

Lab: Binary Search

Summary: In this laboratory, we explore different issues related to searching.

Contents:

- Exercises
 - Exercise 0: Preparation
 - Exercise 1: Observing Binary Search
 - Exercise 2: Counting Recursive Calls
 - Exercise 3: Counting Grades with Binary Search
 - Exercise 4: Binary Search, Revisited

Exercises

Exercise 0: Preparation

- a. Start DrScheme.
- b. If you have not done so already, please scan the reading on searching. In particular, you should look at the sample procedures. *Make sure that you understand the purpose of `get-key` in `binary-search`.*
- c. Create a new file for this lab that contains the `binary-search` procedure from that reading (including the documentation for that procedure).
- d. Here's a vector of lists of cartoon characters and sidekicks, sorted by primary character. Put this vector in your file from step c.

```
(define cartoons
  (vector
    (list "Ash" "Misty")
    (list "Asterix" "Obelix")
    (list "Bart Simpson" "Milhouse Van Houten")
    (list "Batman" "Robin")
    (list "Bullwinkle" "Rocky")
    (list "Dexter" "Didi")
    (list "Dick Dastardly" "Muttley")
    (list "Dogbert" "Dilbert")
    (list "Duck Dodgers" "Porky Pig")
    (list "Fred" "Barney")
    (list "Josie" "The Pussycats")
    (list "Peabody" "Sherman")
    (list "Peter Griffin" "Brian")
    (list "Quick Draw McGraw" "Baba Looey")
    (list "Ren" "Stimp")
    (list "Rocko" "Heffer")
    (list "Scooby Doo" "Scrappy Doo")
    (list "Scooby Doo" "Shaggy")
```

```

(list "Scrooge McDuck" "Huey, Dewey, and Louie")
(list "Secret Squirrel" "Morocco Mole")
(list "Spongebob" "Patrick")
(list "Strong Bad" "The Cheat")
(list "Tennessee Tuxedo" "Chumley")
(list "Yogi" "Booboo")
(list "Yu-Gi-Oh" "Joey")
))

```

Exercise 1: Observing Binary Search

- Verify that binary search can correctly find the entry for "Batman".
- Verify that binary search can correctly find the entry for a character of your choice.
- Verify that binary search can correctly find the first entry.
- Verify that binary search can correctly find the last entry.
- Verify that binary search terminates and returns -1 for something that would fall in the middle of the vector and is not there.
- Verify that binary search terminates and returns -1 for something that comes before the first entry.
- Verify that binary search terminates and returns -1 for something that comes after the last entry.
- What does binary search do if there are duplicate keys? (For example, what does it do when the hero is Scooby Doo?)

Exercise 2: Counting Recursive Calls

It is often useful when exploring a recursive algorithm to observe the steps the algorithm performs. In Scheme, we can sometimes observe steps in recursive calls by inserting code to display the parameters of the procedure at each recursive call.

- Add calls to `display` and `newline` to the definition of `binary-search`, so that it prints out the values of `lower-bound` and `upper-bound` each time the kernel procedure is called.
- Redo steps a-g and report on the number of steps each search took.

Exercise 3: Counting Grades with Binary Search

Professor Smith's teaching assistant has created a sorted vector of the grades on exam 2. For example, in a small class, the TA may have written

```

(define smith-grades
  (vector 10 15 16 17 18 18 18 19
         65 66 66 68 70 71 71 72
         75 75 80 80 80 81 82 83
         95 96 97 98 99 102 103))

```

However, Professor Smith typically teaches classes of hundreds, so you should think about a much longer vector. (No, this Professor Smith does not teach at Grinnell.)

Professor Smith would like to know the number of grades that are less than or equal to some value (say, 90).

a. One way to answer this question is to scan the vector from left to right until you find a number greater than the value. (The grade may not appear in the vector, so you can't stop when you hit the value.) Write a procedure, (`count-grades` `vector-of-grades` `grade`) that implements this strategy.

b. A more efficient way to find the position is to use a variant of binary search. (Again, stop when you hit the smallest value greater than the cutoff grade.) That is, you look in the middle. If the middle element is smaller than the cutoff, you recurse on the right half. If the middle element is larger than the cutoff, you recurse on the left half, but include the middle (since it may be the smallest number greater than the cutoff). Write a procedure, (`grade-count` `vector-of-grades` `grade`) that implements this strategy.

For part b, note that you cannot call the binary-search procedure directly. Rather, you will need to use it as a template for your code. That is, copy the procedure and then make modifications as appropriate.

Exercise 4: Binary Search, Revisited

It is sometimes useful to learn not just that something is not in the vector, but where it would fall if it were in the vector. Revise `binary-search` (both the code and the documentation) so that it returns a "half value" if the value being searched for belongs between two neighboring values. For example, if the key being searched for is larger than the key at position 5 and smaller than the key at position 6, you should return $11/2$. Similarly, if the key being searched for is smaller than the key at position 0, you should return $-1/2$. If the key being searched for is bigger than the largest key, return $(- (\text{vector-length } \text{vec}) 1/2)$.

Copyright © 2007 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.