

### Práctica 2

Límite para la entrega: 21 de octubre, a las 23:59

## Ordenación por inserción y ordenación rápida

### Ordenación por inserción

Algoritmo de ordenación por inserción:

```
procedimiento ordenacionPorInsercion (var v[1..n])  
  para i := 2 hasta n hacer  
    x := v[i] ;  
    j := i-1 ;  
    mientras j > 0 y v[j] > x hacer  
      v[j+1] := v[j] ;  
      j := j-1  
    fin mientras ;  
    v[j+1] := x  
  fin para  
fin procedimiento
```

Se pide:

1. Implemente el algoritmo de ordenación por inserción.

```
void ord_ins (int v [], int n);
```

2. Valide el correcto funcionamiento de la implementación (figura 1).
3. Determine los tiempos de ejecución para distintos tamaños del vector y para tres diferentes situaciones iniciales: (a) el vector ya está ordenado en orden ascendente, (b) el vector ya está ordenado en orden descendente, y (c) el vector está inicialmente desordenado. (véase la figura 2).
4. Calcule empíricamente la complejidad del algoritmo para cada una de las diferentes situaciones iniciales del vector (figura 3).

---

```
> ./test
Ordenacion por insercion con inicializacion aleatoria
3, -3, 0, 17, -5, 2, 11, 13, 6, 1, 7, 14, 1, -2, 5, -14, -2
ordenado? 0
ordenando...
-14, -5, -3, -2, -2, 0, 1, 1, 2, 3, 5, 6, 7, 11, 13, 14, 17
esta ordenado

Ordenacion por insercion con inicializacion descendente
10, 9, 8, 7, 6, 5, 4, 3, 2, 1
ordenado? 0
ordenando...
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
esta ordenado
```

---

Figura 1: Test

---

```
void inicializar_semilla() {
    srand(time(NULL));
}
void aleatorio(int v [], int n) { /* se generan números pseudoaleatorio entre -n y +n */
    int i, m=2*n+1;
    for (i=0; i < n; i++)
        v[i] = (rand() % m) - n;
}
void ascendente(int v [], int n) {
    int i;
    for (i=0; i < n; i++)
        v[i] = i;
}
```

---

Figura 2: Inicialización aleatoria y ascendente

---

Ordenación por inserción con inicialización descendente					
	n	t(n)	$t(n)/n^{1.8}$	$t(n)/n^2$	$t(n)/n^{2.2}$
(*)	500	357.324	0.004954	0.001429	0.000412
	1000	1577.000	0.006278	0.001577	0.000396
	2000	6103.000	0.006977	0.001526	0.000334
	4000	23603.000	0.007749	0.001475	0.000281
	8000	92347.000	0.008707	0.001443	0.000239
	16000	361434.000	0.009786	0.001412	0.000204
	32000	1446534.000	0.011248	0.001413	0.000177

---

Figura 3: Parte de la posible salida por pantalla de la ejecución del programa principal

## Ordenación Rápida

Algoritmo de ordenación rápida (*quicksort*) con selección aleatoria del pivote y un umbral para detectar vectores pequeños:

```

procedimiento ordenarAux (V[izq..der])
  si izq+UMBRAL <= der entonces
    x := {nº aleatorio en el rango [izq..der]};
    pivote := V[x];
    intercambiar (V[izq], V[x]);
    i := izq + 1;
    j := der;
    mientras i <= j hacer
      mientras i <= der y V[i] < pivote hacer
        i := i + 1
      fin mientras;
      mientras V[j] > pivote hacer
        j := j - 1
      fin mientras;
      si i <= j entonces
        intercambiar (V[i], V[j]);
        i := i + 1;
        j := j - 1
      fin si
    fin mientras;
    intercambiar (V[izq], V[j]);
    ordenarAux (V[izq..j-1]);
    ordenarAux (V[j+1..der])
  fin si
fin procedimiento

```

```

procedimiento ordenacionRapida (V[1..n])
    ordenarAux(V[1..n]);
    si (UMBRAL > 1) entonces
        ordenaciónPorInsercion (V[1..n])
    fin si
fin procedimiento

```

Se pide:

1. Implemente el algoritmo de ordenación rápida.

```

void ord_rapida(int v [], int n) {
    rapida_aux(v, 0, n-1);
    if (UMBRAL > 1)
        ord_ins(v, n);
}

```

2. Valide el correcto funcionamiento de la implementación (**con umbral = 1**).
3. Ejecute el algoritmo con vectores de distinto tamaño y en distintas situaciones iniciales (vector ordenado ascendente o descendientemente, o desordenado), y con distintos valores de umbral: 1 (no se realiza la llamada al método de ordenación por inserción), 10 y 100.
4. Compare entre si los tiempos obtenidos para cada umbral usado. En función de la situación inicial del vector, ¿con qué umbral se obtienen los mejores tiempos? ¿por qué?
5. Calcule empíricamente la complejidad del algoritmo para cada una de las diferentes situaciones iniciales del vector y cada uno de los umbrales (i.e., 9 tablas).

Entregue los ficheros con el código C y el fichero .txt con el informe por medio de la tarea *Entrega Práctica 2* en la página de Algoritmos en <https://campusvirtual.udc.gal>. Se recuerda que el límite para completar la tarea es el viernes 21 de octubre a las 23:59, y una vez subidos los archivos no se podrán cambiar. **Todos los compañeros que forman un equipo tienen que entregar el trabajo.**