COMP 250

Lecture 12

recursion 2:

power, binary search

Oct. 4/5, 2017

Recall: Converting to binary (iterative)

```
k   =  0
while  n > 0 {
    b[ k ]  =  n % 2
    n       =  n /  2
    k++
}
```

Recall that n in binary needs approximately $log_2 n$ bits.

Converting to binary (recursive)

```
toBinary( n ){
    if  n > 0 {
        print  n % 2
        toBinary( n / 2 )
    }
}
//   prints  b[ k ],    k = 0, 1, .....
```

Power (x^n)

Let x a positive integer and let n be a positive number.
x has some number of bits e.g. 32.

```
power(x, n){   //  iterative
   result = 1
   for i = 1 to n
       result = result * x
   return  result
}
```

How to compute power() using recursion ?

Example

$$x^{18} = x^{17} * x$$

How to compute power() using recursion ?

More interesting approach:

$$x^{18} = x^9 * x^9$$

$$x^9 = x^4 * x^4 * x$$

$$x^4 = x^2 * x^2$$

```
power( x, n ){        //  recursive
   if n == 0
      return 1
   else if  n == 1
      return x
   else{
      tmp = power( x,  n/2 )       //  n/2 is integer division
      if n is even
         return  tmp*tmp           //    one multiplication
      else
         return  tmp*tmp*x         //    two multiplications
   }
}
```

Example: x^{243}

$$n = (243)_{10} = (11110011)_2$$

Number of multiplies is 5*2 + 2*1 = 12. Why?

The highest order bit is the base case, and doesn't require a multiplication.

ASIDE

The recursive method uses fewer multiplications than the iterative method, and thus the recursive method *seems faster.*

Q: Is this recursive method indeed faster?

A: No. Why not ?

Hint: Let x be a positive integer with M digits.

x^2 has about ? digits.

x^3 has about ? digits.

:

x^n has about ? digits.

Hint: Let x be a positive integer with M digits.

x^2 has about 2M digits.

x^3 has about 3M digits.

:

x^n has about $n * \mathrm{M}$ digits.

We cannot assume that multiplication takes 'constant' time.
Taking large powers gives very large numbers.
In Java, use the BigInteger class.

(See Exercises for more details.)

Binary Search

-75
-31
-26
-4
1
6
25
26
28
39
72
141
290
300

Input:

- a *sorted* list of size n
- the value that we are searching for

Output:

If the value is in the list, *return its index*.
Otherwise, *return -1*.

Binary Search

-75
-31
-26
-4
1
6
25
26
28
39
72
141
290
300

Example: Search for 17.

What is an efficient way to do this ?

Think of how you search for a term in an index. Do you start at the beginning and then scan through to the end? (No.)

Examine the item at the middle position of the list.

	-75	low = 0
	-31	
	-26	
	-4	
	1	
	6	
compare 17 to →	25	mid = (low + high) / 2
	26	
	28	
	39	
	72	
	141	
	290	
	300	high = size - 1

search for 17 here

-75
-31
-26
-4
1
6
25
26
28
39
72
141
290
300

low = 0

mid = (low + high) / 2

high = size - 1

compare 17 to →

-75	low = 0
-31	
-26	mid = (low + high) / 2
-4	
1	
6	high
25	
26	
28	
39	
72	
141	
290	
300	

-75	
-31	
-26	
-4	low = 0
1	mid = (low + high) / 2
6	high
25	
26	
28	
39	
72	
141	
290	
300	

search for 17 here

compare 17 to →

-75
-31
-26
-4
1
6
25
26
28
39
72
141
290
300

low = 0

mid = (low + high) / 2

high

-75
-31
-26
-4
1
6
25
26
28
39
72
141
290
300

search for 17 here

low = high

-75
-31
-26
-4
1

compare 17 to → 6 low = high

25
26
28
39
72
141
290
300

so return index -1
(value 17 not found)

```
binarySearch( list, value){
   low = 0
   high = list.size - 1
   while  low <= high  {

          ....

   }
   return  -1     // value not in list
}
```

```
binarySearch(list, value ){
    low = 0
    high = list.size - 1
    while  low <= high  {
        mid = (low + high)/ 2    //  if high == low + 1, then mid == low
        if list[mid] == value
            return mid
        else{

            //     modify low or high

            }
    }
    return  -1     // value not in list
}
```

```
binarySearch(list, value ){
   low = 0
   high = list.size - 1
   while  low <= high  {
        mid = (low + high)/ 2        //  if high == low + 1,   then mid == low
         if list[mid] == value
              return mid
         else{   if value < list[mid]
                        high = mid - 1        // high can become less than low.
                   else
                        low  = mid + 1
                }
     }
     return  -1     // value not found
}
```

```
binarySearch(list, value ){

}
```

how to make this recursive ?

```
binarySearch( list, value, low, high ){     //  pass as parameters

    if  low <= high {                          //   if instead of while
        mid = (low + high)/ 2
        if value == list[mid]
            return mid
        else   if value < list[mid]
            return  binarySearch(list, value,  low,     mid - 1 )
                                // mid-1  can be less than low
        else
            return  binarySearch(list, value,  mid+1,  high)
        }
    else
        return   -1
}
```

Observations about binary search

Q: How many times through the while loop ? (iterative)

How many recursive calls? (recursive)

A:

Observations about binary search

Q: How many times through the while loop ? (iterative)
How many recursive calls? (recursive)

A: Time to search is worst case O($log_2 n$) where n is size of the list. Why? Because each time we are approximately halving the size of the list.

Three "O($log_2 n$)" problems

- Converting a number to binary
- Power(x, n) -- how many multiplies ?
- Binary search in a sorted array

The binary property is related to the base 2 of the log.