

# lecture 11

more recursion

- 20 questions
  - decimal to binary
  - power
  - binary search
- } all algorithms require  $\sim \log n$  steps

## 20 Questions with numbers

I am thinking of a positive integer  $n$  between 0 and  $2^{20}-1$  ( $\approx 10^6$ )

$$2^{10} = 1024 \approx 10^3$$

Ask me a sequence of yes/no questions to find  $n$ .

Question  $i$ : does the  $i^{\text{th}}$  bit of the binary representation have value 1? ( $i = 0, \dots, 19$ )

Alternatively: does  $19-i^{\text{th}}$  bit have value 1?

(Recall from lecture 2)

Algorithm: given  $n$  in decimal, convert it to binary  
i.e.  $(b_{m-1} \dots b_2 b_1 b_0)$

```
i ← 0
while n > 0 {
  b[i] ← n % 2
  n ← n / 2
  i ← i + 1
}
```

Recursive Algorithm  
(prints  $b_0 b_1 b_2 \dots$  in that order)

```
decimalToBinary(n) {
  if n > 0 {
    print n % 2
    decimalToBinary(n/2)
  }
}
```

How many bits  $m$  are required to write  $n$  in binary?

$n$	$n$ binary	# bits
0	0	1
1	1	1
2	10	2
3	11	2
4	100	3
5	101	3
6	110	3
7	111	3
8	1000	4

How many bits  $m$  are required to write  $n$  in binary?

$$n = b_{m-1}2^{m-1} + b_{m-2}2^{m-2} + \dots + b_12^1 + b_0$$

$$\leq 2^{m-1} + 2^{m-2} + \dots + 2^1 + 2^0$$

$$< 2^m$$

$\therefore \log_2 n < m$   
 ← not necessarily an integer  
 ← an integer

$$n = b_{m-1}2^{m-1} + b_{m-2}2^{m-2} + \dots + b_12^1 + b_0$$

$$\geq 2^{m-1} \quad (\text{Here we assume } b_{m-1} = 1)$$

Thus,  $\log_2 n \geq m-1$

Thus,  $m-1 = \text{floor}(\log_2 n)$

i.e.  $m = \text{floor}(\log_2 n) + 1$

The decimalToBinary( $n$ ) algorithm takes  $\sim \log_2 n$  steps (regardless of whether recursion is used).

Power i.e.  $x^n$

Given double  $x$  and a positive integer  $n$ , compute  $x^n$ .

It is easy to do this with  $n$  steps using a for or while loop.

i.e. see PolyEval(double)

Is there a faster way?

Yes, for example:

$$x^{18} = x^9 \cdot x^9$$

$$x^9 = x^4 \cdot x^4 \cdot x$$

$$x^4 = x^2 \cdot x^2$$

$$x^2 = x \cdot x$$

```
power(x, n) {
  if n == 0
    return 1
  else {
    tmp ← power(x, n/2)
    if n % 2 == 0
      return tmp * tmp
    else
      return tmp * tmp * x
  }
}
```

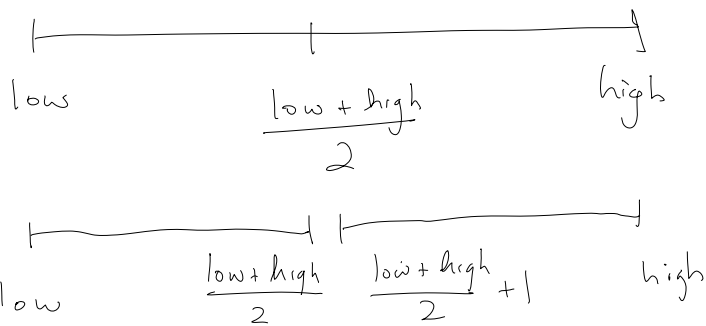
Again,  $\sim \log n$  steps are required.

## Binary Search

Given an array of sorted elements, e.g. numbers, strings, find index (i.e. array position) of given element.  
(Return -1 if element is not present.)

```
binarySearch(a, e, low, high) {
    ...
}
```

- a is the array
- e is the desired element
- position of element is in [low, high]



$e \leq a[(low + high) / 2]$  ?

```
binarySearch(a, e, low, high) {
    if (low < high) {
        mid ← (low + high) / 2
        if e ≤ a[mid]
            return binarySearch(a, e, low, mid)
        else
            return binarySearch(a, e, mid + 1, high)
    }
    else { if e == a[low]
            return low
        else return -1 }
}
```

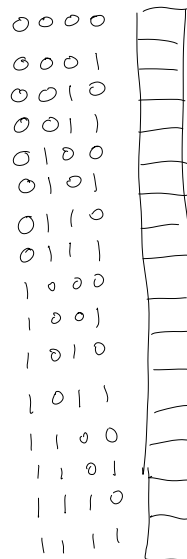
Example:

binarySearch(83, 0, 14)

0	-75
1	-31
2	-26
3	-4
4	1
5	6
6	25
7	26
8	28
9	39
10	72
11	141
12	295
13	296
14	300

low	high	mid
0	14	7
8	14	11
8	11	9
10	11	10
11	11	

returns -1 since 83 ≠ 141



Special case:

If array length is a power of 2,

then the  $i$ th step of binary search

determines if

the  $m-i$ th

bit of the result

index is 0 or 1.