02. Divide-and-Conquer 1 / 25

02. Divide-and-Conquer CPSC 535 ∼ Fall 2019

Kevin A. Wortman



September 9, 2019



02. Divide-and-Conquer 2 / 25

Divide-and-Conquer

One of the big ideas of computer science problem solving

- 1. **Divide** a problem into smaller parts
- 2. Conquer the smaller problems recursively
- Combine the smaller solutions into one solution for the original problem

(The term carries some baggage from the age of imperialism.)

Divide-and-conquer, outside of algorithm design

- Software design; breaking features into classes, functions
- Networking; OSI seven layer model
- ► Parallel processing; MapReduce
- Software process; agile methods; sprints



02. Divide-and-Conquer 3 / 25

Divide-and-conquer at a high level

```
1: function DIVIDE-AND-CONQUER(INPUT)
        if INPUT is base case then
 2:
            return trivial base case solution
 3:
 4.
        else
            x_1, x_2, \dots, x_k = \text{divide INPUT into } k \text{ pieces (often 2)}
 5:
            s_1 = \text{DIVIDE-AND-CONQUER}(x_1)
 6:
 7:
            s_k = \text{DIVIDE-AND-CONQUER}(x_k)
8:
            S = \text{combine } s_1, \ldots, s_k \text{ into one solution}
9:
            return S
10:
        end if
11:
12: end function
```

02. Divide-and-Conquer 4 / 25

Time complexity recurrences

Recursive pseudocode leads to recurrences in run-time functions

Suppose base case is n=1 and takes $\Theta(1)$ time; in the recursive case we divide evenly into k pieces of size $\approx n/k$, recurse once on each, and spend f(n) time in the *divide* and *conquer* phases:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ kT(n/k) + f(n) & \text{if } n > 1. \end{cases}$$

Recall merge sort divides into k = 2 pieces, merge takes $\Theta(n)$ time:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

02. Divide-and-Conquer 5 / 25

Taking liberties with recurrences

General math: bound recurrences precisely including constant factors

Algorithm analysis: ordinarily bounding asymptotically; Θ notation will hide constant factors anyway; drop math details that can only impact constants and add clutter

- ▶ drop ceilings/floors, so write e.g. n/2 in lieu of $\lceil n/2 \rceil$ or $\lceil n/2 \rceil$ is more precise
- when the base case is $\Theta(1)$ time for n < c for some $c \in \Theta(1)$, don't bother writing it explicitly; so

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

is abbreviated as

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

02. Divide-and-Conquer 6 / 25

Maximum subarray problem

Maximum subarray problem

input: an array $\langle p_1, p_2, \dots, p_n \rangle$ where each $p_i \in \mathbb{R}$ is a *profit* (or

loss) on day i

output: indices s, e with $s \le e$, maximizing the total profit

$$\sum_{i=s}^{e} p_i$$

Applications

- buy then sell a stock/security
- pick opening/closing time of a retail store with slow periods
- computer vision, data mining: identify region most consistent with a pattern e.g. street striping

02. Divide-and-Conquer 7 / 25

Greedy fails

Straightforward greedy algorithm would be:

- buy at the lowest price or sell at the highest price
- incorrect; best "run" could be elsewhere
- (example)
- ▶ not always correct ⇒ not actually an algorithm

Brute force

```
Exhaustive search: try every legal start/end
 1: function BRUTE-FORCE-MAX-SUBARRAY(P)
 2:
       s = e = 1
    for i from 1 to n do
 3:
           for j from i to n do
 4:
               if (\sum p_i \dots p_i) > (\sum p_s \dots p_e) then
 5:
                  s = i, e = i
 6:
               end if
 7:
           end for
 8:
       end for
 9.
10:
        return (s, e)
11: end function
\Theta(n^3) time as written; can cache sums to achieve \Theta(n^2)
```

02. Divide-and-Conquer 9 / 25

Divide-and-conquer brainstorm

Divide: chop array in half into two smaller arrays L, R

Conquer: recursively compute maximum subarray in L and in R

Combine: maximum subarray of entire P could be

- 1. subarray entirely in *L*;
- 2. subarray entirely in R; or
- 3. crossing subarray that starts in L and ends in R

(exhaustive case analysis)

Theme with **combine**: choose best among small solutions (easy) or a distinct solution that crosses boundaries (trickier)

02. Divide-and-Conquer 10 / 25

Identify crossing subarray — try 1

Suppose the two pieces of P are P[low ...mid] and P[mid + 1...high]

Tempting to try all pairs of $s \in \{low, ..., mid\}$ and $e \in \{mid + 1, ..., high\}$

Would work, but

- ▶ time becomes $T(n) = 2T(n/2) + \Theta(n^2)$ which is $\Theta(n^2)$ by master theorem
- ▶ same time as brute force, but more complicated ⇒ not a win

02. Divide-and-Conquer 11 / 25

Identify crossing subarray — insight

Theme in algorithm design: in general, a more specific problem admits a faster and/or simpler algorithm

First try is not using the fact that a *crossing* subarray <u>must</u> cross *mid*

- substantially simplifies the search
- s is how far before mid; separately, e is how far after mid?
- ightharpoonup two separate 1D searches \implies two linear loops
- \triangleright $\Theta(n) + \Theta(n) = \Theta(2n) = \Theta(n)$ time
- versus: s is where, and e is how much later?
- ▶ 2D search \implies two nested loops $\implies \Theta(n^2)$ time
- ▶ location of the "2" is profound; $\Theta(2n) \ll \Theta(n^2)$

Identify crossing subarray — try 2

```
    function MAX-CROSSING-SUBARRAY(P, low, mid, high)

2:
3:
4:
5:
6:
8:
11:
12:
13:
        leftsum = rightsum = -\infty
        sum = 0
        for i from mid down to low do
           sum = sum + P[i]
           if sum > leftsum then
               leftsum = sum
               maxleft = i
           end if
         end for
          sum = 0
         for i from mid + 1 to high do
             sum = sum + P[i]
14:
15:
             if sum > rightsum then
                 rightsum = sum
16:
17:
18:
19:
                rightleft = i
             end if
         end for
         return (maxleft, maxright, leftsum + rightsum)
20: end function
```

 $\Theta(n)$ time

(Note scoping of maxleft, maxright, and that they are inevitably initialized.)

02. Divide-and-Conquer 13 / 25

Maximum subarray algorithm

```
1: function MAX-SUBARRAY(P. low, high)
2:
3:
4:
5:
6:
7:
8:
       if low == high then
           return (low, high, P[low])
       else
           mid = \lceil (low + high)/2 \rceil
           (leftlow, lefthigh, leftsum) = MAX - SUBARRAY(P, low, mid)
           (rightlow, righthigh, rightsum) = MAX - SUBARRAY(P, mid + 1, high)
           (crosslow, crosshigh, crosssum) = MAX - CROSSING - SUBARRAY(A, low, mid, high)
9:
           if leftsum > rightsum and leftsum > crosssum then
10:
                return (leftlow, lefthigh, leftsum)
                                                                                          ▷ entirely-left subarray
11:
            else if rightsum > leftsum and rightsum > crosssum then
12:
                return (rightlow, righthigh, rightsum)
                                                                                        entirely-right subarray
13:
14:
            else
                return (crosslow, crosshigh, crosssum)
                                                                                        ▷ mid-crossing subarray
15:
16:
            end if
         end if
     end function
```

02. Divide-and-Conquer 14 / 25

Maximum subarray analysis

D&C runtime is

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

Solves to $\Theta(n \log n)$, by master theorem, same as merge sort.

Brute force was $\Theta(n^2)$

- D&C is much faster
- perhaps counterintuitive due to recursion's reputation for sloth
- ▶ D&C benefits from observation that subarrays are contiguous, so extend in two directions from a middle
- brute force is oblivious to this
- human mathematical insight eliminates wasted effort

02. Divide-and-Conquer 15 / 25

Matrix multiplication

Matrix multiplication problem input: A, B each an $n \times n$ matrix output: matrix product C = AB

Recall notation: element at row i and column j of matrix A is denoted a_{ij}

Definition of matrix multiplication:

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

02. Divide-and-Conquer 16 / 25

Naïve matrix multiplication

```
1: function MATRIX-MULTIPLY(A, B)
        C = \text{new } n \times n \text{ matrix}
 2:
 3:
    for i from 1 to n do
            for j from 1 to n do
 4:
 5:
                c_{ii}=0
                for k from 1 to n do
 6:
                    c_{ii} = c_{ii} + a_{ik} \cdot b_{ki}
 7:
                end for
 8:
            end for
 9.
        end for
10:
11:
        return C
12: end function
\Theta(n^3) time
```

02. Divide-and-Conquer 17 / 25

Is naïve optimal?

The definition of matrix multiplication involves a sum that is iterated n times, for each of the $n \times n$ elements of C, which might seem to require exactly n^3 scalar multiply instructions, and imply an $\Omega(n^3)$ lower bound for matrix multiplication.

Surprise! Strassen's algorithm (1969) takes $O(n^{\lg 7}) = O(n^{2.81})$ time; more complicated Williams-Le Gall algorithm (2014) takes $O(n^{2.37})$ time

Insight: per the definition of matrix multiplication, some elements of A and B are multiplied together more than once; avoid duplicating these efforts.

02. Divide-and-Conquer 18 / 25

Moving to divide-and-conquer

Suppose n is an even power of 2, i.e. $n=2^k$ for $k \ge 0$ (Can preprocess A, B by adding padding zeroes, then trim the zeroes out of C.)

Divide A into four equal-size submatrices, and same for B, C.

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix},$$

so we can compute C as

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}.$$

02. Divide-and-Conquer 19 / 25

Moving to divide-and-conquer (continued)

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

can be broken down into four separate computations

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

each of which can be performed recursively.

02. Divide-and-Conquer 20 / 25

Divide-and-conquer matrix multiplication — try 1

```
1: function MMR(A, B)
        C = \text{new } n \times n \text{ matrix}
 2:
 3:
    if n == 1 then
 4:
             c_{11} = a_{11} \cdot b_{11}
        else
 5:
6.
             quadrisect A, B, C
             C_{11} = MMR(A_{11}, B_{11}) + MMR(A_{12}, B_{21})
 7:
             C_{12} = MMR(A_{11}, B_{12}) + MMR(A_{12}, B_{22})
8.
             C_{21} = MMR(A_{21}, B_{11}) + MMR(A_{22}, B_{21})
9:
             C_{22} = MMR(A_{21}, B_{12}) + MMR(A_{22}, B_{22})
10:
        end if
11:
12.
        return C
13: end function
```

02. Divide-and-Conquer 21 / 25

Analysis

- \triangleright each of the submatrices A_{11} , etc. has size n/2
- quadrisecting A, B is $\Theta(n^2)$ time; same for assembling C
- each matrix + takes $\Theta((\frac{n}{2})^2) = \Theta(\frac{n^2}{4}) = \Theta(n^2)$ time
- 8 recursive calls

$$T(n) = 8T(n/2) + \Theta(n^2)$$

Solves to $T(n) \in \Theta(n^3)$ by master theorem; same as naïve

Observe: the 8 factor is meaningful, but the $\frac{1}{4}$ isn't \implies it's a win to have fewer recursive calls, but more work (by a constant factor) in the **combine** step

Strassen's insight

Use algebra to refactor into 7 recursive multiplies instead of 8

- 1. quadrisect A, B, C as before
- 2. create 10 $(n/2) \times (n/2)$ submatrices S_1, \dots, S_{10} using matrix + and -
- 3. recursively compute 7 submatrix products P_1, \ldots, P_7 in terms of the matrices from steps 1, 2
- 4. compute C_{11} , C_{12} , C_{21} , C_{22} using matrix + and -

$$T(n) = \Theta(n^2) + \Theta(10\frac{n}{4}) + 7T(n/2) + T(4\frac{n}{4})$$

= $7T(n/2) + \Theta(n^2)$
 $\in \Theta(n^{\lg 7})$

by master theorem

Details of Strassen's algorithm

$$S_1 = B_{12} - B_{22}$$

$$S_2 = A_{11} + A_{12}$$

$$S_3 = A_{21} + A_{22}$$

$$S_4 = B_{21} - B_{11}$$

$$S_5 = A_{11} + A_{22}$$

$$S_6 = B_{11} + B_{22}$$

$$S_7 = A_{12} - A_{22}$$

$$S_8 = B_{21} + B_{22}$$

$$S_9 = A_{11} - A_{21}$$

$$S_{10} = B_{11} + B_{12}$$

$$P_{1} = A_{11} \cdot S_{1}$$

$$P_{2} = S_{2} \cdot B_{22}$$

$$P_{3} = S_{3} \cdot B_{11}$$

$$P_{4} = A_{22} \cdot S_{4}$$

$$P_{5} = S_{5} \cdot S_{6}$$

$$P_{6} = S_{7} \cdot S_{8}$$

$$P_{7} = S_{9} \cdot S_{10}$$

$$C_{11} = P_{5} + P_{4} - P_{2} + P_{6}$$

$$C_{12} = P_{1} + P_{2}$$

$$C_{21} = P_{3} + P_{4}$$

$$C_{22} = P_{5} + P_{1} - P_{3} - P_{7}$$

02. Divide-and-Conquer 24 / 25

Editorial Commentary

- proof that 7 recursive multiplies suffice, instead of 8, is surprising and therefore interesting
- equations on previous slide are relatively uninteresting (though not unimportant) technical detail
- $o(n^3)$ matrix multiply is of great theoretical interest (because surprise)
- but the naïve alg. has substantially better constant factors, and the gap between $\Theta(n^3)$ and $\Theta(n^{2.81})$ is narrow
- Strassen (and descendants) are only practical for very large n
- ▶ in practice: naïve alg. for base case n < 128 (say)

02. Divide-and-Conquer 25 / 25

Takeaways

Recall

- ▶ insertion sort is $\Theta(n^2)$; D&C merge sort is $\Theta(n \log n)$
- brute force maximum subarray is $\Theta(n^2)$; D&C alg. is $\Theta(n \log n)$
- ▶ naïve matrix multiply is $\Theta(n^3)$; Strassen's alg. is $\Theta(n^{2.81})$

In each case study,

- first try was no faster; just using D&C isn't an automatic improvement
- master method analyses hinted at the bottleneck
- shift work around to decrease asymptotic time complexity (but increase constant factors); beneficial trade-off
- optimization comes from human insight into the problem
- ▶ unclear how to make these insights w/o the D&C framing

