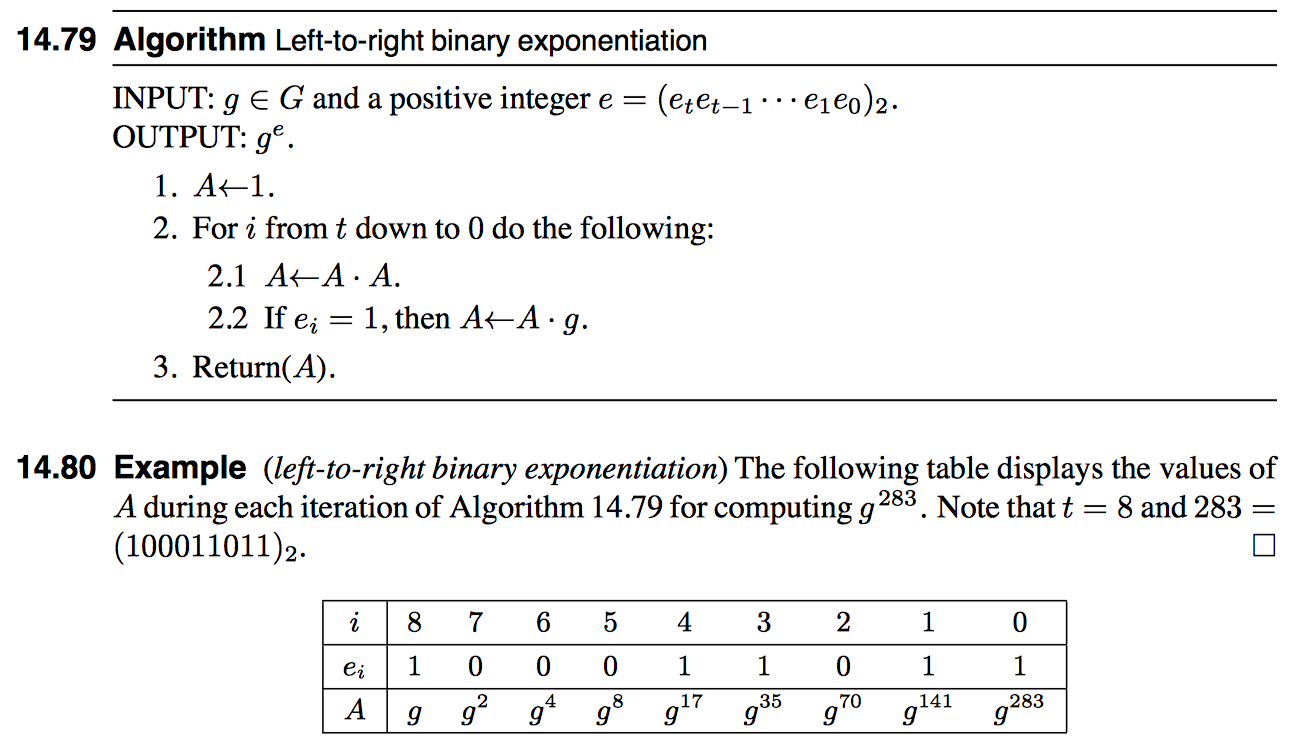
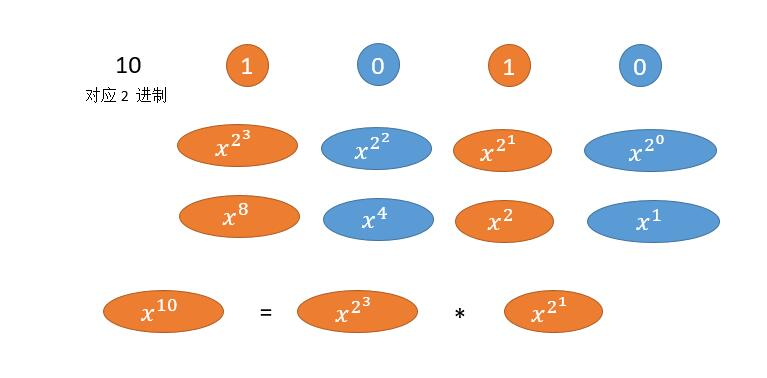
1. -1 \* -2147483648 (or any lowest possible int/double) will result in overflow and you still get a -1 \* 2147483648.
2. An article about 1’s complement and 2’s complement (java uses 2’s complement). <https://medium.com/@malaybiswas/binary-addition-multiplication-subtraction-and-division-55ad8d27ff02>
3. Remember to due with the lowest int: <https://leetcode.com/explore/learn/card/recursion-i/256/complexity-analysis/2380/discuss/19563/Iterative-Log(N)-solution-with-Clear-Explanation>
4. Multiplication by bit manipulation: 



1. Unique binary tree II.
2. Square root: avoid using \* and + (which might cause overflow), and use / and – instead.
3. middle = (left + right) >> 1 （or /2） might result in overflow.  use low+(high-low)/2 instead. （or (int)(((long)low+(long)high)/2); ）Notice that (int) has higher precedence than any algebraic operations.
4. Notice there are two templetes of binary search!
5. Normally you will want to set the condition to left <= right, so that you can check every element in the array. <https://leetcode.com/explore/learn/card/binary-search/125/template-i/950/>
6. Sometimes you will want to have access to elements in another location while you are checking. So that you want to set the condition to left < right. And do a post-iteration check.

<https://leetcode.com/explore/learn/card/binary-search/126/template-ii/947/>

1. Find pivot?

int left = 1; int right = n;

while (left < right) {

int middle = left + (right - left) / 2;

if (!isBadVersion(middle)) {

left = middle + 1;

} else {

right = middle;

}

}

return left;

1. Sometimes in binary search, you’ll encounter situation where you can’t decide left/right should equal mid + 1/mid/mid-1. What you can do is simply right – 1 or left + 1 so that the process can eventually terminate. See this example: