Recursive Algorithms

ICS 6D
Sandy Irani
Pseudo-code

• Algorithms for solving problems
• Specified in a language between English and programming language
  – The syntax is informal – meant to convey the meaning of steps.
  – Should be complete enough that it can be unambiguously translated into code.
  – Use elements of computer languages:
    • If-then-else, For, Assignment (x := 6).
Recursive Algorithm

• A *recursive algorithm* is an algorithm that calls itself.

• A recursive algorithm has
  – *Base case*: output computed directly on small inputs
  – On larger input, the algorithm calls itself using smaller inputs and then uses the results to construct a solution for the large input.

  • The calls to itself are called *Recursive Calls*. 
Computing an Exponent Recursively

• **Power(a, n)**
• **Input:** real number, non-negative integer **n**
• **Output:** $a^n$

• If $(n = 0)$, return $(1)$
• $p := \text{Power}(a, n-1)$
• Return $(p*a)$
Power(a,n) Proof of Correctness

• **Theorem**: for integer $n \geq 0$, $\text{Power}(a, n)$ returns $a^n$

• Proof: By induction on $n$.

• **Base case**: $n = 0$. $\text{Power}(a, 0)$ returns $1 = a^0$.

• **Inductive Step**: We will show that for $k \geq 0$,
  – If $\text{Power}(a, k)$ returns $a^k$,
  – then $\text{Power}(a, k+1)$ returns $a^{k+1}$. 
Power(a,n) multiplication count

• **Theorem**: for integer $n \geq 0$, Power(a, n) performs $n$ multiplications.

• **Proof**: By induction on $n$.

• Base case: $n = 0$. Power(a, 0) performs no multiplications.

• Inductive Step: We will show that for $k \geq 0$,
  – If Power(a, $k$) performs $k$ multiplications
  – then Power(a, $k+1$) performs $k+1$ multiplications.
Power(a,n) multiplication count

- Inductive Step: We will show that for \( k \geq 0 \),
  - If \( \text{Power}(a, k) \) performs \( k \) multiplications
  - then \( \text{Power}(a, k+1) \) performs \( k+1 \) multiplications.

- The number of multiplications performed by
  \( \text{Power}(a, k+1) = \)
Faster Recursive Exponentiation

**FastPower**(a, n)

**Input:** real number a, non-negative integer n

**Output:** \(a^n\)

- If \(n = 0\), return(1)
- \(d := \text{n DIV 2}\)
- \(p := \text{FastPower}(a, d)\)
- If n is even
  - Return\((p^2)\)
- If n is odd
  - Return\((a \cdot p^2)\)
Faster Recursive Exponentiation

*FastPower*(a, n)

**Input**: real number a, non-negative integer n

**Output**: \(a^n\)

- If \(n = 0\), return(1)
- \(d := n \text{ DIV } 2\)
- \(p := \text{FastPower}(a, d)\)

- If n is even
  - Return\(p^2\)
- If n is odd
  - Return\((a \cdot p^2)\)
FastPower(a,n) Proof of Correctness

- **Theorem**: for integer $n \geq 0$, 
  \[ \text{FastPower}(a, n) \text{ returns } a^n \]

- **Proof**: By induction on $n$.

- **Base case**: $n = 0$. \[ \text{FastPower}(a, 0) \text{ returns } 1 = a^0. \]

- **Inductive Step**: We will show that for $k \geq 0$,
  - If \[ \text{FastPower}(a, j) \text{ returns } a^j \text{ for every } j = 0,\ldots,k \]
  - then \[ \text{FastPower}(a, k+1) \text{ returns } a^{k+1}. \]
FastPower(a,n) Proof of Correctness

• Inductive Step: We will show that for \( k \geq 0 \),
  – If FastPower(a, j) returns \( a^j \) for every \( j = 0,\ldots,k \)
  – then FastPower(a, k+1) returns \( a^{k+1} \).

• Case 1: \( k+1 \) is even

  – \( k+1 = 2m \) for integer \( m \)
  – \( d := k+1 \ \text{DIV} \ 2 = m \)
  – \( m \) is in the range 0 through \( k \), by the inductive hypothesis,
    FastPower(a, m) returns \( a^m \)
  – FastPower(a, k+1) returns \([\text{FastPower}(a, m)]^2\)
Faster Recursive Exponentiation
Case 1: \( k+1 \) is odd

**FastPower**\((a, k+1)\)

Input: real number \( a \), non-negative integer \( n \)

Output: \( a^n \)

- If \( n = 0 \), return(1)
- \( d := k+1 \) DIV 2
- \( p := \text{FastPower}(a, d) \)

- If \( n \) is even
  - Return\((p^2)\)

- If \( n \) is odd
  - Return\((a \cdot p^2)\)
FastPower($a,n$) Proof of Correctness

- **Inductive Step:** We will show that for $k \geq 0$,
  - If FastPower($a, j$) returns $a^j$ for every $j = 0,\ldots,k$
  - then FastPower($a, k+1$) returns $a^{k+1}$

- **Case 2:** $k+1$ is odd
  - $k+1 = 2m+1$ for integer $m$
  - $d := k+1 \text{ DIV } 2 = m$
  - $m$ is in the range 0 through $k$, by the inductive hypothesis,
    FastPower($a, m$) returns $a^m$
  - FastPower($a, k+1$) returns $a \cdot [\text{FastPower}(a, m)]^2$
Faster Recursive Exponentiation
Case 2: k+1 is odd

*FastPower*(a, k+1)

**Input:** real number a, non-negative integer n

**Output:** $a^n$

- If (n = 0), return(1)
- $d = k+1 \text{ DIV } 2$
- $p := \text{FastPower}(a, d)$

- If n is even
  - Return($p^2$)

- If n is odd
  - Return($a \cdot p^2$)
Recursive Algorithm to Compute
SuperPower(a, n) = \(a^{(3^n+1)}\)

- SuperPower(a,n)

  //a is a real number, n is a non-negative int

If __________ then __________  //Base Case

p := SuperPower(a,n-1)
Return(_________)
Recursive Algorithm to Compute the Power Set of a Set

• PowerSet(A)

• Input: a set A

• Output: P(A)

• If A = ∅, return( {∅} )
Recursive Algorithm to Compute the Power Set of a Set

- PowerSet(A)
- Input: a set A
- Output: P(A)

- If A = ∅, return( {∅} )
- Select an element a ∈ A
- A' := A – {a}
- P := PowerSet(A')
- For each S ∈ P
  - Add {a} ∪ S to P
- Return(P)