Lecture notes for Week 3: Complexity

by Ken Clowes

Table of contents

1 Topics....2
1.1 Textbook portions covered....2
2 External links....2
3 Lecture 7 (Friday, 21 January 2005)....2
3.1 Announcements....2
3.2 Remarks....2
3.3 Program performance: an example....4
3.4 Big-O notation....4
4 Lectures 8/9 (Tuesday, 25 January 2005)....5
4.1 Announcements....5
4.2 Examples BigO....5
4.3 Examples BigOmega....5
4.4 Examples BigTheta....5
4.5 Math....5
4.6 Analyzing pseudocode or C....5
4.7 Solving recurrences....6
5 Suggested Problems....6

1. Topics

1. Big-O notation
2. Big-Omega notation
3. Big-Theta notation

1.1. Textbook portions covered

Introduction to Algorithms (Cormen et al.)

Chapters 3 and 4

Engineering Algorithms...(Clowes "online book")

Chapter 4

2. External links

- Wikipedia (http://en.wikipedia.org/wiki/Big_O_Notation)
- NIST (http://www.nist.gov/dads/HTML/theta.html) (U.S. National Institute of Standards)

3. Lecture 7 (Friday, 21 January 2005)

3.1. Announcements

- Lecture notes now available (http://www.ee.ryerson.ca/~courses/ele428/clowesNotes) .

3.2. Remarks

1. The nitty-gritty of rigorous mathematical analysis of algorithmic complexity can be intimidating.
2. The formal mathematical definitions might obscure the the basic simplicity and importance of this topic.
3. Hence, we begin with some simple, informal examples of where we are going.

3.2.1. Sample objectives

- Once we have finished, you will be able to determine the complexity of many C source code functions *by inspection*. (i.e. You will be able to determine the BigTheta complexity in less than 30 seconds.)
- Here are the examples:

1. ```
   int f(n)
   ```
   ```

```
{
   return n+5;
}
```

Constant time. i.e.: BigTheta(1)

2.
```
int f(n)
{
   int i;
   if (n > 123)
      i = n*n*n*n + 6;
   else
      i = n;
   return i;
}
```

Constant time. i.e.: BigTheta(1)

3.
```
int f(n)
{
   int i, t = 0;
   int vector[10000000];
   for(i = 0; i < n; i++)
     t += vector[i];
   return t;
}
```

Linear time. i.e.: BigTheta(n)

4.
```
int f(n)
{
   int i, j, t = 0;
   int matrix[10000000][10000000];
   for(i = 0; i < n; i++)
     for(j = 0; j < n; j++)
       t += matrix[i][j];
   return t;
}
```

Quadratic time. i.e.: BigTheta(n^2)

5.
```
int f(n)
{
   int i, t = 0;
   double x;
   int vector[10000000];
   for(i = 0; i < n; i++)
     for(x = 1; x < n;  x *= 1.1)
       t += vector[i];
```

```
   return t;
}
```

Supralinear time. i.e.: BigTheta(n log n)

3.3. Program performance: an example

Suppose 5 programmers write software to solve some problem. All programs work. When the size of the problem is 1000, the following times are measured:

A	B	C	D	E
1	10	600	1000	10000

Table 1: Measured times (in microseconds) for n = 1000

Which is best (for, say, n=10,000,000)?

Answer: Insufficient data! We also need to know the algorithmic complexity of each implementation.

For example, if we know the complexity is *linear*, then increasing the problem size by a factor of *x* will increase the time by a factor of *x*. Similarly, if the size of a *quadratic* algorithm is increased by a factor of *x*, the time will be multiplied by *x^2*.

Knowing the complexity of an implementation, we can calculate the approximate times as summarized in the following table:

n	A (*k1 1.01^n*) exponential	B (*k2 n^2*) quadratic	C (*k3 n* log *n*) n log n (supralinear)	D (*k4 n*) linear	E (*k5 n*) linear
1000	1	10	600	1000	10000
10,000,000	> 10^1000 years	10^9 microseconds = 16 minutes	14 seconds	10^7 microseconds = 10 seconds	10^8 microseconds = 100 seconds

Table 2: Measured times for n = 1000/Calculated times for n = 10,000,000

3.4. Big-O notation

1. Summary of Big-O definition: (refer to textbook/links for formal definition)

   f(n) is O(g(n)) if and only if
   f(n) < *k g(n)* for very large *n*.

4. Lectures 8/9 (Tuesday, 25 January 2005)

4.1. Announcements

- **QUIZ:** The quiz is next week: Friday, Feb 4. The quiz covers the first 4 lecture weeks (up to and including Pointers and Dynamic allocation) and the first 3 labs.
- My lecture notes now include all 13 weeks of the course. The notes for lectures that have not yet been delivered provide only preliminary outlines.

4.2. Examples BigO

1. The values of *k* and *n_0* are arbitrary (anything that proves the inequality).
2. If *f(n) = O(g(n))* then *f(n)* is also Big-O of any function that grows faster than *g(n)*.

4.3. Examples BigOmega

1. If *f(n) = BigOmega(g(n))* then *f(n)* is also Big-Omega of any function that grows slower than *g(n)*.

4.4. Examples BigTheta

1. Tip: If *f(n))* is the sum of terms, then it is Big-Theta (and hence BigO and BigOmega) of the fastest growing term (after eliminating multiplicative constants).

4.5. Math

You are responsible for:

1. Floor and ceiling functions.
2. O(log n!) = n log n
3. Closed-form equations for arithmetic and geometric series.
4. lg means log_2.
5. log^k n means (log n)^k
6. Note: log n grows more slowly than *any* polynomial. (That's why we say that BigTheta(n log n) is *almost* as good as linear complexity.)
7. Note: Any exponential grows faster than any polynomial.
8. Note: log log n grows more slowly than log^k n. (k # 1)

4.6. Analyzing pseudocode or C

1. Primitive statements take constant (BigTheta(1)) time.
2. Any combination of primitive statements which are executed only once (or not at all!) is BigTheta(1).

3. When loops are present, the problem is to determine the number of iterations as a function of problem size (i.e. linear, quadratic, logarithmic, etc.)
4. **Examples**
   - Refer to Sample Objectives section earlier in this week's notes.
   - See also *Engineering Algorithms...(Clowes "online book")* Chapter 4, Section 5.

4.7. Solving recurrences

1. Analysis of recursive algorithms (eg. mergeSort) often requires finding the closed-form solution to a recurrence.
2. **Method 1:**
   - Calculate values of T(n) by hand starting from the base case and then directly using the recurrence to obtain result for larger values of n.
   - Find a pattern and guess the closed-form solution.
   - Prove it by mathematical induction.
   - Example: analysis of mergeSort done previously. (see also *Engineering Algorithms...(Clowes "online book")* problem 4.6 and solution which was also done in class.)
3. Other methods will be covered in the next lecture.

5. Suggested Problems

Introduction to Algorithms (Cormen et al.)

- 3-1
- 3-3 (except iterated logarithm)
- 3-4

Engineering Algorithms...(Clowes "online book")

- 4.1
- 4.2
- 4.3
- 4.4
- 4.6
- 4.7