Math 135 - Recursive Algorithms

Note Title 3/15/2010

Announcements

- HW due next Monday
- Review for exam 2 is Monday
- Exam - next Wed. in class

Recursive Algorithms

Dfn: A recursive algorithm solves a problem by reducing it to an instance of the same problem with smaller input.

Note – similar to induction!

A function that calls itself.
(on a smaller input)

→ Don't forget base case!

Recall:

Recursive definition of $n!$ was:

(define $n!$ in terms of $(n-1)!$)

$$n! = n \cdot (n-1)!$$

Base case: $1! = 1$

$0! = 1$

Write a function to compute $n!$

1. Base case
2. Recursive call

Pseudo code:

```
procedure factorial(n):
    if n = 1
        return 1
    else
        return n · factorial(n-1)
```

factorial(5) = $5 \cdot$ factorial(4) $= 5 \cdot 4 \cdot$ factorial(3)

$= 5 \cdot 4 \cdot 3 \cdot$ factorial(2) $= 5 \cdot 4 \cdot 3 \cdot 2 \cdot$ factorial(1)

$= 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$

Run time: $F(n)$ = run time of this alg. on input n

$F(n) = 3 + F(n-1)$ $\quad x - 1 = 0$ $\quad g(n) = 3 \cdot 1^n$ $\quad F(n) = c_1 + c_2 \cdot n$

(Ex) $= O(n)$

Recall: recursive dfn for $a^n = a \cdot a^{n-1}$

base case: $a^0 = 1$

Pseudocode:

```
procedure power(a, n):
    if n = 0
        return 1
    else
        return a · power(a, n-1)
```

runtime: $P(n) = P(n-1) + 3$

$\Rightarrow P(n) = O(n)$

Computing Fibonacci Numbers

$f_n = f_{n-1} + f_{n-2}$, $f_0 = 0$, $f_1 = 1$ fib(20)

```
procedure fib(n):
    if n <= 1
        return n
    else
        return fib(n-1) + fib(n-2)
```

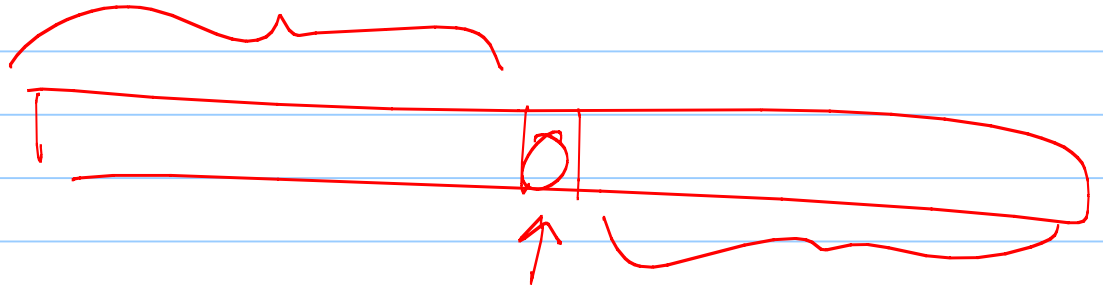
Runtime: Let $F(n)$ = run time for fib(n)

$F(n) = F(n-1) + F(n-2) + 2$

$= O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$ exponential

Recall: Binary Search

Suppose we have a sorted list, & want to find if an element is in there.



if found, done

otherwise if middle element is too big, check left half

if $n = 1$ check a_n

```
procedure BinarySearch(a_1, ..., a_n, x):
    middle := ⌈n/2⌉
    if a_middle = x
        return found
    else
        if a_middle > x
            BinarySearch(a_1, ..., a_middle - 1, x)
        else
            BinarySearch(a_middle + 1, ..., a_n, x)
```

Run time: $B(n)$ = binary search on n elements

$B(n) \geq B\left(\frac{n}{2}\right) + 6$

Master theorem or recursion tree

Merge Sort:

Idea: Recursive sorting algorithm.

If size is 0 or 1, done.

If size ≥ 2, recursively sort left half & right half.

Then merge: 2 3 11 20 | 4 10 15 19

→ need n comparisons to merge these 2 lists

Running time:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$= O(n \log n)$$