Algorithms Course Notes
Divide and Conquer 1

Ian Parberry*

Fall 2001

Summary

Divide and conquer and its application to

- Finding the maximum and minimum of a sequence of numbers
- Integer multiplication
- Matrix multiplication

Divide and Conquer

To solve a problem

- Divide it into smaller problems
- Solve the smaller problems
- Combine their solutions into a solution for the big problem

Example: merge sorting

- Divide the numbers into two halves
- Sort each half separately
- Merge the two sorted halves

Finding Max and Min

Problem. Find the maximum and minimum elements in an array $S[1..n]$. How many comparisons between elements of S are needed?

To find the max:

```
max:=S[1];
for i := 2 to n do
  if S[i] > max then max := S[i]
```

(The min can be found similarly).

max of n	?
min of $n-1$	?
TOTAL	?

Divide and Conquer Approach

Divide the array in half. Find the maximum and minimum in each half recursively. Return the maximum of the two maxima and the minimum of the two minima.

```
function maxmin(x, y)
  comment return max and min in S[x..y]
  if y − x ≤ 1 then
    return(max(S[x], S[y]),min(S[x], S[y]))
  else
    (max1,min1):=maxmin(x, ⌊(x + y)/2⌋)
    (max2,min2):=maxmin(⌊(x + y)/2⌋ + 1, y)
    return(max(max1,max2),min(min1,min2))
```

Correctness

The size of the problem is the number of entries in the array, $y - x + 1$.

We will prove by induction on $n = y - x + 1$ that maxmin(x, y) will return the maximum and minimum values in $S[x..y]$. The algorithm is clearly correct when $n \leq 2$. Now suppose $n > 2$, and that maxmin(x, y) will return the maximum and minimum values in $S[x..y]$ whenever $y - x + 1 < n$.

In order to apply the induction hypothesis to the first recursive call, we must prove that $\lfloor (x+y)/2 \rfloor - x + 1 < n$. There are two cases to consider, depending on whether $y - x + 1$ is even or odd.

Case 1. $y - x + 1$ is even. Then, $y - x$ is odd, and

hence $y + x$ is odd. Therefore,

$$\begin{aligned} & \lfloor \frac{x+y}{2} \rfloor - x + 1 \\ = \; & \frac{x+y-1}{2} - x + 1 \\ = \; & (y - x + 1)/2 \\ = \; & n/2 \\ < \; & n. \end{aligned}$$

Case 2. $y - x + 1$ is odd. Then $y - x$ is even, and hence $y + x$ is even. Therefore,

$$\begin{aligned} & \lfloor \frac{x+y}{2} \rfloor - x + 1 \\ = \; & \frac{x+y}{2} - x + 1 \\ = \; & (y - x + 2)/2 \\ = \; & (n+1)/2 \\ < \; & n \quad \text{(see below)}. \end{aligned}$$

(The last inequality holds since

$$(n+1)/2 < n \Leftrightarrow n > 1.)$$

To apply the ind. hyp. to the second recursive call, must prove that $y - (\lfloor (x+y)/2 \rfloor + 1) + 1 < n$. Two cases again:

Case 1. $y - x + 1$ is even.

$$\begin{aligned} & y - (\lfloor \frac{x+y}{2} \rfloor + 1) + 1 \\ = \; & y - \frac{x+y-1}{2} \\ = \; & (y - x + 1)/2 \\ = \; & n/2 \\ < \; & n. \end{aligned}$$

Case 2. $y - x + 1$ is odd.

$$\begin{aligned} & y - (\lfloor \frac{x+y}{2} \rfloor + 1) + 1 \\ = \; & y - \frac{x+y}{2} \\ = \; & (y - x + 1)/2 - 1/2 \\ = \; & n/2 - 1/2 \\ < \; & n. \end{aligned}$$

Procedure maxmin divides the array into 2 parts. By the induction hypothesis, the recursive calls correctly find the maxima and minima in these parts. Therefore, since the procedure returns the maximum of the two maxima and the minimum of the two minima, it returns the correct values.

Analysis

Let $T(n)$ be the number of comparisons made by maxmin(x, y) when $n = y - x + 1$. Suppose n is a power of 2.

What is the size of the subproblems? The first subproblem has size $\lfloor (x+y)/2 \rfloor - x + 1$. If $y - x + 1$ is a power of 2, then $y - x$ is odd, and hence $x + y$ is odd. Therefore,

$$\begin{aligned} \lfloor \frac{x+y}{2} \rfloor - x + 1 &= \frac{x+y-1}{2} - x + 1 \\ &= \frac{y-x+1}{2} = n/2. \end{aligned}$$

The second subproblem has size $y - (\lfloor (x+y)/2 \rfloor + 1) + 1$. Similarly,

$$\begin{aligned} y - (\lfloor \frac{x+y}{2} \rfloor + 1) + 1 &= y - \frac{x+y-1}{2} \\ &= \frac{y-x+1}{2} = n/2. \end{aligned}$$

So when n is a power of 2, procedure maxmin on an array chunk of size n calls itself twice on array chunks of size $n/2$. If n is a power of 2, then so is $n/2$. Therefore,

$$T(n) = \begin{cases} 1 & \text{if } n = 2 \\ 2T(n/2) + 2 & \text{otherwise} \end{cases}$$

$$\begin{aligned} T(n) &= 2T(n/2) + 2 \\ &= 2(2T(n/4) + 2) + 2 \\ &= 4T(n/4) + 4 + 2 \\ &= 8T(n/8) + 8 + 4 + 2 \\ &= 2^i T(n/2^i) + \sum_{j=1}^{i} 2^j \\ &= 2^{\log n - 1} T(2) + \sum_{j=1}^{\log n - 1} 2^j \end{aligned}$$

$$
\begin{aligned}
&= n/2 + (2^{\log n} - 2) \\
&= 1.5n - 2
\end{aligned}
$$

Therefore function maxmin uses only 75% as many comparisons as the naive algorithm.

Multiplication

Given positive integers y, z, compute $x = yz$. The naive multiplication algorithm:

$x := 0$;
while $z > 0$ **do**
  if $z \bmod 2 = 1$ **then** $x := x + y$;
  $y := 2y$; $z := \lfloor z/2 \rfloor$;

This can be proved correct by induction using the loop invariant $y_j z_j + x_j = y_0 z_0$.

Addition takes $O(n)$ bit operations, where n is the number of bits in y and z. The naive multiplication algorithm takes $O(n)$ n-bit additions. Therefore, the naive multiplication algorithm takes $O(n^2)$ bit operations.

Can we multiply using fewer bit operations?

Divide and Conquer Approach

Suppose n is a power of 2. Divide y and z into two halves, each with $n/2$ bits.

y	a	b
z	c	d

Then

$$
\begin{aligned}
y &= a2^{n/2} + b \\
z &= c2^{n/2} + d
\end{aligned}
$$

and so

$$
\begin{aligned}
yz &= (a2^{n/2} + b)(c2^{n/2} + d) \\
&= ac2^n + (ad + bc)2^{n/2} + bd
\end{aligned}
$$

This computes yz with 4 multiplications of $n/2$ bit numbers, and some additions and shifts. Running time given by $T(1) = c$, $T(n) = 4T(n/2) + dn$, which has solution $O(n^2)$ by the General Theorem. No gain over naive algorithm!

But $x = yz$ can also be computed as follows:

1. $u := (a + b)(c + d)$
2. $v := ac$
3. $w := bd$
4. $x := v2^n + (u - v - w)2^{n/2} + w$

Lines 2 and 3 involve a multiplication of $n/2$ bit numbers. Line 4 involves some additions and shifts. What about line 1? It has some additions and a multiplication of $(n/2 + 1)$ bit numbers. Treat the leading bits of $a + b$ and $c + d$ separately.

$a + b$	a_1	b_1
$c + d$	c_1	d_1

Then

$$
\begin{aligned}
a + b &= a_1 2^{n/2} + b_1 \\
c + d &= c_1 2^{n/2} + d_1.
\end{aligned}
$$

Therefore, the product $(a+b)(c+d)$ in line 1 can be written as

$$
\underbrace{a_1 c_1 2^n + (a_1 d_1 + b_1 c_1)2^{n/2}}_{\text{additions and shifts}} + b_1 d_1
$$

Thus to multiply n bit numbers we need

- 3 multiplications of $n/2$ bit numbers
- a constant number of additions and shifts

Therefore,

$$
T(n) = \begin{cases} c & \text{if } n = 1 \\ 3T(n/2) + dn & \text{otherwise} \end{cases}
$$

where c, d are constants.

Therefore, by our general theorem, the divide and conquer multiplication algorithm uses

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

bit operations.

Matrix Multiplication

The naive matrix multiplication algorithm:

```
procedure matmultiply(X, Y, Z, n);
comment multiplies n × n matrices X := YZ
  for i := 1 to n do
    for j := 1 to n do
      X[i, j] := 0;
      for k := 1 to n do
        X[i, j] := X[i, j] + Y[i, k] * Z[k, j];
```

Assume that all integer operations take $O(1)$ time. The naive matrix multiplication algorithm then takes time $O(n^3)$. Can we do better?

Divide and Conquer Approach

Divide X, Y, Z each into four $(n/2) \times (n/2)$ matrices.

$$\begin{aligned} X &= \begin{bmatrix} I & J \\ K & L \end{bmatrix} \\ Y &= \begin{bmatrix} A & B \\ C & D \end{bmatrix} \\ Z &= \begin{bmatrix} E & F \\ G & H \end{bmatrix} \end{aligned}$$

Then

$$\begin{aligned} I &= AE + BG \\ J &= AF + BH \\ K &= CE + DG \\ L &= CF + DH \end{aligned}$$

Let $T(n)$ be the time to multiply two $n \times n$ matrices. The approach gains us nothing:

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 8T(n/2) + dn^2 & \text{otherwise} \end{cases}$$

where c, d are constants.

Therefore,

$$\begin{aligned} T(n) &= 8T(n/2) + dn^2 \\ &= 8(8T(n/4) + d(n/2)^2) + dn^2 \\ &= 8^2T(n/4) + 2dn^2 + dn^2 \\ &= 8^3T(n/8) + 4dn^2 + 2dn^2 + dn^2 \\ &= 8^iT(n/2^i) + dn^2 \sum_{j=0}^{i-1} 2^j \\ &= 8^{\log n}T(1) + dn^2 \sum_{j=0}^{\log n - 1} 2^j \\ &= cn^3 + dn^2(n-1) \\ &= O(n^3) \end{aligned}$$

Strassen's Algorithm

Compute

$$\begin{aligned} M_1 &:= (A + C)(E + F) \\ M_2 &:= (B + D)(G + H) \\ M_3 &:= (A - D)(E + H) \\ M_4 &:= A(F - H) \\ M_5 &:= (C + D)E \\ M_6 &:= (A + B)H \\ M_7 &:= D(G - E) \end{aligned}$$

Then,

$$\begin{aligned} I &:= M_2 + M_3 - M_6 - M_7 \\ J &:= M_4 + M_6 \\ K &:= M_5 + M_7 \\ L &:= M_1 - M_3 - M_4 - M_5 \end{aligned}$$

Will This Work?

$$\begin{aligned} I &:= M_2 + M_3 - M_6 - M_7 \\ &= (B + D)(G + H) + (A - D)(E + H) \\ &\quad - (A + B)H - D(G - E) \\ &= (BG + BH + DG + DH) \\ &\quad + (AE + AH - DE - DH) \\ &\quad + (-AH - BH) + (-DG + DE) \\ &= BG + AE \end{aligned}$$

$$J := M_4 + M_6$$

$$
\begin{aligned}
&= A(F-H) + (A+B)H \\
&= AF - AH + AH + BH \\
&= AF + BH
\end{aligned}
$$

$$
\begin{aligned}
K &:= M_5 + M_7 \\
&= (C+D)E + D(G-E) \\
&= CE + DE + DG - DE \\
&= CE + DG
\end{aligned}
$$

$$
\begin{aligned}
L &:= M_1 - M_3 - M_4 - M_5 \\
&= (A+C)(E+F) - (A-D)(E+H) \\
&\quad - A(F-H) - (C+D)E \\
&= AE + AF + CE + CF - AE - AH \\
&\quad + DE + DH - AF + AH - CE - DE \\
&= CF + DH
\end{aligned}
$$

Analysis of Strassen's Algorithm

$$
T(n) = \begin{cases} c & \text{if } n = 1 \\ 7T(n/2) + dn^2 & \text{otherwise} \end{cases}
$$

where c, d are constants.

$$
\begin{aligned}
T(n) &= 7T(n/2) + dn^2 \\
&= 7(7T(n/4) + d(n/2)^2) + dn^2 \\
&= 7^2 T(n/4) + 7dn^2/4 + dn^2 \\
&= 7^3 T(n/8) + 7^2 dn^2/4^2 + 7dn^2/4 + dn^2 \\
&= 7^i T(n/2^i) + dn^2 \sum_{j=0}^{i-1} (7/4)^j \\
&= 7^{\log n} T(1) + dn^2 \sum_{j=0}^{\log n - 1} (7/4)^j \\
&= cn^{\log 7} + dn^2 \frac{(7/4)^{\log n} - 1}{7/4 - 1} \\
&= cn^{\log 7} + \frac{4}{3} dn^2 \left(\frac{n^{\log 7}}{n^2} - 1\right) \\
&= O(n^{\log 7}) \\
&\approx O(n^{2.8})
\end{aligned}
$$

State of the Art

Integer multiplication: $O(n \log n \log\log n)$.

Schönhage and Strassen, "Schnelle multiplikation grosser zahlen", *Computing*, Vol. 7, pp. 281–292, 1971.

Matrix multiplication: $O(n^{2.376})$.

Coppersmith and Winograd, "Matrix multiplication via arithmetic progressions", *Journal of Symbolic Computation*, Vol. 9, pp. 251–280, 1990.

Assigned Reading

CLR Chapter 10.1, 31.2.

POA Sections 7.1–7.3