

**UNIVERSIDAD NACIONAL DE LA MATANZA**

Departamento de Ingeniería e Investigaciones Tecnológicas

***PROGRAMACION AVANZADA***

**Jefe de Cátedra: Verónica Aubin**

**Jefe de TP: Leonardo Blautzik**

**Docentes: Lucas Videla, Lucas Ponce de León**

**TP 2B – “ORDENADORA”**

**ENTREGA 25/09/2015**

**GRUPO Nº 8**

**INTEGRANTES**

* BLANCO, JUAN IGNACIO DNI. 35272529
* DIGIACOMO, GASTON DNI. 36075262
* MIRANDA, CRISTIAN NAHUEL DNI. 35349257
* MARTIN, GONZALO JAVIER DNI. 36170285

# OBJETIVO

Desarrollar la class Ordenadora que contenga todos los algoritmos de ordenamiento vistos en clase y permita ordenar cualquier vector de objetos comparables.

Graficar el tiempo de respuesta en función del tamaño de la entrada para los algoritmos elementales y para los tres casos posibles: entrada ordenada, aleatoria o en orden inverso. Extraer conclusiones. Separar en el gráfico la implementación con y sin centinela del método de inserción.

Ídem para algoritmos de ordenamiento no elementales: Shell, QuickSort y Fusion.

# CONCLUSIONES GENERALES

## ELEMENTALES

### BURBUJEO

Es el algoritmo de ordenamiento más sencillo de implementar, pero como vimos en el análisis de rendimiento si tratamos de ordenar un conjunto de elementos de tamaño considerablemente grande, no es para nada recomendable, ni siquiera si están ordenados los elementos (en esto caso nos ahorraríamos el intercambio). Entonces sería un algoritmo aplicable sobre dimensiones reducidas.

### SELECCION

En este caso el algoritmo es una ligera mejora del algoritmo de burbujeo, dado que realiza menos intercambios, de hecho en el mejor de los casos los intercambios son nulos. Podemos verlo reflejado esto en los análisis de rendimiento, aunque sigue siendo un algoritmo poco eficiente para dimensiones considerablemente grandes, es más eficiente que el algoritmo de burbujeo. Y al igual que el de burbujeo no tiene notables mejoras si el algoritmo esta ordenado o no.

### INSERCION

Dentro de los algoritmos elementales sin dudas, basándonos en la teoría y en nuestro análisis de rendimiento, determinamos que es sustancialmente más performante que los anteriores, ya sea implementado con “centinela” (ahorra una comparación buscando el valor más chico y llevándolo al principio inicialmente) o sin él. En este caso si podemos utilizarlo para manejar dimensiones considerablemente grandes, a diferencia del burbujeo y selección. Lo que lo transforma en el algoritmo de ordenamiento elemental más óptimo.

## NO ELEMENTALES

### SHELL

Dentro de los algoritmos no elementales, basándonos en nuestro análisis de rendimiento y en base a la codificación de los mismos, se aprecia una clara superioridad de performance con respecto a las otros dos, quicksort y fusión (esto puede deberse a que estos dos últimos utilizan recursividad a diferencia de Shell). Soporta sin problemas conjunto de elementos de dimensiones considerablemente grandes.

### QUICK SORT

Dentro de los algoritmos no elementales, basándonos en nuestro análisis de rendimiento y en base a la codificación de los mismos, al haber implementado la solución recursiva esto afecta considerablemente su performance, siendo el menos eficiente de los tres. De hecho para un conjunto de elementos de dimensiones considerablemente grandes es muchísima la diferencia con los otros dos.

### FUSION

Dentro de los algoritmos no elementales, basándonos en nuestro análisis de rendimiento y en base a la codificación de los mismos, su rendimiento es bastante aceptable, pese a ser recursivo. Para un conjunto de elementos de dimensiones considerablemente grandes no fue tanto más lento que el Shell (el más performante de los tres).