Lecture 10: Divide and Conquer algorithms

CS303: Algorithms

Last update: January 26, 2014

1 Review

- Efficiency Analysis: Asymptotic notations (O, Θ, Ω), Summation approach, substitution method, recurrence tree, and master theorem
- Algorithms: Fibonacci sequence generation, insertion sort, merge sort.

2 Divide and Conquer algorithms

Starting from this chapter, different algorithms will be discussed. We will use the mathematical analysis we have learned so far quite often (usually in a simple form)

2.1 divide and conquer idea

History: to conquer a land with a big powerful structure, you might want to divide the big land into small political/religious groups, preferably with family feuds involved. And if you can separate this big power structure into little power structures such that you dominate each little power structure then you can conquer all of them individually, as long as you make sure they don't get back together. That is divide and conquer as practiced, say, by the British.
It is a basic and powerful algorithm design techniques. $\rightarrow$ leads to recurrence
Steps:

1. Divide: split the big problem into small instances (smaller subproblems)
2. Conquer:solve each subproblem recursively
3. Combine: combine solutions into the solution of the original sub-problem

We have learned merge sort

1. Divide: divide the array into two halves (this is constant time step since we only look at the array as two halves)
2. Conquer: recursively sort each subarray
3. Combine: merge two sorted subarrays (linear time), this is the non-recursive part

$T(n) = 2T(n/2) + \Theta(n) \rightarrow$ there are 2 subproblems of size n/2. based on the master theorem, it is $\Theta(nlg(n))$.

2.2 Binary search

Find x in a sorted array $A[1, 2, ..., n]$ $left = 1, right = n$

1. Divide: compare x with $A[mid]$ where $mid = \lfloor (\rfloor (left + right)/2)$
2. Conquer: recurse in one subarray $A[left, \ldots, mid - 1]$ or $A[mid + 1, \ldots, right]$
3. Combine: do nothing (trivial)

Thus $T(n) = T(n/2) + \Theta(1) \rightarrow T(n) = \Theta(lgn)$

2.3 Powering of a number

Given a number x and an integer $n \geq 0$, compute x^n
Naive algorithm: $x * x * x.... * x \rightarrow$ so it is $\Theta(n)$ time
Using divide and conquer: Let's try to divide on n
Here we have to be careful with floor and ceiling
$x^n = (x^{n/2})^2$ for even n, $x^n = (x^{(n-1)/2})^2 * x$for odd n
Thus $T(n) = T(n/2) + \Theta(1) \rightarrow T(n) = \Theta(lgn)$

2.4 Fibonacci sequence

$$F_n = \begin{cases} 0 & \text{if n=0} \\ 1 & \text{if n=1} \\ F_{n-1} + F_{n-2} & \text{if n > 1} \end{cases}$$

We designed a linear algorithm to compute the nth term in the Fibonacci sequence. But if we use $F_n = \frac{\phi^n}{\sqrt{5}}$ for the Fibonacci sequence, we can find that and round to the nearest number. Where ϕ is the golden ratio $\frac{(1+\sqrt{(5)})}{2}$
Q:/ What's the efficiency?
A:/ if we use the log algorithm to calculate the power, then it can be $\Theta(lgn)$
Note: in a normal machine, we can't achieve this efficiency due to rounding in floating point numbers. Another approach can be used from matrix multiplication to achieve log efficiency.

2.5 Matrix multiplication

Given two matrices $A[a_{ij}]$ and $B[b_{ij}]$ of size $n(1 \leq i, j \leq n)$, find $C[c_{ij}] = A * B$
where $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$
Obvious algorithm: use brute force

```
for i=1 → n
    for j=1 → n
        c_ij = 0
        for k=1 → n
            c_ij = c_ij + a_ik  b_kj
```

Q:/ What's the efficiency? A:/$\Theta(n^3)$
Divide and conquer: nxn matrix can be divided into block matrices of size $n/2$ by $n/2$
So using the divide and conquer approach, we can compute the result using smaller matrix multiplications.

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

C A B

where all blocks are of size $n/2$.

e.g. $r = ae + bg$, $s = af + bh$, $t = ce + dg$, $u = cf + dh$

e.g. if $A = \begin{bmatrix} 1 & 2 & 0 & 2 \\ 0 & 0 & 0 & 1 \\ 1 & 3 & 0 & 5 \\ 2 & 5 & 0 & 1 \end{bmatrix}$ $B = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 0 & 3 \\ 3 & 0 & 0 & 5 \\ 2 & 1 & 1 & 2 \end{bmatrix}$

if you multiply A w/ B, the result (upper left corner) is

$$C = A * B = \begin{bmatrix} 8 & 2 & \cdots \\ 2 & 1 & \\ \cdots & & \cdots \end{bmatrix}$$

Using block method, upper left block $= \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix} * \begin{bmatrix} 2 & 0 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 2 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix}$

$$= \begin{bmatrix} 8 & 2 \\ 2 & 1 \end{bmatrix}$$

Thus there are 8 multiplication of matrices of size $n/2$ 4 additions$\rightarrow \Theta(n^2)$ Thus $T(n) = 8 * T(n/2) + \Theta(n^2) \rightarrow$ still $\Theta(n^3)$

Strassen's algorithm (divine inspiration, really clever)

Idea: reduce 8 to a smaller number. Basically reduce # of multiplications $\rightarrow$ reduce to 7

$P_1 = a(f-h)$ $P_3 = (c+d)e$ $P_5 = (a+d)(e+h)$

$P_2 = (a+b)h$ $P_4 = d(g-e)$ $P_6 = (b-d)(g+h)$

$P_7 = (a-c)(e+f)$

Thus

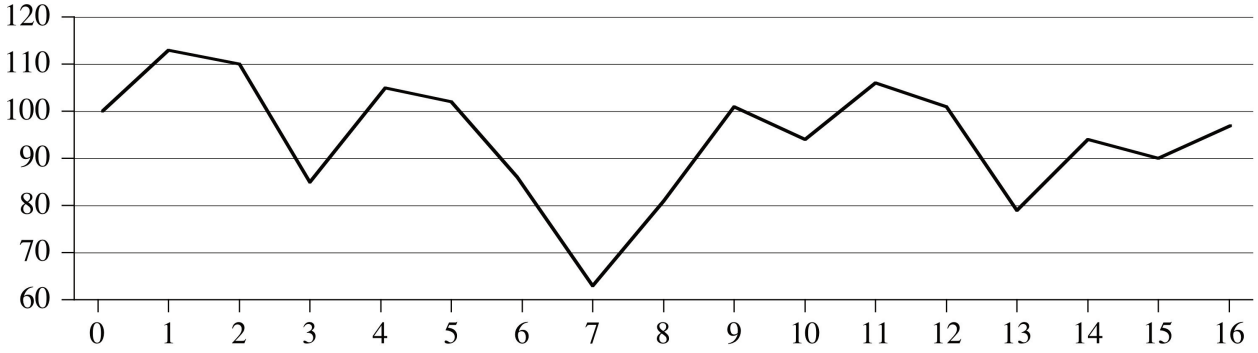
$r = P_5 + P_4 - P_2 + P_6$ $s = P_1 + P_2$

$t = P_3 + P_4$ $u = P_5 + P_1 - P_3 - P_7$

check
$S = p_1 + p_2 = a(f-h) + (a+b)h = af - ah + ah + bh = af + bh$
Thus $T(n) = 7T(n/2) + \Theta(n^2) \rightarrow T(n)$ is $\Theta(n^{lg7})$ which is $o(n^{2.81})$. It is better than naïve way for $n \geq 32$
Note: the best is $\Theta(n^{2.376})$

2.6 Max subarray

Given an array $A[1, ..., n]$, find a continuous segment of A such that the sum of that segment is max of any continuous segment in the array.
Application of sub-array problem can be used as how to buy and sell stocks



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

Brute force:
check each sub-array (there are ${}_nC_2$ subarrays. Thus $\Theta(n^2)$

Divide and conquer:
Idea: divide the array into two halves. Find the max subarray in the first half and second half or the one crossing the middle.. Pick the largest one
Q:/ Any problem with it?
A:/ It is possible that the max subarray can be corssing the middle point.

It turns out that we can find the one crossing the middle with linear time$\rightarrow$ just expand to the left and right until the sum doesn't increase.
$T(n) = 2T(n/2) + \Theta(n) \rightarrow T(n)$ is $\Theta(nlg(n))$