Binary Search

CS 5010 Program Design Paradigms
“Bootcamp”
Lesson 8.3
Introduction

• Binary search is a classic example that illustrates generative recursion
• We will look at a function for binary search
Learning Objectives

• At the end of this lesson you should be able to:
  – explain what binary search is and when it is appropriate
  – explain how the standard binary search works, and how it fits into the framework of generative recursion
  – write variations on a binary search function
Binary Search

• You probably learned about binary search in an array: given an array $A[0:N]$ of increasing values and a target $tgt$, find an $i$ such that $A[i] = tgt$, or else report not found.
Arrays can be modeled as functions

- Racket has arrays (called vectors), but we don't need them.
- Instead of having an array, we'll have a function

  \[ f : [0..N] \rightarrow \text{Integer} \]

  which will give the value of the array at any index.
- We will require that \( f \) be non-decreasing: that is:

  \[ i \leq j \implies f(i) \leq f(j) \]
Let's do the obvious generalization

- Clearly the 0 and N don't matter, so we'll add them as arguments to our function.
Contract and Purpose Statement

;; binary-search-loop
;; : NonNegInt NonNegInt
;; (NonNegInt -> Integer)
;; Integer
;; -> MaybeNonNegInt
;; GIVEN: two numbers lo and hi, a function f,
;; and a target tgt
;; WHERE: f is monotonic
;; (ie, i<=j implies f(i) <= f(j))
;; RETURNS: a number k such that lo <= k <= hi
;; and f(k) = tgt if there is such a k,
;; otherwise false.
Once we've written that, we can write the main function

;; binary-search :
;; NonNegInt (NonNegInt -> Integer) Integer
;; -> MaybeNonNegInt
;; GIVEN: a number N,
;; a function f : NonNegInt -> Integer,
;; and a number tgt
;; WHERE: f is monotonic (ie, i<=j implies f(i) <= f(j))
;; RETURNS: a number k such that 0 <= k <= N
;; and f(k) = tgt if there is such a k,
;; otherwise false.

;; STRATEGY: functional composition
(define (binary-search N f tgt)
  (binary-search-loop 0 N f tgt))
What are the easy cases for binary-search-loop?

• if \( lo > hi \), the search range \([lo, hi]\) is empty, so the answer must be **false**.

• if \( lo = hi \), the search range has size 1, so it's easy to figure out the answer.
What if the search range is larger?

• Insight of binary search: divide it in half.

• Choose a midpoint $p$ in $[lo, hi]$.
  – $p$ doesn't have to be close to the center—any value in $[lo, hi]$ will lead to a correct program
  – but choosing $p$ to be near the center means that the search space is divided in half every time, so you'll only need about $\log(hi-lo)$ steps.
What are the cases?

• $f(p) < tgt$
  – so we can rule out $p$, and all values less than $p$
    (because if $p' < p$, $f(p') \leq f(p) < tgt$).
  – So the answer $k$, if it exists, is in $[p+1, \text{hi}]$

• $tgt < f(p)$
  – so we can rule out $p$ and all values greater than $p$, because if $p < p'$, $tgt < f(p) \leq f(p')$.
  – So the answer $k$, if it exists, is in $[lo, p-1]$

• $tgt = f(p)$
  – then $p$ is our desired $k$. 
As code:

(define (binary-search-loop lo hi f tgt)
  (cond
    [(> lo hi)
      ;; the search range is empty, return false
      false]
    [ (= lo hi)
      ;; the search range has size 1
      (if (equal? (f lo) tgt) lo false)]
   [else (local
            ((define midpoint (floor (/ (+ lo hi) 2)))
             (define f-of-midpoint (f midpoint)))
             (cond
              [(< f-of-midpoint tgt)
                ;; the tgt is in the right half
                (binary-search-loop (+ midpoint 1) hi f tgt)]
              [(> f-of-midpoint tgt)
                ;; the tgt is in the left half
                (binary-search-loop lo (- midpoint 1) f tgt)]
              [else
               ;; midpoint is the one we're looking for
               midpoint]]))])

12
Watch this work

\[
\text{(binary-search-loop 0 40 sqr 49)} \\
= \text{(binary-search-loop 0 19 sqr 49)} \\
= \text{(binary-search-loop 0 8 sqr 49)} \\
= \text{(binary-search-loop 5 8 sqr 49)} \\
= \text{(binary-search-loop 7 8 sqr 49)} \\
= 7
\]
What's the termination argument?

- If lo > hi, the function terminates immediately. Otherwise (- hi lo) decreases at every recursive call.
Summary

• You should now be able to:
  – explain what binary search is and when it is appropriate
  – explain how the standard binary search works, and how it fits into the framework of generative recursion
  – write variations on a binary search function