Solve the following recurrence equations. For each recurrence, write a method that receives $n$ as parameter and has the running time described by that recurrence.

1. $T(n) = T(n - 1) + n$
2. $T(n) = T(n - 1) + 1$
3. $T(n) = 2T(n - 1) + 1$
4. $T(n) = T(n/2) + 1$
5. $T(n) = T(n/2) + n$
6. $T(n) = 2T(n/2) + 1$
7. $T(n) = 2T(n/2) + n$
8. $T(n) = 2T(n/2) + n^2$
9. $T(n) = 4T(n/2) + n$
10. $T(n) = 8T(n/2) + n$
11. $T(n) = 8T(n/2) + n^3$

For each method, plot the running times for various values of $n$. 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.), we will use a global variable to keep track of the number of times the most frequent operation is executed and use that operation as an estimate of the running time. See the example that illustrates the behavior of a $O(n^2)$ method, with running time recurrence $T(n) = T(n - 1) + n$. 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). Notice that the ranges of values of $n$ that my be used for various methods are different. Here are the results:
As usual, write a report describing your work.
public class lab2{
    public static long op_count;
    
    // Quadratic running time method described by recurrence
    // T(n) = T(n-1) + n
    public static void p1(int n){
        if(n>0){
            for(int i=0;i<n;i++){
                op_count++;
            }
            p1(n-1);
        }
    }
    
    public static void main(String[] args) {
        System.out.println("n Operations performed: ");
        for (int n= 0;n<=1000;n+=10){
            op_count =0;
            p1(n);
            System.out.println(n" +op_count);
        }
    }
}