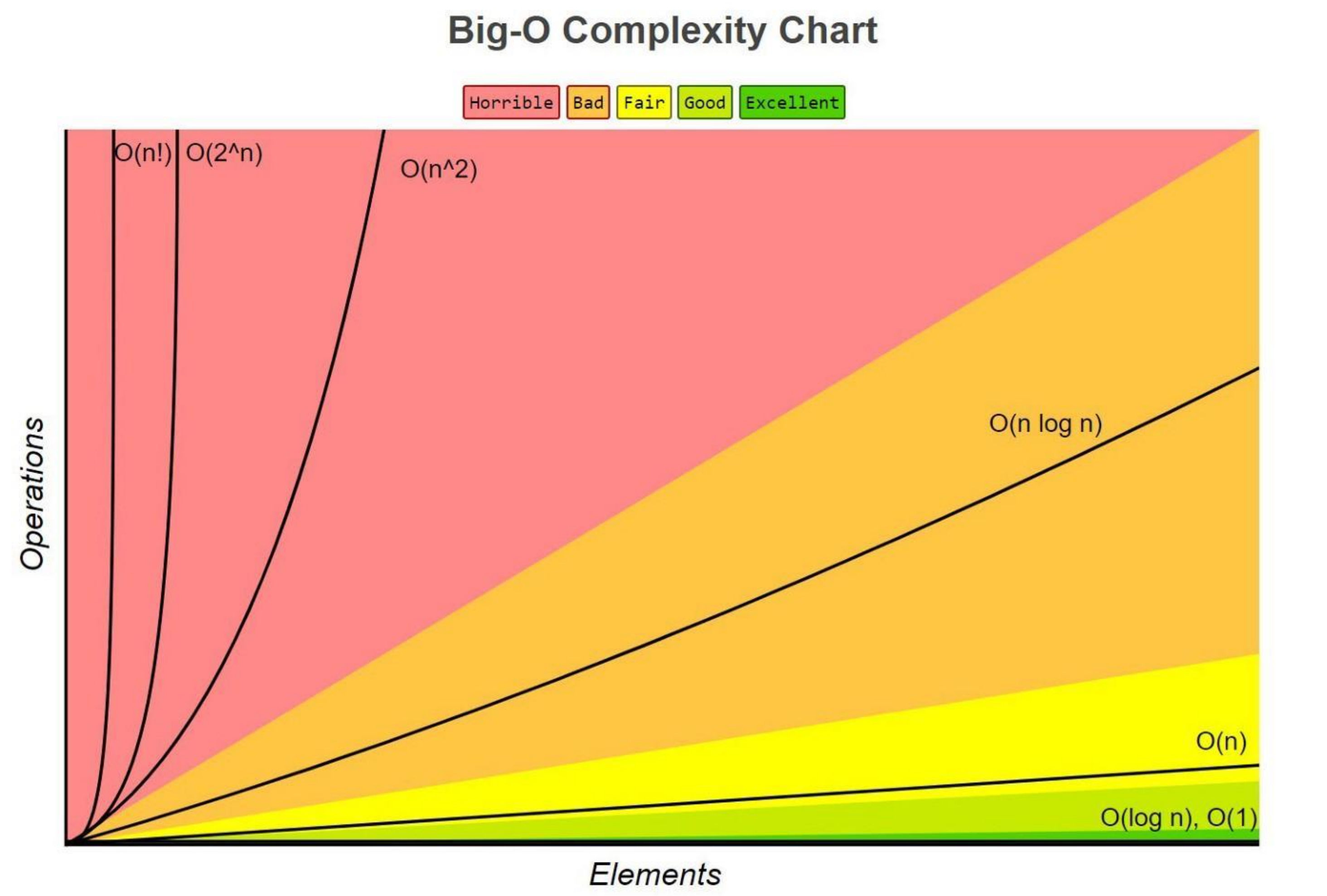
General Rules to Determine:

- Ignore constants
- Certain terms dominate others:

O(1) < O (log n) < O(n) < O(n log n) <
O(n2) < O(2n) < O(n!)

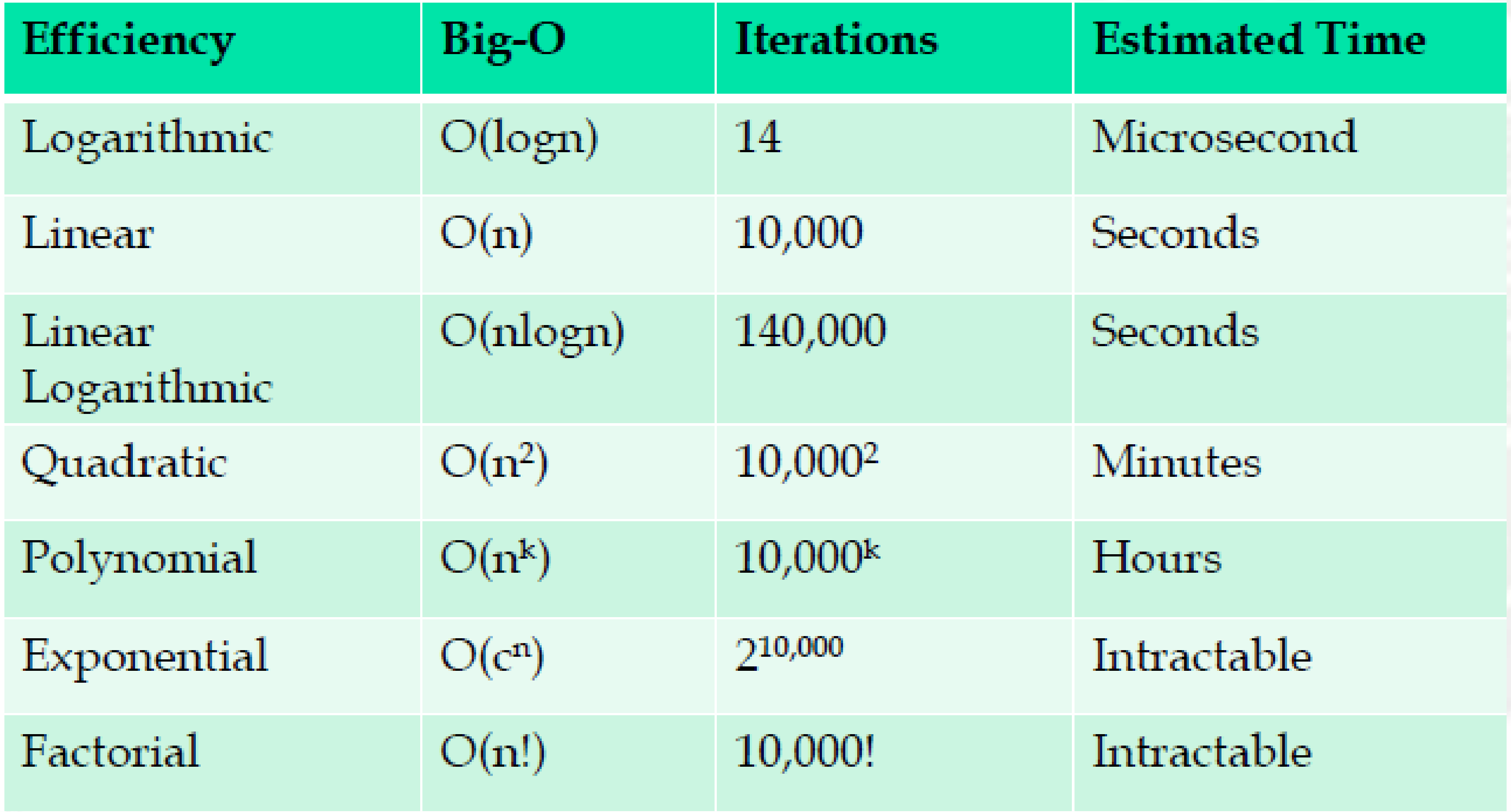
- Ignore low-order terms when the high-order terms are present

Big O Complexity Chart



Big O Measure of Efficiency

- Measure of efficiency for n = 10,000



Efficiency	Big-O	Iterations	Estimated Time
Logarithmic	$O(\log n)$	14	Microsecond
Linear	$O(n)$	10,000	Seconds
Linear Logarithmic	$O(n\log n)$	140,000	Seconds
Quadratic	$O(n^2)$	$10{,}000^2$	Minutes
Polynomial	$O(n^k)$	$10{,}000^k$	Hours
Exponential	$O(c^n)$	$2^{10{,}000}$	Intractable
Factorial	$O(n!)$	10,000!	Intractable

Big O Complexity Chart

- Looks into the time that is taken to finish a set of operations
- Usually states the efficiency of the algorithm
  - “Horrible” notations can be faster depending on the number of operations
  - “Good” notations are usually better on shorter operations, but worse at larger ones

Big O Complexity Chart

- Constant time
- Linear time
- Quadratic time

Big O Complexity Chart

Constant Time

- Independent of input size
- It doesn’t matter what the operation is, it will have the same complexity
- Notation(s): O(1)
- Sample operations:
  - Assignment to a variable
  - Printing

Big O Complexity Chart

Linear Time

- The more operations, the slower the time
- Better for shorter operations
- Notation(s): O(log N), O(N), O(N log N)
- Sample operations:
  - Decision control structures
  - One-layered iterative control structures

Big O Complexity Chart

Quadratic Time

- The more operations, the shorter the time
- Better for longer operations
- Notation(s): O(N2), O(2N), O(N!)
- Sample operations:
  - Nested iterative control structures
  - Recursive loops
  - Permutations

Big O Notation

- O(1)
  - Constant; most instructions are executed once or at most only a few times
- O(log n)
  - Program slightly slower as N grows; normally in programs that solve a big problem by transforming it into a small problem, cutting the size by some constant factor
- O(n)
  - Linear; proportional to the size of N

Big O Notation

- O(n log n)
  - Occurs in algorithms that solve a problem by breaking it up into smaller sub-problems, solve them independently and then combining the solution
- O(n^2)
  - Quadratic; can be seen in algorithms that process all pairs of data items.
- O(n^k)
  - Polynomial; algorithms that process polynomial of data items.

Big O Notation

- O(2^n)
  - Exponential; brute-force situation
- O(n!)
  - Factorial

Example: given 2 algorithms performing the same task on N inputs, which is faster and efficient?

P1	P2
10n	$n^2/2$
O(n)	**O(n^2)**

Big O Notation

N	**P1**	**P2**
1	__	__
5	__	__
10	__	__
15	__	__
20	__	__
30	__	__

Substitute values in N and determine which among the two algorithms is more efficient and faster.

Big O Notation

N	**P1**	**P2**
1	10	0.5
5	50	12.5
10	100	50
15	150	112.5
20	200	200
30	300	450

P2 is faster and more efficient for N <= 20, but for N>20, P1 proves to be faster than P2.

Example – Big O Notation

```
public static int returnSum (int a[], n){
    int s = 0;
    for (int i = 0; i < n; i++){
        s = s + a[i];
    }
    return s;
}
```

Example – Big O Notation

```
for(j=1; j<=n; j++){
   for(k=1; k<=n; k++){
      c[j][k] = 0;
      for(l=1; l<=n; l++){
         c[j][k] = c[j][k] * b[l][k];
      }
   }
}
```