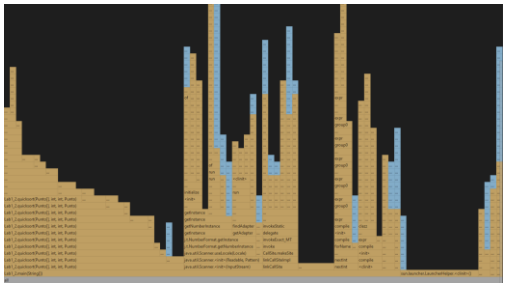
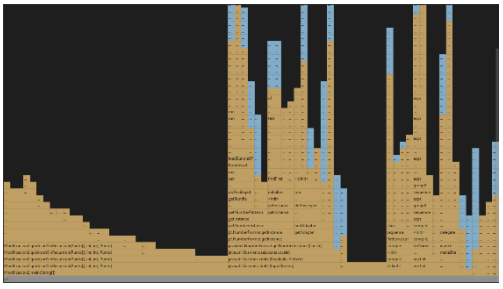
1. ¿Qué problemas introduce en una ordenación comparar distancias a un punto determinado en vez de directamente valores enteros? ¿Qué ventajas e inconvenientes tienen las alternativas que usan menos o más memoria y por qué? ¿Qué coste/complejidad tiene cada alternativa en tiempo y memoria?

El principal problema que nos encontramos a la hora de la ordenación con las distancias son dos:

- El algoritmo Quicksort original esta creado para coger un elemento del vector y compararle con el siguiente. En nuestro caso, no se puede como es lógico ya que lo que tenemos que comparar las distancias entre los vectores primero.
- Para poder resolver este problema tenemos dos posibles soluciones que son: calcular durante la ordenación o calcular antes de la ordenación.

En mi caso he aplicado la primera opción, ya que tenía un objeto Punto que tenía un método de cálculo de distancias.
Por lo que, a la hora de aplicar los algoritmos, solo tenía que aplicar a los vectores que quería comparar el cálculo de distancia, por lo que era la manera más eficiente y el coste temporal sería menor que habiéndolo calculado antes de implementar el InsertionSort o el QuickSort. Pese a ello el gasto de memoria es similar ya que realmente es como si se almacenase del estilo a un atributo de Punto.
Podemos ver el gasto de memoria con JProfiler, mas o menos es parecido a niveles generales.





2. ¿Cómo has eliminado la recursión final en el algoritmo de QuickSort? ¿Cómo has introducido la inserción directa en QuickSort?
Primero vamos a ver como es el algoritmo QuickSort original:

```java
public static Punto[] quicksort(Punto[] arr, int lower, int upper, Punto x) {
    if (!(lower >= upper || lower < 0)) {

        int t = partition(arr, lower, upper, x);

        quicksort(arr, lower,  upper: t - 1, x);
        quicksort(arr,  lower: t + 1, upper, x);
    }
    return arr;
}
```

He aplicado al algoritmo del segundo laboratorio (QuickSort original) el llamado tail call, esto hace que tras la primera recursividad, el pivote lower, crezca una posición, hasta el pivote +1, es decir, siendo = que upper+1.

```java
public static Punto[] quicksortSinRecursion(Punto[] arr, int lower, int upper, Punto x) {
    while (!(lower >= upper || lower < 0)) {

        int t = partition(arr, lower, upper, x);

        quicksortSinRecursion(arr, lower, upper: t - 1, x);
        //quicksort(arr, t + 1, upper, x);
        lower=t+1;
    }
    return arr;
}
```

En el segundo método, he introducido InsertionSort en el Quicksort con un condicional en el que comprobando que el pivote upper – lower +1 sea mayor que c. De ser así se usara el Quicksort común, mientras que si no se utiliza el InsertionSort.

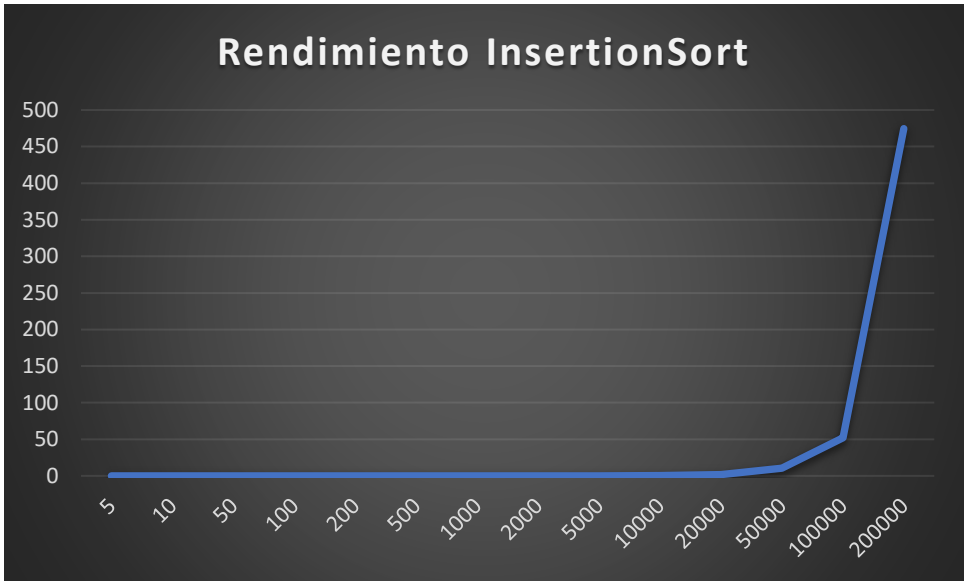
```java
public static Punto[] quicksortC(Punto[] arr, int lower, int upper, Punto x, int c){
    if (upper-lower+1>c){
            if (!(lower >= upper || lower <0)){
                int t = partition(arr, lower, upper, x);
                quicksortC(arr, lower, upper: t - 1, x, c);
                quicksortC(arr, lower: t + 1, upper, x, c);
            }
            else{
                insercionDirectaArray(arr, x, lower, upper);
            }
        }
        return arr;
    }
```

3. ¿Cómo has diseñado los experimentos para comprobar la corrección de los programas y para medir qué versión es mejor o peor? ¿Qué tamaños de problema has usado y que tiempos aproximados has obtenido?

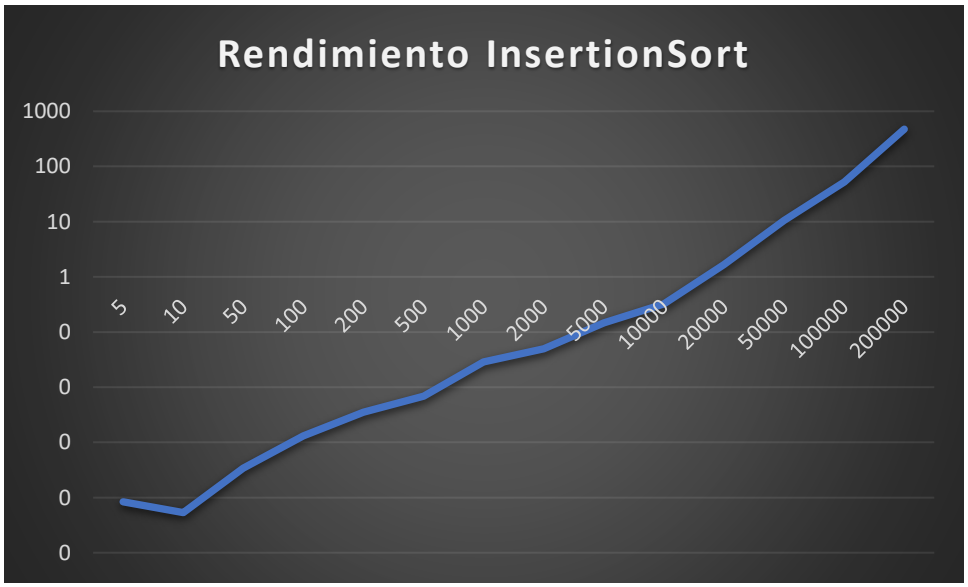
Para probar el correcto funcionamiento de los algoritmos, he creado una clase en java a la que le pasas un vector y te iba calculando las distancias entre los elementos correspondientes, de esta forma si te imprimía las distancias de una forma ordenada quería decir que el vector de Puntos estaba bien ordenado, y si no quería decir que no. Tras ello, cogía las salidas que imprimía cada uno de los algoritmos de ordenación, la copiaba en la otra clase manualmente, en lugar de creando el vector de manera aleatoria y si de esta manera imprimía las distancias de la misma manera que en la clase de prueba, es que funcionaba de manera correcta.
Para ver el tiempo que tardaba cada uno de los algoritmos de ordenación he iniciado en cada uno de ellos una variable llamada inicio y otra fin. La primera la he colocado justo antes de realizar la llamada al algoritmo, y la otra tras finalizar el algoritmo. Tras ello, hacia la resta a fin de inicio y con ello ya teníamos el tiempo correspondiente. Para ello, he utilizado un System.nanotype().

Por último, he ido probando con distintos tamaños de variables en los algoritmos de ordenación. He tomado unas 10 veces los resultados por cada valor y algoritmo y he plasmado el valor medio de ellos en una hoja de Excel cuyos resultados son los siguientes.

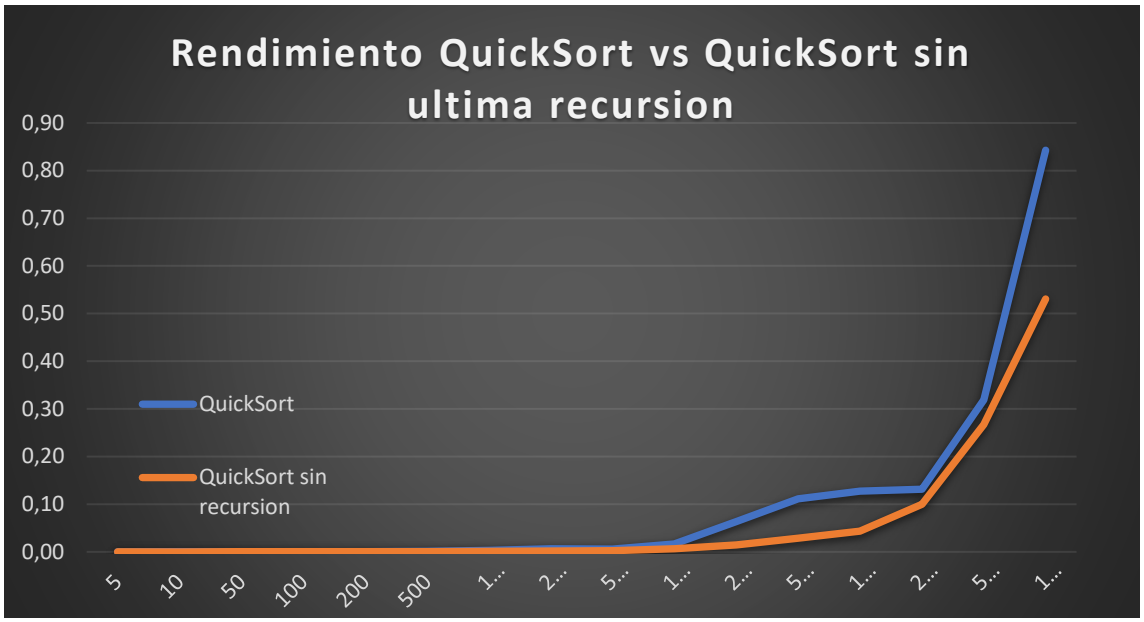
Como vemos en la primera gráfica el gasto temporal aumenta considerablemente al aumentar el número de Puntos.



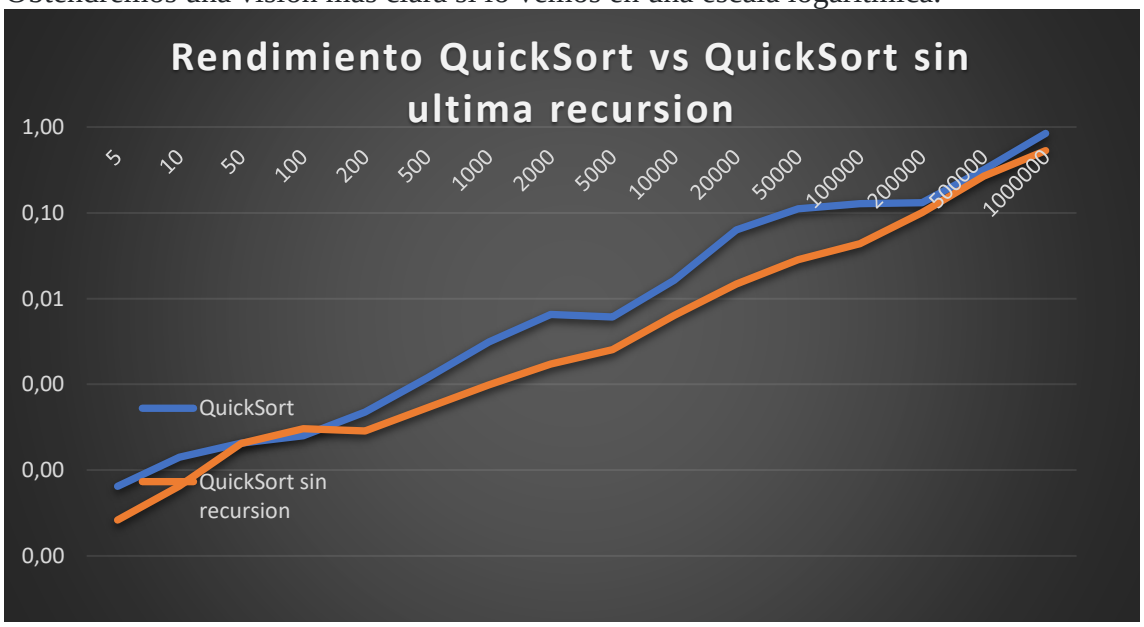
Con una escala logarítmica lo veremos mucho mejor.



En este grafico que hay a continuación, vemos el rendimiento del algoritmo QuickSort original representado por la línea azul. Vemos que pese a aumentar al final, no tiene ni punto de comparación con la del algoritmo InsertionSort. Por otro lado, tenemos en la línea naranja la segunda modificación pedida. Vemos que mejora sobre todo a mayor tamaño de vector, el algoritmo QuickSort original quitando la última recursión.



Obtendremos una visión mas clara si lo vemos en una escala logarítmica.



Por último, en la siguiente gráfica podemos ver los distintos rendimientos de tiempo para un tamaño de vector de 50000 con distintos valores de C.

4. ¿Qué conclusiones has obtenido sobre los cambios en el coste al eliminar la recursión final o introducir el algoritmo de inserción directa en el QuickSort? ¿Qué versiones son mejores o peores y por qué? ¿Qué lenguaje has usado, qué problemas has tenido?

Como conclusión podemos obtener que el método 2 mejora el algoritmo de QuickSort original a los valores mas altos del vector de puntos.
Por otro lado, vemos que la modificación 1 muestra mejoras para distintos valores de C, que están comprendidos en si casi todos entre 20 y 30.

Respecto al lenguaje utilizado, ha sido java y lo único que me ha aportado es facilidad, ya que es con el que mejor manejo tengo.