Quinn Roemer

CISP - 430

Assignment 2

2/8/2018

Program 2.0 - Factorial

Description:

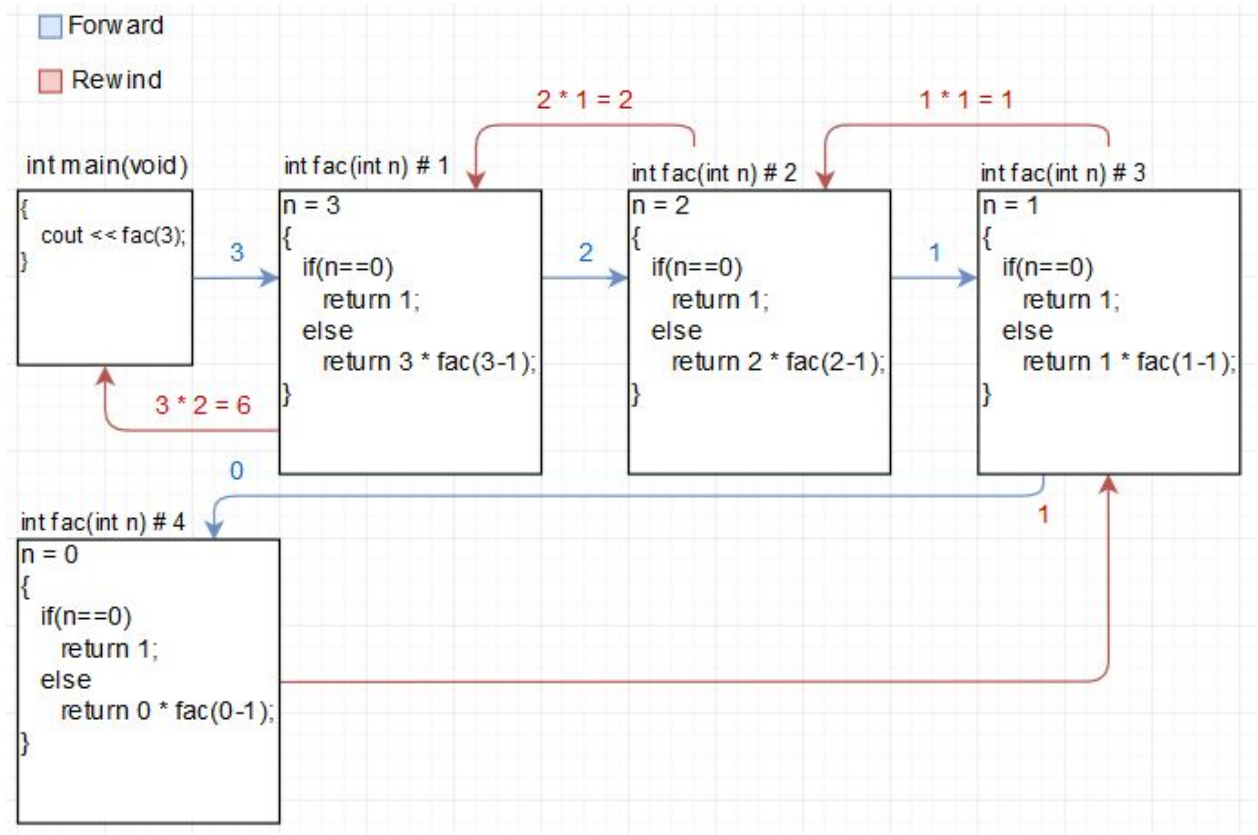
The goal for this part of the assignment was to implement a recursive function to calculate factorial numbers. The professor provided the code needed. However, my task was to create a detailed diagram that showed how the recursive function actually performed its work. In addition, I needed to explain the BigO of the function.

Source Code:

```
//Code written by Professor Ross.

#include <iostream>

using namespace std;

int fac(int n)
{
 if(n == 0)
 {
        return 1;
 }
 else
 {
        return n * fac(n - 1);
 }
}

 int main(void)
{
 cout << fac(3) << endl;
}
```

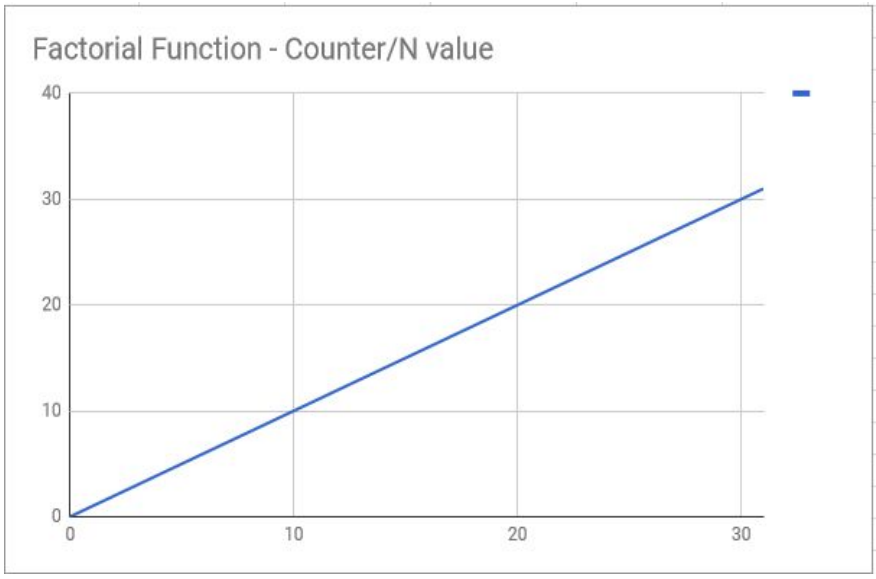
Output:

Diagram:



BigO of Function:

Because N is directly proportional to how many individual instances of the function is required we can assume that the BigO of this function is O(N). To help prove this I analyzed the function empirically. Here are my results on a graph.



Program 2.1 - Fibonacci Numbers

Description:

The goal for this part of the assignment was to implement a recursive function to calculate fibonacci numbers. The professor provided the code needed. However, my task was to create a detailed diagram that showed how the recursive function actually performed its work. In addition, I needed to explain the BigO of the function.

Source Code:

```
//Code written by Professor Ross
#include <iostream>

using namespace std;

int fib(int n)
{
 if(n == 0)
 return 0;
 if (n == 1)
 return 1;
 else
 return fib(n - 1) + fib(n - 2);
}

int main(void)
{
 cout << fib(6) << endl;
}
```

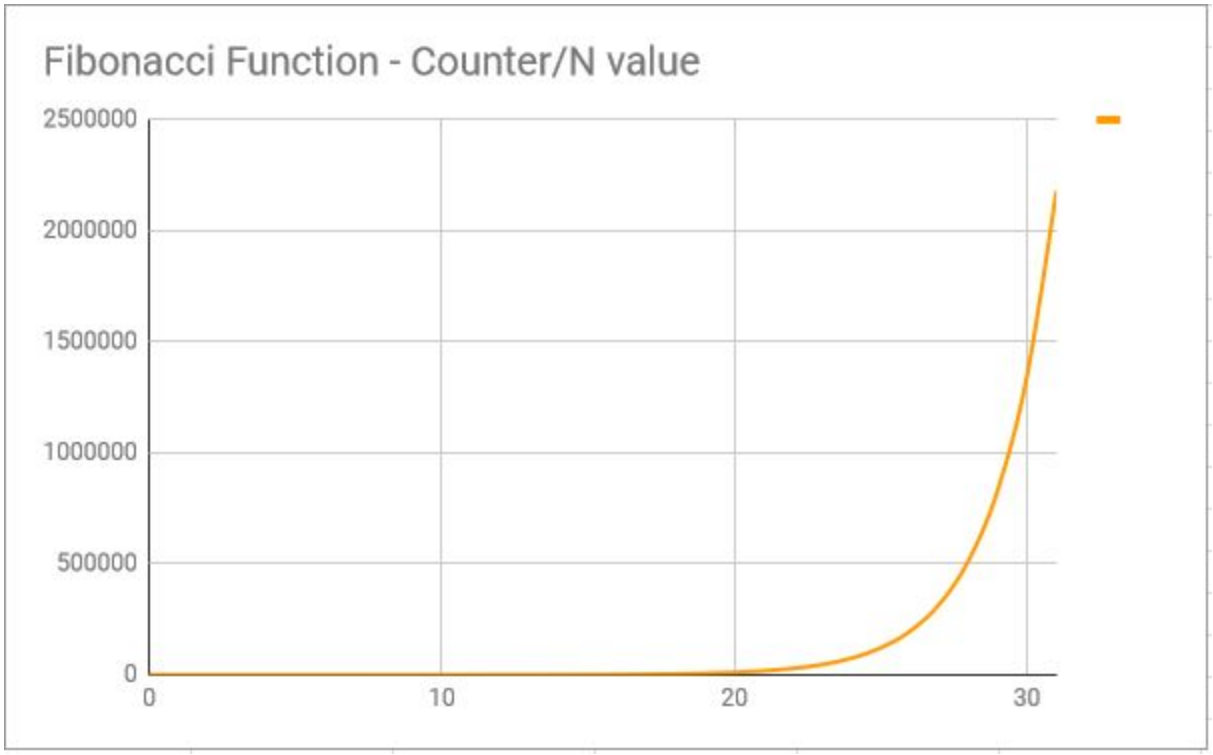
Output:

Diagram:

The following diagram is very large. To contain it in this document I printed on its own pages. These pages flip out from the following pages so that the entire diagram can be viewed.

BigO of Function:

Because N is directly proportional to how many individual branches of the tree we need we can assume that the BigO of this function is O(2^N). To help prove this I analyzed the function empirically. Here are my results on a graph.



Program 2.2 - Bob

Description:

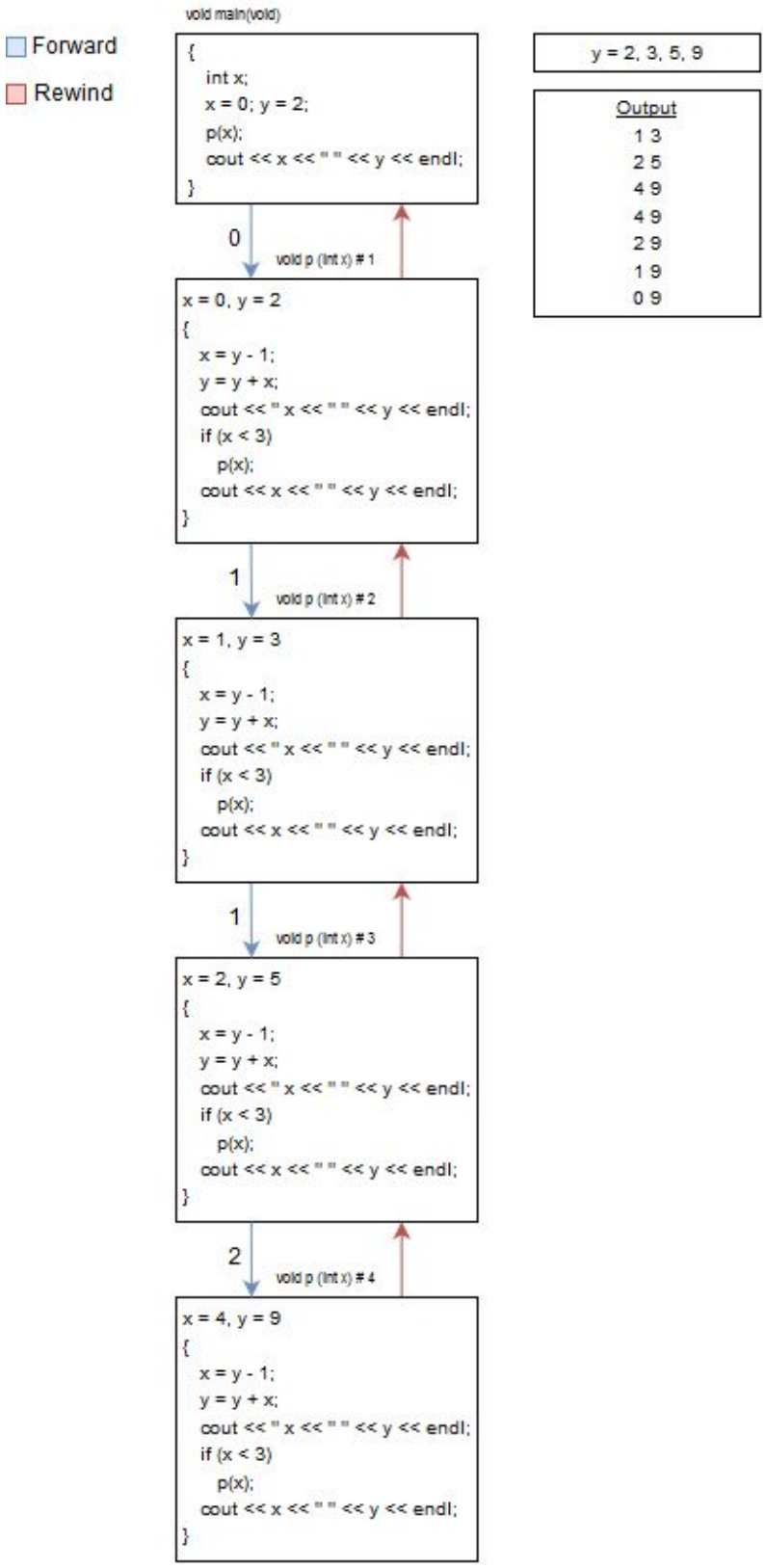
The goal for this part of the assignment was to implement a recursive function programmed by my professor. My task was to create a detailed diagram that showed how the recursive function actually performed its work. In addition, I needed to explain the BigO of the function.

Source Code:

```
//Code written by Professor Ross
#include <iostream>
using namespace std;
int y;
void p (int x)
{
 x = y - 1;
 y += x;
 cout << x << " " << y << endl;
 if (x < 3)
 p(x);
 cout << x << " " << y << endl;
}
int main (void)
{
 int x;
 x = 0;
 y = 2;
 p(x);
 cout << x << " " << y << endl;
}
```

Output:

Diagram:


Forward
Rewind
void main(void)
{
int x;
x = 0; y = 2;
p(x);
cout << x << " " << y << endl;
}
0
void p (int x) # 1
x = 0, y = 2
{
x = y - 1;
y = y + x;
cout << " x << " " << y << endl;
if (x < 3)
p(x);
cout << x << " " << y << endl;
}
1
void p (int x) # 2
x = 1, y = 3
{
x = y - 1;
y = y + x;
cout << " x << " " << y << endl;
if (x < 3)
p(x);
cout << x << " " << y << endl;
}
1
void p (int x) # 3
x = 2, y = 5
{
x = y - 1;
y = y + x;
cout << " x << " " << y << endl;
if (x < 3)
p(x);
cout << x << " " << y << endl;
}
2
void p (int x) # 4
x = 4, y = 9
{
x = y - 1;
y = y + x;
cout << " x << " " << y << endl;
if (x < 3)
p(x);
cout << x << " " << y << endl;
}
y = 2, 3, 5, 9
Output
1 3
2 5
4 9
4 9
2 9
1 9
0 9

BigO of Function:

After analyzing this function by using different x and y values I have determined that it has to be O(logN). It might even by O(1). Unfortunately I had difficulty analyzing this function using the empirical method. As a result, I was unable to produce a graph.

Program 2.3 - Hanoi Towers

Description:

The goal for this part of the assignment was to implement a recursive function programmed by my professor. My task was to create a detailed diagram that showed how the recursive function actually performed its work. In addition, I needed to explain the BigO of the function.

Source Code:

```
//Code written by Professor Ross.
#include <iostream>
using namespace std;
void Hanoi (int N, int Start, int Goal, int Spare)
{
 if (N == 1)
 {
        cout << "Move disk from peg " << Start << " to peg " << Goal << endl;
 }
 else
 {
        Hanoi(N - 1, Start, Spare, Goal);
        cout << "Move disk from peg " << Start << " to peg " << Goal << endl;
        Hanoi(N - 1, Spare, Goal, Start);
 }
}
int main()
{
 Hanoi(3, 1, 3, 2);
}
```

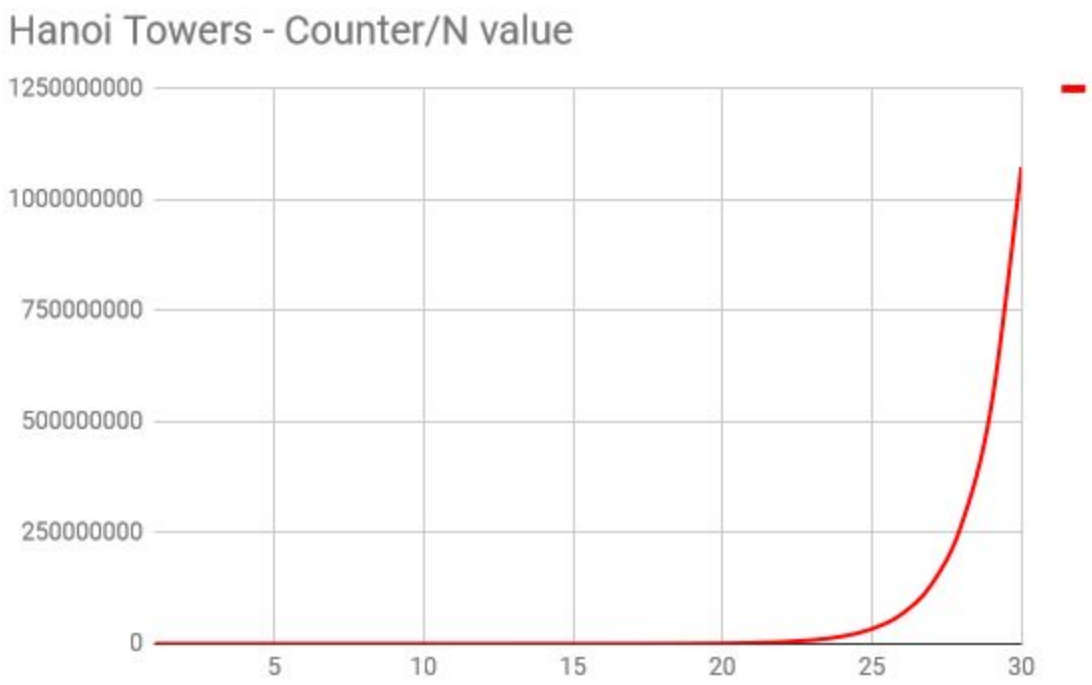
Output:

Diagram:

Since the following diagram is very large I have added it on its own page. Please refer to the next page for the diagram.

BigO of Function:

After analyzing this function by using my diagram I have determined since the function outputs in a tree like pattern that the BigO has to be O(2^N). I also analyzed this function empirically and produced a graph to further confirm this.



Program 2.4 - 4x4 Queens

Description:

The goal for this part of the assignment was to implement a recursive function programmed by my professor. My task was to create a detailed diagram that showed how the recursive function actually performed its work. In addition, I needed to explain the BigO of the function.

Source Code:

```
//Code Written by Professor Ross.
#include <iostream>
using namespace std;
int sol[5];

void printSolution(void)
{
 for (int i = 1; i < 5; i++)
 {
        cout << sol[i] << " ";
 }

 cout << endl;
}

bool cellok(int n)
{
 int i;

 for (i = 1; i < n; i++)
 {
        if (sol[i] == sol[n])
        {
        return false;
        }
 }

 for (i = 1; i < n; i++)
 {
        if (((sol[i] == (sol[n] - (n - i)))) || (sol[i] == (sol[n] + (n - i))))
        {
        return false;
        }
 }

 return true;
```

```
}

void build(int n)
{
 int p = 1;

 while (p <= 4)
 {
        sol[n] = p;

        if (cellok(n))
        {
        if (n == 4)
        {
        printSolution();
        }
        else
        {
        build(n + 1);
        }
        }
        p++;
 }
}

int main(void)
{
 build(1);
}
```

Output:

Diagram:

Since the following diagram is very large I have added it on its own page. Please refer to the next page for the diagram.

BigO of Function:

After analyzing this function by using my diagram I have determined since the function outputs in a proportional pattern the BigO has to be 0(N). This function proved hard to analyze empirically since it was difficult to increase the functions workload to see how a counter responded.

Conclusion

Before this assignment the idea behind recursive function for me was vague. However, as a result of this assignment and the hours that I spent diagramming each individual function I feel a lot more comfortable with the concept. It will be interesting to see if I am able to program my own recursive function in the next assignment.