Binary Search example


```plaintext
BINARY-SEARCH(A, w)
low = 0
high = length[A]

while low < high
do  try = int ((low + high) / 2)
   if  A[try] > w
       then high = try
   else if  A[try] < w
       then low = try + 1
   else
       return try
   end if
end do
return NO_ELEMENT
```
Binary Search

In our program, each word is represented in Perl as a scalar, which can be an integer, a floating-point number, or (as in this case) a string of characters.

The list of words is stored in a Perl array: an ordered list of scalars.

Perl Notation:

- Scalars begin with a $ sign,
- Arrays begin with an @ sign.
- Hashes begin with a % sign.

Recall that hashes (aka associative arrays) “map” one set of scalars (the “keys”) to other scalars (the “values”).
Binary Search

# $index = binary_search( @array, $word )
# @array is a list of lowercase strings in alphabetical order.
# $word is the target word that might be in the list.
# binary_search() returns the array index such that $array[$index] is $word.
sub binary_search {
    my ($array, $word) = @_;
    my ($low, $high) = ( 0, @array - 1 );
    while ( $low <= $high ) {
        my $try = int( ($low+$high) / 2 );  # Try the middle element
        $low = $try+1, next if $array->[ $try ] lt $word;  # Raise bottom
        $high = $try-1, next if $array->[ $try ] gt $word;  # Lower top
        return $try;  # We’ve found the word!
    }
    return;  # The word isn’t there.
Binary Search

next
The `next` command is like the `continue` statement in C; it starts the next iteration of the loop:

1. `LINE: while (<STDIN>) {`
2. `next LINE if /^#/;` # discard comments
3. `#...`
4. `}`
Binary Search

- my creates a local (scope) variable
- \@array is a reference to the array named.
- @_ arguments to the subroutine.
- my ($array, $word) = @_;, assigns the first two subroutine arguments to the scalars $array and $word.
- my ($low, $high) = ( 0, @$array - 1 ); declares and initializes two more scalars.
  - $low is initialized to 0—actually unnecessary, but good form.
  - $high is initialized to @$array - 1, which dereferences the scalar variable $array to get at the array underneath. In this context, the statement computes the length (@$array) and subtracts 1 to get the index of the last element.
#!/usr/bin/perl
#
# bsearch - search for a word in a list of alphabetically ordered words
#
# Usage: bsearch word filename

$word = shift;                           # Assign first argument to $word
chomp( @array = <> );                   # Read in newline-delimited words,
                                             # truncating the newlines
($word, @array) = map lc, ($word, @array); # Convert all to lowercase
$index = binary_search(@array, $word);   # Invoke our algorithm

if (defined $index) { print "$word occurs at position $index.\n" } else { print "$word doesn’t occur.\n" }

sub binary_search {
  my ($array, $word) = @_;
  my $low = 0;
  my $high = @array - 1;
  while ( $low <= $high ) {
    my $try = int( ($low+$high) / 2 );
    $low = $try+1, next if $array->[[$try] lt $word;
    $high = $try-1, next if $array->[[$try] gt $word;
    return $try;
  }
  return;
}
Binary Search

Try it out

- % perl bsearch.pl binary /usr/dict/words
- % perl bsearch.pl binary /usr/share/dict/words # OS X
- binary occurs at position 22369.