Algorithms and Data Structures

Chapter 4

Catherine Durso

cdurso@cs.du.edu

Chapter 4

Chapter 4 examines methods for obtaining asymptotic bounds on functions for which a recurrence is known. These methods apply to bounding the running times of Divide and Conquer algorithms. The first section provides two more examples of Divide and Conquer algorithms. Three approaches to obtaining asymptotic bounds are presented and applied.

- substition method: exact, requires potentially challenging guess and check steps
- recurrence tree method: flexible, increases intuition, potentially time-consuming
- master method: quick, limited

Maximum-Subarray

Motivation: A manufacturing process is intended to produce items with approximately a given mass, say. You are concerned that the process may drift, producing runs of items that tend to be too heavy. To monitor the process, you have collected the difference between the actual and the target mass for a chronological sequence of items.

You will find the largest sum of these for a subset of consecutive items, and compare it to the largest sums produced by taking the values in randomized orders. If the actual maximum sum is larger that most of the maxima from randomized sequences, you have evidence that the process drifted.

Set up

The values are in an array $A =< a_1, a_2..a_n >$. The goal is to find $j \leq k$ with the property that, for all $r \leq s$, $\sum_{i=j}^{k} a_i \geq \sum_{i=r}^{s} a_i$, where $j, k, r, s \in \{1, 2..n\}$.

Exhaustive Solution

A solution to an optimization problem that simply examines all possibilities and identifies the best is called an exhaustive solution (or a brute force solution). What is the exhaustive solution to the maximum subarray sum problem? What is its asymptotic running time?

Divide and Conquer

The basic idea is to recursively find the maximum sum in the first half of the arrray and the maximum sum in the upper half of the array, and also the maximum sum that includes both the last value in the lower half and the first value in the upper half. Choosing the maximum of these three solves the problem.

Crossing Sum

The task of finding the maximum sum of a subarray of consecutive values that crosses halves is straightforward. Consider all possible sums in the lower half that include the last element and save the largest. Examine all possible sums in the top half that include the first element, and save the largest. The sum of these is the maximum sum that crosses halves.

Find Max Crossing Subarray

```
FIND-MAX-CROSSING_SUM(A, low, mid, high)
1     leftsum = −∞
2     sum = 0
3     for i = mid downto low
4         sum = sum + A[i]
5         if sum > leftsum
6             leftsum = sum
7     rightsum = −∞
8     sum = 0
9     for j = mid + 1 downto high
10        sum = sum + A[j]
11        if sum > rightsum
12            rightsum = sum
13    return leftsum + rightsum
```

(The routine in the book also returns an optimal start and stop index.)

Recursive Routine

```
FIND-MAX-SUBSUM(A, low, high)
1     if low == high
2         if A[low] > 0
3             return A[low]
4         else
5             return 0
6     else mid = ⌊(low + high)/2⌋
7         leftsum = FIND-MAX-SUBSUM(A, low, mid)
8         rightsum = FIND-MAX-SUBSUM(A, mid + 1, high)
9         crosssum = FIND-MAX-CROSSING-SUM(A, low, mid, high)
10    return max(leftsum, rightsum, crosssum)
```

T(n) for Find-Max-Subsum

The recurrence relation for $T(n)$ for the FIND-MAX-SUBSUM function is

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

We know $T(n) = \Theta(n \lg n)$. Why? Is this better than brute force?

Polynomial Representation

Store the polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$ in an array $P = [a_0, a_1..a_n]$ and store the polynomial $q(x) = \sum_{i=0}^{m} b_i x^i$ in an array $Q = [b_0, b_1..b_m]$. Then the product $p(x)\,q(x)$can be calculated with a nested loop. (For this example, let's start the indexing of arrays at 0.)

Polynomial Multiplication

POLYNOMIAL-PRODUCT1(A, B)

1 $n = A.length - 1$

2 $m = B.length - 1$

3 $C =$ new array of 0's length $n + m + 1$

4 for $i = 0$ to n

5 for $j = 0$ to m

6 $C[i + j] = C[i + j] + A[i] * B[j]$

If $n = m$, what is the running time?

Divide

If the polynomials p and q are of the same degree n we can create polynomials p_{low}, p_{high}, q_{low}, and q_{high} with corresponding arrays $P_{low} = \left[a_0..a_{\lfloor \frac{n}{2} \rfloor}\right]$, $P_{high} = \left[a_{\lfloor \frac{n}{2} \rfloor + 1}..a_n\right]$, $Q_{low} = \left[b_0..b_{\lfloor \frac{n}{2} \rfloor}\right]$, and $Q_{high} = \left[b_{\lfloor \frac{n}{2} \rfloor + 1}..b_n\right]$.

Conquer?

Then $p = p_{low} + x^{\lfloor \frac{n}{2} \rfloor + 1} p_{high}$ and $q = q_{low} + x^{\lfloor \frac{n}{2} \rfloor + 1} q_{high}$.

So $pq = p_{low}q_{low} + x^{\lfloor \frac{n}{2} \rfloor + 1} p_{low}q_{high} + x^{\lfloor \frac{n}{2} \rfloor + 1} p_{high}q_{low} + x^{2\lfloor \frac{n}{2} \rfloor + 2} p_{high}q_{high}$

Doing this recursively gives $T(n) = 4T(n/2) + \Theta(n)$.

Rats!

If $k = \lg n$, the recurrence relation implies $T(n) \geq 4^k T(1) + c \sum_{i=0}^{k-1} 2^i n = O\left(n^2\right)$. We have gained nothing!

Conquer

But $x^{\lfloor \frac{n}{2} \rfloor + 1} p_{low} q_{high} + x^{\lfloor \frac{n}{2} \rfloor + 1} p_{high} q_{low} =$
$\left(\left(p_{low} + p_{high} \right) \left(q_{low} + q_{high} \right) - p_{low} q_{low} - p_{high} q_{high} \right) x^{\lfloor \frac{n}{2} \rfloor + 1}$.

Now $T(n) = 3T(n/2) + \Theta(n)$.

Conquer!

We can show

$$T(n) \leq 3^k T(1) + c \sum_{i=0}^{k-1} \left(\frac{3}{2}\right)^i n = O\left(3^{\lg n} + n\left(\frac{3}{2}\right)^{\lg n}\right)$$

$$= O\left(n^{\lg 3} + n\left(n^{\lg \frac{3}{2}}\right)\right) = O\left(n^{\lg 3} + n\left(n^{\lg 3 - \lg 2}\right)\right)$$

$$= O\left(n^{\lg 3}\right)$$

Matrix Notation

A matrix is a rectangular array of numbers, like $A = \begin{bmatrix} 5 & 0 & -1 & 0.5 \\ 1 & 2 & 0 & -3 \end{bmatrix}$. The value in the i^{th} row and j^{th} column is referred to using the row and column subscripts. For example, here $a_{14} = 0.5$. The size of a matrix is specified by (number of rows)$\times$(number of columns). Here, A is 2×4.
An $n \times n$ matrix is said to be *square*.

One major application of matrices is in solving systems of linear equations.

Matrix Multiplication

If A is $n \times m$ and B is $m \times k$, the product $C = AB$ is $n \times k$. The value of the entry in C i^{th} row and j^{th} column, c_{ij} is defined to be $\sum_{l=1}^{m} a_{il} b_{lj}$.

For example, with $A = \begin{bmatrix} 5 & 0 & -1 & 0.5 \\ 1 & 2 & 0 & -3 \end{bmatrix}$ and

$B = \begin{bmatrix} 0 & 0 & -1 \\ 1 & 2 & 3 \\ 0 & -1 & .5 \\ 2 & 0 & 1 \end{bmatrix}$, C is 2×3. The entry

$c_{23} = 1(-1) + 2(3) + 0(.5) + (-3)(1) = 2$. $C = \begin{bmatrix} 1 & 1 & -5 \\ -4 & 4 & 2 \end{bmatrix}$.

Basic Algorithm

SQUARE-MATRIX-MULTIPLY(A, B)

1 $n = A.rows$
2 let C be a new $n \times n$ matrix
3 for $i = 1$ to n
4 for $j = 1$ to n
5 $c_{ij} = 0$
6 for $k = 1$ to n
7 $c_{ij} = c_{ij} + a_{ik} * b_{kj}$
8 return C

T(n)

What is the running time of this basic multiplication routine for $n \times n$ matrices as a function of n? Give a Θ-bound.

Strassen's Method

By using a fiendishly clever arrangement of sums and differences of recursive products of sums and differences of $\frac{n}{2} \times \frac{n}{2}$ subarrays, Strassen's Method achieves a running time with the bounds

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 7T(n/2) + \Theta\left(n^2\right) & n > 1 \end{cases}.$$

Recurrence

$$
\begin{aligned}
&T(n) \leq 7T(n/2) + cn^2 \\
&\leq 7\left(7T(n/4) + c(n/2)^2\right) + cn^2 \\
&= 7^2 T\left(\tfrac{n}{2^2}\right) + c\left(n^2 + 7\left(\tfrac{n}{2}\right)^2\right) \\
&\leq 7^2\left(7T\left(\tfrac{n}{2^3}\right) + c\left(\tfrac{n}{2^2}\right)^2\right) + c\left(n^2 + 7\left(\tfrac{n}{2}\right)^2\right) \\
&= 7^3 T\left(\tfrac{n}{2^3}\right) + c\left(n^2 + 7\left(\tfrac{n}{2}\right)^2 + 7^2\left(\tfrac{n}{2^2}\right)^2\right) \\
&= 7^3 T\left(\tfrac{n}{2^3}\right) + c\left(n^2 + \tfrac{7}{4}n^2 + \left(\tfrac{7}{4}\right)^2 n^2\right) \\
&\vdots \\
&\leq 7^k T\left(\tfrac{n}{2^k}\right) + cn^2\left(1 + \tfrac{7}{4} + \left(\tfrac{7}{4}\right)^2 + \ldots\left(\tfrac{7}{4}\right)^{k-1}\right)
\end{aligned}
$$

O- bound

Setting $k = \lg n$, get

$$T(n) \leq 7^k T(1) + cn^2 \left(1 + \tfrac{7}{4} + \left(\tfrac{7}{4}\right)^2 + \ldots \left(\tfrac{7}{4}\right)^{k-1}\right)$$

$$= 7^{\lg n} + cn^2 \frac{\left(\frac{7}{4}\right)^k - 1}{\left(\frac{7}{4}\right) - 1}$$

$$= O\left(n^{\lg 7}\right) + O\left(n^2 \left(\tfrac{7}{4}\right)^{\lg n}\right)$$

$$= O\left(n^{\lg 7}\right) + O\left(n^2 n^{\lg\left(\frac{7}{4}\right)}\right)$$

$$= O\left(n^{\lg 7}\right) + O\left(n^2 n^{\lg(7)} n^{\lg(-4)}\right)$$

$$= O\left(n^{\lg 7}\right)$$

Ω- bound

Using $T(n) \geq 7T(n/2) + cn^2$, get $T(n) = \Omega\left(n^{\lg 7}\right)$.
This allows us to conclude $T(n) = \Theta\left(n^{\lg 7}\right)$.
This is better than the asymptotic behavior of the naive algorithm.

More Recurrences

What asymptotic behavior does $T(n) = \begin{cases} \Theta(1) & n = 1 \\ 8T(n/2) + \Theta(n) & n > 1 \end{cases}$ give?

Using a recursion-tree, or repeated applications of $T(n) \leq 8T(n/2) + cn$, conclude

$T(n) \leq 8^k T(1) + cn\left(1 + 4 + 4^2 + \ldots 4^{k-1}\right)$

$= O\left(n^{\lg 8}\right) + cn\frac{4^{\lg n} - 1}{4-1}$

$= O\left(n^3\right)$

Using $T(n) \geq 4T(n/2) + cn$ gets us same Ω-bound, and so $T(n) = \Theta\left(n^3\right)$

O-bound

What asymptotic behavior does $T(n) = \begin{cases} \Theta(1) & n = 1 \\ 2T(n/2) + \Theta(n^2) & n > 1 \end{cases}$ give?

Using a recursion-tree, or repeated applications of $T(n) \leq 2T(n/2) + cn^2$, conclude

$T(n) \leq 2^k T(1) + cn^2 \left(1 + \frac{1}{2} + \frac{1}{2}^2 + \ldots \frac{1}{2}^{k-1}\right)$

$\leq O(n) + cn^2 \sum_{i=0}^{\infty} \frac{1}{2}$

$= O(n) + cn^2 \left(\frac{-1}{\frac{1}{2}-1}\right) = O(n^2)$

Ω and Θ-bounds

Using a recursion-tree, or repeated applications of $T(n) \geq 2T(n/2) + cn^2$, conclude

$$T(n) \geq 2^k T(1) + cn^2 \left(1 + \tfrac{1}{2} + \tfrac{1}{2}^2 + \ldots \tfrac{1}{2}^{k-1}\right)$$
$$\geq O(n) + cn^2$$
$$= O\left(n^2\right)$$

Since $T(n) = O\left(n^2\right)$ and $T(n) = \Omega\left(n^2\right)$, $T(n) = \Theta\left(n^2\right)$

Substitution Method

In this method, applied when a recurrence relation for $T(n)$is known, you guess a bound for $T(n)$ then verify it by induction. By verifying an upper bound, a big-O bound can be proved. By verifying an lower bound, an Ω bound can be proved. If both bounds can be shown by substitution for the same bounding function, then a Θ-bound has been proved.

Substitution Example

Consider $T(n) = 4T\left(\frac{n}{2}\right) + n$.
To get a big-Ω bound, guess that $T(n) \geq cn^2$ for some c small enough that the inequality is true for $n = 1$.
This is the basis step.

Next, show that if the bound holds for $m < n$ then it holds for n.
$T(n) = 4T\left(\frac{n}{2}\right) + n \geq 4c\left(\frac{n}{2}\right)^2 + n$
$= cn^2 + n \geq cn^2$, as required.
Conclude $T(n) = \Omega\left(n^2\right)$.

big-O by Substitution

The calculation above shows that an inductive argument for the bound $T(n) \leq cn^2$ will not work.

Try $T(n) \leq cn^2 - dn$. Figure out c and d as you go. The limitation is that it must be possible to make $cn^2 - dn$ arbitrarily large for $n = 1$ for the basis step.

$$T(n) = 4T\left(\tfrac{n}{2}\right) + n \leq 4\left(c\left(\tfrac{n}{2}\right)^2 - d\left(\tfrac{n}{2}\right)\right) + n$$

$$= cn^2 - 2dn + n \leq cn^2 - dn$$

if we choose any $d \geq 1$, say $d = 1$.

big-O, cont.

So a proper inductive proof would run like this: Set $d = 1$, and choose c such that $T(n) \leq cn^2 - n$ for $n = 1$. Claim $T(n) \leq cn^2 - n$ for all n.
The base case (or cases) are true by choice of c and d.
Induction step:

$$T(n) = 4T\left(\tfrac{n}{2}\right) + n \leq 4\left(c\left(\tfrac{n}{2}\right)^2 - \tfrac{n}{2}\right) + n$$

$$= cn^2 - 2n + n \leq cn^2 - n.$$

Θ-bound

Thus $T(n) = O\left(cn^2 - n\right)$. Conclude $T(n) = O\left(cn^2\right)$.
Now we have proved $T(n) = \Omega\left(n^2\right)$ and
$T(n) = \Omega\left(n^2\right)$, so $T(n) = \Theta\left(n^2\right)$.

Master Theorem Context

The Master Theorem identifies three types of recurrence relations and gives solutions for those types. The theorem applies to recurrence relations of the form
$T(n) = aT\left(\frac{n}{b}\right) + f(n)$
where this notation is interpreted to include
$T(n) = a_1 T\left(\lfloor \frac{n}{b} \rfloor\right) + a_2 T\left(\lceil \frac{n}{b} \rceil\right) + f(n)$
for $a_1, a_2 \in \mathbb{N}$, $a \in \mathbb{Z}^+$, and $a_1 + a_2 = a$.

Case 1

If $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$ for some $\varepsilon > 0$ then
$T(n) = \Theta\left(n^{\log_b a}\right)$. (Base case costs dominate.)

Example: $T(n) = 4T\left(\frac{n}{2}\right) + n$
$a = 4$, $b = 2$, $n^{\log_b a} = n^{\log_2 4} = 2$. $n = n^{2-1}$, so case 1 applies.
$T(n) = \Theta\left(n^2\right)$

Case 2

If $f(n) = \Theta\left(n^{\log_b a}\right)$ then $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$.
(Base case and recombination costs balance.)
Example: $T(n) = 3T\left(\frac{n}{3}\right) + n$
$a = 3$, $b = 3$, $n^{\log_b a} = n^{\log_3 3} = 1$. $n = n^1$, so case 2 applies.
$T(n) = \Theta(n \lg n)$

Case 3

If $f(n) = \Omega\left(n^{\log_b a+\varepsilon}\right)$ for some $\varepsilon > 0$, and if $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.
(Recombination costs dominate.)

Example: $T(n) = 9T\left(\frac{n}{3}\right) + n^3$
$a = 9$, $b = 3$, $n^{\log_b a} = n^{\log_3 9} = 2$. $n^3 = n^{2+1}$,
and $af\left(\frac{n}{b}\right) = 9\left(\left(\frac{n}{3}\right)\right)^3 = 9\frac{n^3}{27} = \frac{1}{3}n^3 = \frac{1}{3}f(n)$, so case 3 applies.
$T(n) = \Theta\left(n^3\right)$

What if?

Take $T(n) = 9T\left(\frac{n}{3}\right) + f(n)$, with $f(n) = 9^{3\lceil \log_9 n \rceil} \geq n^3$.

Does case 3 apply?

Not quite.
For $n = 3^{2k}$, $f\left(\frac{n}{3}\right) = f(n)$, so $9f\left(\frac{n}{3}\right) = 9f(n)$.