EECS 3101 M: Design and Analysis of Algorithms

Suprakash Datta
Office: LAS 3043

Course page: `http://www.eecs.yorku.ca/course/3101M`
Also on Moodle

More on Correctness of Algorithms: Binary Search

Precondition: *A* an array of sorted integers, *key* an integer
Postcondition: Index in which *key* is found, if it exists in the array

Algorithmic idea:

1. Cut sublist in half
2. Determine which half the key would be in
3. Keep that half.

Note: LI must not assume that the element is present in the list. So it should say something like
If the key is contained in the original list, then the key is contained in the sublist

Pinning down the algorithm: Binary Search

- If $key \leq A[mid]$ then key is in the left half
  Else key is in the right half
- Maintain sublist from i to j
- Which element is mid? Must be consistent
- Subtle issue: Suppose we use
  $mid = \lfloor \frac{i+j}{2} \rfloor$
  If $key \geq A[mid]$ then $i = mid$
  Else $j = mid$
  A $= \boxed{10}\boxed{20}\boxed{30}$, $key = 20$

$\boxed{\text{Similar issue may arise in Q2 of the assignment!}}$

GCD of 2 Natural Numbers m, n

- Precondition: $m, n \in \mathbb{N}$
  Postcondition: returns $GCD(m, n)$
- Idea: if $(m > n)$, $GCD(m, n) = GCD(m–n, n)$
  Proof: k divides $m - n, n \iff k$ divides m, n
- Can design iterative (or recursive) algorithm using this idea

Efficiency of GCD algorithm

$$\begin{aligned}
GCD(999999999999, 2) &= GCD(999999999997, 2) \\
&= GCD(999999999995, 2) \\
&= GCD(999999999993, 2) \\
&= \ldots \\
&= GCD(1, 2) \\
&= GCD(2, 1) \\
&= GCD(1, 1) \\
&= 1
\end{aligned}$$

Running time $= \Theta(m)$. $\boxed{\text{Is this a linear time algorithm?}}$

GCD(m, n): Better Intuition

$$\begin{aligned} GCD(m,n) &= GCD(m-n,n) \\ &= GCD(m-2n,n) \\ &= \ldots \\ &= GCD(m-in,n) \text{ such that } m-in<n \end{aligned}$$

So $i = \lfloor \frac{m}{n} \rfloor$, $m - in = m \mod n$, and
$GCD(m,n) = GCD(m \mod n, n) = GCD(n, m \mod n)$

GCD(m, n): Euclid's Algorithm (c 300 BC)

```
GCD($m, n$)
1  $x = m$
2  $y = n$
3  while $y > 0$
4      $xnew = y$
5      $ynew = x \mod y$
6      $x = xnew$
7      $y = ynew$
8  return $x$
```

Proof of correctness: Use LI $GCD(m, n) = GCD(x, y)$

Euclid's Algorithm: Running time

Try a few cases

Case 1:

$$\begin{aligned} GCD(999999999, 2) &= GCD(1, 2) \\ &= GCD(2, 1) \\ &= GCD(1, 1) = 1 \end{aligned}$$

Case 2:

$$\begin{aligned} GCD(999999999, 999999991) &= GCD(8, 999999999991) \\ = GCD(999999999991, 8) &= GCD(7, 8) \\ = GCD(8, 7) &= GCD(1, 7) \\ = GCD(7, 1) &= 1 \end{aligned}$$

Euclid's Algorithm: Running time - contd.

- **Key Insight:** Every two iterations, the value x decreases by at least a factor of 2
  i.e., the size of x decreases by at least one bit.
- Proof by cases.
  <u>Case 1</u>: $n \leq \lfloor m/2 \rfloor$. Since $GCD(m, n) = GCD(n, m \mod n)$, so $n \leq \lfloor m/2 \rfloor$ implies n has 1 fewer bit than m after 1 iteration
  <u>Case 2</u>: $n > \lfloor m/2 \rfloor$. Again $GCD(m, n) = GCD(n, m \mod n) = GCD(m \mod n, n \mod (m \mod n))$, and $m \mod n = m - n < \lceil m/2 \rceil$.
  Therefore the first argument has reduced by a factor of 2 and is thus 1 bit smaller after 2 iterations
- Running time: $O(\log_2 m + \log_2 n) = O(\log m)$

Multiplying Complex Numbers

(From Jeff Edmonds' slides)

- INPUT: Two pairs of integers, $(a, b), (c, d)$ representing complex numbers, $a + ib, c + id$ respectively.
- OUTPUT: The pair $[(ac - bd), (ad + bc)]$ representing the product $(ac - bd) + i(ad + bc)$
- Naive approach: 4 multiplications, 2 additions.
  Suppose a multiplication costs \$1 and an addition cost a penny. The naive algorithm costs \$4.02.

Q: Can you do better?

Multiplying Complex Numbers: Gauss' Idea

- $m_1 = ac$
  $m_2 = bd$
  $A_1 = m_1 - m_2 = ac - bd$
  $m_3 = (a+b)(c+d) = ac + ad + bc + bd$
  $A_2 = m_3 - m_1 - m_2 = ad + bc$
- Saves 1 multiplication! Uses more additions. The cost now is \$3.03.
  This is good (saves 25% multiplications), but it leads to more dramatic asymptotic improvement elsewhere!
  (aside: look for connections to known algorithms)
- Q: How fast can you multiply two n-bit numbers?

Multiplying 2 n-bit Numbers

- Elementary school algorithm: $\Theta(n^2)$ time complexity

- Faster Algorithm: uses Divide-and-conquer strategy

Divide and Conquer

- DIVIDE: the problem into smaller instances to the same problem.
- CONQUER: (Recursively) solve them.
- COMBINE: Glue the answers together so as to obtain the answer to your larger instance. Sometimes the last step may be trivial.

Multiplying 2 n-bit Numbers using Divide and Conquer

- $X = \boxed{A}\boxed{B}$, $Y = \boxed{C}\boxed{D}$
- $X = A2^{n/2} + B$, $Y = C2^{n/2} + D$,
  A, B, C, D are $n/2$ bit numbers
- Naive approach: $XY = AC2^{n} + (AD + BC)2^{n/2} + BD$
  This gives $\Theta(n^2)$ time complexity – same as before

Faster Multiplication (Karatsuba 1962)

Uses Gauss' Idea

- $X = A2^{n/2} + B, Y = C2^{n/2} + D$,
  A, B, C, D are $n/2$ bit numbers
- $e = AC, f = BD$
- $XY = e2^n + ((A+B)(C+D) - e - f)2^{n/2} + f$
  This gives $\Theta(n^{\log_2 3})$ time complexity
  – asymptotically faster than before; $n^{1.58}$ vs n^2
- Fastest known: $O(n \log n \log \log n)$

Matrix Multiplication

```
MatMult($A, B$)
1  // return $AB$ where $A, B$ are $n \times n$ matrices
2  $n = A.rows$
3  $C = \textsc{CreateMatrix}(n, n)$
4  for $i = 1$ to $n$
5      for $j = 1$ to $n$
6          $C[i, j] = 0$
7          for $k = 1$ to $n$
8              $C[i, j] = C[i, j] + A[i, k] * B[k, j]$
9  return $C$
```

the running time is $\Theta(n^3)$

Towards Faster Matrix Multiplication

- Divide A, B into 4 $n/2 \times n/2$ matrices
- $$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$
- This gives $\Theta(n^3)$ time complexity – same as before
- Need a better idea

Faster Matrix Multiplication: Using Gauss' Idea

- $M_1 = (A_{11} + A_{22})(B11 + B_{22}$
  $M_2 = (A_{21} + A_{22})B_{11}$
  $M_3 = A_{11}(B_{12} - B_{22})$
  $M_4 = A_{22}(B_{21} - B_{11})$
  $M_5 = (A_{11} + A_{12})B_{22}$
  $M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$
  $M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$
- We now express the C_{ij} in terms of M_k:
  $C_{11} = M_1 + M_4 - M_5 + M_7$
  $C_{12} = M_3 + M_5$
  $C_{21} = M_2 + M_4$
  $C_{22} = M_1 - M_2 + M_3 + M_6$

Faster Matrix Multiplication: Strassen's Algorithm

- only using 7 multiplications (one for each M_k) instead of 8
- This gives $\Theta(n^{\lg 7})$ time complexity
  Proof needs the Master Theorem to analyze recurrences
- Divide and conquer approach provides unexpected improvements

Merge Sort

To sort n numbers

- if $n = 1$ done!
- DIVIDE: Divide the array into 2 lists of sizes $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$
- CONQUER: recursively sort the 2 lists
- COMBINE: merge 2 sorted lists in $\Theta(n)$ time

Merge Sort

```
MERGESORT($A, p, r$)
1  if $p < r$
2      $q = \lfloor \frac{p+r}{2} \rfloor$
3      MERGESORT(A,p,q)
4      MERGESORT(A,q+1,r)
5      MERGE(A,p,q,r)
```

MERGE(A, p, q, r)

Take the smallest of the two topmost elements of sequences $A[p..q]$ and $A[q+1..r]$ and put into the resulting sequence. Repeat this, until both sequences are empty. Copy the resulting sequence into $A[p..r]$.

Merge Sort: Analysis

- Correctness: combine induction and loop invariants
- Run time: Can only express it recursively:
  $T(1) = \Theta(1)$
  $T(n) = 2T(n/2) + \Theta(n)$

Finding the Maximum in an Array

- Divide into 2 (approximate) halves
- Find the maximum of each half
- Return the greater of these two values

Similar Problem: Finding the Maximum Subarray

Input: an array of integers
Output: find a contiguous subarray with the maximum sum

- Brute force: $\Theta(n^3)$ or $\Theta(n^2)$
- Can we do better using divide and conquer?
- Problem: The answer may not lie in either!
- Key question: What information do we need from the two halves to solve the big problem?
- Related question: how do we get this information?

Finding the Maximum Subarray

Ask 3 questions to each half:

- What is the maximum subarray for each half?
- What is the maximum “left-aligned subarray”?
- What is the maximum “right-aligned subarray”?

Questions:

- Is this enough? Proof of correctness?
- What is the running time of this algorithm?