ESO208A: Computational Methods in Engineering

Richa Ojha

Department of Civil Engineering

IIT Kanpur

Acknowledgements: Profs. Abhas Singh and Shivam Tripathi (CE)

Comparison

Which of these two algorithms is better?

1. Minimum Round-off errors (Condition no. is small)
2. Minimum storage requirement
3. Minimum computational time
4. Programming ease- Subjective

Computing Time

- Speed of computer
- Programming language
- Input Data
- Algorithm

Comparison

Computational or Algorithm Complexity

- Instead of measuring time in micro-seconds, we measure time in terms of number of basic steps executed by algorithm.
- Basic steps: (+, -, ×, /, assignment, comparison)
- Instead of representing algorithm complexity as a single no. we represent it in terms of size of data

Comparison: Algorithm Complexity

Example 1: Sum of n numbers, X=[x1,x2,x3….xn]

```
Sum = 0
For i = 1 to n
    sum = sum + x(i)
end
```

Operations

- Sum =0 (Assignment operation)
- Within the for loop (n assignments, n summations)

Total no. of operations = n

Comparison: Algorithm Complexity

Example 2: Sum and product of n numbers, X=[x1,x2,x3….xn]

```
Sum =0
product =1
for i=1 to n
    Sum = Sum + x(i)
    product = product * x(i)
end
```

Operations

- Sum =0, product =0 (Assignment operation=2)
- Within the for loop (2n assignments, n summations, n products= 2n)

Total no. of operations = 2n

Comparison: Algorithm Complexity

Example 3: Sum of all possible pairs, X=[x1,x2,x3….xn]

```
for i = 1 to n
    for j = 1 to n
        sum(i,j) = x(i) + x(j)
    end
end
```

Total no. of operations = n^2

Comparison: Algorithm Complexity

Two things:

1) Worst Case Scenario

Find a number x_0 in the vector X

```
f = 0 ; i = 0
while f == 0
    i = i + 1
    if x(i) == x0
        f = 1
    end
end
```

The number of basic steps depends on the location of x_0

Comparison: Algorithm Complexity

Two things:

2) Asymptotic Analysis

- Any algorithm is sufficiently efficient for small input.
- When comparing algorithms for computational time one is interested in very large inputs
- As a proxy for "very large" asymptotic analysis that consider size of input data tending to infinity
- "Big O" gives an upper bound on the asymptotic growth of the algorithm
- The complexity of the function/algorithm is $O(n^2)$ it means that for the worst case $O(n^2)$ steps are needed to estimate function value when n is very large

Comparison: Algorithm Complexity

Two things:

2) Asymptotic Analysis

- If the computation time is the sum of multiple terms. Keep the number which has the largest growth rate and drop the others.
- So, if no. of basic steps are n^2+n+c
- As $n \rightarrow \infty$, n^2 is what we are worried about.

Comparison: Algorithm Complexity

Common Complexity Classes

	Constant	$O(1)$
	Logarithmic	$O(\log(n))$
	Linear	$O(n)$
	log. linear	$O(n \log n)$
Polynomial	Quadratic	$O(n^2)$
	Cubic	$O(n^3)$
	Exponential	$O(c^n)$ $c > 1$

Comparison: Algorithm Complexity

Computational Complexity of GE and GJ

Recap.

$$\sum_{i=1}^{n} 1 = n$$

$$\sum_{i=k}^{n} 1 = n-k+1$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$$

$$O(n^2/2)$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \quad O\left(\frac{n^3}{3}\right)$$

Comparison: Algorithm Complexity

Gauss Elimination

(a)

```
DOFOR k = 1, n − 1
  DOFOR i = k + 1, n
    factor = a_{i,k} / a_{k,k}
    DOFOR j = k + 1 to n
      a_{i,j} = a_{i,j} − factor · a_{k,j}
    END DO
    b_i = b_i − factor · b_k
  END DO
END DO
```

(b)

```
x_n = b_n / a_{n,n}
DOFOR i = n − 1, 1, −1
  sum = b_i
  DOFOR j = i + 1, n
    sum = sum − a_{i,j} · x_j
  END DO
  x_i = sum / a_{i,i}
END DO
```

Pseudo code for Gauss elimination
(Source: Chapra and Canal

Comparison: Algorithm Complexity

Gauss Elimination

(a)

```
DOFOR k = 1, n − 1
  DOFOR i = k + 1, n
    factor = a_i,k / a_k,k
    DOFOR j = k + 1 to n
      a_i,j = a_i,j − factor · a_k,j
    END DO
    b_i = b_i − factor · b_k
  END DO
END DO
```

On the first pass, k=1

- The limits of middle loop are 2 to n
- The number of iterations in the middle loop will be

$$\sum_{i=2}^{n} 1 = n - 2 + 1 = n - 1$$

- For every iteration in the middle loop,
  - The number of multiplication/division operations
    $1 + n - 2 + 1 + 1 = n + 1$
  - The number of subtraction
    $n - 2 + 1 + 1 = n$
- Total multiplication for the first pass $= (n - 1)(n + 1)$
- The total number of subtraction operations
  $= (n - 1)n$

Comparison: Algorithm Complexity

Gauss Elimination

(*a*)

```
DOFOR k = 1, n − 1
  DOFOR i = k + 1, n
    factor = a_{i,k} / a_{k,k}
    DOFOR j = k + 1 to n
      a_{i,j} = a_{i,j} − factor · a_{k,j}
    END DO
    b_i = b_i − factor · b_k
  END DO
END DO
```

Outer Loop k	Middle Loop i	Addition/Subtraction flops	Multiplication/Division flops
1	2, n	$(n-1)(n)$	$(n-1)(n+1)$
2	3, n	$(n-2)(n-1)$	$(n-2)(n)$
.	.		
.	.		
.	.		
k	$k+1, n$	$(n-k)(n+1-k)$	$(n-k)(n+2-k)$
.	.		
.	.		
.	.		
$n-1$	n, n	$(1)(2)$	$(1)(3)$

Comparison: Algorithm Complexity

Gauss Elimination

The total addition/subtraction operations can be computed as

$$\sum_{k=1}^{n-1}(n-k)(n+1-k) = \sum_{k=1}^{n-1}[n(n+1) - k(2n+1) + k^2]$$

Applying some of the relationships mentioned earlier:

$$[n^3 + O(n)] - [n^3 + O(n^2)] + \left[\frac{1}{3}n^3 + O(n^2)\right] = \frac{n^3}{3} + O(n)$$

By doing similar analysis for multiplication and division.

$$[n^3 + O(n^2)] - [n^3 + O(n)] + \left[\frac{1}{3}n^3 + O(n^2)\right] = \frac{n^3}{3} + O(n^2)$$

Total number of floating point operations:

$$\frac{2n^3}{3} + O(n^2)$$

Comparison: Algorithm Complexity

Gauss Elimination

Backward substitution

No of steps $\quad n^2 + O(n)$

Total Gauss elimination

$$\frac{2n^3}{3} + O(n^2) + n^2 + O(n)$$

$$= \boxed{\frac{2}{3}n^3 + O(n^2)} \qquad O(n^3) \quad O\left(\frac{2}{3}n^3\right)$$

Gauss Jordan

No of steps $\quad n^3 + n^2 - n$

$$= \boxed{n^3 + O(n^2)}$$

Summary

- How to determine algorithm complexity?