CS2102 Analysis and Design of Algorithms

Divide and Conquer II

M.Sc. Bryan Gonzales Vega
bgonzales.vega@gmail.com

University of Engineering and Technology

Lecture Content

1. Divide and conquer algorithms
    - Maximum Subarray Sum
    - Karatsuba Method
    - Strassen’s Algorithm

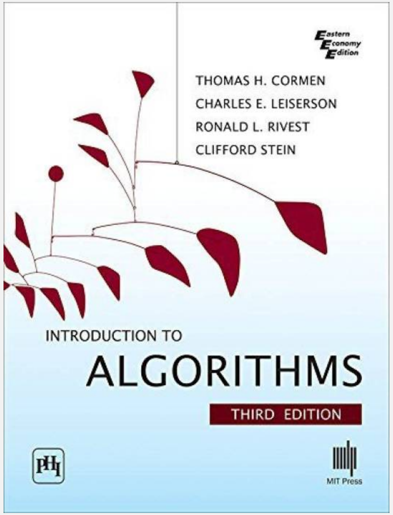
Divide and conquer algorithms

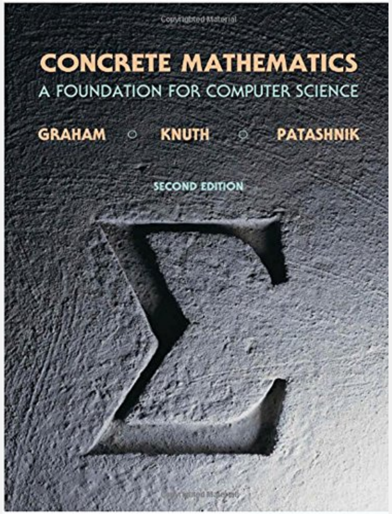
Introduction to Algorithms
[Cormen et al., 2009]

- Chap 02: Getting started
- Chap 03: Growth of functions
- Chap 04: Divide and conquer

Concrete Mathematics
[Graham et al., 1989]

- Chap 01: Recurrent problems
- Chap 02: Sums

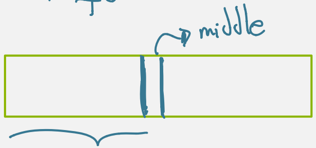




Warm up

Binary Search

Search for x



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

m.t $\Theta(\log n)$

divide : compare x against middle

conquer : recursively call bs. in 1 subarray → left / → right

combine : check if found.

Powering a number

given x, n compute x^n

Naive : $\underbrace{x \cdot x \ldots x}_{n \text{ times}}$ $\Theta(n)$ multiplications.

Divide and conquer

$$x^n \begin{cases} x^{n/2} \cdot x^{n/2} , & n \text{ is even} \\ x^{\frac{n-1}{2}} \cdot x^{\frac{n-1}{2}} \cdot x^1 , & n \text{ is odd} \end{cases}$$

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

m.t. $\Theta(\log n)$

Maximum Subarray Sum

Maximum Subarray Problem

The **maximum subarray** problem (MSP) involves selection of a segment of **consecutive** array elements that has the largest possible sum over all other segments in a given array of numbers.

rates of change

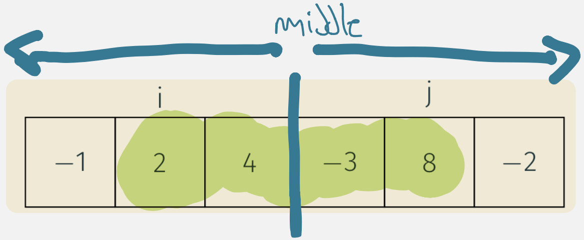
$$\begin{array}{cccccc} & i & & & j & \\ \hline -1 & \underbrace{2 \quad 4 \quad -3 \quad 8}_{\text{Maximum contiguous subarray}} & & & & -2 \end{array}$$

i.e. given an *array* we need to find the indices *i* and *j* such that the sum below is as large as possible.

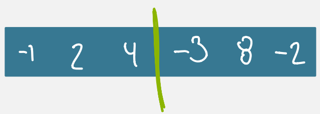
$$\sum_{x=i}^{j} array[x]$$

Q. What considerations should we have if all numbers are positives or negatives?

Cont'd



Divide and Conquer intuition

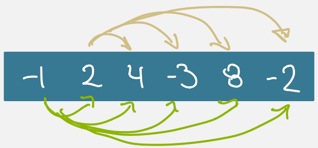


max sum is:

- entirely in the left side ←
- entirely in the right side ←
- <u>somewhere in the middle</u>

Brute force approach

$n = 6$



$i = 0$

$\}-1\{$	-1
$\}-1, 2\{$	1
$\}-1\ 2\ 4\{$	5
$\}-1\ 2\ 4\ -3\{$	2
$\}-1\ 2\ 4\ -3\ 8\{$	10
$\}-1\ 2\ 4\ -3\ 8\ -2\{$	8

$$n + n-1 + n-2 + \dots + 1 = \frac{n(n+1)}{2}$$

$$\Theta(n^2)$$

Cont'd

	i			j	
−1	2	4	−3	8	−2

low	i		middle	j	high
-1	2	4	-3	8	-2
0	1	2	3	4	5

$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
left-sum = −∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = −∞
sum = 0
for j = mid + 1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```

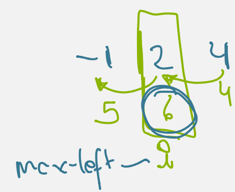
$\Theta(1)$

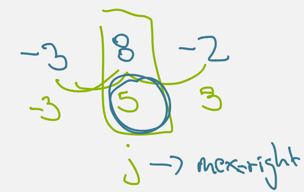
left $(n/2)$

right $(n/2)$

1 4 11

How can we get the max subarray that crosses the center line (middle)





crossing:

$T(n) = \Theta(n)$

Cont'd

```
FIND-MAXIMUM-SUBARRAY(A, low, high)                                   T(n)
if high == low
    return (low, high, A[low])      Θ(1)  // base case: only one element
else mid = ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY(A, low, mid)                            T(n/2)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)                       T(n/2)
    (cross-low, cross-high, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)                 Θ(n)
    if left-sum ≥ right-sum and left-sum ≥ cross-sum
        return (left-low, left-high, left-sum)                        Θ(1)
    elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
        return (right-low, right-high, right-sum)
    else return (cross-low, cross-high, cross-sum)
```

low	i		middle	j	high
-1	2	4	-3	8	-2
0	1	2	3	4	5

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

$$T(n) = 2T(n/2) + \Theta(n)$$

m.t. $\Theta(n \log n)$

```
if high = low
    return A[low];
else
    mid = ⌊low + high / 2⌋
    left = MS(A, low, mid)
    right = MS(A, mid+1, high)
    middle = crossing(A, low, mid, high)
    return max(left, right, middle)
```

Kadane's algorithm $O(n)$

↳ dynamic programming

Karatsuba Algorithm

Is a **fast multiplication** algorithm proposed by the Russian mathematician Anatoly Karatsuba that improves the *grade school* multiplication procedure given below:

			1	2	3	4
		x	5	6	7	8
			9	8	7	2
		8	6	3	8	
	7	4	0	4		
6	1	7	0			
7	0	0	6	6	5	2

Figure 1: Anatoly Karatsuba.

- As it is shown above if we consider the multiplication of 2 numbers of **n-digits**, the complexity of the brute force algorithm is $\Theta(n^2)$

Q. How can we apply divide and conquer strategy to the multiplication of 2 n-digit numbers?

Intuition

$$\begin{array}{r} p \\ \times\, q \\ \hline r \end{array} = \begin{array}{r} a\,b \\ c\,d \\ \hline \end{array}$$

Divide and Conquer

$p = \boxed{1\ 2}\boxed{3\ 4}$ $a=12$ $b=34$

$q = \boxed{5\ 6}\boxed{7\ 8}$ $c=56$ $d=78$

$1234 = 12 \times 10^2 + 34$ (shift)

$5678 = 56 \times 10^2 + 78$

$$\begin{array}{r} a \cdot 10^{n/2} + b \\ c \cdot 10^{n/2} + d \\ \hline \end{array}$$

$$p \cdot q = (a\,10^{n/2} + b)(c \cdot 10^{n/2} + d)$$

$$= \underbrace{ac\,10^{n/2}}_{O(n)} + \underbrace{ad\,10^{n/2}}_{O(n)} + \underbrace{bc\,10^{n/2}}_{O(n)} + bd$$

$$T(n) = 4\,T(n/2) + O(n)$$

m.t. $\Theta(n^2)$

$$\underbrace{(a+b)(c+d)}_{\text{only 1 multiplication}} = ac + ad + bc + bd$$

$$= ac\,10^{n/2} + ad\,10^{n/2} + bc\,10^{n/2} + bd$$

$$= ac\,10^{n/2} + (ad + bc)\,10^{n/2} + bd$$

we don't care about ad or bc but sum.

$K_{ac} = a \cdot c$

$K_{bd} = b \cdot d$ — 3 multiplications

$K_{(a+b)(c+d)} = (a+b)(c+d)$

$K_{(ad+bc)} = K_{(a+b)(c+d)} - K_{ac} - K_{bd}$

$$= ac\,10^{n/2} + (ad + bc)\,10^{n/2} + bd$$

$$T(n) = 3T(n/2) + O(n)$$

m.t. $\Theta(n^{\log_2 3})$

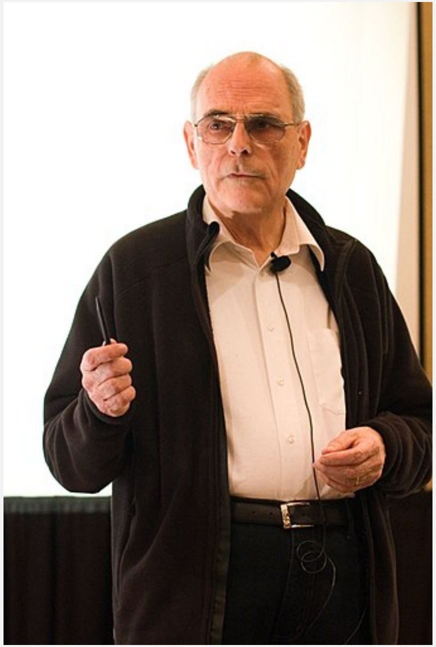


Figure 2: Volker Strassen.

Strassen's Algorithm

Strassen's algorithm is a **matrix multiplication** method proposed by the German mathematician Volker Strassen, that improves the asymptotic complexity of the regular multiplication between 2 matrices of size *nxn* presented below.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} r & s \\ t & u \end{pmatrix} \tag{1}$$

Strassen's expansions:

$$\begin{aligned}
k_1 &= a(f-h) & k_5 &= (a+d)(e+h) & r &= k_5 + k_4 - k_2 + k_6 \\
k_2 &= (a+b)h & k_6 &= (b-d)(g+h) & s &= k_1 + k_2 \\
k_3 &= (c+d)e & k_7 &= (a-c)(e+f) & t &= k_3 + k_4 \\
k_4 &= d(g-e) & & & u &= k_5 + k_1 - k_3 - k_7
\end{aligned}$$

Matrix Multiplication.

Def. Given 2 matrices A, B

$$C_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

for i=1 to n
  for j=1 to n
    for k=1 to n
      $c_{ij} = a_{ik} \cdot b_{kj}$

Brute force: $\Theta(n^3)$

Divide and Conquer

$$\underset{A}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \times \underset{B}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}} = \underset{C}{\begin{bmatrix} r & s \\ t & u \end{bmatrix}}$$

$r = ae + bg$

$s = af + bh$

$t = ce + dg$

$u = cf + dh$

recursive matrix multiplication

$$T(n) = 8T(n/2) + O(n^2)$$

m.t. $\Theta(n^3)$

Cont'd

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

$$\begin{aligned} k_1 &= a(f-h) & k_5 &= (a+d)(e+h) & r &= k_5 + k_4 - k_2 + k_6 \\ k_2 &= (a+b)h & k_6 &= (b-d)(g+h) & s &= k_1 + k_2 \\ k_3 &= (c+d)e & k_7 &= (a-c)(e+f) & t &= k_3 + k_4 \\ k_4 &= d(g-e) & & & u &= k_5 + k_1 - k_3 - k_7 \end{aligned}$$

$r = k_5 + k_4 - k_2 + k_6$

$r = (ae + \cancel{ah} + \cancel{de} + \cancel{dh}) + (\cancel{dg} - \cancel{de}) - (\cancel{ah} + \cancel{bh}) + (bg + \cancel{bh} - \cancel{dg} - \cancel{dh})$

$\underline{r = ae + bg}$

$T(n) = 7T(n/2) + \Theta(n^2)$

$r = ae + bg$

$s = af + bh$

$t = ce + dg$

$u = cf + dh$

Strassen: m.t. $\Theta(n^{\log_2 7})$

References i

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009).
Introduction to algorithms.
MIT press.

Graham, R., Graham, R., Knuth, D., Knuth, D., and Patashnik, O. (1989).
Concrete Mathematics: A Foundation for Computer Science.
A foundation for computer science. Addison-Wesley.

Cont'd