Amy Jiang

CS 211

March 16, 2017

Homework#3 Recursion

|  |
| --- |
|  |
|  |

#1. The getNumberEqual calls itself.

This recursive case has a base value: when n <= 0 we return 0;

The recursive call is able to diminish the size of the problem by

Using this statement:

return getNumberEqual(x, n-1, desiredValue) + count;

At each recursive call the getNumberValue’s integer which at the first call is the array’s length is diminished by 1.

This checks every value in the array by having the n-1 check the index before it because we essentially start at the end of the array and go backwards until we hit index 0 which is the base case. At that point the function stops and we return all the values and check to see if the desired value was in the array, if so the count increases by one each time our desired value is in the array.

#6. In this recursive function we do not have a base case so the recursive call will be never ending. Another issue is that the recursive call is not reducing its problem into smaller problems. And it never reaches its base case since it doesn’t have any.

#11.

In this program we want to check if both a & b added together and then divided by 2 is equal or less than the n value. The recursive call reduces the b value by one which is the calculated c = (a+b)/2 value.

OUTPUT: given getValue(1, 7, 7);

Enter: a = 1 b = 7 //therefore n is not < = c \*c

Enter: a = 1 b = 3 / n is < = c \*c

Leave: a = 1 b = 3

Leave: a = 1 b = 7

2 //This is the value that b was changed to which made the base case:

c\*c <= n true

#18a.

If anything, the array should be in sorted order so that the binary search can be applied effectively. For binary search we need an ordered array.

#18b

The index of the integer in the array that binary search would examine first is the middle of the array. If the middle is the target then it is returned, if not it checks if the middle value with the desired value and if less than it searches first half of the array. If desired value is larger than middle value it searches second half of the array.

#21a

|  |
| --- |
|  |

f(6) = f(6-1) + 3 x f(6-5)

= f(5) + 3 x f(1)

= 5 + 3

f(6) = 8

|  |
| --- |
|  |

f(7) = f(7-1) + 3 x f(7-5)

= f(6) + 3 x f(2)

= 8 + 3

f(7) = 11

|  |
| --- |
|  |

f(12) = f(12-1) + 3 x f(12-5)

= f(11) + 3 x f(7)

= 62 + 3 x 11

= 62 + 33

f(12) =95

|  |
| --- |
|  |

f(15) = f(15-1) + 3 x f(15-5)

= f(14) + 3 x f(10)

= 206 + 3 x 38

= 206 + 114

f(15) = 320

Note: f(7) = 11 | f(8) = 14 | f(9) = 23 | f(10) = 38 | f(11) = 62 | f(12) = 95 | f(13) = 137 | f(14) = 206

|  |
| --- |
|  |

#21b

#include <iostream>

#include <string>

using namespace std;

int recur(int n) ;

int main()

{

cout << recur(15);

}

int recur(int n) {

if(n <= 3 && n > 0)

return 1;

else if (n == 4)

return 3;

else if (n == 5)

return 5;

else {

return recur(n-1) + (3\*recur(n-5));

}

}