Lec1 Analysis of Algorithms

Analysis of algorithms: the theoretical study of computer program performance(*main) and resource usage (e.g. communication, memory).

What's more important than performance?

correctness, cost (programmer time), simplicity, robustness, maintenance, modularity, security, scalability

Why performance is important?

- line between feasible and infeasible (e.g. requirements for real time streaming)
- universal standard to quantify code

  e.g. use Java for greater functionality, but Java is slow

Sorting Problem

Input: a sequence $< a_1, a_2, ..., a_n >$ of numbers

Output: permutation $< a_1', a_2', ..., a_n' >$ such that $a_1' <= a_2' <= ... <= a_n'$

Insertion sort

- maintain an invariant of an array that part of array is sorted
- goal for the loop is to increase the length of sorted array
- how to insert? 1) move the things and copy until find the right place 2) insert

Ex.

Input	8	2	4	9	3	6
R1	2	8	4	9	3	6
R2	2	4	8	9	3	6
R3	2	4	8	9	3	6
R4	2	3	4	8	9	6
R5	2	3	4	6	8	9

worst-case analysis

when input in **reverse order**

如何对操作数目计数？ counting memory references: how many times access some variables

$T(n) = \sum_{j=2}^{n} \theta(j)$ (i 从j-1 循环到0) $= \theta(n^2)$ 算数级数arithmetic series

Is insertion sort fast?

- moderately so, for small n
- not at all for large n

Merge sort

Merge sort $< a_1, a_2, \ldots, a_n >$

Steps: Time T(n)

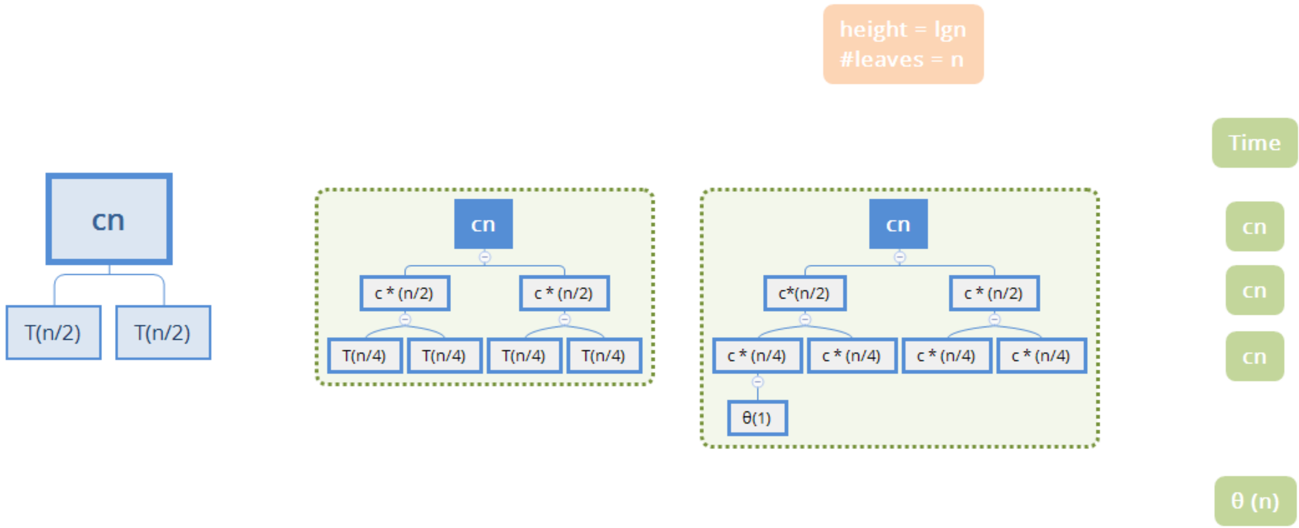
1. If $n = 1$, done θ(1)
2. recursively sort $< a_1, \ldots, a_{[n/2]} >$ and $< a_{[n/2]+1, \ldots, a_n} >$ 递归排序 前半 后半数据
   2*T(n/2)
3. Merge 2 sublists θ(n)
   1. Ex. sublist 1: 2, 7, 13, 20 sublist 2: 1, 9, 11, 12
      where is the smallest element? **always at the head**
   2. merge过程的每一步 比较两个数 at the offset的大小
   3. $T(n) = \theta(n)$ on total n elements LINEAR TIME 线性时间

Recurrence 递归表达式:

$$T(n) = \begin{cases} \theta(1) & n = 1 \\ 2T(n/2) + \theta(n) & n > 1 \end{cases}$$

Simply recurrence: recursion tree 递归树

$T(n) = 2T(n/2) + cn$, const $c > 0$



$$leaves = 1 + 2^1 + s^2 + ... + 2^{lgn} = \frac{1*(1-2^{lgn})}{1-2} = n$$

$Time = lgn * cn + \theta(n) = \theta(nlgn)$ faster than $\theta(n^2)$

在输入规模充分大，归并排序更快!

Runtime analysis

- depends on input (already sorted, worst case: reverse order)
- depends on input size (参数化parametrize input size)
- upper bounds = represent guarantee to the users e.g.这个程序最多需要多久

Kinds of analysis

- worst-case (most time)
  - $T(n)$ = max time n an input of size n
- average-case (sometimes)
  - $T(n)$ = expected time over all inputs of size n
  - weighted average: runtime * probability of runtime
  - Need assumption of distribution e.g. uniform distribution
- best-case (bogus 假象)
  - cheat: take any slow algorithm and check for some particular input
  - not reflect majority cases how algorithm works

Runtime depends on computer

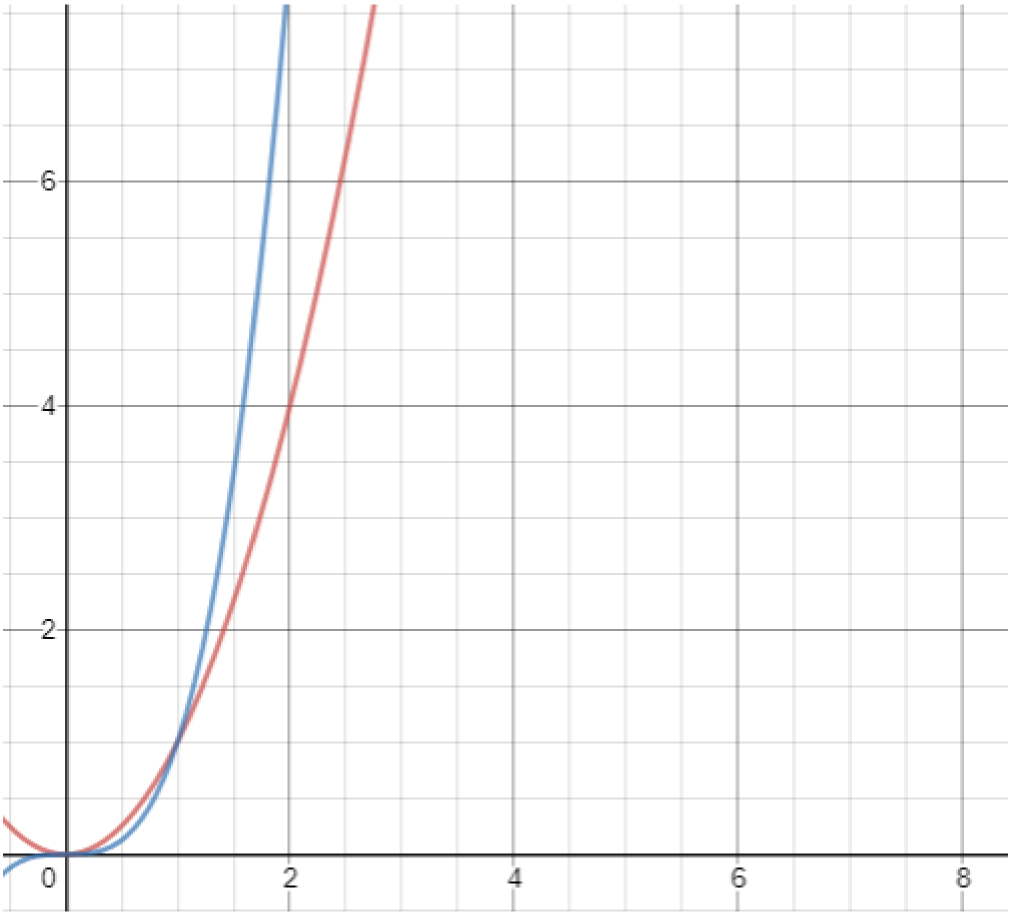
- usually compare in relative speed (on same machine)
- absolute speed (on different machines)

Asymptotic analysis 渐进分析

1. Ignore machine-dependent constants
2. look at the growth of running time $T(n)$ instead of actual running time

Asymptotic Notations

- θ notation
  - drop low order terms and ignore leading constants 弃去低阶项，忽略常数因子
  - e.g. $3n^3 + 90n^2 - 5n = \theta(n^3)$
  - as $n \to \infty$, $\theta(n^2)$ always beat $\theta(n^3)$ algorithms



  - at some point n_0, $\theta(n^2)$ algorithms always cheaper than $\theta(n^3)$
  - engineering view: n_0 might be too large that computers couldn't execute -> also interested in slower algorithms (asymptotically slower but still faster on reasonable size of inputs)
  - balance between math and engineering sense