Stephen Woodbury   4/30/17   CMPS 102
I have read and agree to the collaboration policy. Stephen Woodbury.
Collaborators: none

Assignment 2_2 : Divide and Conquer II

Assumptions: 1) Guesser knows that the number resides in [1,n].
2) Guesser knows that on first guess, if number guessed doesn't equal the chosen number, that **Guess(x)** will return "*warmer*", signifying that the number guessed wasn't the right answer.
3)Guesser knows that if on the first guess, the guessed number was correct, **Guess(x)** would have returned, "*You guessed it!*".
4) Guesser knows the following behavior of the **Guess(x)** method on guess #2 and above.
  a) "*Warmer*" is returned if the guessed number is closer to the chosen number than the last guess passed to **Guess(x)**.
  b) "*Colder*" is returned if the guessed number is farther from the chosen number than the last guess passes to **Guess(x)**.
  c) "*Colder*" is returned if the guessed number has a distance to the chosen number that is equal to the distance between the chosen number and the last guess sent to **Guess(x)**.
5) **Guess(x)** has access to int variable L, which holds the last guess sent to **Guess(x)**. It also has access to int variable C, which holds the number the guesser must guess. This variable is not accessible to the user.
Components: int number(number holder), int b (start of range), int e (end of range), string response (store result of ***Guess(x)****).*

Algorithm - Function form

```
1) initialize b = 1, e = n, response = "", number = uninitialized
2) int find(b, e)
3)   response = Guess{floor[(b+e)/2]};
4)   if(response == "you guessed it!")
5)     number = floor[(b+e)/2];
6)   else
7)     response = Guess{floor[(b+e)/2]+1};
8)     if(response == "you guessed it!")
9)       number = (floor[(b+e)/2]+1);
10)    else if(response == "warmer")
11)      number = find( (floor[(b+e)/2]+1), e );
12)    else
13)      number = find( b, floor[(b+e)/2] );
14)  return number;
```

Algorithm Description: Our algorithm is very similar to a binary search. Given a range 1->n, our algorithm calls find(1,n). Then the median is calculated on b and e and guessed. If the guess is correct, then number is set to the median. If the guess is incorrect (the output stored in response from guess is anything but 'you guessed it!'), the median+1 is guessed. If the output stored in response is, 'you guessed it!', set number equal to median+1. If instead the response says, 'warmer', set number equal to recursive call on find on bounds median+1 to e. Why? Because we know that if the median and median+1 isn't the right number and that if median+1 is closer to our wanted number than the median, that anything before the median can be excluded from our search. If the response doesn't say 'warmer' and doesn't say 'you guessed it!', then set number equal to recursive call on find on bounds b to median. Why? Because we know that if the median and median+1 isn't the right number and that if median+1 is farther from our wanted number than the median, that anything after the median can be excluded from our search. Eventually, we'll zero in on our wanted number and recursively return it through the calls.

<u>Proof of Correctness : Termination</u>

<u>Claim:</u> Algorithm terminates after at most log(n) recursive calls.
<u>Pf:</u> Each time the function find is recursively called, n is halved. There are only log(n) possible recursion calls before n=1. When n=1, our number will have been found and the function returned.

<u>Proof of Correctness : Choice Accuracy</u>

<u>Claim:</u> The correct number we want to guess is in every recursive call's domain just as it is in our original domain
<u>Pf:</u> Analyze when our algorithm cuts the domain in any way.
Case 1: Line 11: The condition for line 11 to be triggered is that response == "warmer". The response is set to "warmer" when Guess is called on (the median of the domain) + 1 in line 7. The only way guess would have returned warmer is if the number previously guessed was further away from our wanted number than our median+1. The number previously guessed in every scenario where response is set to "warmer" in line 7 is the median itself, which we know to not be the correct number from line 4 (wouldn't have triggered the else statement). Since median+1 is closer to our wanted number than median and that median isn't our wanted number, we know that anything before or is our median cannot possibly be the number we want so we cannot possibly cut out the number we want by the domain cut in line 11.
Case 2: Line 13: Very much the same argument for line 11, however, instead of our response being set to 'warmer, it's set to anything but "warmer" and "you guessed it!", "colder" is the only other thing **Guess(x)** can return. We again analyze Median+1 and Median and if we find that neither are the number we want and that response has been set to "colder" by line 13, then we know that anything after Median cannot possibly hold our wanted number. This means we cannot possibly cut out our wanted number. QED
NOTE: Another thing that one would be worried about is when our guess is the same distance to our wanted number as is between our previous guess and wanted number. However, this doesn't matter as the only time this can happen is when we select our new median from new bounds in line 3. Yet, in life 4, we only check to see if response stores "you guessed it!" which it won't unless the median we have selected is our wanted number. Then in line 7 we set a new guess without cutting bounds. Since it's right next to our previous guess, there's no way we could run into this problem.

<u>Time Complexity:</u>

1 : no cost
2 : no cost
3 : O(1)
4 : O(1)
5 : O(1)
6 : no cost
7 : O(1)
8 : O(1)
9 : O(1)
10: O(1)
11: T(n/2)
12: no cost
13: T(n/2)
14: no cost

<u>Space Complexity</u>

Guess uses 2 variables, one to hold last guess and one to hold chosen number : O(1).
find uses 4 variables, one to hold start of range, one for the end of range, one to hold output of Guess, and one to store number which holds the number we want : O(1).
Total: O(1).

Recurrence Statement:T(n)=T(n/2)+4 = T(n/(2^k))+4k. N/(2^k)=4, k=log_2(4n). T(n)=4log_2(4n) + 4; *O(log(n))*, Shows us #guesses=log_2(n) + O(1).