15CSE201 : Data Structures and Algorithms

Lecture 2 : Complexity Analysis

Ritwik M

Based on the reference materials by Prof. Goodrich, OCW METU, Dr. Jeyakesavan Veerasamy and Dr. Vidhya Balasubramanian

Analysis of Data Structures

- Data structures have many functions
  - Each function is a set of simple instructions
- Analysis
  - Determine resources, time and space the algorithms requires
- Goal
  - Estimate time required to execute the functionalities
  - Reduce the running time of the program
  - Understand the space occupied by the data structure

Issues in Analysis

- Running time grows with input size
  - Varies with different inputs
  - Actual running time can be calculated in seconds or milliseconds
- Issues
  - The system setup must be same for all inputs
  - Same hardware and software must be used
  - Actual time maybe affected by other programs running on the same machine
- A theoretical analysis is sometimes preferable

Average Case and Worst Case

- The running time and memory usage of a program is not constant
  - Depends on input
  - Can run fast for certain inputs and slow for others
    - e.g linear search
- Average Case Cost
  - Cost of the algorithm (time and space) on average
  - Difficult to calculate
- Worst Case
  - Gives an upper limit for the running time and memory usage
  - Easier to analyze the worst case

Method for analyzing complexity

- Model of Computation
  - Mathematical Framework
- Asymptotic Notation
  - What to Analyze
- Running Time Calculations
- Checking the analysis

Counting Primitives to analyze time complexity

- Primitive operations are identified and counted to analyze cost
- Primitive Operations
  - Assigning a value to a variable
  - Performing an arithmetic operation
  - Calling a method
  - Comparing two numbers
  - Indexing into an array
  - Following an object reference
  - Returning from a method

What about an IF -Then -Else Statement?

Example

Algorithm FindMax(S, n)

Input : An array S storing n numbers, n>=1

Output: Max Element in S

curMax <-- S[0] (2 operations)

i ← 0 (1 operations)

while i< n-1 do (n comparison operations)

if curMax <A[i] ***then*** (2(n-1) operations)

curMax <-- A[i] (2(n-1) operations)

i ← i+1; (2 (n-1) operations)

return curmax (1 operations)

Complexity between 5n and 7n-2

Some Details

- Loops
  - cost is linear in terms of number of iterations
  - Nested loops – product of iteration of the loops
    - If outer loop has n iterations, and inner m, cost is mn
  - Multiple loops not nested
    - Complexity proportional to the longest running loop
- If Else
  - Cost of if part of else part whichever is higher

Exercises

1)
```
sum = 0;
for( i=0; i<n; i++ )
    sum++;
```

2)
```
sum = 0;
for( i=0; i<n; i++ )
    for( j=0; j<n; j++ )
        sum=sum+1;
```

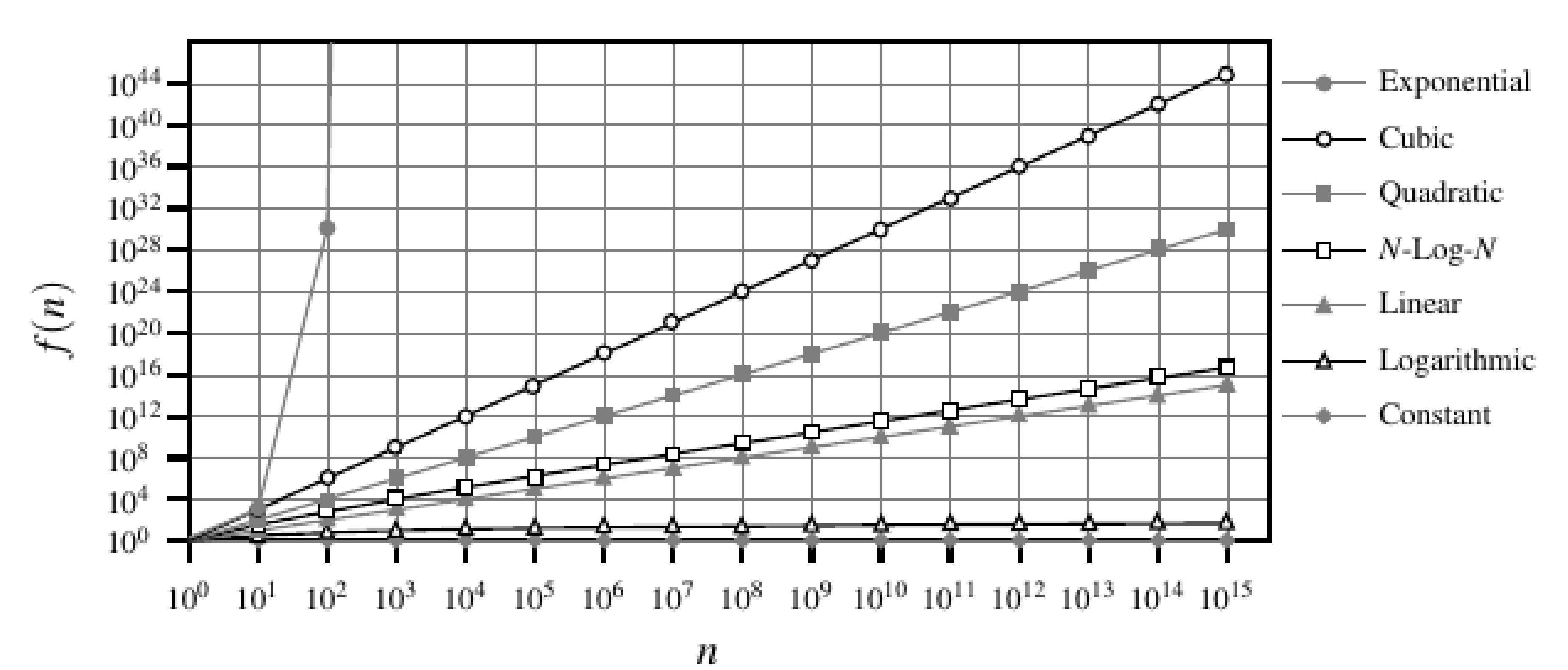
3)
```
sum = 0;
for( i=0; i<n; i++ )
    for( j=0; j<n*n; j++ )
        sum+=1;
```

4)
```
for (i = 20; i <= 30; i++) {
    for (j=1; j<=n; j++){
        x = x + 1;
    }
}
```

Growth Rates and Complexity

- Important factor to be considered when estimating complexity
- When experimental setup (hardware/software) changes
  - Running time/memory is affected by a constant factor
  - 2n or 3n or 100n is still linear
  - Growth rate of the running time/memory is not affected
- Growth rates of functions
  - Linear
  - Quadratic
  - Exponential

Comparing Growth Rates



Comparing Growth Rates Contd...

n	O(1)
10^2	1 μsec
10^3	1 μsec
10^4	1 μsec
10^5	1 μsec
10^6	1 μsec
10^7	1 μsec
10^8	1 μsec

n	O(n^2)	O(2^n)
100	1 μsec	1 μsec
110	1.2 μsec	1 msec
120	1.4 μsec	1 sec
130	1.7 μsec	18 min
140	2.0 μsec	13 d
150	2.3 μsec	37 yr
160	2.6 μsec	37,000 yr

O(n^2)
1 μsec
100 μsec
10 msec
1 sec
1.7 min
2.8 hr
11.7 d

Ideal Logic / From the Growth Rates

- Data structure operations to run in times proportional to the constant or logarithm function
- Algorithms to run in linear or n-log-n time

More Exercises

- Consider the following three algorithms for determining whether anyone in the room has the same birthday as you.

  - *Algorithm 1*: You say your birthday, and ask whether anyone in the room has the same birthday. If anyone does have the same birthday, they answer yes.

  - *Algorithm 2*: You tell the first person your birthday, and ask if they have the same birthday; if they say no, you tell the second person your birthday and ask whether they have the same birthday; etc, for each person in the room.

  - *Algorithm 3*: You only ask questions of person 1, who only asks questions of person 2, who only asks questions of person 3, etc. You tell person 1 your birthday, and ask if they have the same birthday; if they say no, you ask them to find out about person 2. Person 1 asks person 2 and tells you the answer. If it is no, you ask person 1 to find out about person 3. Person 1 asks person 2 to find out about person 3, etc.

More Exercises

Questions:

1. For each algorithm, what is the factor that can affect the number of questions asked (the "problem size")?
2. In the worst case, how many questions will be asked for each of the three algorithms?
3. For each algorithm, say whether it is constant, linear, or quadratic in the problem size in the worst case.

Solution:

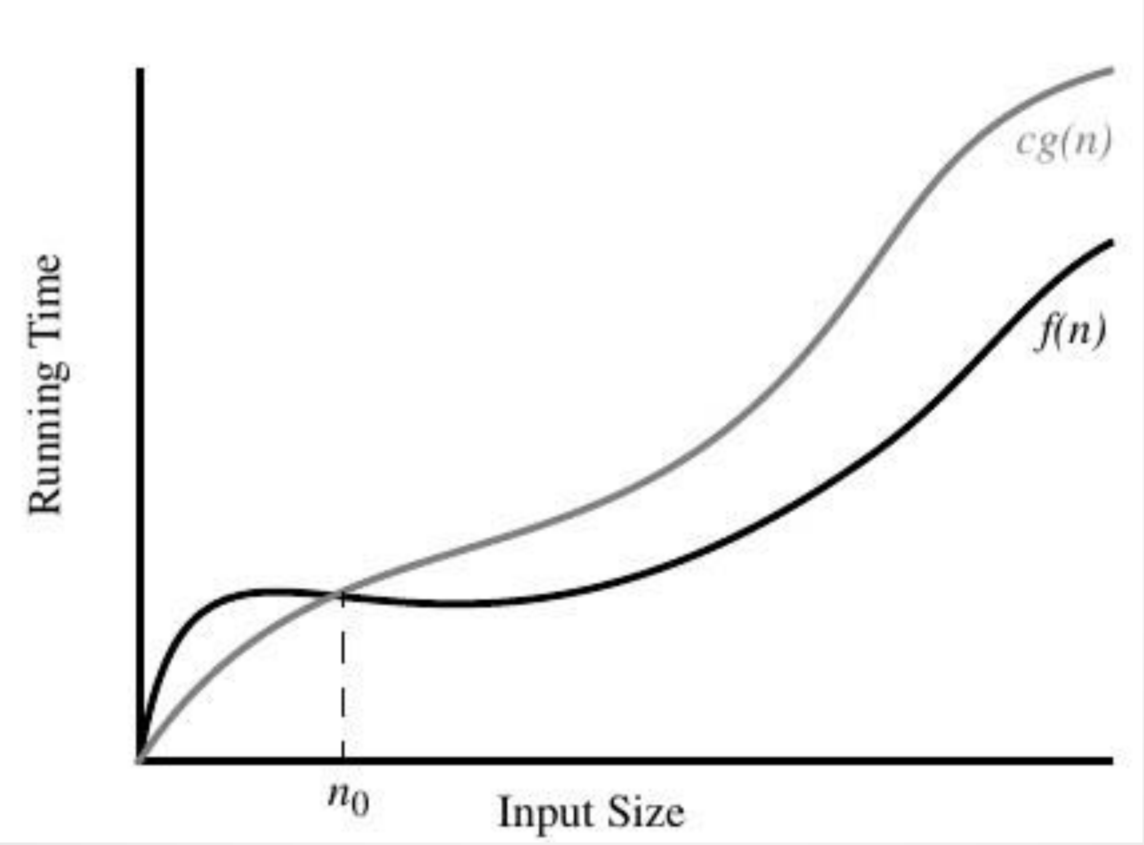
1. The number of people in the room.
2. Assume there are N people in the room. In algorithm 1 you always ask <u>1 question</u>. In algorithm 2, the worst case is if no one has your birthday. Here you have to ask every person to figure this out. <u>This is N questions.</u> In algorithm 3, the worst case is the same as algorithm 2. The number of questions is 1 + 2 + 3 + ... + N-1 + N = <u>N*(N+1)/2</u>.
3. Constant, Linear and Quadratic

Asymptotic Analysis

- Can be defined as a method of describing limiting behavior
- Used for determining the computational complexity of algorithms
  - A way of expressing the main component of the cost of an algorithm using the most determining factor
  - e.g if the running time is *5n2+5n+3,* the most dominating factor is *5n2*
- Capturing this dominating factor is the purpose of asymptotic notations

Big-Oh Notation

- Given a function f(n) we say, f(n) is O(g(n)) if there are positive constants c and n_0 such that f(n)<= cg (n) when n>= n_0



Big-Oh Example

- Show 7n-2 is O(n) – [Hint: prove that f(n)<=cg(n)]
  - need $c > 0$ and $n_0 >= 1$ such that $7n-2 <= cn$ for $n >= n_0$
  - this is true for $c = 7$ and $n_0 = 1$
- Show $3n^3 + 20n^2 + 5$ is $O(n^3)$
  - need $c > 0$ and $n0 >= 1$ such that $3n^3 + 20n^2 + 5 <= cn^3$ for $n >= n0$
  - this is true for $c = 4$ and $n0 = 21$
- n^2 is not O(n)
  - Must prove $n^2 <= cn$
  - $n <= c$
  - The above inequality cannot be satisfied since c must be a constant
  - Hence proof by contradiction

Big-Oh Significance

- The big-Oh notation gives an upper bound on the growth rate of a function
- The statement "f(n) is O(g(n))" means that the growth rate of f(n) is not more than the growth rate of g(n)
  - We are guaranteeing that f(n) grows at a rate no faster than g(n)
  - Both can grow at the same rate
  - Though 1000n is larger than n^2, n^2 grows at a faster rate
    - n^2 will be larger function after n = 1000
    - Hence $1000n = O(n^2)$
- The big-Oh notation can be used to rank functions according to their growth rate

Big-Oh Significance

- Table of max-size of a problem that can be solved in one second, one minute and one hour for various running time measures in microseconds [Goodrich]

Running Time	Maximum Problem Size (n)		
	1sec	1 min	1 hour
$400n$	2500	150000	9000000
$20n\log n$	4096	166666	7826087
$2n^2$	707	5477	42426
n^4	31	88	244
2^n	19	25	31

- Take away from the table
  - Algorithms with quadratic or cubic running times are less practical, and algorithms with exponential running times are infeasible for all but the smallest sized inputs

Common Rules for Big-Oh

- If is f(n) a polynomial of degree d, then f(n) is $O(n^d)$, i.e.,
  - Drop lower-order terms
  - Drop constant factors
- Use the smallest possible class of functions to represent in big Oh
  - "2n is O(n)" instead of "2n is $O(n^2)$"
- Use the simplest expression of the class
  - "3n+ 5 is O(n)" instead of "3n + 5 is O(3n)"

Some Good Examples

Complexity Class	Growth Function	Example
O(1)	Constant Time	One arithmetic operation (eg., +, *) , a print statement
O(log n)	Logarithmic Time	Binary search in a sorted array of n elements.
O(n)	Linear Time	Traversing an array, Linear search, etc
O(n log n)	"n log n" Time	MergeSort, QuickSort, etc
O(n^2)	Quadratic Time	Worst case time complexity of Bubble sort, selection sort, etc
O(a^n) (a>1)	Exponential Time	Recursive Fibonacci implementation, towers of Hanoi

Some More Exercises

- Show that 8n+5 is O(n)
- Show that $20n^3 + 10n\log n + 5$ is $O(n^3)$
- Show that 3logn+2 is O(logn).

Yet Another Exercise

- Consider a set of numbers from 1 to n. All the values except one value are present
  - Goal: Find the missing number
  - Give 3 solutions to find the missing number
  - What is the time and <u>space</u> complexity in terms of n?

Even More Exercises

- **Algorithm** arrayFind(x,A);

```
//Given an element x and an n element array A, output pos if present in A
for i = 0 to n-1 do
      if x = A[i] then
            return I
return -1
```

- There is an algorithm find2D to find an element x in an nxn array A. The algorithm iterates over the rows of A and calls the algorithm arrayFind on each one, until x is found or it has searched all rows of A.
  - What is the time and space complexity of the algorithm

The Exercises Continue

- Calculate the value returned by total

```
def example4(S):
"""Return the sum of the prefix sums of sequence S."""
n = len(S)
prefix = 0
total = 0
for j in range(n):
    prefix += S[j]
    total += prefix
return total
```

And More Exercises.. They Don’t Stop!!

- Given an n-element sequence S, Algorithm D calls Algorithm E on each element S[i]. Algorithm E runs in O(i) time when it is called on element S[i]. What is the worst-case running time of Algorithm D?
- A sequence S contains n−1 unique integers in the range [0,n−1], that is, there is one number from this range that is not in S. Design an O(n)-time algorithm for finding that number. You are only allowed to use O(1) additional space besides the sequence S itself.

Anything Else?

- For More exercises and problems you can refer:
  - Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.
  - Data Structures and Algorithms in Python by Michael T. Goodrich

  - Have Fun Solving!!