Stav Rones
001201196
2/15/20
HW03

Summary of approach:

For problem 1, I used the method of dividing a decimal number by 2 and returning the remainder until the number is less than 1. For the recursive method, the recursion is called before the remainder is printed so that the binary number is in the correct order with the MSBs on the left hand side. For the iterative method, I used the stack code from class to implement a template class for a stack data structure. First, the size of the binary number was calculated with $\log_2(n)$, and the number was divided by 2 this many times. Each time, the remainder was pushed to the stack. Then, a while loop that executed when the stack was not empty printed the top value and then popped it. For the seconds problem, a for loop that iterated through a list of test values in main was checked with each function. A tabular format was used so the results could be easily analyzed. To determine the time complexity of these functions, I analyzed the recursive arguments. Arguments of n-1 meant O(N) complexity, while arguments of n/2 meant O(logn) complexity. For F3, this time complexity was 2^n because each round of recursion created another 2 call, meaning that after 3 calls there will be 8 recursive cases. For the Fibonacci dynamic recursive function, I noticed that in the regular case, the same Fibonacci numbers were being calculated many times over as a result of needing to calculate both n-1 and n-2. Utilizing dynamic memory allocation, each time the argument was called an array of size n was created with all values being -1. Every time the Fibonacci case was called with k, the array would change from -t to k to indicate that this value does not need to be another recursive case. When the recursive cases were being called, it passed through a conditional statement that checked if the value was -1.

Summary of Skills acquired and challenges faced

This assignment required the knowledge of how to implement recursion, a stack data structure, object oriented principals and time complexity calculations. One challenge that I faced was figuring out how to drastically reduce the time complexity of the Fibonacci sequence calculator just by using a single array, but in the end I figured it out by analyzing the time complexity of the normal recursion and realizing that a simple fix could yield tremendous improvements.

```
Stavs-MacBook-Pro:HW03 stavrones$ ./output
Enter a decimal number: 38
The binary number using recursive method = 100110
The binary number using iterative method = 0100110

-------- PROBLEM 2 -------

n    F1                  F2                  F3
===  ===                 ===                 ===
1    2                   2                   2
3    8                   8                   8
5    32                  32                  32
16   65536               65536               65536
24   16777216            16777216            16777216
26   67108864            67108864            67108864
30   1073741824          1073741824          1073741824

Each function returns 2^n:
Tiem Complexity:
  F1: O(log(n))
  F2: O(n)
  F2: O(2^n)
Therefor, the worse run time is F3

-------- PROBLEM 3 -------

n    Recursion           Dynamic Recursion   Runtime1 (microseconds)       Runtime2 (microseconds)
===  ===                 ===                 ===                           ===
5    5                   5                   0                             0
10   55                  55                  1                             0
20   6765                6765                85                            0
30   832040              832040              7723                          0
40   102334155           102334155           978946                        1

The runtime of the purely recursive method is O(2^n).
With the help of a dynamic array that keeps track of which numbers
were already calculated recursion, this cuts the runtime down significantly
because the time complexity is now O(n)

Stavs-MacBook-Pro:HW03 stavrones$
```