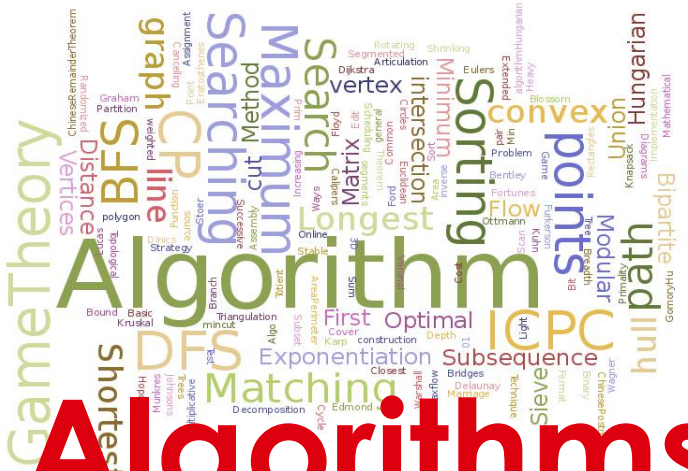
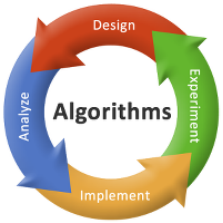


Algorithms - Examples and The Best Practices

Course: Algorithms

Faculty: Dr. Rajendra Prasath



Autumn 2018

Algorithms - Examples and The Best Practices

This class covers a few **Examples** with their complexity analysis and **the Best Practices**. This lecture illustrates the derivation of the **Worst Case Complexity** of a few selected problems and their analysis in detail.

Recap: Complexity Analysis

- **Computational Complexity of the Algorithms**
  - **Best Case Analysis**
  - **Average Case Analysis and**
  - **Worst Case Analysis**

  - **Big O – Notation, Omega Notation, Theta Notation and the choice to Data Structures**

- How do we optimize the running time and space needed to solve the given problem?
  - Running Time - **Minimize** or Maximize?
  - Space Needed - **Minimize** or Maximize?

Simple / Efficient Algorithms

- Running Times of the algorithms may vary based on n

- Estimating runtime
  - Considering the input size
  - Growth in the order of magnitude of the input
  - Perform tight / upper bound analysis

  - Compute the space requirements and choose appropriate data structures
  - Efficient implementations with the right choice of the programming language for the given task

Estimating Runtime – 1

- What is the runtime of g(n)?

```
void g(int n) {
        for (int i = 0; I < n; ++i)
                f( );
}
```

Runtime g(n) ≈ n • runtime (f(n))

Estimating Runtime – 2

- What is the runtime of g(n)?

```
void g(int n) {
        for (int i = 0; i < n; ++i)
                for (int j = 0; j < n; ++j)
                        f( );
}
```

Runtime g(n) ≈ n^2 • runtime (f(n))

Estimating Runtime – 3

- What is the runtime of g(n)?

```
void g(int n) {
        for (int i= 0; i < n; ++i)
                for(int j = 0; j <= i; ++j)
                        f( );
}
```

Runtime g(n) ≈ (1 + 2 + 3 + ••• + n) • runtime (f(n))

≈ (n (n+1) / 2) • runtime (f(n))

Complexity: $O(n^2)$

Big O - Algorithms

A method to characterize the execution time of an algorithm:

- Adding two square matrices is $O(n^2)$
- Searching in a dictionary is O(log n)
- Sorting a vector is O(n log n)
- Solving Towers of Hanoi is $O(2^n)$
- Multiplying two square matrices is $O(n^3)$
-
-
-

- Count the dominating terms … discard constants

Ranking Complexity

Function	Common name
$n!$	factorial
2^n	exponential
n^d, $d > 3$	polynomial
n^3	cubic
n^2	quadratic
$n\sqrt{n}$	
$n \log n$	quasi-linear
n	linear
$\sqrt{n}$	root - n
$\log n$	logarithmic
1	constant

Big-O: Examples

$T(n)$	Complexity
$5n^3 + 200n^2 + 15$	$O(n^3)$
$3n^2 + 2^{300}$	$O(n^2)$
$5\log_2 n + 15\ln n$	$O(\log n)$
$2\log n^3$	$O(\log n)$
$4n + \log n$	$O(n)$
2^{64}	$O(1)$
$\log n^{10} + 2\sqrt{n}$	$O(\sqrt{n})$
$2^n + n^{1000}$	$O(2^n)$

- Count dominating terms … discard constants

Complexity Analysis

- Let us assume that f() has complexity O(1)

- for(int i= 0; i< n; ++i) f();
  ➔ O(n)

- for(int i= 0; i< n; ++i)
  for(int j = 0; j < n; ++j) f();
  ➔ $O(n^2)$

- for(int i= 0; i< n; ++i)
  for(int j = 0; j <= i; ++j) f();
  ➔ $O(n^2)$

Complexity Analysis

- Let us assume that f() has complexity O(1)

- ```
  for(int i= 0; i< n; ++i)
      for(int j = 0; j < n; ++j)
          for(int k = 0; k < n; ++k) f();
  ```
  ➔ $O(n^3)$

- ```
  for(int i= 0; i< m; ++i)
      for(int j = 0; j < n; ++j)
          for(int k = 0; k < p; ++k) f();
  ```
  ➔ O(mnp)

Best Practices

- Understanding the given problem
- Choosing right approach for solving the given problem
  - Perform tight bound analysis
  - Adaptable algorithm with increasing size of input
- Know the limitations
  - Explore ways to handle the optimization of the limitations
- Code Profiling
  - In terms of runtime… Representative Simulation
- Handling Memory Management
  - How to handle increasing number of variables?
- How to avoid wasting CPU Cycles?
  - Efficient use of CPU cycles … even in Parallel executions
- Handle Loops efficiently
  - Avoid branching and I/O operations inside a loop
  - Process Data in blocks

Pitfalls

- How do we use the available memory efficiently?
  - Compute the required CPU cycles and plan your algorithm execution
  - Use of excessive memory will decrease the performance and increase the run time
  - Avoid performing I/O operations
    - Avoid writing to and/or reading from Disks too often
  - Perform trade-off between memory and space available in the given system
  - Avoid overloading the file systems
    - This would slowdown the simulation or the execution of the algorithm

A few problems

Let us try a Few Problems:

- Write an algorithm to add two numbers
- Write an algorithm to find the largest among three numbers
- Write an algorithm to find all the roots of quadratic equation
- Write an algorithm to find the factorial
- Write an algorithm to check prime number

Add two numbers

- Write an algorithm to add two numbers

Begin
1: Declare variables num1, num2 and sum
2: Read values num1 and num2
3: Add num1 and num2 and assign result to sum

sum←num1+num2

4: Print sum
End

Largest among 3 numbers

- Write an algorithm to find the largest among three different numbers

Begin

- 1: Declare variables a, b and c
- 2: Read variables a, b and c
- 3: If a>b

```
        If a>c
            Display a is the largest number
        Else
            Display c is the largest number.
    Else
        If b>c
            Display b is the largest number.
        Else
            Display c is the greatest number.
```

End

Solve a Quadratic Equation

- Write an algorithm to find all roots of a quadratic equation $ax^2+bx+c=0$

Begin
1: Declare variables a, b, c, D, x1, x2, rp and ip;
2: Calculate discriminant
    D←b2 - 4ac
3: If D ≥ 0
        r1←(-b+√D)/2a
        r2←(-b-√D)/2a
        Display r1 and r2 as roots.
    Else
        Calculate real part and imaginary part
        rp←b/2a
        ip←√(-D)/2a
        print rp+j(ip) and rp-j(ip) as roots
End

Help among Yourselves?

- **Perspective Students** (having CGPA above 8.5 and above)
- **Promising Students** (having CGPA above 6.5 and less than 8.5)

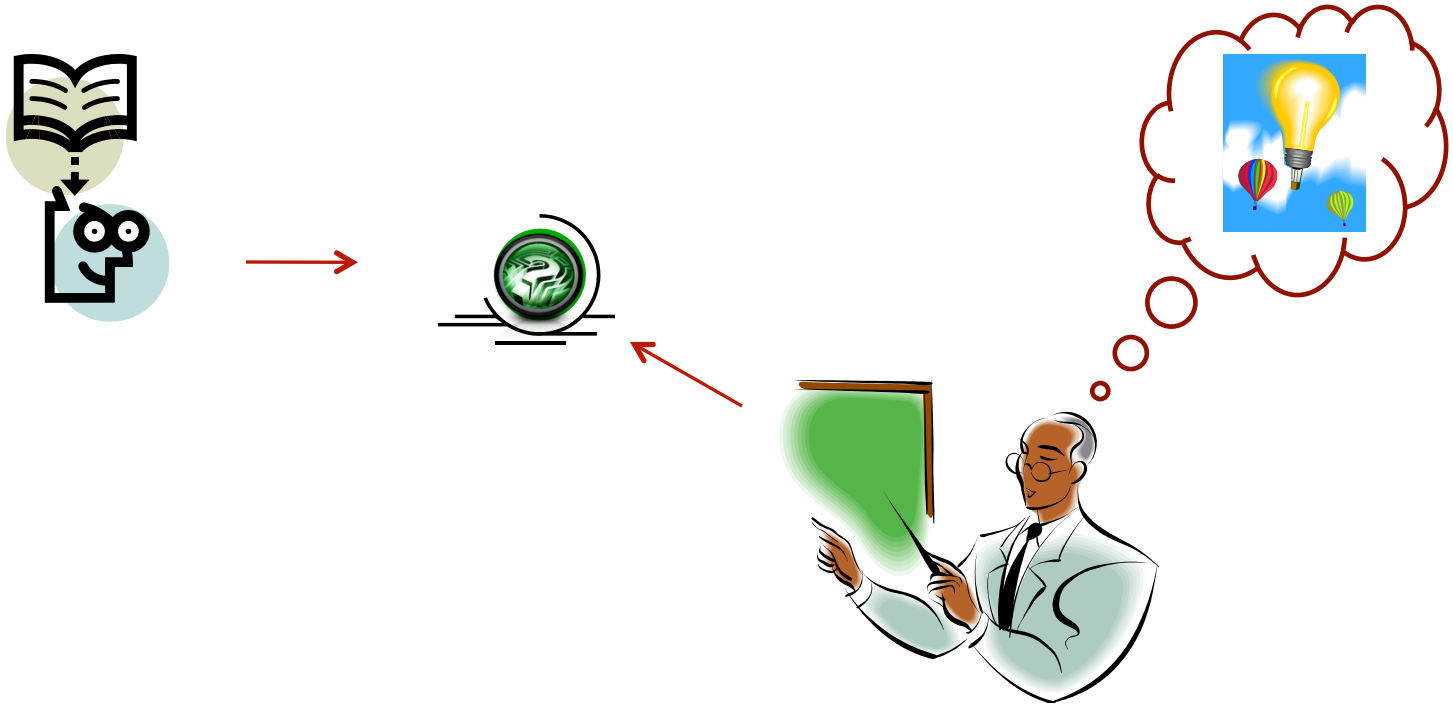
- **Needy Students** (having CGPA less than 6.5)
  - Can the above group help these students? (Your work will also be rewarded)

- You may grow a culture of **collaborative learning** by helping the needy students

Assistance

- You may post your questions to me at any time
- You may meet me in person on available time or with an appointment
- TA s would assist you to clear your doubts.
- You may leave me an email any time (email is the best way to reach me faster)

Thanks …



… Questions ???