CS314: Recursive Algorithms

Announcements

- Homework 1 up (probably) tomorrow
  - due next Friday
  - written this time
    (HW 2 will be oral grading)

- Turn in HW 0 now

- Picnic next week!
  (4pm next Wed)
Another (old) example: Merge Sort

According to Knuth, suggested by von Neumann around 1945.

Idea:
1. Subdivide array into 2 parts.
2. Recursively sort the 2 parts.
3. Merge them back together.

Input: SORTINGEXAMPL
Divide: SORTINGEXAMPL
Recurse: I N O S R T | A E G L M P X
Merge: A E G I L M N O P S R T X
Keys: If thinking recursively only step 3 is non-trivial!

\begin{verbatim}
 MERGE_SORT(A[1..n]):
  if (n > 1)
    m ← \lfloor n/2 \rfloor
    MERGE_SORT(A[1..m])
    MERGE_SORT(A[m+1..n])
    MERGE(A[1..n], m)
\end{verbatim}

(Again, avoid unrolling.)

What's my base case here?

size 1 (or less)
How to merge?

| Input:  | SORTINGEXAMPLE |
| Divide: | SORTINGEXAMPLE |
| Recurse: | INSORTAEGLMPX |
| Merge:  | AEGINLMNOPSRTX |
Write a subroutine:

\[
\text{MERGE}(A[1..n], m): \\
i \leftarrow 1; j \leftarrow m + 1 \\
\text{for } k \leftarrow 1 \text{ to } n \\
\quad \text{if } j > n \\
\quad \quad B[k] \leftarrow A[i]; \ i \leftarrow i + 1 \\
\quad \text{else if } i > m \\
\quad \quad B[k] \leftarrow A[j]; \ j \leftarrow j + 1 \\
\quad \text{else if } A[i] < A[j] \\
\quad \quad B[k] \leftarrow A[i]; \ i \leftarrow i + 1 \\
\quad \text{else} \\
\quad \quad B[k] \leftarrow A[j]; \ j \leftarrow j + 1 \\
\text{for } k \leftarrow 1 \text{ to } n \\
\quad A[k] \leftarrow B[k]
\]

\[O(n) + O(n) + 2 = O(n)\]
Proof of correctness: Actually, 2 of them.

Lemma: MERGE results in sorted order.

pf: given 2 sorted subarrays. induction on sizes of $A[i..m]$ and $A[j..n]$

Base case $A[i..m]$ empty or $A[j..n]$ empty correct thing is to just keep taking elements from non-empty list, which is what our first

II: $A$ is correct for smaller inputs
cont.: IS: Consider $A[i...m] + A[i...n]$ not base case, so neither is empty.

Since these are sorted, first element of one of them must be minimum (or else not sorted).

Alg: Finds min a moves it to B. Shrinks one of the sub-arrays by 1. By I, correct on rest.
PF that mergesort works.

Base case: size 0 or 1, do nothing.

IH: works for lists of size \( k < n \).

IS: Consider \( A[1 \ldots n] \).

By IH, \( A[1 \ldots m] \cup A[m+1 \ldots n] \) are sorted.

Now need to show \( A \) ends in sorted order, which it does since Merge works (by prev. lemma).
Runtime:

Let \( M(n) \) = runtime on \( n \) elements of merge sort.

\[
M(0) = M(1) = \mathcal{O}(1) \quad (1 \text{ comparison})
\]

\[
M(n) = 3 + M\left(\frac{n}{2}\right) + M\left(\left\lceil \frac{n}{2} \right \rceil \right) + \mathcal{O}(n) \quad \text{(runtime of merge)}
\]

\[
= 2M\left(\frac{n}{2}\right) + \mathcal{O}(n)
\]

\[
\Rightarrow M(n) = \mathcal{O}(n \log n)
\]
Multiplication is fundamental

\[
\begin{align*}
31415962 \\
\times 27182818 \\
\hline
251327696 \\
31415962 \\
251327696 \\
62831924 \\
\underline{219911734} \\
62831924 \\
\hline
853974377340916
\end{align*}
\]

or

\[
\begin{array}{ccc}
\text{x} & \text{y} & \text{prod} \\
\hline
0 & & 0 \\
123 & +456 & = 456 \\
61 & +912 & = 1368 \\
30 & +824 & \\
15 & +3648 & = 5016 \\
7 & +7296 & = 12312 \\
3 & +14592 & = 26904 \\
1 & +29184 & = 56088
\end{array}
\]

How fast? (n-bit number)

\(\mathcal{O}(n^2)\)
Divide & conquer strategy:

\[(10^m a + b)(10^m c+d) = 10^{2m} ac + 10^m (bc+ad) + bd\]

\[
\frac{963245}{10^3} = 10^3 \cdot 963 + 245
\]

How to turn this into an algorithm?

\[
\frac{624197}{c \quad d}
\]

\[ac = 963 \cdot 624\]

\[bd = 245 \cdot 197\]

\[bc + ad = \]
Pseudo code

MULTIPLY(x, y, n):
    if n = 1
        return x \cdot y
    else
        m \leftarrow \lfloor n/2 \rfloor
        a \leftarrow \lfloor x/10^m \rfloor;
        b \leftarrow x \mod 10^m
        d \leftarrow \lfloor y/10^m \rfloor;
        c \leftarrow y \mod 10^m
        e \leftarrow MULTIPLY(a, c, m)
        f \leftarrow MULTIPLY(b, d, m)
        g \leftarrow MULTIPLY(b, c, m)
        h \leftarrow MULTIPLY(a, d, m)
        return 10^{2m}e + 10^m(g + h) + f

Time Function:
\[ T(1) = 1 \]
\[ T(n) = 4T(\frac{n}{2}) + O(n) \]
\[ aT(\frac{n}{2}) + f(n) \]

Runtime:
- \( f(n) \) to \( n^{\log_{64}2} \)
- \( n^{\log_{64}2} \)
- Adding \( n \) bit shifts: \( O(n^2) \)

= \( O(n^2) \)
Hmm... not better after all...

Another trick:

\[ ac + bd - (a-b)(c-d) = bc + ad \]

Why will this help?

Recall:

\[ (10^m a + b)(10^m c + d) = 10^{2m} ac + 10^m (bc + ad) + bd \]
Now, pseudo code only has 3 recursive calls!

\[
\text{FASTMULTIPLY}(x, y, n):
\]
\[
\text{if } n = 1 \\
\quad \text{return } x \cdot y \\
\text{else} \\
\quad m \leftarrow \lfloor n/2 \rfloor \\
\quad a \leftarrow \lfloor x/10^m \rfloor; \quad b \leftarrow x \mod 10^m \\
\quad d \leftarrow \lfloor y/10^m \rfloor; \quad c \leftarrow y \mod 10^m \\
\quad e \leftarrow \text{FASTMULTIPLY}(a, c, m) \\
\quad f \leftarrow \text{FASTMULTIPLY}(b, d, m) \\
\quad g \leftarrow \text{FASTMULTIPLY}(a - b, c - d, m) \\
\quad \text{return } 10^{2m}e + 10^m(e + f - g) + f
\]

Runtimes?

\[
T(n) = 3T(\frac{n}{2}) + O(n)
\]

(4 instead of n)

\[
T(n) = O(n \log_3^2)
\]

\[
\log_3 3 < 2
\]

\[
= n^1.5...
\]
Notes:

- In practice, this is done in binary - replace 10's with 2's.
- This idea can be broken down recursively even further for an eventual $O(n \log n)$ time.

(Ever heard of Fast Fourier transforms?)