Calculating polynomials

Example:

$$p(x) = 2\mathrm{x}^4 - x^3 + 3\mathrm{x}^2 + x - 5$$

$$Evaluate\ for\ x = 3$$

The traditional, obvious, brute force way:

$$p(3) = 2(3)^4 - 3^3 + 3 \times 3^2 + 3 - 5$$

Brute force polynomial

- For a polynomial of size n, just the first term $a_n x^n$ requires n multiplications using brute force.
- We can improve on this by efficiently calculating x^n
- But Horner's rule does even better for large polynomials and it's dead easy.

Horner's rule

Factor x out of as much as possible. Example:

$$p(x) = 2x^4 - x^3 + 3x^2 + x - 5$$

$$= (2x^3 - x^2 + 3x + 1)x - 5$$

$$= ((2x^2 - x + 3)x + 1)x - 5$$

$$= (((2x - 1)x + 3)x + 1)x - 5$$

So what?

Factor x out of as much as possible.   Example:

$$p(x) = 2x^3 - x^2 - 6x + 5$$

$$= (2x^2 - x - 6)x + 5$$

$$= ((2x - 1)x - 6)x + 5$$

Example: Find p(x) at x=3

	$2x^3$	$- x^2$	$- 6x$	$+ 5$
C[ ]:	2	-1	-6	5
p:	2	2*3 + (-1) = 5	5*3 + (-6) = 9	9*3 + 5 = 32

Efficiency?

Horner's rule pseudocode

```
double horner(coefficients[0..n], x):

    p = coefficients[n]

    for i = n – 1 downto 0:
        p = x * p + coefficients[i]
    return p
```

Can you think of a polynomial where Horner's rule is no help at all ?

Horner's rule efficiency

- Basic operations are multiplication and addition
- Let number of multiplications = M(n)
- Let number of additions = A(n)

$$M(n) = A(n) = \sum_{i=0}^{n-1} 1 = n$$

For the entire polynomial it makes as many multiplications as the brute force method on the first coefficient.

Horner's rule is extremely efficient.

Horner's Rule: Representation Change

- Addresses the problem of evaluating a polynomial $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ at a given point $x = x_0$
- Re-invented by W. Horner in early 19th Century
- Approach:
  - Convert to $p(x) = (\ldots(a_n x + a_{n-1}) x + \ldots) x + a_0$
- Algorithm:

```
p ← P[n]
for i ← n -1 downto 0
        p ← x * p + P[i]
return p
```

- Example:
  - $Q(x) = 2x^3 - x^2 - 6x + 5$ at $x = 3$

P[ ]:	2	-1	-6	5
p:	2	3*2 + (-1) = 5	3*5 + (-6) = 9	3*9 + 5 = 32

Notes on Horner's Rule

- An optimal algorithm

- Intermediate results are coefficients of the quotient of *p(x)* divided by x - *x0*

- Used by binary exponentiation algorithm

  Integer n in binary seen as polynomial

  $n = b_l \dots b_i \dots b_0$

$$p(x) = b_k x^k + \dots + b_i x^i + \dots + b_0$$
$$where\ x = 2$$

"next level" Horner !

$$p(x) = b_k x^k + \ldots + b_i x^i + \ldots + b_0$$
$$where\ x = 2$$

Example: binary13 is 1101

$p(x) = 1x^3 + 1x^2 + 0x + 1$ at $x=2$

C[ ]:	**1**	**1**	**0**	**1**
p:	**1**	**1*2 + 1 = 3**	**3*2 + 0 = 6**	**6*2 + 1 = 13**
a^p	**a^1**	**$a^2 * a^1$**	**$(a^3)^2 * a^0$**	**$(a^6)^2 * a^1$**
a^p	**a^1**	**$a^2 * a^1$**	**$(a^3)^2 * a^0$**	**$(a^6)^2 * a^1$**

Horner for exponentiation

$$p(x) = b_k x^k + \ldots + b_i x^i + \ldots + b_0$$
$$where\ x = 2$$

Example: binary13 is 1101

$p(x) = 1x^3 + 1x^2 + 0x + 1$ at $x=2$

C[ ]:	**1**	**1**	**0**	**1**
p:	**1**	**1*2 + 1 = 3**	**3*2 + 0 = 6**	**6*2 + 1 = 13**
a^p	**a^1**	**$a^2 * a^1$**	**$(a^3)^2 * a^0$**	**$(a^6)^2 * a^1$**

because $a^{2n+d} = (a^n)^2 * a^d$ ***And d is only 0 or 1 !***

$a^n = a^{p(2)}$

Horner's rule for p(2)

p = 1 //leading digit is 1

for i = k – 1 down to 0:
    p = 2 * p + b[i]

Implications for $a^n = a^{p(2)}$

$a^p = a^1$

for i = k – 1 down to 0:
    $a^p = \boldsymbol{a^{2p+b[\,i\,]}}$

We are going to look at $a^{2p+b[i]}$ *in two next slides.*

Horner's rule for a^n

$$a^{2\mathrm{p}+b[i]} = a^{2\mathrm{p}} \times a^{b[i]} = (a^p)^2 \times a^{b[i]} = \begin{cases} (a^p)^2 \, if \, b[i] = 0 \\ (a^p)^2 \times a \, if \, b[i] = 1 \end{cases}$$

This is equal to these 2 lines of code from our algorithm.

```
product = product * product
if b[i]: product = product * a
```

HORNER'S RULE FOR p(n)

```
p = coefficients[n]

for i = n – 1 down to 0:
        p = x * p + coefficients[i]


return p
```

HORNER'S RULE FOR a^n

```
p = a

for i = k down to 0:
        p = p * p
        if b[i]: p = p * a

return p
```

Efficiency of Horner's rule for calculating a^n

- Basic operation is multiplication.
- Express number of multiplications as M(n)
- At most two multiplications on each iteration of the loop. Sometimes only one.

$$(b-1) \leq M(n) \leq 2(b-1) \; b \textit{ is length of bit representation of } n$$

$$b-1 = floor(\log_2 n)$$

$$Therefore{:} M(n) \leq 2(\log_2 n)$$
$$Therefore{:} M(n) \in \theta(\log n)$$

Use this pseudcode to calculate 5^{13}

```
Pow(a, b(n)):
    // a is any number
    // b(n) is binary representation of exponent
    // d is number of digits in b
    // 13 is 1101: b(3) = 1, b(2) = 1, b(1) = 0 and b(0) = 1

    product = a
    for i = d – 1 downto 0:
        product = product * product
        if b[i]==1 product = product * a
    return product
```

Only 5 multiplications to get 5^{13}!

13 is 1101 in binary

B(i)	1	1	0	1
i		2	1	0
product	5			
square it		**25**	**15,625**	**244,140,625**
times by a if B(i) is 1		**125**		**1,220,703,125**

Much more efficient than brute force power(a, n).

Notes on Horner's Rule

Efficiency:

- Brute Force = $\Theta(n^2)$
- Transform and Conquer = $\Theta(n)$

Has useful side effects:

- Intermediate results are coefficients of the quotient of $p(x)$ divided by $x - x0$

An optimal algorithm

Binary exponentiation:

- Also uses ideas of representation change to calculate a^n by considering the binary representation of n