15CSE201 : Data Structures and Algorithms

Lecture 2.5 : Complexity Analysis -- A Recap

By
Ritwik M

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

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 ← 1 (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

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

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

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

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

More Exercises

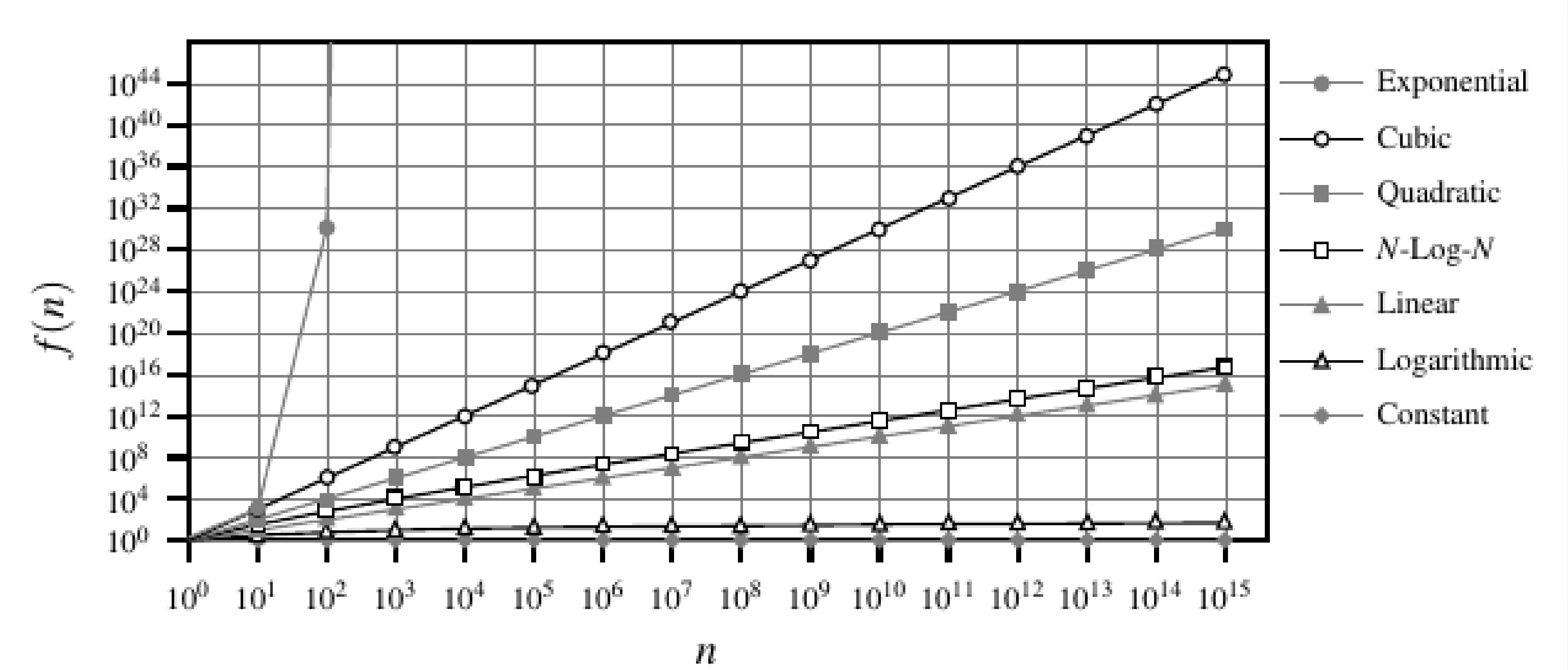
Identify the basic operation for each of these algorithms

1) Computing sum of ‘n’ numbers
2) Computing the cube of a number
3) Computing n!
4) Max(2,3,7,6,5,8)
5) Pen and paper method for multiplying 2 n-digit numbers

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



Ideal Logic / From the Growth Rates

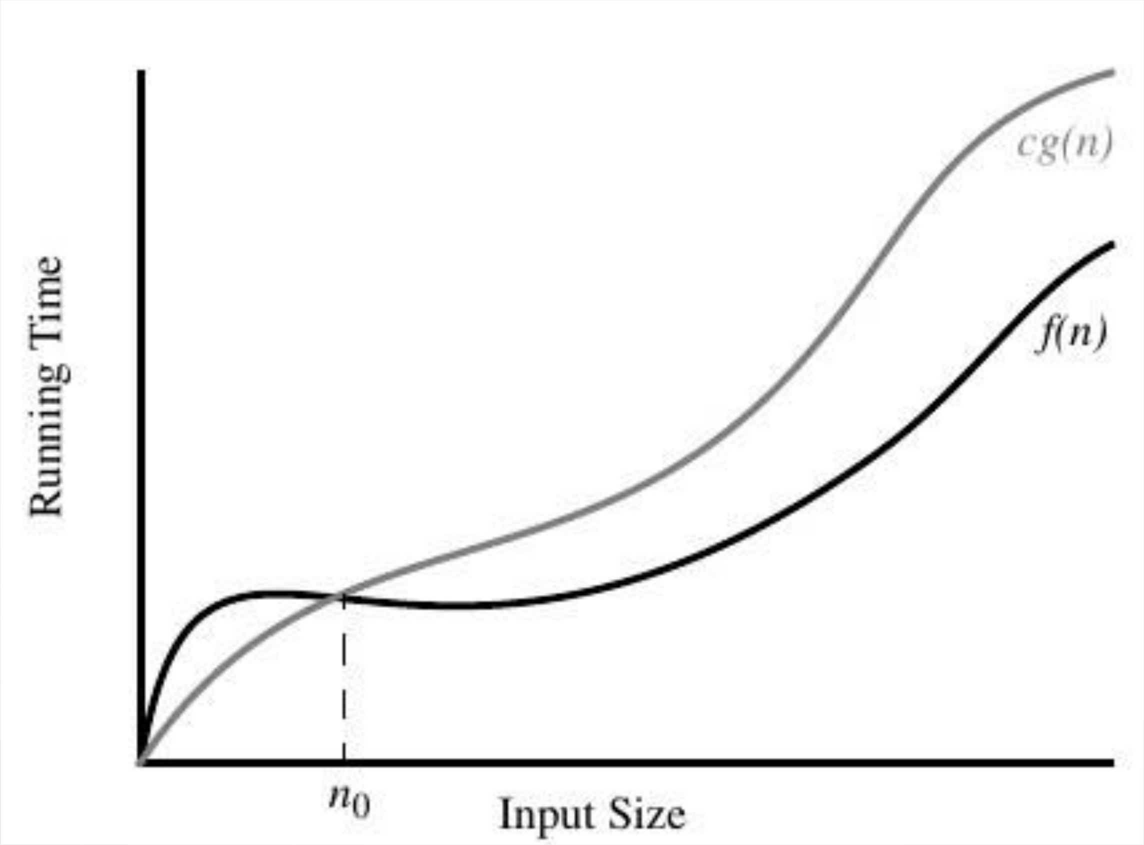
- The exponential functions are increasing at an extremely fast rate when compared to linear functions.
- Data structure operations to run in times proportional to the constant or logarithm function
- Algorithms to run in linear or n-log-n time

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 $n_0 = 21$
- 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
20nlogn	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)"

Exercises

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

Try This Out

- 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?

Exercises

- 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
```

QUIZ - 1

- Kindly solve the following quiz on your own.
- Copying is strictly prohibited and any discovered attempt will be viewed seriously
- Answer any 5 of 6 questions
- If all 6 are attempted and correct, bonus points will be awarded
- All questions carry 2 points.
- Total 10 points

QUIZ

1. Calculate time complexity by counting primitives

    Input: An array A storing n ≥ 1 integers.
    Output: The sum of the elements in A.
    1: s ← A[0]
    2: for i ← 1 to n − 1 do
    3:     s ← s + A[i]
    4: end for
    5: return s

Quiz

2. Calculate time complexity by counting primitives

   Input: An array A storing $n \geq 1$ integers.
   Output: The sum of the elements at even cells in A.
   1: $s \leftarrow A[0]$
   2: for $i \leftarrow 2$ to $n - 1$ by increments of 2 do
   3: $\quad s \leftarrow s + A[i]$
   4: end for
   5: return s

Quiz

3. Calculate time complexity by Asymptotic notation

    Input: An array A storing $n \geq 1$ integers.
    Output: The sum of the prefix sums in A.

```
s ← A[0]
t ← s
for i ← 1 to n − 1 do
    s ← s + A[i]
    t ← t + s
end for
return t
```

4. Show that $f(n) = n^2 + 2n + 1$ is $O(n^2)$.

5. Show that $f(n) = (n + 1)^3$ is $O(n^3)$.

6. Assume that you are the teacher and write as if you are explaining the concept of growth rates to a student.

QUIZ –Solution

1.

Input: An array A storing n ≥ 1 integers.
Output: The sum of the elements in A.

```
s ← A[0] {1}
for i ← 1 to n − 1 do {n-1}
    s ← s + A[i] {n-1}
end for
return s {1}
```

2.

Input: An array A storing n ≥ 1 integers.
Output: The sum of the elements at even cells in A.

```
s ← A[0] {1}
for i ← 2 to n − 1 by increments of 2 do {(n-1)/2}
    s ← s + A[i] {(n-1)/2}
end for
return s {1}
```

QUIZ –Solution

3. O(n)
4. Choose C =4 and n > 1
5. Choose C = 8. whenever n > 1
6. The scoring for this is based on the answer that you provide, namely the depth and clarity of the explanation given.