Instructor comments do not write in this section

CS412 - The Maximum Subarray Problem

The Problem: Given an array (contiguous input sequences) of 'n' real numbers, find the maximum sum in any subarray of the input.

· eg., A = [31, -41, 59, 26, -53, 58, 97, -93, -23, 84]

Let's see the trivial cases:

i) When all elements in the array ($\mathbb{R}^+$) are positive real numbers.
   Solution: The entire array has the max sum.
   eg.,
   P = [31, 41, 59, 26, 53, 58, 97] : $\Theta(n)$

ii) When all elements of the array are negative realnumbers.
   Solution: The min element is the max sum subarray
   Again: $\Theta(n)$ {Find_Min}

· Non-trivial: When the numbers can be any $\mathbb{R}$ (real numbers). We need to decide should a negative number be included hoping that positive numbers on both sides will compensate for the negative contribution?

Let's begin with checking Brute-Force (or naive) first.

Instructor comments do not write in this section

The (Naïve) Brute-Force: $O(n^3)$

For each pair of integers L, U ($1 \leq L \leq U \leq N$) for an array of size N [1…N], compute the sum of A[L…U] and report the subarray with greatest (max) sum.

```
MaxSoFar ← 0
For L: 1 to N do
    For U: L to N do
        Sum ← 0
        For I: L to U do
            Sum ← Sum + A[I]

        MaxSoFar ← max(MaxSoFar, Sum)
```

Can we do better? Of course!

The (Smart) Brute-Force:

Exploiting that the array (contiguous) has the property that the sum of a subarray

$A[i \ldots j]$ is $A[i \ldots j-1] + A[j]$

e.g., $X = [A, B, C]$ or $X = [\underset{1}{11}, \underset{2}{-15}, \underset{3}{19}]$

There are $\frac{n(n+1)}{2}$ or 6 subarrays for this array of size 3.

Instructor comments do not write in this section

Left-to-right scan $X = [\underset{1}{11} \quad \underset{2}{-15} \quad \underset{3}{19}]$

Psuedocode

```
Max ← 0
For i: 1 to N do
    Sum ← 0
    For j: i to N do
        Sum ← Sum + X[j]
    Max ← max(Max, Sum)
```

Dry-run

i	j	X[i...j]	Sum	Max
1	1	X[1...1]	11	11
1	2	X[1...2]	-4	11
1	3	X[1...3]	15	15
2	2	X[2..2]	-15	15
2	3	X[2...3]	4	15
3	3	X[3...3]	19	19

Max Sum: 19 for X[3...3]

Can we do better? Of course!

The divide-and-conquer approach by Michael Shamos

$$T(n) = 2T\left(\frac{n}{2}\right) + n = \Theta(n \lg n)$$

Can we do better? Of course!

Kadane's linear time algorithm!

Instructor comments do not write in this section

Multiplying Long Integers:

The Karatsuba's Algorithm (Integer Multiplication)

Consider the problem of multiplying two numbers X and Y, say X = 23, Y = 17.

We know from the school.

$$\begin{array}{r} 23 \\ \times\ 17 \\ \hline 161 \\ 23\times \\ \hline 391 \end{array}$$

Let's assume that X and Y are two-bits (or in this case, two digits) each.
then,
$X \pm Y$ involves computing 2 partial products of size 2 and this is an $O(n^2)$ algorithm.

Can we do better?

Kartasuba in 1960s proposed a divide-and-conquer approach to improve this bound.

Instructor comments do not write in this section

Let X and Y are two n-bits (or n-digit).
Both can be represented in two integers each of n/2 bits.

Let's assume that n is in the powers of 2

X : $\boxed{A \mid B}$, $X = A \cdot 2^{n/2} + B$

$\underbrace{Y}_{n\text{-bit}}$: $\boxed{C \mid D}$, $Y = C \cdot 2^{n/2} + D$

eg., $X = \underbrace{9724}$ $= \underset{X_H}{97} \times \underset{r^{n/2}}{10^2} + \underset{X_L}{24}$

or, $X = X_H \cdot r^{n/2} + X_L$
$Y = Y_H \cdot r^{n/2} + Y_L$
{where, 'r' is 10 in case of base-10, or 2 in case of binary}

then,

$$XY = (X_H \cdot r^{n/2} + X_L)(Y_H \cdot r^{n/2} + Y_L)$$

$$= \underset{1}{X_H \cdot Y_H} \cdot r^n \underset{A}{+} (\underset{2}{X_H \cdot Y_L} \underset{B}{+} \underset{3}{X_L \cdot Y_H}) r^{n/2}$$

$$\underset{C}{+} \underset{4}{X_L \cdot Y_L}$$

Four multiplications and three additions: (of atmost n-bits)

$$T(n) = 4T\left(\frac{n}{2}\right) + \underbrace{O(n)}_{c.n \text{ (n bits)}}, \quad T(1) = 1$$

or,

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

Gauss' Trick: $bc + ad = (a+b)\cdot(c+d) - ac - bd$

Karatsuba's Insight: Instead of 04 subproblems, we can exploit Gauss' trick to reduce it to 3 subproblems

Note: In divide-and-conquer, reducing 'a' (the branching factor) will give a performance boost!

Reconciling

$$XY = (\underbrace{X_H \cdot Y_H}_{a})\, r^n + (\underbrace{X_H Y_L + \cancel{X_L} Y_H}_{e}) \cdot r^{n/2}$$

$$+ \underbrace{X_L Y_L}_{d}$$

Three subproblems: $XY = a r^n + e r^{n/2} + d$

[Nonhomogeneous recurrence relation]

Now,

$$e = [(X_H + X_L)\cdot(Y_H + Y_L) - \underbrace{a - d}_{\text{already computed}}]$$

$$\therefore T(n) = 3T\left(\frac{n}{2}\right) + O(n) \text{ or } O(n^{\log_2 3})$$

$$\text{or } O(n^{1.584\ldots})$$

Instructor comments do not write in this section

$$XY = \underbrace{(X_H Y_H)}_{1} r^n + \Big[\underbrace{(X_H + X_L) \cdot (Y_H + Y_L)}_{2}$$

$$- \underbrace{X_H Y_H}_{1} - \underbrace{X_L Y_L}_{3} \Big] r^{n/2}$$

$$+ \underbrace{X_L Y_L}_{3}$$

A huge pefomance gain when $n \to \infty$?

The Kartasuba's algorithm outpeforms the $O(n^2)$ [naïve] when n becomes really large $[\underbrace{n \geq 500 \text{ bits}}_{\text{astronomical scale!}}]$

Design Implications

* Keep Your # subproblems small (à as small)

* Keep your subproblems as balanced, i.e, not as lopsided as

$$\underbrace{T(n) = T\left(\frac{n}{16}\right) + T\left(\frac{n}{12}\right)}_{\text{lopsided!}} + O(n)$$