For each of the following items, write a method having that running time:

1. $O(1)$
2. $O(\log n)$
3. $O(n)$
4. $O(n \log n)$
5. $O(n^2)$
6. $O(n^2 \log n)$
7. $O(n^3)$
8. $O(n^3 \log n)$
9. $O(2^n)$

For each method, write the recurrence that describes the running time and plot the running times for various values of $n$. See the example that illustrates the behavior of a $O(n^2)$, with running time recurrence $T(n) = T(n-1) + n$.

Since running times vary depending on circumstances that are independent of the method being evaluated (other applications run, garbage collection, O.S. operations, cache misses, etc.), you can obtain more reliable estimates of running times by repeating the experiment a few times and computing either the median or the minimum time for a particular value of $n$. In the code provided, I use the minimum as the estimate of the running time.

To generate the plot, I simply ran the attached program and cut and pasted the results from the screen into my favorite plotting program (I used Matlab, you may want to try that or Excel). Here are the results:

As usual, write a report describing your work.
/** Program to illustrate the behavior of a method with quadratic running time ***/
****** Programmed by Olac Fuentes ******
****** Last modified September 13, 2012 ******

import java.util.*;
import java.lang.*;

public class lab2{

    public static long min(long [] t){
        long tempMin = t[0];
        for(int i=1;i<t.length;i++)
            if (t[i]<tempMin)
                tempMin = t[i];
        return tempMin;
    }

    public static void p1(int n){
        // Method with O(n^2) running time
        // T(n) = T(n-1) + n
        if(n>1){
            for(int i=0;i<n;i++)
                {} // Do nothing
            p1(n-1);
        }
    }

    public static void main(String[] args) {
        int trials = 51;
        long [] time = new long [trials];
        for (int i=1;i<=10000;i+=100){
            int sum =0;
            for (int t =0;t<trials;t++){
                long start = System.nanoTime();
                p1(i);
                long end = System.nanoTime();
                time[t] = end-start;
            }
            //System.out.println("Executing p1("+i+"). Min running time: "+median(time)+" nanoseconds");
            System.out.println(i+" "+min(time));
        }
    }
}