CSE 326
Lecture 5
Spring Quarter 1997

Example 7

- Recursive Fibonacci
- Fib(n) {

  case {
    n = 0: return(0);
    n = 1: return(1);
    n > 1: return(Fib(n-1) + Fib(n-2)}}
- Lower bound analysis

  $T(0), T(1) \geq b$

  $T(n) \geq T(n-1) + T(n-2) + c$ if $n > 1$
- Analysis

  let $\phi = \frac{1+\sqrt{5}}{2}$ which satisfies the property $\phi^2 = \phi + 1$

  We show by induction on n that $T(n) \geq b\phi^{n-1}$

Example 7 (Cont')

- Basis: $T(0) \geq b > b\phi^{-1}$ and $T(1) \geq b = b\phi^0$
- Inductive Step:

  Assume that $T(m) \geq b\phi^{m-1}$ for all $m < n$.

  $T(n) \geq T(n-1) + T(n-2) + c$

  $\geq b\phi^{n-2} + b\phi^{n-3} + c$

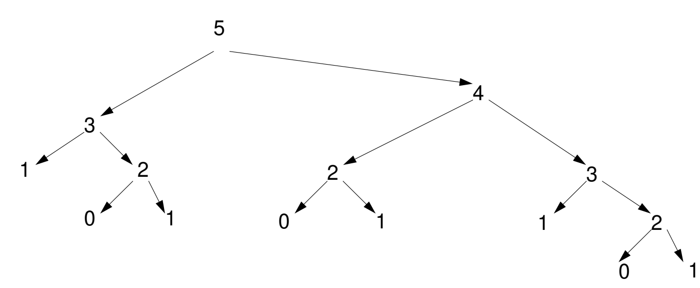
  $\geq b\phi^{n-3}(\phi + 1) + c$

  $= b\phi^{n-3}\phi^2 + c$

  $\geq b\phi^{n-1}$
- The bottom line is that this version of recursive Fibonacci runs in exponential time.

  This is an example where the same recursive call is made numerous times.

Fibonacci Call Tree



Strategy to Avoid Repeated Recursive Calls

- Create an array or other storage structure called the *memoization structure* and initialize it to some value that cannot be returned by any recursive call.

1. All the basis values are stored in the memoization structure.
2. The return value of a recursive call is remembered in the memoization structure.
3. Before making any recursive call check the memoization structure to see if the value has already been computed. If not compute it, otherwise use the already computed value.

Efficient Fibonacci using Memoization

- Array F[0..N] where N is larger than any Fibonacci number to be computed, initialize F[0] = 0, F[1] = 1 and F[i] = -1 for $2 \leq i \leq N$.
- Fib(n) {

```
case {
  n = 0:  return(F[0]);
  n = 1:  return(F[1]);
  n > 1: { if F[n-1] = -1 then F[n-1] := Fib(n-1);
           if F[n-2] = -1 then F[n-2] := Fib(n-2);
           return(F[n-1] + F[n-2])}}}
```

- This version of recursive Fibonacci runs in O(n) time instead of exponential time!
- Naturally, there is a fairly simple iterative version of Fibonacci which only remembers the last two values computed.

Polynomial Multiplication

- Suppose we have two polynomials one with m terms and the other with n terms.

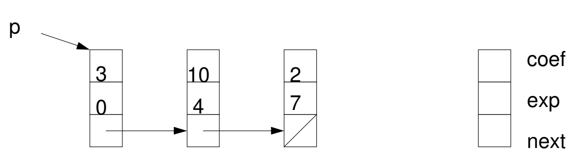
  Their product will no more than mn terms.

  If the polynomials are stored as sparse lists we would like to compute the product in time O(mn).

  Note that adding two polynomial is O(m+n).

  Recall data structure for polynomial.

  $3 + 10x^4 + 2x^7$



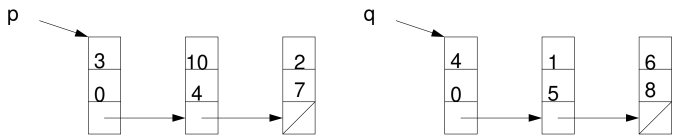
Basic Polynomial Multiplication Algorithm

- Mult(p,q) {

```
local pointer r, t,

r := null;
t := p;
while not(t=null) do {
    r := Add(r, Term_Mult(t,Copy(q)));
    t := t.next}
return(r) }
```

Example



$3 + 10x^4 + 2x^7$ $\quad\quad$ $4 + x^5 + 6x^8$

Time Analysis

- How does the size of r grow with each iteration? Let m be the number of terms in the first polynomial and n the number of terms in the second polynomial. Each iteration may create as many as n new terms in r.

  n terms in iteration 1
  2n terms in iteration 2
  3n terms in iteration 3
  .....
  mn terms in iteration m

- How many steps does the addition take for each step? The number of steps in proportional to the number of terms in the result r.

$$Time = \sum_{i=0}^{m} in = n \cdot \sum_{i=0}^{m} i = O(m^2 \cdot n)$$

- Thus, it is a good idea to choose p to be the smaller of the two polynomials to reduce the running time.

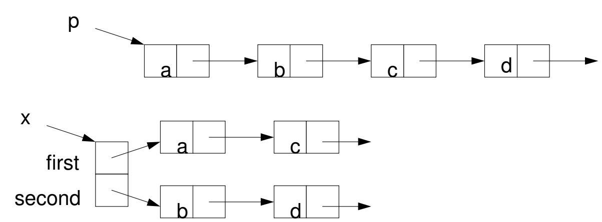
A Better Polynomial Multiplication Algorithm

- Better_Mult(p,q)

```
local duo d;

case {
  p := null :  return(null);
  p.next = null: return(Term_Mult(p,copy(q)));
  otherwise: {
     d := split(p);
     return( Add( Better_Mult(d.first, q), Better_Mult(d.second,q)) }
 }
```


Recall Split



Example of Better_Mult

$(4 + x^3 + x^5 + 3x^{10}) * (3 + 2x + x^4)$

$(4 + x^5) * (3 + 2x + x^4)$ $\quad$ $(x^3 + 3x^{10}) * (3 + 2x + x^4)$

$(4) * (3 + 2x + x^4)$ $\quad$ $(x^5) * (3 + 2x + x^4)$ $\quad$ $(x^3) * (3 + 2x + x^4)$ $\quad$ $(3x^{10}) * (3 + 2x + x^4)$

term_mult

$12 + 8x + 4x^4$ $\quad$ $3x^5 + 2x^6 + x^9$ $\quad$ $3x^3 + 2x^4 + x^7$ $\quad$ $9x^{10} + 6x^{11} + 3x^{14}$

add

$12 + 8x + 4x^4 + 3x^5 + 2x^6 + x^9$ $\quad$ $3x^3 + 2x^4 + x^7 + 9x^{10} + 6x^{11} + 3x^{14}$

$12 + 8x + 3x^3 + 6x^4 + 3x^5 + 2x^6 + x^7 + x^9 + 9x^{10} + 6x^{11} + 3x^{14}$

Complexity of Better_Mult

- m recursive calls with splits costing m log m
- m Term_Mults costing mn
- $\log_2 m$ levels of addition.

1. m/2 additions of polynomials of n terms each costing (m/2)2n
2. m/4 additions of polynomials of 2n terms each costing (m/4)4n
3. m/8 additions of polynomials of 4n terms each costing (m/8)8n

k. $m/2^k$ additions of polynomials of 2^{k-1}n terms each costing mn

$$Time = \sum_{k=1}^{\log m} \frac{m}{2^i} \cdot 2^i \cdot n = \sum_{k=1}^{\log m} mn = O(nm \cdot \log m)$$

Analysis with Recurrences

$$T(m, n) \le 2T\left(\frac{m}{2}, n\right) + cmn$$

$$T(1, n) \le bn$$