A Change of View

Efficiency as a sequence?

Exercise
Write the basic operation count $C(n)$ for $n = 0, 1, 2, 3, 4, 5$ as terms of a math sequence.

Algorithm *Factorial*
Input Integer $n \geq 0$
Output $n!$

```
fact ← 1
for i ← 1 to n do
    fact ← fact × i
return fact
```

n			
$C(n)$			

Another Useful Tool

- Mathematical sequence
- Generic (nth) term

Examples

$0, 1, 1, 2, 3, 5, 8, 13, \ldots$ (Fibonacci)

Position of **term** is indicated by an index.

0 1 2 3 4 5 6 7

Quiz
What's the difference between a set and a sequence?

$x(n) = 2n, n > 0$ $\quad$ $2, 4, 6, 8, \ldots$

Characterization

3 ways to do the same thing; each leads to the other two, but maybe more convenient or helpful in some cases.

Explicit sequence

0	1	3	6	10	15	21	…
0	1	2	3	4	5	6	

The sequence seems off somehow! Write some terms. How to *correct* it?

Generic term ***x*(6)=?**

$$x(n) = n(n+1)/2, \quad n \geq 1$$

Exercise
Use the **recurrence** to find the 8th term (position 7) of the sequence. Verify from the generic term.

Recurrence ***x*(6), *x*(7)=?**

$$x(n) = x(n-1) + n \text{ for } n > 0, \quad x(0) = 0$$

Recurrence Relations

- General solution
- Particular solution

Recurrence relations are used in analysis of recursive algorithms.

Definitions, examples

$$\boldsymbol{x(n)} = x(n-1) + n \text{ for } n > 0$$

$$x(0) = 0$$

$$x(n) = x(n/2) + n \text{ for } n > 1$$

$$x(1) = 1$$

The condition on the generic term (previous slide) specifies a different (particular) sequence.

Recurrence solution (the generic term)

0 1 3 6 10 15 21 …
1 3 6 10 15 21 …

Standard Recurrences

Quiz
Identify the terms related to the **generic term** in each recurrence (no math, use words).

➪ Decrease-by-one

$x(n) = x(n-1) + n \text{ for } n > 0$

$$T(n) = T(n-1) + f(n)$$

➪ Decrease-by-constant factor

$$T(n) = T(n/b) + f(n) \quad b > 1, n = b^k, k = 0, 1, 2, \cdots$$

Exercise
Write the recurrence relation that describes the *Fibonacci* sequence.

➪ General divide-conquer

$$T(n) = aT(n/b) + f(n) \quad a \geq 1, b \geq 2$$

cs681fig3-4.cdr
Monday, February 26, 2024 12:20:17 PM
Color profile: Disabled

Composite Default screen

Master Theorem

Not quite the general form (note condition on f).

No need to solve, sometimes

If $f(n) \in \Theta(n^d)$ with $d \geq 0$ in recurrence

$$T(n) = aT(n/b) + f(n), a \geq 1, b > 1$$

Quiz
Use theorem to determine order of growth for $a = b = 2$ and d=1. Write the **divide-conquer recurrence**.

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

cs681fig3-4.cdr
Monday, February 26, 2024 12:20:17 PM
Color profile: Disabled
Composite Default screen

Solving A Recurrence

Quiz
What is the rationale (basis or reasoning) behind this approach?
Hint: **not lazy!**

➪ Recognize recurrence?

Lookup solution or growth results *first*

➪ Strategy to solve?

- ✎ Use Maple? or Wolfram MathWorld
- ✎ Backward substitutions (standard method)

➪ Examples

WolframAlpha

Challenge Exercise

Use *WolframAlpha* to inductively deduce/guess a general form for solution of 2nd example (try 3,4,5 in const-fac term, i.e., n/3, n/4, n/5), write a formal proof by induction to prove guessed result.

- ⇨ **Inductive thinking in action**
- ⇨ **Guessing a result**
- ⇨ **Proof by induction**
- ⇨ **Optional bonus**

Recurrence relations are used in analysis of recursive algorithms

Efficiency can be determined from the order of growth of the dominant term of a count $C(n)$ of a basic operation as a function of input size n, which should match the order of growth of run time on any machine.

- ✎ Sometimes useful to view algorithm efficiency as a math sequence of terms with input size as position in sequence
- ✎ A family of sequences can be described by a recurrence (+ init condition to specify a particular one)
- ✎ A recurrence relates a generic term with 1+ other terms of an underlying sequence

Solving a recurrence involves finding the generic term of its underlying sequence.

General Plan Review

❶ Select suitable input size parameter *n*

❷ Identify a suitable basic operation

❸ Check basic operation count dependancy

❹ Setup a sum <u>or a recurrence</u> for $C(n)$

❺ <u>Determine order of growth</u> of $C(n)$
(may need to solve sum or recurrence)

Analysis of Recursive Algorithms
Simple Example

Quiz
Suggest another suitable input size parameter beside the magnitude of *n*?

Exercise
Discuss possible basic operation choices, why would the multiplication be preferred?

Algorithm *Factorial*
Input Integer $n \geq 0$
Output $n!$

```
if n = 0 then
    return 1
else return Factorial(n − 1) × n
```

?

Algorithm *Factorial*
Input Integer $n \geq 0$
Output $n!$

```
fact ← 1
for i ← 1 to n do
    fact ← fact × i
return fact
```

Algorithm *Factorial* (n, val)

```
if n = 0 then
    return val
val ← val × n
return Factorial (n − 1, val)
```

$f(n)$

A **tail recursion** involving <u>one</u> recursive call to a smaller instance is often easy to specify.

Exercise
Compare to the definition-based recursive version.

Design clues
A simple recursive plan

Analysis of Recursive Algorithms
Example 2

3 14 27 31 39 **42** 55 70 74 81 85 93 98

Classic top-down

➪ **Operation review**

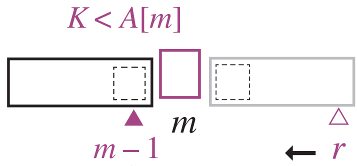
➪ **Efficiency?**

Exercise
Write the sequence of lengths of searched lists in each iteration (code and generate the log below).

```
? [0 ?], m = ? (?)
6 [0 5], m = 2 (27)
3 [3 5], m = 4 (39)
1 [5 5], m = 5
```

Algorithm *BinarySearch*
Input $A[0..n-1]$ sorted in ascending order
Input Search key K
Output Index of key in A if found, -1 otherwise

$l \leftarrow 0, r \leftarrow n-1$
while $l \leq r$ **do**
    $m \leftarrow \lfloor \frac{l+r}{2} \rfloor$
    if $K = A[m]$ **then** **return** m
    else if $K < A[m]$ **then** $r \leftarrow m-1$
    else $l \leftarrow m+1$
return -1



3-way key-comp counts once regardless of its run time cost (why?)

Analysis of Recursive Algorithms
Example 2

Quiz
Determine the relationship between an instance *n* and the immediately following one(s) in the iteration.

Algorithm *BinarySearchRec*
Input Subrange $[l\,..\,r]$ of $A[0\,..\,n-1]$ sorted ascending
Input Search key K
Output Index of key in A if found, -1 otherwise

```
if l > r then
    return −1
m ← ⌊(l+r)/2⌋
if K = A[m] then
    return m
else if K < A[m] then
    r ← m − 1
else
    l ← m + 1
return BinarySearchRec(A, l, r, K)
```

$f(n) = ?$

Check dependency on instances. Determine the efficiency case (always, worst, or best).

Check the solution to the standard form next slide.

```
l ← 0, r ← n − 1
while l ≤ r do
    m ← ⌊(l+r)/2⌋
    if K = A[m] then
        return m
    else if K < A[m] then
        r ← m − 1
    else
        l ← m + 1
return −1
```

```
?  [0 ?], m = ? (?)
6  [0 5], m = 2 (27)
3  [3 5], m = 4 (39)
1  [5 5], m = 5
```

cs681fig3-4.cdr
Monday, February 26, 2024 12:20:19 PM
Color profile: Disabled

Composite Default screen

Solutions

Exercise
Lookup the solution for the standard form.

➪ Decrease-by-one

$$T(n) = T(n-1) + f(n)$$

➪ Decrease-by-constant factor

$$T(n) = T(1) + \sum_{i=1}^{k} f(2^i)$$

$$T(n) = T(1) + \sum_{i=1}^{k} 1$$

$$T(n) = T(n/b) + f(n)$$

$$b > 1, n = b^k, k = 0, 1, 2, \cdots$$

$$\boxed{T(n) = T(1) + \sum_{i=1}^{k} f(b^i)}$$

➪ General divide-conquer

$$T(n) = aT(n/b) + f(n) \quad a \geq 1, b \geq 2$$

Recursive Algorithms
Exercise

In the *Mergesort*, all we know about step 6 is that it depends on n as shown.

Exercise
In the *InsertionSort*, write a recurrence relating $C(n)$ with the immediately following instance, then solve. Hint: Write the sequence of instances generated by the outer loop and note sizes.

Algorithm *Mergesort*
Input $\ldots A[0 \mathinner{..} n-1] \ldots$

1: **if** $n > 1$ **then**
2: copy $A[0 \mathinner{..} \lfloor n/2 \rfloor - 1]$ to $B[0 \mathinner{..} \lfloor n/2 \rfloor - 1]$
3: copy $A[\lfloor n/2 \rfloor \mathinner{..} n-1]$ to $C[0 \mathinner{..} \lfloor n/2 \rfloor - 1]$
4: $Mergesort(B[0 \mathinner{..} \lfloor n/2 \rfloor - 1])$
5: $Mergesort(C[0 \mathinner{..} \lfloor n/2 \rfloor - 1])$
6: $Merge(B, C, A)$

n $O(n)$

?

Algorithm *InsertionSort*
Input $\cdots A[0..n-1] \cdots$
Output

for $i \leftarrow 1$ **to** $n-1$ **do**
  $v \leftarrow A[i]$
  $j \leftarrow i-1$
  while $j \geq 0$ **and** $A[j] > v$ **do**
    $A[j+1] \leftarrow A[j]$
    $j \leftarrow j-1$
  $A[j+1] \leftarrow v$

➪ **Write a recurrence**

➪ **Determine efficiency**