CSE 230: Data Structures

Lecture 2: Complexity Analysis

Dr. Vidhya Balasubramanian

Amrita School of Engineering
Amrita Vishwa Vidyapeetham

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 analyse 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

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 (2n 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 6n and 8n-2

Some Points

- 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

Try these

1) sum = 0;

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

current ← 0

for i ← 0 to n - 1 do

current ← current+A[i]

return current/n

2) prod ← 0

for i ← 0 to n-1 do

prod ← prod + A[i]*B[i]

return prod

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

Try These

- for (i = 0; i < n; i++) do

  if (A[i] == x) then

  return true

  return false

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

- sum = 0;

  for(i=0; i<n; i++)

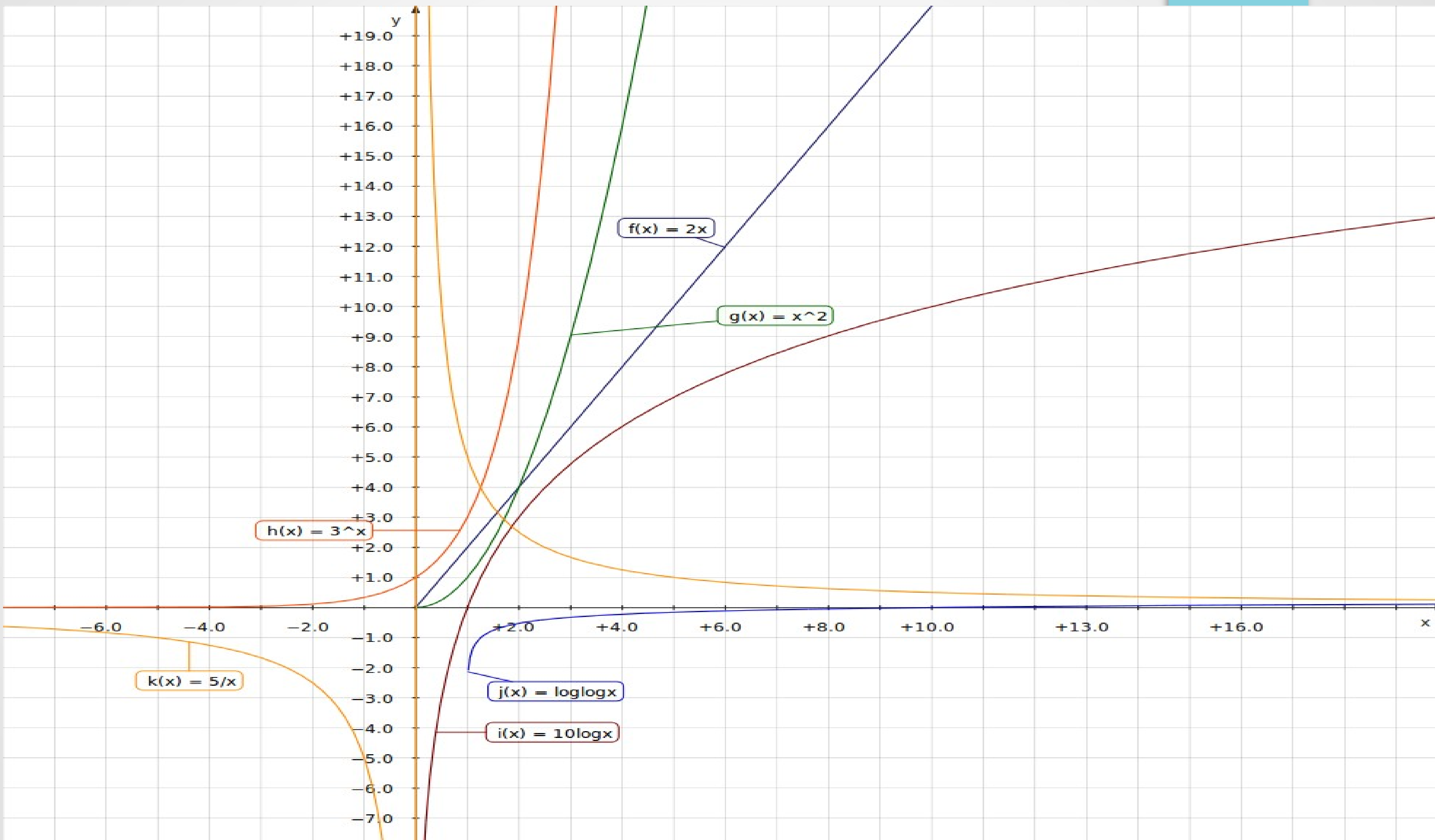
  for(j=0; j<n; j++)

  sum++;

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

Sample Growth Functions

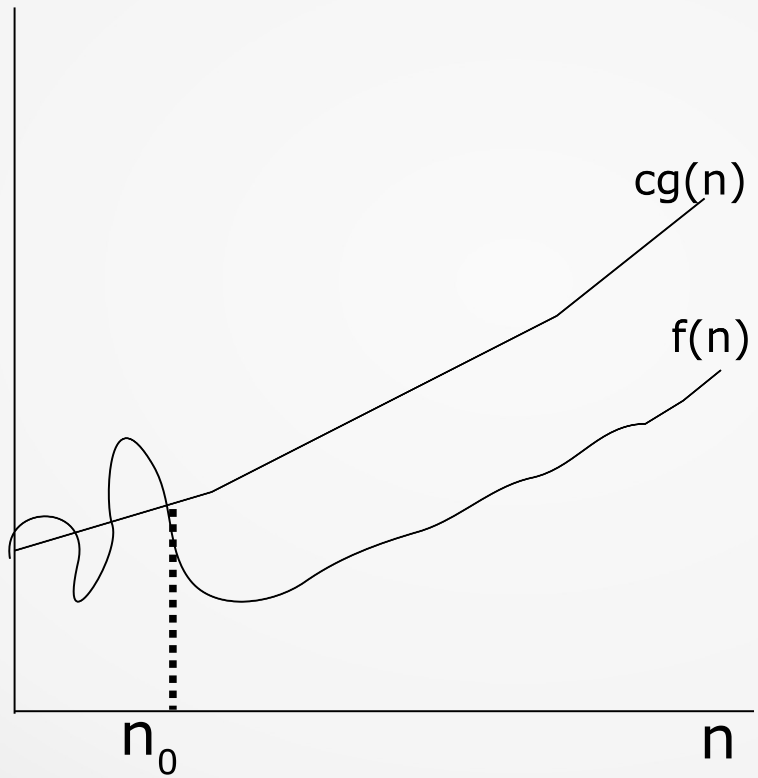


Asymptotic Analysis

- 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 $5n^2+5n+3$, the most dominating factor is $5n^2$
- Capturing this dominating factor is the purpose of asymptotic notations

Big-Oh Notation

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



$f(n) = O(g(n))$

Big-Oh Example

- Show 7n-2 is O(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 $n_0 >= 1$ such that $3n^3 + 20n^2 + 5 <= cn^3$ for $n >= n_0$
  - 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

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).

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 no 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

- Growth rate for different functions [Goodrich]

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,096	262,144	1.84×10^{19}
128	7	128	896	16,384	2,097,152	3.40×10^{38}
256	8	256	2,048	65,536	16,777,216	1.15×10^{77}
512	9	512	4,608	262,144	134,217,728	1.34×10^{154}

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

- 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.