

MIT OpenCourseWare  
<http://ocw.mit.edu>

6.006 Introduction to Algorithms  
Spring 2008

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.

## Lecture 2: More on the Document Distance Problem

### Lecture Overview

Today we will continue improving the algorithm for solving the document distance problem.

- Asymptotic Notation: Define notation precisely as we will use it to compare the complexity and efficiency of the various algorithms for approaching a given problem (here Document Distance).
- Document Distance Summary - place everything we did last time in perspective.
- Translate to speed up the ‘Get Words from String’ routine.
- Merge Sort instead of Insertion Sort routine
  - Divide and Conquer
  - Analysis of Recurrences
- Get rid of sorting altogether?

### Readings

CLRS Chapter 4

### Asymptotic Notation

#### General Idea

For any problem (or input), parametrize problem (or input) size as  $n$ . Now consider many different problems (or inputs) of size  $n$ . Then,

$$\begin{aligned} T(n) &= \text{worst case running time for input size } n \\ &= \max_{X: \text{Input of Size } n} \text{running time on } X \end{aligned}$$

How to make this more precise?

- Don’t care about  $T(n)$  for small  $n$
- Don’t care about constant factors (these may come about differently with different computers, languages, ...)

For example, the time (or the number of steps) it takes to complete a problem of size  $n$  might be found to be  $T(n) = 4n^2 - 2n + 2 \mu s$ . From an asymptotic standpoint, since  $n^2$  will dominate over the other terms as  $n$  grows large, we only care about the highest order term. We ignore the constant coefficient preceding this highest order term as well because we are interested in rate of growth.

### Formal Definitions

1. **Upper Bound:** We say  $T(n)$  is  $O(g(n))$  if  $\exists n_0, \exists c$  s.t.  $0 \leq T(n) \leq c \cdot g(n) \forall n \geq n_0$

Substituting 1 for  $n_0$ , we have  $0 \leq 4n^2 - 2n + 2 \leq 26n^2 \forall n \geq 1$

$\therefore 4n^2 - 2n + 2 = O(n^2)$

Some semantics:

- Read the ‘equal to’ sign as “is” or  $\epsilon$  belongs to a set.
- Read the  $O$  as ‘upper bound’

2. **Lower Bound:** We say  $T(n)$  is  $\Omega(g(n))$  if  $\exists n_0, \exists d$  s.t.  $0 \leq d \cdot g(n) \leq T(n) \forall n \geq n_0$

Substituting 1 for  $n_0$ , we have  $0 \leq 4n^2 + 22n - 12 \leq n^2 \forall n \geq 1$

$\therefore 4n^2 + 22n - 12 = \Omega(n^2)$

Semantics:

- Read the ‘equal to’ sign as “is” or  $\epsilon$  belongs to a set.
- Read the  $\Omega$  as ‘lower bound’

3. **Order:** We say  $T(n)$  is  $\Theta(g(n))$  iff  $T(n) = O(g(n))$  and  $T(n) = \Omega(g(n))$

Semantics: Read the  $\Theta$  as ‘high order term is  $g(n)$ ’

### Document Distance so far: Review

To compute the ‘distance’ between 2 documents, perform the following operations:

**For each of the 2 files:**

Read file

Make word list

Count frequencies

Sort in order

+ op on list  $\Theta(n^2)$

double loop  $\Theta(n^2)$

insertion sort, double loop  $\Theta(n^2)$

**Once vectors  $D_1, D_2$  are obtained:**

Compute the angle

$\arccos\left(\frac{D_1 \cdot D_2}{\|D_1\| \|D_2\|}\right)$   $\Theta(n)$

The following table summarizes the efficiency of our various optimizations for the Bobsey vs. Lewis comparison problem:

<u>Version</u>	<u>Optimizations</u>	<u>Time</u>	<u>Asymptotic</u>
V1	initial	?	?
V2	add profiling	195 s	
V3	wordlist.extend(...)	84 s	$\Theta(n^2) \rightarrow \Theta(n)$
V4	dictionaries in count-frequency	41 s	$\Theta(n^2) \rightarrow \Theta(n)$
V5	process words rather than chars in get words from string	13 s	$\Theta(n) \rightarrow \Theta(n)$
V6	merge sort rather than insertion sort	6 s	$\Theta(n^2) \rightarrow \Theta(n \lg(n))$
V6B	eliminate sorting altogether	1 s	a $\Theta(n)$ algorithm

The details for the version 5 (V5) optimization will not be covered in detail in this lecture. The code, results and implementation details can be accessed at [this link](#). The only big obstacle that remains is to replace Insertion Sort with something faster because it takes time  $\Theta(n^2)$  in the worst case. This will be accomplished with the Merge Sort improvement which is discussed below.

## Merge Sort

Merge Sort uses a divide/conquer/combine paradigm to scale down the complexity and scale up the efficiency of the Insertion Sort routine.

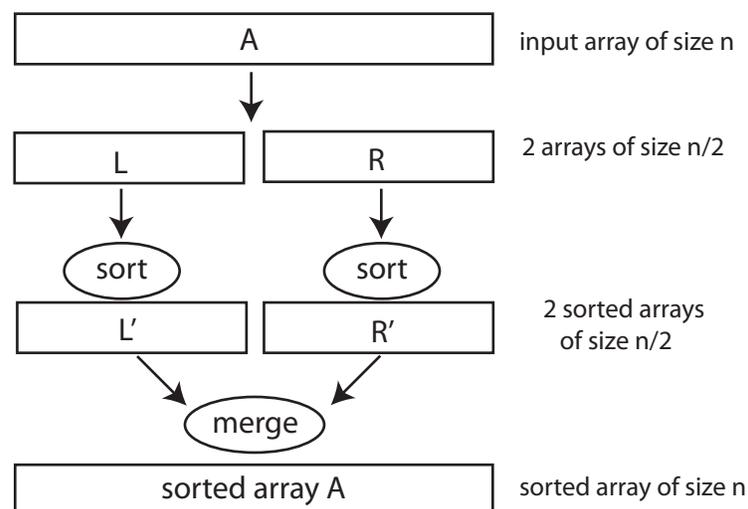


Figure 1: Divide/Conquer/Combine Paradigm

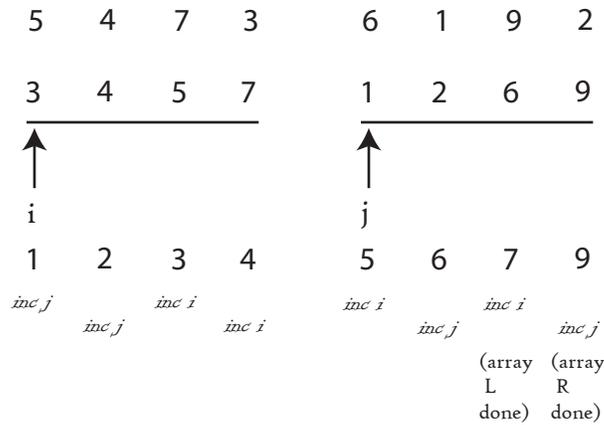


Figure 2: “Two Finger” Algorithm for Merge

The above operations give us  $T(n) = \underbrace{C_1}_{\text{divide}} + \underbrace{2T(n/2)}_{\text{recursion}} + \underbrace{C \cdot n}_{\text{merge}}$

Keeping only the higher order terms,

$$\begin{aligned}
 T(n) &= 2T(n/2) + C \cdot n \\
 &= C \cdot n + 2 \times (C \cdot n/2 + 2(C \cdot (n/4) + \dots))
 \end{aligned}$$

Detailed notes on implementation of Merge Sort and results obtained with this improvement are available [here](#). With Merge Sort, the running time scales “nearly linearly” with the size of the input(s) as  $n \lg(n)$  is “nearly linear” in  $n$ .

## An Experiment

Insertion Sort  $\Theta(n^2)$   
 Merge Sort  $\Theta(n \lg(n))$  if  $n = 2^i$   
 Built in Sort  $\Theta(n \lg(n))$

- Test Merge Routine: Merge Sort (in Python) takes  $\approx 2.2n \lg(n) \mu\text{s}$
- Test Insert Routine: Insertion Sort (in Python) takes  $\approx 0.2n^2 \mu\text{s}$
- Built in Sort or sorted (in C) takes  $\approx 0.1n \lg(n) \mu\text{s}$

The 20X constant factor difference comes about because Built in Sort is written in C while Merge Sort is written in Python.

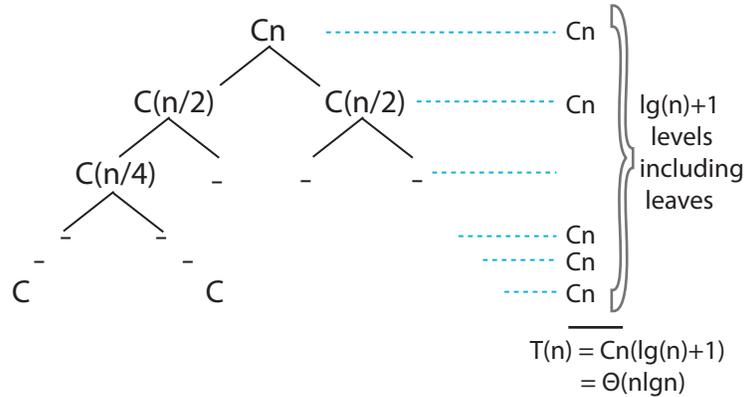


Figure 3: Efficiency of Running Time for Problem of size  $n$  is of order  $\Theta(n \lg(n))$

**Question:** When is Merge Sort (in Python)  $2n \lg(n)$  better than Insertion Sort (in C)  $0.01n^2$ ?

**Aside:** Note the 20X constant factor difference between Insertion Sort written in Python and that written in C

**Answer:** Merge Sort wins for  $n \geq 2^{12} = 4096$

**Take Home Point:** A better algorithm is much more valuable than hardware or compiler even for modest  $n$

See recitation for more Python Cost Model experiments of this sort ...