Week 2

The Master Method

general mathematical tool for analysing running time of D&C algorithms

Integer multiplication

grade-school mult. algo – $\Theta(n^2)$

recursive way

$$x \cdot y = 10^n ac + 10^{n/2}(ad + bc) + bd \quad *$$

x: a, b $\quad$ y: c, d

$T(n)$ – max number of operators

$T(1) \le a$ – constant (base case)

$$n > 1: T(n) \le \underbrace{4T\left(\frac{n}{2}\right)}_{\text{recursive}} + \underbrace{O(n)}_{+} \quad \text{(general case)}$$

(Gauss) recursive algo

$$ac^{1},\ bd^{2},\ (a+b)(c+d)^{3}$$

$$ad + bc = 3 - 1 - 2$$

Base Case: $T(1) \leq C$

Gen Case: $T(n) \leq \underbrace{3T\left(\frac{n}{2}\right)}_{\text{rec. call}} + \underbrace{O(n)}_{+}$

The Master Method (Master Theorem)

assumption: all problems have equal size

$$T(n) \leq aT\left(\frac{n}{b}\right) + O(n^d)$$

don't depend on n
$\begin{cases} a - \text{number of recursive calls } (\geq 1) \\ b - \text{shrinkage factor } (>1) \\ d - \text{exp running time done outside of recursive calls} \end{cases}$

$$T = \begin{cases} O(n^d \log n) & \text{if } a = b^d \text{ (case 1)} \\ O(n^d) & \text{if } a < b^d \text{ (case 2)} \\ O(n^{\log_b a}) & \text{if } a > b^d \text{ (case 3)} \end{cases}$$

Examples

Merge Sort

$$a=2,\ \underbrace{b=2,\ d=1}$$

$$\underbrace{a=2 \qquad b^d=2}_{\text{case 1.}}$$

$$\Downarrow$$

$$T(n) \le O(n^d \log n) = O(n \log n)$$

Binary Search

$$a=1,\ b=2,\ d=0,\ \text{case 1}$$

$$a=b^d=1$$

$$T(n) = O(\log n)$$

Recursive algo for int. multiplic #1

$$a=4,\ b=2,\ d=1$$

$$b^d=2<a,\ \text{Case 3}$$

$$T(n) = O(n^{\log_b a}) = O(n^{\log_2 4}) = O(n^2)$$

same as grade-school

Rec. algo for Int mult #2 (Gauss')

$a = 3, \; b = 2, \; c = 1$

$a = 3, \; b^d = 2$

$a > b^d$ (Case 3)

$T(n) = O(n^{\log_2 3}) \simeq O(n^{1.59})$

better than $O(n^2)$ the grade-school algo.

Strassen's Matrix Multiplication

$a = 7, \; b = 2 \; d = 2$

$b^d = 4 < a$ (case 3)

$T(n) = O(n^{\log_2 7}) = O(n^{2.81})$

better than the naive iterative algo

Fictitious Recurrence

$$T(n) = 2T\left(\frac{n}{2}\right) + \underbrace{O(n^2)}$$

same as merge sort, but n^2 instead of n

$a = 2$ $\quad b^d = 4 > a$ (case 2)
$b = 2$
$d = 2$

$$T(n) = O(n^2)$$

Proof

Assumptions

(i) $T(1) \leq c$

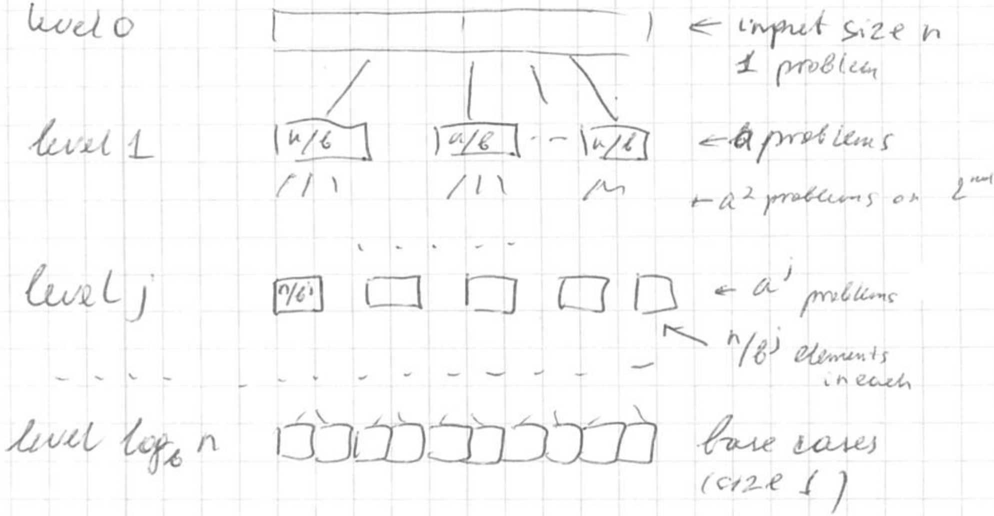
(ii) $T(n) \leq aT\left(\frac{n}{b}\right) + cn^d$

n - is a power of b

Idea: generalize MergeSort

At level j there are a^j subproblems and each of size n/b^j

The Recursion Tree



Total work at level j [ignoring work in recursive calls]

$$\le \underbrace{a^j}_{\text{number of level-}j\text{ subproblems}} \cdot \underbrace{c \cdot \left(\frac{n}{b^j}\right)^d}_{\text{work per level-}j\text{ subproblem}} = cn^d \cdot \left(\frac{a}{b^d}\right)^j$$

size of each level-j subproblem

Total Work:

$$c n^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j \quad (*)$$

Total work $\leq (*)$

forces of Evil

a = rate of subproblems proliferation (RSP)

b^d = rate of work shrinkage (RWS)

force of Good

if RSP < RWS
then the amount of work is decreasing with the recursion level j

if RSP > RWS
then the amount of work is increasing with the recursion level j

if RSP == RWS
then the amount of work is the same at every level

Intuition for the 3 cases

1. RSP = RWS   same amount $\Rightarrow$ expect $O(n^d \log(n))$

2. RSP < RWS

outdoes, most work at the root $\Rightarrow$ expect $O(n^d)$

3. RSP > RWS $\Rightarrow$ expect $O(\#\text{leaves})$

more work at each level, most work at the leaves

Proof II

$$TW \leq cn^d \times \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j \qquad (*)$$

a - number of rec. calls
b - shrinkage factor
c - work done outside of recursion

Case I

if $a = b^d$

$$cn^d \times \underbrace{\sum_{j=0}^{\log_b n} \left[\left(\frac{a}{b^d}\right)^j \right]}_{\log_b n + 1}$$

$\frac{a}{b^d} \to 1$, $\left(\frac{a}{b^d}\right)^j = 1$

$$(*) = cn^d \cdot (\log_b n + 1) =$$

$$O(n^d \cdot \log n)$$

Basic Sum Facts

for $r \neq 1$, we have

$$\underbrace{1 + r^2 + r^2 + \ldots + r^k}_{\text{RHS}} = \frac{r^{k+1} - 1}{\underbrace{r-1}_{\text{RHS}}}$$

1. if $r < 1$,

$$\text{RHS} \leq \frac{1}{1-r}$$ — a constant (independent of k)

2. if $r > 1$

$$\text{RHS} \leq r^k \underbrace{\left(1 + \frac{1}{r-1}\right)}_{\text{a constant}}$$

r^k: latest term

Case 2 $\quad a < b^d$

$$cn^d \times \underbrace{\sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j}_{\text{constant, independent of } n}$$

<1

constant,
independent of n
[by basic sum fact]

$$(*) = O(n^d)$$

(dominated by root!)

Case 3

$a > b^d$

>1

$$cn^d \times \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$

constant × largest term

$$(*) = O\left(n^d \cdot \left(\frac{a}{b^d}\right)^{\log_b n}\right) =$$

$$\cancel{(*) = O\left(n^d \log_b n\right)}$$

$$b^{-d\log_b n} = \left(b^{\log_b n}\right)^{-d} = n^{-d}, \text{ so}$$

$$(*) = O\left(a^{\log_b n}\right)$$

$a^{\log_b n} \leftarrow$ number of leaves of the recursion tree as $a^{\log_b n} = n^{\log_b a}$ ← simple to apply

$$\left[\text{since } (\log_b n)(\log_b a) = (\log_b a)(\log_b n)\right]$$

QED!

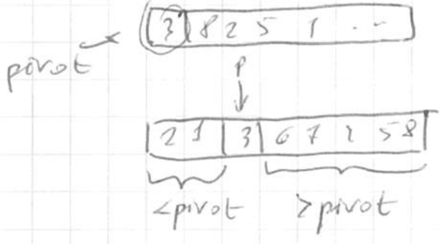
Quick Sort

$O(n \log n)$ on average
operates at place

Assume: all array elements are distinct

Key idea: Partition around a pivot

- pick an element
- rearrange array so
  - left of pivot => less than pivot
  - right of pivot => greater than pivot



1 partition is done in $O(n)$ time

2. reduces problem size

↑

D&C approach

Quick Sort (array A, length n):

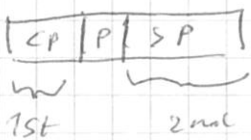
if $n = 1$ return

p = choose Pivot (A, n)

Partition A around p

Recursively sort 1^{st} part

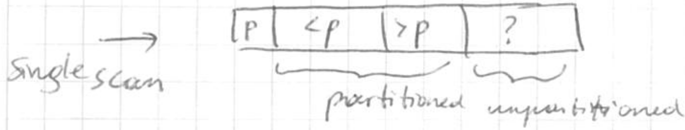
Recursively sort 2^{nd} part



Choose Pivot:

1^{st} element

Partition (in-place impl)



invariant:

```
Partition(A, l, r)          input: A[l...r]
    p = A[l]
    i = l+1
    for j = l+1 to r        // if A[j] > p, do nothing
        if A[j] < p
            swap A[j] A[i]
            i = i+1
    swap A[l] and A[i-1]
```

running time $O(n)$ where $n = r - l + 1$

Pivot

Running time depends on the quality of the pivot

good quality – divides into 2 equal halves

if always matches median ⇒

$\Rightarrow \Theta(n \log n)$

$$\left(T(n) = 2T\left(\tfrac{n}{2}\right) + \underbrace{O(n)}_{\text{choose pivot}} \right.$$

$$\left. T(n) = O(n \log n) \right)$$

Random Pivot.

Random pivots! ← It's „good enough" (not perfect)

Probability Review

(See Wikibook on Discrete Probability)

Sample Space Ω - all possible outcome
(finite set)

$$i \in \Omega \, ; \quad p(i) \geq 0$$

$$\sum p(i) = 1$$

Events

$S \subseteq \Omega$ - subset of the sample sets

$$p(S) = \sum_{i \in S} p(i)$$

Random Variable

$$X : \Omega \to \mathbb{R}$$

$$E[X] = \sum_{i \in \Omega} X(i) \cdot p(i)$$

expectation

$E[X+Y] =$
$= E[X] + E[Y]$