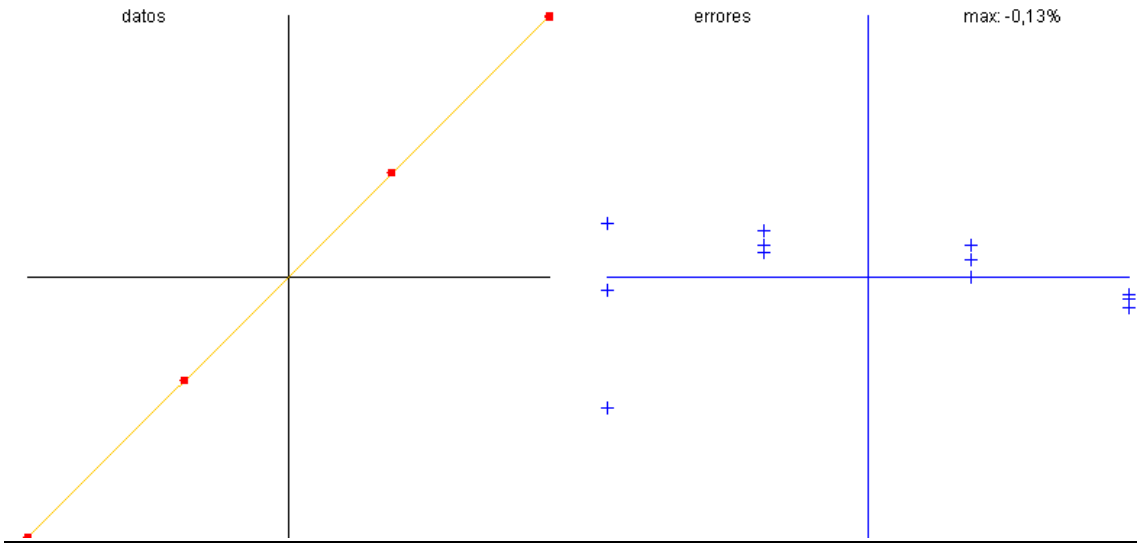
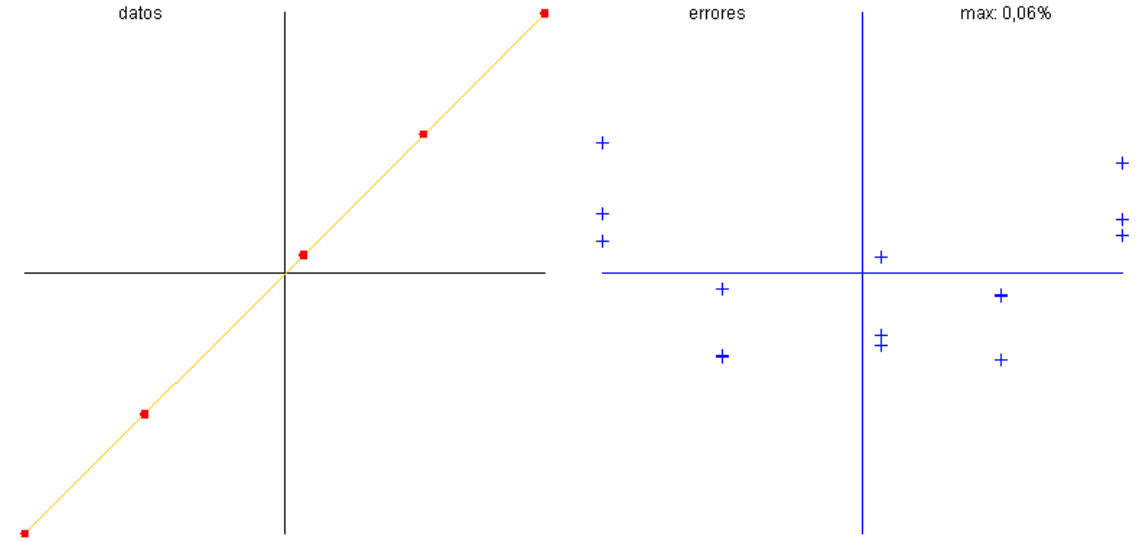
BubbleSort: En el peor de lo casos, es decir que el array de n elementos esté ordenado a la inversa, el primer bucle será recorrido n veces, mientras que el segundo bucle, encargado de hacer el “swap”, será recorrido n-1 veces, luego la complejidad total será de O(n^2). En el mejor de los casos, un array ya ordenado, el array será recorrido n veces, luego la complejidad del algoritmo es de O(n)



InsertionSort: Analogo a la explicación anterior, en el peor de los casos, el array será recorrido n veces por el bucle exterior, mientras que el bucle interior, deberá ser recorrido n-1 veces, siendo la complejidad total O(n^2), en el mejor de los casos, un array ya ordenado, solo será necesario una pasada, luego la complejidad total será O(n)



Merge Sort : Este algoritmo lo que hace es dividir cada array de n elementos a ordenar en dos sub arrays de n/2 elementos cada uno, este proceso se repite hasta que la longitud de los arrays es 1. El hecho de dividir el array a la mitad toma un tiempo constante independientemente del numero de elementos, O(1). Este proceso debe hacerse n veces luego se obtiene una complejidad de O(n). Cada vez que se separa un array en dos, generas 2 subproblemas con los que el algoritmo debe lidiar. Se observa que independientemente del

numero de datos que se meta, el numero de subproblemas va a ser menor que n, luego la complejidad de esta parte del algoritmo es O(logn). Siendo la complejidad total O(n*logn)