Big O Calculation:

At first it was a bit confusing to interprete the data seeing as how the default only displayed 10 instances of each function running. The First thing I did is plot out 100 instances of each Function.

The first function only gave me 31 instances, seing as how it blew up in count at an exponential rate. I then ploted theise values and verified that indeede it grew exponentialy. Upon seeing the counts, i was able to actualy calculate the exact Big O formula. 2^n - 1. I say exact because of all the formulas, this is the only one that gives me the exact count given n.

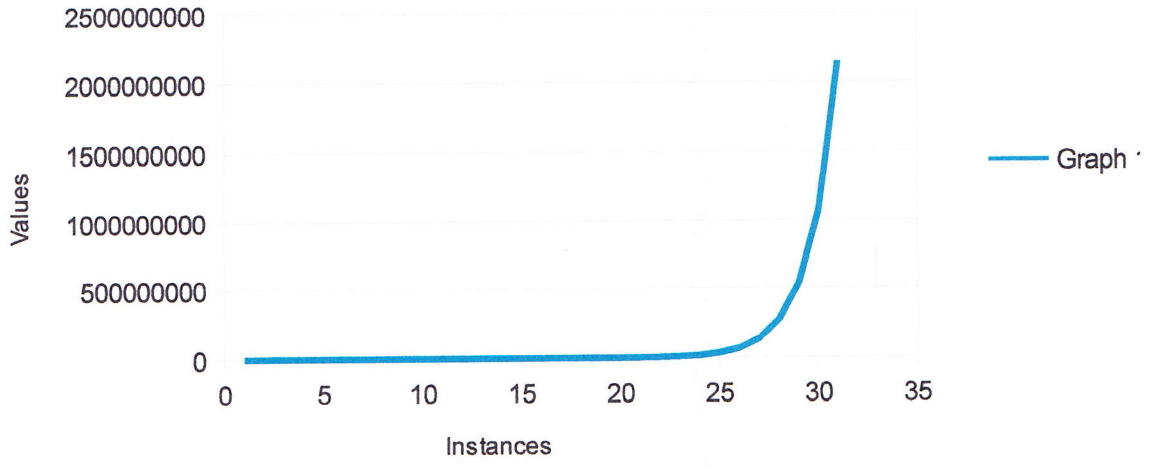
The Second Formula was a bit trickyer. It grows just like the first one, but at half the rate or less. It is of exponential order 2^n but it dose not grow as fast. I say its more of the order (2*2^(ln n/ln 2)) - 1.

The Third is hard to see, but its of the order of ln n / ln 2
(or log (base 2) n). Everytime n makes the whole number increase, it literaly doubles the count.

Confusing if you dont graph.

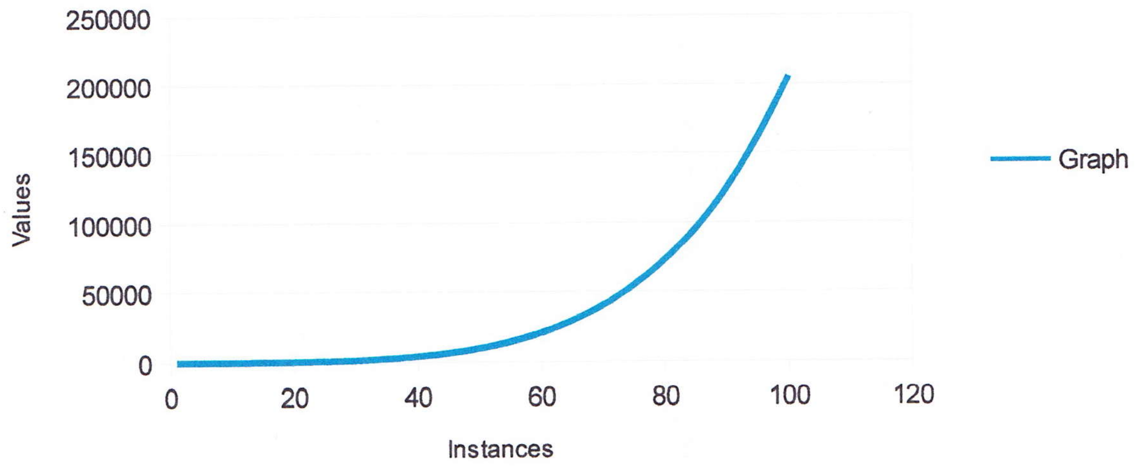
Function 1 - Big O

(2^n)-1



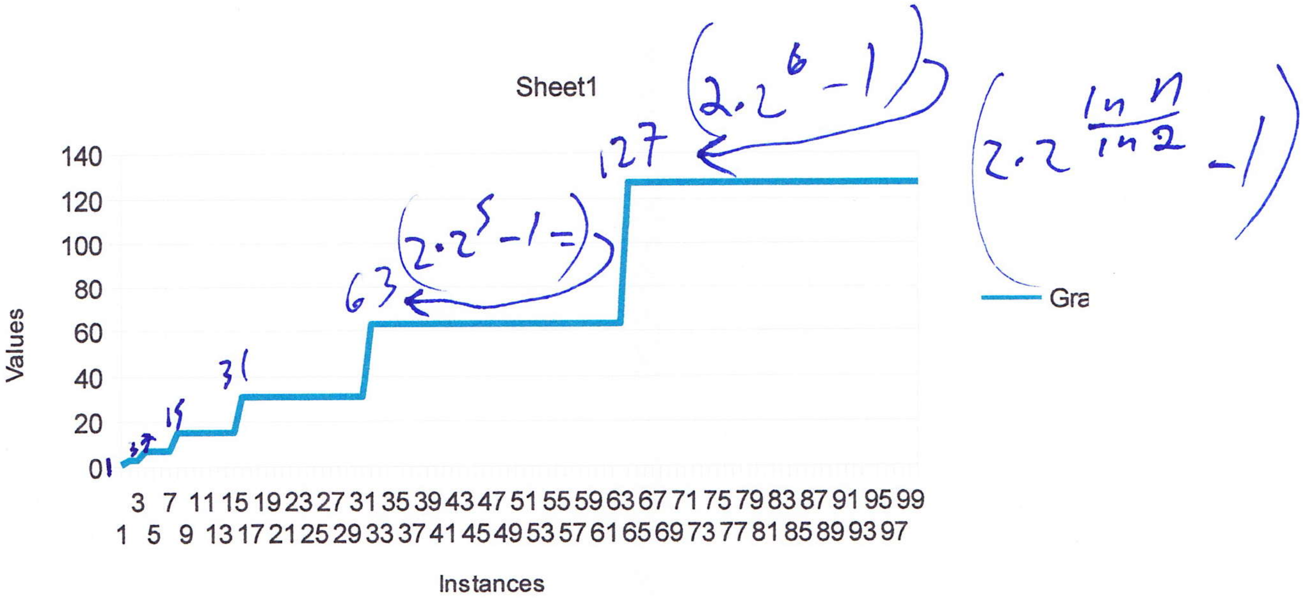
Finction 2 - Big O

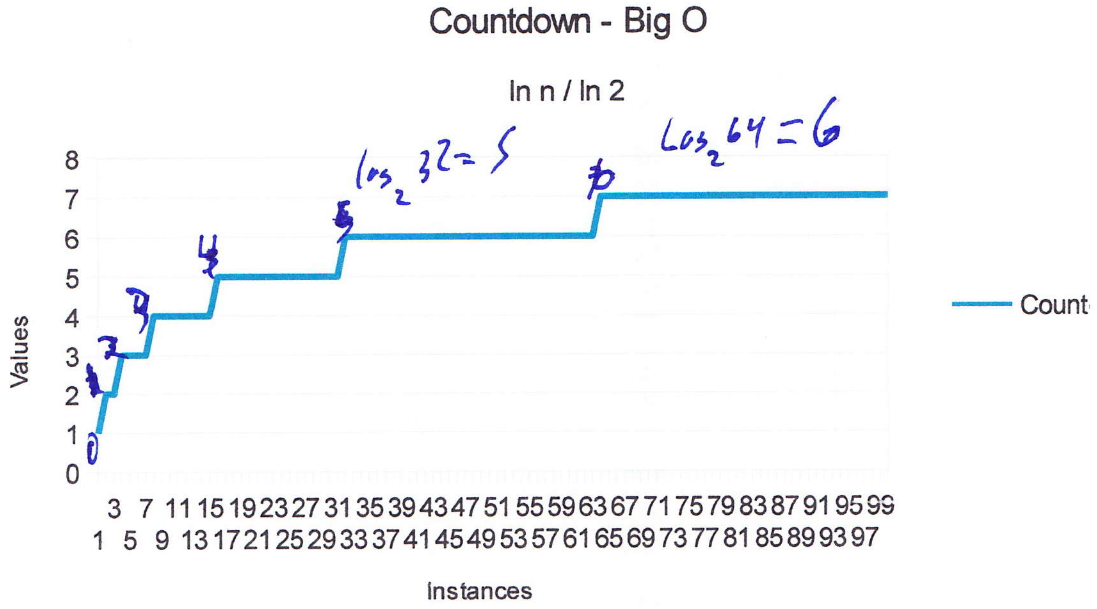
((2^n)-1)/2



Function 3 - Big O

ln n / ln 2


Sheet1
140
120
100
80
60
40
20
0
Values
3 7 11 15 19 23 27 31 35 39 43 47 51 55 59 63 67 71 75 79 83 87 91 95 99
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97
Instances
Gra
1
3
7
15
31
63
$(2\cdot 2^5-1=)$
127
$(2\cdot 2^6-1)$
$\left(2\cdot 2^{\frac{\ln n}{\ln 2}}-1\right)$


Countdown - Big O
ln n / ln 2
8
7
6
5
4
3
2
1
0
Values
3 7 11 15 19 23 27 31 35 39 43 47 51 55 59 63 67 71 75 79 83 87 91 95 99
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97
Instances
Count
0
1
2
3
4
5
6
$\log_2 32=5$
$\log_2 64=6$

```
 *
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package assignmentanalysis;

/**
 *
 * @author Paul
 */
public class AssignmentAnalysis {

    static int count;

    public static void function_one(int n)
    {
```

2^n-1

```
        count++;
        System.out.print(n+" ");
        if ( n > 1 )
        {
            function_one(n-1);
            function_one(n-1);
        }
    }//End Function 1

        public static void function_two(int n)
        {
            count++;
            System.out.print(n+" ");
            if ( n > 1 )
            {
                function_two(n-1);
                function_two(n/2);
            }
        }//end function 2

    public static int function_three(int n)
    {
        count++;
        System.out.print(n+" ");
        if ( n > 1 )
        {
            return function_three(n/2)+function_three(n/2);
        }
        return 1;
    }//end Function 3

                    public static void count_down(int n)
                    {
                        count++;
                        System.out.println(n);
                        if ( n > 1 )
                            count_down(n/2);
                    }//End Count Down

    public static void main(String[] args) {
        for(int i=1;i<10;i++)
        {
            count=0;
            function_one(i);
            System.out.printf("n=%d, count=%d\n", i,count);
        }

        for(int i=1;i<10;i++)
        {
```

n	Count				check
1	1	2	2^n-1	2^1-1	2-1=1
2	3	4		2^2-1	4-1=3
3	7	8		2^3-1	8-1=7
4	15	16		2^4-1	16-1=15
5	31	32		2^5-1	32-1=31
6	63			2^6-1	$2^6-1=63$
7	127	64		2^7-1	$2^7-1=128-1=127$
8	255	128		2^8-1	$2^8-1=256-1=255$
9	511	256		2^9-1	$2^9-1=512-1=511$

4
5

	n	Count	2^n-1
	1	1	
	2	3	
	3	5	
	4	9	
	5	13	
	6	19	
7·5	7	25	
8·5+4	8	35	
9·5	9	45	