Lecture 3: Merge sort and Asymptotic Notation Introduction

CS303: Algorithms

Prepared on January 13, 2014

1 Review

We learned insertion sort and studied its correctness. We also discussed how to analyze the efficiency of algorithms: worst case, best case, and average case. We prefer the worst case in general because it offers the upper bound of an algorithm's efficiency.

2 Introduction to Asymptotic Notations

When we compare algorithm efficiency, we can propose to compare the running time of algorithms. But that depends on the computer/programmer $\rightarrow$ compare them for relative speed (on the same machine with the same programmer) $\rightarrow$ Big idea: asymptotic analysis (why algorithm analysis is so successful)
Adopt asymptotic notation
$\theta(n)$: drop low order terms and ignore leading constants
e.g. As n $\rightarrow$ inf, $\theta(n^2)$ always beats $\theta(n^3)$

2.1 Difference amongst various efficiency classes

Hypothetically, if we are dealing with a problem and we design several algorithms with different efficiencies, What do we expect about their performances?
We compare $log(n)$, n and n^2 If I use three computers with the following speed,
My first pc Pentium 3: 300M Hz (2^{28} operations per second, flops)
My current computer: 3G Hz (2^{32} flops)
One of the supper computers: 2^{66} flops

Input Size n	log(n)	n	n^2
100,000	62ns	2.33×10^4ns	.14ns
10,000,000	87ns	2.33×10^6ns	1360ns
1,000,000,000	111ns	.23s	.013s
100,000,000,000	136ns	23.3s	2.26m
1,000,000,000,000	149ns	233s	3.76h

Now let's analyze the worst case running time of the insertion sort
T(n): the total work in the big for loop

$T(n) = \sum_{j=2}^{n} j * c = c \times \sum_{j=2}^{n} j = \frac{(n-1)(n+1)}{2} = \theta(n^2)$
Q:/ Is insertion fast?

A:/ moderately so for small n (less than 30), not at all for large n
Can we do better?

2.2 Merge sort

```
Merge sort A[1,... , n]
1. if n==1, done
2. recursively sort A[1,... ,⌈n/2⌉] and A[⌈n/2⌉+1, ..., n]
3. merge 2 sorted sublist
```

2.2.1 Merge sub-routine

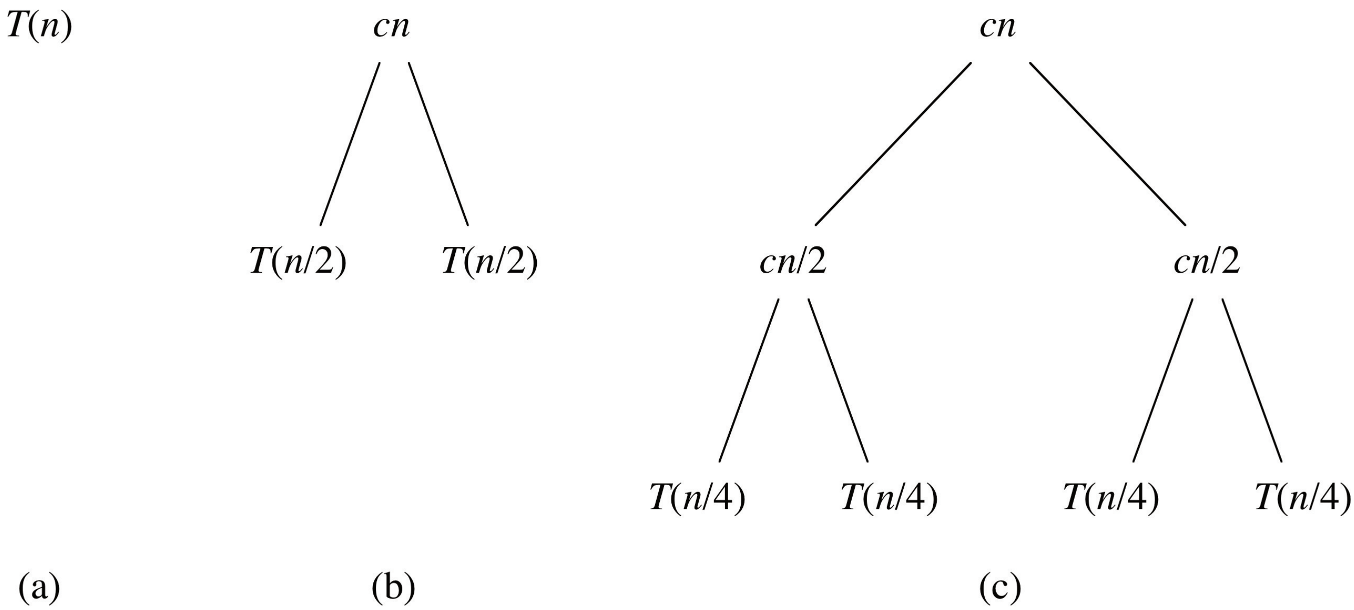
sublist 1: 2 7 13 20
sublist2: 1 4 11 12

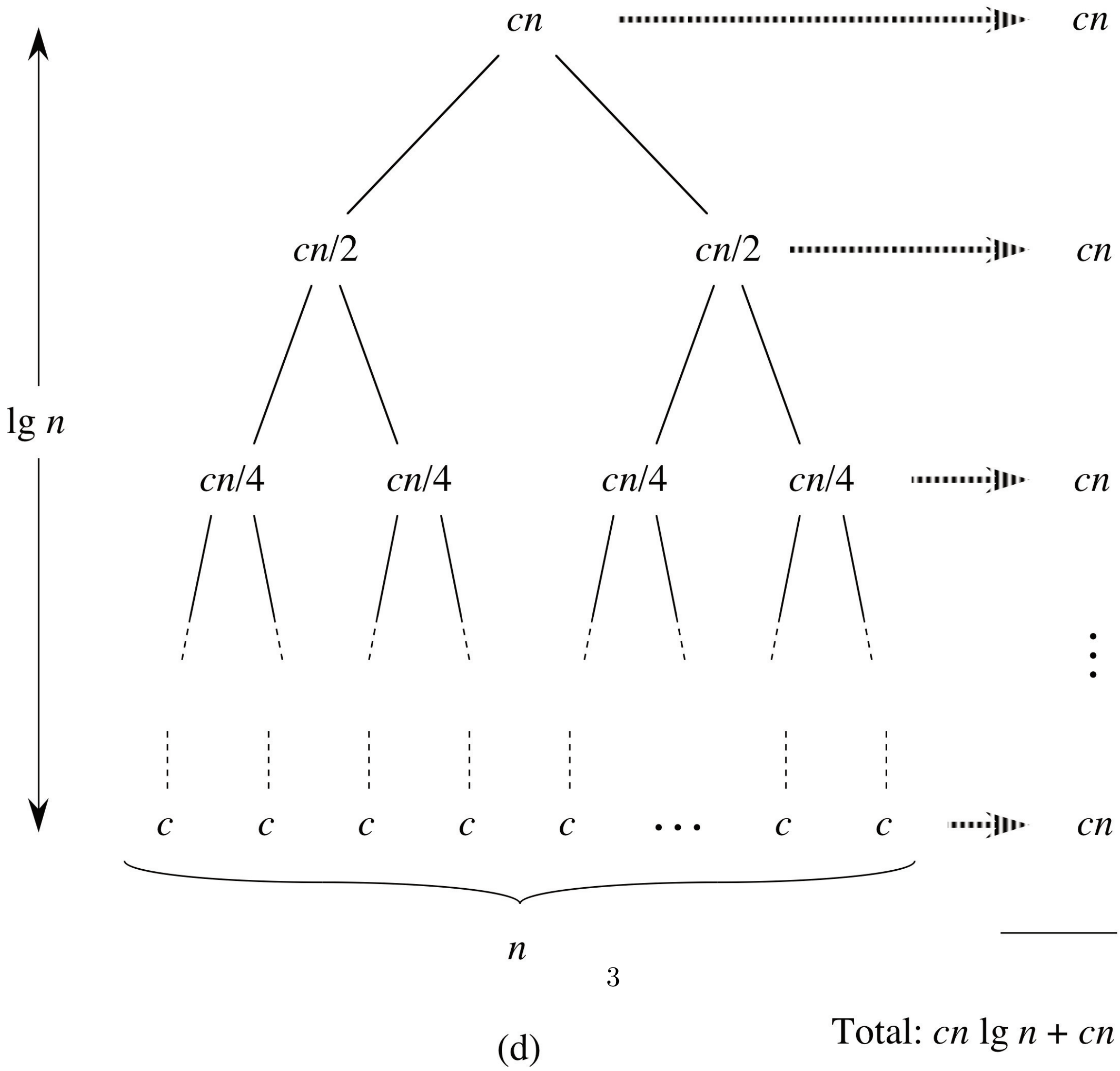
Idea: pick the smallest element in the two list and put it in the output array. Use i, j as pointers that points to the next element of the two sub arrays

Q:/ What is the worst case for this merge sub-routine?
A:/ Time is $\theta(n)$ on n total elements

Q:/ What is the running time of merge sort?
A:/ T(n)=constant time ($\theta(1)$ for step 1) + $T(n/2)$ + $T(n/2)$ +$\theta(n)$
note: we are throwing away the ceil here. It is ok and provable. $\rightarrow$ **for algorithms, so long as you are strict and precise, you can be as sloppy as you want.**

If we rewrite $\theta(n)$ as cn then from the recursion tree on the next page, we can see that the running time of merge sort is $\theta(nlgn)$


T(n)
cn
T(n/2) T(n/2)
cn
cn/2 cn/2
T(n/4) T(n/4) T(n/4) T(n/4)
(a)
(b)
(c)


cn
cn/2 cn/2
cn/4 cn/4 cn/4 cn/4
c c c c c … c c
lg n
n
cn
cn
cn
cn
Total: cn lg n + cn
(d)