COSC 290 Discrete Structures

Lecture 23: Recurrence Relations

Prof. Michael Hay

Friday, Oct. 27, 2017

Colgate University

Plan for today

1. More analysis of non-recursive algorithms
2. Recurrence relations
3. Solving recurrence relations

More analysis of non-recursive algorithms

Analyzing while loops

How do we analyze algorithms with while loops? Example:

procedure BinarySearch(A, x) ▷ Find x in *sorted* array A of length n
  $l := 0$, $u := n - 1$.
  while True **do**
    if $l > u$ **then**
      break
    $m := \lfloor (l + u)/2 \rfloor$. ▷ integer division
    if $x < A[m]$ **then**
      $u := m - 1$.
    else if $x > A[m]$ **then**
      $l := m + 1$.
    else ▷ It must be that $A[m] = x$
      return m
  return -1

Analysis: binary search runtime

Claim: the runtime is at most $c \cdot (\log_2 n + 1)$ where c is some constant (representing work done per iteration of while loop).

Proof: Let w denote the width of the interval that remains to be searched.

$w := u - l + 1$

Lemma: After i iterations of while loop, $w \leq \frac{n}{2^i}$.

Assume (for now) lemma is true. Then after $\log_2 n$ iterations, $w \leq 1$ (because $\frac{n}{2^{\log_2 n}} = 1$).

Once $w \leq 1$, the algorithm iterates *at most* once more.

Thus, algorithm terminates after at most $\log_2 n + 1$ iterations and each iteration incurs constant runtime cost.

Proof of lemma

Lemma: After i iterations of while loop, $w \leq \frac{n}{2^i}$.

Proof by induction on i.

Base case: $i = 0$, then $w = u - l + 1 = (n-1) - 0 + 1 = n$ and $w \leq \frac{n}{2^0} = n$.

Inductive case: assume lemma is true for iteration i. Want to show it is true for iteration $i + 1$.

Let $w = u - l + 1$ denote width after i iterations of while loop. Inductive hypothesis: $w \leq \frac{n}{2^i}$

If w is odd, then there are $(w-1)/2$ elements on either side of midpoint m. If w is even, then right side has $w/2$ elements and the left side has $w/2 - 1 = (w-2)/2$ elements.

In the worst case, the interval is $w/2$ at the end of the loop. Thus, it is $w \leq \frac{n}{2^{i+1}}$ after $(i+1)^{th}$ iteration.

Poll: Asymptotic notation

We have just shown that the runtime of BinarySearch is

A) $O(\log n)$ only
B) $\Omega(\log n)$ only
C) $\Theta(\log n)$ only
D) $O(\log n)$ and $\Omega(\log n)$
E) We haven't shown anything since BinarySearch *might* terminate in a single iteration (if x is in the exact middle of the array)

Recursive algorithms

How do we analyze runtime when the algorithm is recursive? Example:

function RecBinarySearch(A, x, l, u)
  ▷ Given *sorted* array A, find x within $A[l \ldots u]$
  if $l > u$ **then**
    return -1
  $m := \lfloor (l+u)/2 \rfloor$
  if $x = A[m]$ **then**
    return m
  else if $x < A[m]$ **then**
    return RecBinarySearch($A, x, l, m-1$)
  else ▷ It must be that $x > A[m]$
    return RecBinarySearch($A, x, m+1, u$)

Initially called with arguments RecBinarySearch($A, x, 0, n-1$).

Recurrence relations

Recurrence relations

To analyze runtime of recursive algorithm, we can express runtime *recursively*.

A recurrence relation is a function $T(n)$ that is defined (for some n) in terms of the values $T(k)$ for input values $k < n$.

We will express runtime using a recurrence relation $T(n)$ (where n typically captures some measure of input size).

Recurrence relation for RecBinarySearch

To analyze the runtime of RecBinarySearch, we will *not* use the size of the array A. (why not?)

Instead, we will use $w := u - l + 1$, the width of the interval that needs to be searched.

Let $T(w)$ denote an upper bound on the runtime of the algorithm on an input with width w. Let c be some constant.

- $T(0) := c$ ($w = 0$ means $l > u$ and algorithm terminates)
- $T(w) := \underbrace{T(\lfloor w/2 \rfloor)}_{\text{recursive call}} + c$
  (we can apply the same kind of analysis we used for the non-recursive case to show that new width at most $\lfloor w/2 \rfloor$ after checking midpoint)

Poll: Recurrence relation

```
function StackSearch(S, x)
        ▷ Return true if x in stack S
    if S is empty then
        return False
    y := S.pop()
    if y = x then
        result := True
    else
        result := StackSearch(S, x)
    S.push(y)
    return result
```

Let $T(n)$ denote runtime of StackSearch(S, x) on an input of size n. Given $T(0) = c$, what is the correct recurrence for $T(n)$?

A) $T(n) = c$

B) $T(n) = T(\lfloor n/2 \rfloor) + c$

C) $T(n) = T(n-1) + c$

D) $T(n) = T(n) + c$

E) More than one / None of the above

Poll: Recurrence relation

function TREESEARCH(T, x) ▷ Return true if x in tree T
    if T is empty **then**
        return False
    $y :=$ root of T
    if $y = x$ **then**
        return True
    else
        left := TREESEARCH(T_ℓ, x)
        right := TREESEARCH(T_r, x)
        return *left* || *right*

Let $T(h)$ denote the worst-case runtime of TREESEARCH(T, x) on a tree of height h. Given $T(-1) = c$, what is the correct recurrence for $T(h)$?

A) $T(h) = c$
B) $T(h) = T(h/2) + c$
C) $T(h) = T(h-1) + c$
D) $T(h) = 2 \cdot T(h-1) + c$
E) More than one / None of the above

Solving recurrence relations

Solving a recurrence relation

Suppose we have the following recurrence

- $T(0) = c$
- $T(n) = T(\lfloor n/2 \rfloor) + c$

How do we *solve* it? I.e., express it is a (non-recursive) function of n.

Two methods:

1. Guess. Then verify (using proof by induction).
2. Master method.

In COSC 290, we will only look at option 1. (The master method is discussed in Ch. 6.5, and covered in COSC 302.)

Guess + verify

Recurrence for RECBINARYSEARCH:

- $T(0) = c$
- $T(n) = T(\lfloor n/2 \rfloor) + c$

Guess: "Iterate" the recurrence starting at $T(0)$ and look for a pattern. $T(n) = c \cdot (\lfloor \log_2 n \rfloor + 2)$.

Verify: We will prove that $\forall n \geq 1 : T(n) = c \cdot (\lfloor \log_2 n \rfloor + 2)$

Base case: $n = 1$, $T(1) = T(0) + c = 2c$ and $c \cdot (\lfloor \log_2 1 \rfloor + 2) = 2c$.

Inductive case: assume true for $1 \leq m \leq n-1$. Show it's true for n.

$$\begin{aligned} T(n) &= T(\lfloor n/2 \rfloor) + c && \text{def of recurrence} \\ &= c \cdot (\lfloor \log_2 \lfloor n/2 \rfloor \rfloor + 2) + c && \text{inductive hypothesis} \\ &= c \cdot (\lfloor \log_2 n \rfloor - 1 + 2) + c && \text{math tricks with logs and floors} \\ &= c \cdot (\lfloor \log_2 n \rfloor + 2) && \text{simplify} \end{aligned}$$

Poll: guess for stack search

Recurrence for StackSearch:

- $T(0) = c$
- $T(n) = T(n-1) + c$

What is your guess for the solution to $T(n)$?

1. $T(n) = c$
2. $T(n) = c \cdot n$
3. $T(n) = c \cdot (n+1)$
4. $T(n) = c \cdot (n+2)$
5. None of above / More than one

Poll: guess for tree search

Recurrence for TreeSearch:

- $T(-1) = c$
- $T(h) = 2T(h-1) + c$

What is your guess for *asymptotic* solution to $T(h)$?

1. $T(h) = O(1)$
2. $T(h) = O(h)$
3. $T(h) = O(h^2)$
4. $T(h) = O(2^h)$
5. None of the above / more than one