

# Actividades adicionales

## Algoritmos y Estructuras de Datos

### *Tema 2: algoritmos no recursivos*

En cada apartado evaluable, se valorará también la “buena programación” (comentarios, nomenclatura, precondiciones, máxima cohesión, mínimo acoplamiento, etc.), tal como la hemos explicado en clase. Por cada apartado, el 50% de la nota será la funcionalidad, y el 50% restante será la buena programación. Como en las actividades realizadas en clase, sólo se valorará la buena programación si el apartado cumple completamente su funcionalidad.

#### **Actividad adicional 2.1**

Realiza en C++ un programa que permita ordenar un array de números racionales:

1. Programa la clase Racional, que tendrá únicamente lo siguiente:
  - a. (2 mn) Atributos int privados: numerador y denominador (0,1 puntos)
  - b. (3 mn) Un constructor que reciba como parámetro el numerador y el denominador del Racional que se quiere construir. El denominador debe ser distinto de 0 (0,2 puntos)
  - c. (3 mn) Un método público “getValor()” que devuelva la división real del numerador entre el denominador (0,2 puntos)
  - d. (2 mn) Un método público para devolver el numerador, y otro método público para devolver el denominador (0,1 puntos)
2. Realiza un main que permita (puedes añadir varias funciones más aquí):
  - a. (2 mn) Preguntar al usuario por el tamaño del array (0,1 puntos)
  - b. (5 mn) Crear un array de Racionales de ese tamaño. Cada racional tendrá un numerador y un denominador aleatorio. (0,4 puntos)
  - c. (2 mn) Se imprime el array por la pantalla. Por cada racional, se imprimirá su numerador, una raya (la de la tecla del 7), y su denominador. Los racionales se separan entre ellos por un espacio. (0,1 puntos)
  - d. (15 mn) Ordenar el array de racionales mediante la función qsort() de la biblioteca estándar de C. La ordenación se hará de mayor a menor valor real de cada racional (el valor real de un racional se obtiene llamando a la función “getValor”). (2,5 puntos)
  - e. (1 mn) Se imprime el array, ya ordenado, por pantalla (0,1 puntos)

#### **Actividad adicional 2.2**

(1 hora) Realiza un programa en C++ que calcule a posteriori el tiempo que se tarda en ordenar de menor a mayor (en el caso medio) con el algoritmo de burbuja inversa. Para ello, sigue los siguientes pasos:

1. Programa el algoritmo de ordenación de números enteros de menor a mayor por burbuja inversa. Es como el algoritmo típico de la burbuja, pero en este caso la “burbuja” se desplaza de derecha a izquierda en vez de izquierda a derecha. Es decir, los mínimos se van acumulando al principio del vector en vez de acumularse los máximos al final.
2. Programa un método que imprima por pantalla un array de enteros.
3. Realiza un main que haga lo siguiente:
  - a. Pide por teclado el tamaño de los arrays (mayor o igual que 1) y el número de arrays a considerar para calcular el caso medio (mayor o igual que 1). Los llamaremos “n” y “m” respectivamente.
  - b. Se crean dinámicamente (con malloc obligatoriamente) m arrays aleatorios, cada uno de tamaño n. Hay que utilizar obligatoriamente dobles punteros.
  - c. Se imprimen todos ellos por pantalla.
  - d. Se ordenan todos ellos de menor a mayor con el algoritmo de la burbuja inversa y se calcula el tiempo medio que se tarda en ordenar cada uno.
  - e. Se imprime el tiempo medio por pantalla.
  - f. Se imprimen todos los arrays (ya ordenados) por pantalla.

### **Actividad adicional 2.3**

Vamos a comparar “a posteriori” el tiempo que se tarda en ordenar de menor a mayor un vector de enteros en el caso mejor, medio y peor. Lo haremos con dos algoritmos: quicksort e inserción.

Para ello:

1. Programa una función (sin clase) que ordene de menor a mayor un array de enteros según el método de inserción ([https://es.wikipedia.org/wiki/Ordenamiento\\_por\\_inserci%C3%B3n](https://es.wikipedia.org/wiki/Ordenamiento_por_inserci%C3%B3n)). Recibirá por parámetro un puntero a la zona de memoria en donde comienza el vector que queremos ordenar, y el número de componentes de dicho vector.
2. Programa una función (sin clase) que imprima por pantalla un array de enteros en el caso de que su número de componentes sea menor o igual que 10. En caso contrario imprimirá “<demasiados componentes para mostrar>”. Se le pasa la dirección de comienzo del vector y su número de componentes.
3. Programar un main() que haga lo siguiente:
  - a. Se pide por teclado el número de componentes de los arrays que se ordenarán (al menos 1 componente). A este número lo llamamos n.
  - b. Para hacer la comparación de tiempos en el caso medio, lo que vamos a hacer es calcular cuánto se tarda en ordenar varios arrays aleatorios, y haremos la media de dichos tiempos. Se pide por teclado cuántos arrays aleatorios se

crearán para calcular el tiempo del caso medio (al menos 1). A este número lo llamamos “cantidadArraysParaMedia”.

- c. Se creará un array de “cantidadArraysParaMedia” punteros a enteros. Cada uno de estos punteros apuntará al comienzo de un array de n enteros. Todos estos arrays se utilizarán para medir lo que tarda el caso medio en el algoritmo quicksort. En cada componente de los arrays de enteros metemos un valor aleatorio.
- d. Se hará lo mismo que en el paso “c”, para medir lo que tarda el caso medio en el algoritmo de inserción. Los arrays creados en este paso deben de tener los mismos valores que en el paso “d”. Para ello, se pueden copiar los valores (ojo: no los punteros) de los arrays del paso “c” en los arrays que hemos creado en este paso, o bien podemos hacer los dos pasos (“c” y “d”) a la vez. Esto último es la solución más sencilla.
- e. Para medir el caso mejor, crearemos un array aleatorio de tamaño n. Lo ordenamos con qsort() para que quede ordenado de menor a mayor (el caso mejor). Creamos otro array que sea copia de éste: uno nos servirá para medir el caso mejor posteriormente en qsort, y otro para medir el caso mejor posteriormente en inserción.
- f. Para medir el caso peor, crearemos un array de tamaño n que sea copia (en valores, no en punteros) del obtenido en el apartado “e”. Lo ordenamos con qsort() para que quede ordenado de mayor a menor (el caso peor). Recuerda que para que qsort() ordene de mayor a menor sólo debes reprogramar la función que compara dos enteros para que devuelva justo lo opuesto a cuando la programamos para que qsort() ordene de menor a mayor. Creamos otro array que sea copia de éste: uno nos servirá para medir el caso peor posteriormente en qsort, y otro para medir el caso peor posteriormente en inserción.
- g. Imprimimos por pantalla los arrays creados. No hace falta imprimir todos, sino sólo la mitad de ellos (pues una mitad es copia de la otra mitad).
- h. Se ordena el caso mejor con los algoritmos de inserción y qsort y se muestra el tiempo empleado en cada caso, al estilo de la actividad 1.3. Se imprime el array ya ordenado.
- i. Lo mismo con el caso peor.
- j. Se ordenan todos los arrays aleatorios del caso medio con los dos algoritmos de ordenación, y se hace y se muestra la media de los tiempos para cada uno de los dos algoritmos. Se supone que todos los casos son equiprobables, así pues la media se calculará, por cada algoritmo, sumando todos los tiempos y dividiéndolo entre el número de casos (“cantidadArraysParaMedia”). Se imprimen los arrays ya ordenados, y esta vez para los dos algoritmos probados (para asegurarnos de que los dos han funcionado).

Algunas consideraciones:

- En este ejercicio no hay clases. Lo metemos todo en el archivo “main.cpp”. Sí que crearemos varias funciones sin clase, todas ellas dentro de “main.cpp”.

- Utiliza `malloc()` en todo momento. No puedes utilizar `new`.
- Una vez que tenemos un array ordenado de menor a mayor (o de mayor a menor), reordenarlo al revés se puede hacer muy fácilmente y de forma muy eficiente intercambiando sobre el mismo vector los elementos  $(0, n-1)$ ,  $(1, n-2)$ ,  $(2, n-3)$ , etc. No obstante, en vez de así, nosotros lo hacemos con `qsort()` para practicar.
- Estadísticamente la relevancia para calcular una media empieza a partir de los 100 casos. Pero por rapidez nosotros sólo utilizaremos 15 en las pruebas que se describen a continuación.

En la primera prueba decimos que los arrays tendrán 10 componentes, y decimos que habrá 15 arrays para calcular el caso medio:

```
C:\WINDOWS\system32\cmd.exe
Numero de componentes de los arrays (>= 1): 10
Cantidad de arrays aleatorios para el caso medio (>= 1): 15

Array para calcular el caso mejor:
 3184 8444 9508 12953 19724 24565 24817 27193 28203 32426
Array para calcular el caso peor:
 32426 28203 27193 24817 24565 19724 12953 9508 8444 3184
Arrays para calcular caso medio:
17506 15213 22232 7656 27633 2713 32008 16498 21572 13330
25225 20074 16936 10558 13239 270 3829 3819 12214 21054
26396 20882 31659 21214 22272 20189 13868 16508 6663 14438
24309 7730 24560 5058 26332 21710 31847 31678 31751 7729
13526 1768 26204 16373 8319 30546 2105 22867 20168 6034
22373 18499 10197 26893 25159 27108 15196 1910 13654 14879
29109 26001 13521 5767 21297 18237 16048 10372 22957 7345
10458 15025 103 8966 28568 32130 1552 28150 10314 32554
5410 16099 3161 23228 18035 19738 12396 25232 12811 25434
15327 18920 562 30352 13425 4563 15566 6629 12860 15698
14060 29690 4075 4282 26269 6514 27431 24665 1795 28850
2386 12892 17067 740 29816 14320 22557 24130 25730 25000
27826 28049 18605 4895 422 26859 1339 14814 16238 17346
14697 19328 2892 15394 8608 28436 6944 28470 27119 21954
31617 5902 1209 1714 15372 21919 22955 8745 13820 12556

Tiempos para el caso mejor:
Insercion: 0.000 segundos
QuickSort: 0.000 segundos
Tiempos para el caso peor:
Insercion: 0.000 segundos
QuickSort: 0.000 segundos
Tiempos para el caso medio:
Insercion: 0.000 segundos (de media)
QuickSort: 0.000 segundos (de media)

Arrays para calcular el caso mejor:
Insercion: 3184 8444 9508 12953 19724 24565 24817 27193 28203 32426
QuickSort: 3184 8444 9508 12953 19724 24565 24817 27193 28203 32426
Arrays para calcular el caso peor:
Insercion: 3184 8444 9508 12953 19724 24565 24817 27193 28203 32426
QuickSort: 3184 8444 9508 12953 19724 24565 24817 27193 28203 32426
Arrays para calcular caso medio:
Insercion:
2713 7656 13330 15213 16498 17506 21572 22232 27633 32008
270 3819 3829 10558 12214 13239 16936 20074 21054 25225
6663 13868 14438 16508 20189 20882 21214 22272 26396 31659
5058 7729 7730 21710 24309 24560 26332 31678 31751 31847
1768 2105 6034 8319 13526 16373 20168 22867 26204 30546
1910 10197 13654 14879 15196 18499 22373 25159 26893 27108
5767 7345 10372 13521 16048 18237 21297 22957 26001 29109
103 1552 8966 10314 10458 15025 28150 28568 32130 32554
3161 5410 12396 12811 16099 18035 19738 23228 25232 25434
562 4563 6629 12860 13425 15327 15566 15698 18920 30352
1795 4075 4282 6514 14060 24665 26269 27431 28850 29690
740 2386 12892 14320 17067 22557 24130 25000 25730 29816
422 1339 4895 14814 16238 17346 18605 26859 27826 28049
2892 6944 8608 14697 15394 19328 21954 27119 28436 28470
1209 1714 5902 8745 12556 13820 15372 21919 22955 31617
QuickSort:
2713 7656 13330 15213 16498 17506 21572 22232 27633 32008
270 3819 3829 10558 12214 13239 16936 20074 21054 25225
6663 13868 14438 16508 20189 20882 21214 22272 26396 31659
5058 7729 7730 21710 24309 24560 26332 31678 31751 31847
1768 2105 6034 8319 13526 16373 20168 22867 26204 30546
1910 10197 13654 14879 15196 18499 22373 25159 26893 27108
5767 7345 10372 13521 16048 18237 21297 22957 26001 29109
103 1552 8966 10314 10458 15025 28150 28568 32130 32554
3161 5410 12396 12811 16099 18035 19738 23228 25232 25434
562 4563 6629 12860 13425 15327 15566 15698 18920 30352
1795 4075 4282 6514 14060 24665 26269 27431 28850 29690
740 2386 12892 14320 17067 22557 24130 25000 25730 29816
422 1339 4895 14814 16238 17346 18605 26859 27826 28049
2892 6944 8608 14697 15394 19328 21954 27119 28436 28470
1209 1714 5902 8745 12556 13820 15372 21919 22955 31617

Presione una tecla para continuar . . . _
```

[illegible]

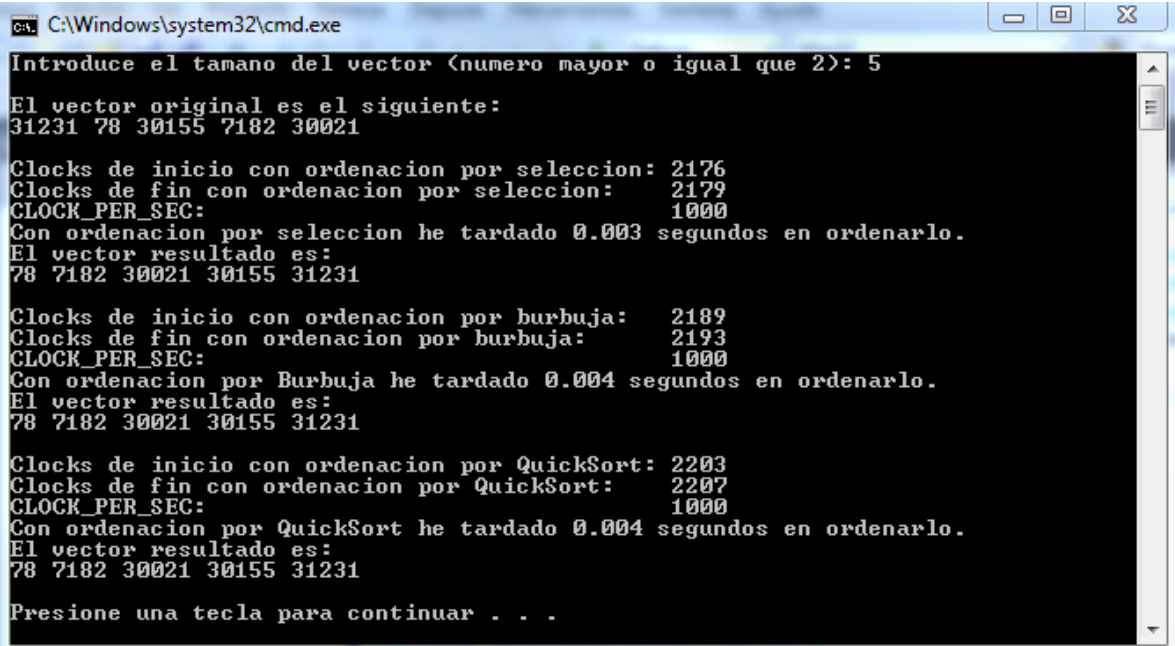
## Actividad adicional 2.4

Vamos a ampliar la actividad 2.2 añadiendo un método de ordenación más. Esta vez se trata del método de ordenación por burbuja (ver la animación del principio de [https://es.wikipedia.org/wiki/Ordenamiento\\_de\\_burbuja](https://es.wikipedia.org/wiki/Ordenamiento_de_burbuja)). El método tendrá la precondition de que el tamaño del vector sea al menos 2.

El main será sustancialmente el mismo que en la actividad 2.2, pero añadiendo también la ordenación por burbuja. Fíjate (en la prueba con un vector de 30.000 componentes) en que la burbuja tarda aún más que el método de ordenación por selección.

Para analizar por qué Burbuja tarda más que Selección, calcula el  $T(n)$  de la burbuja, para el caso mejor y peor. En base a ello, razona por qué tarda más uno que otro.

La prueba con 5 componentes daría el siguiente resultado:



```
C:\Windows\system32\cmd.exe
Introduce el tamaño del vector (numero mayor o igual que 2): 5
El vector original es el siguiente:
31231 78 30155 7182 30021

Clocks de inicio con ordenacion por seleccion: 2176
Clocks de fin con ordenacion por seleccion: 2179
CLOCK_PER_SEC: 1000
Con ordenacion por seleccion he tardado 0.003 segundos en ordenarlo.
El vector resultado es:
78 7182 30021 30155 31231

Clocks de inicio con ordenacion por burbuja: 2189
Clocks de fin con ordenacion por burbuja: 2193
CLOCK_PER_SEC: 1000
Con ordenacion por Burbuja he tardado 0.004 segundos en ordenarlo.
El vector resultado es:
78 7182 30021 30155 31231

Clocks de inicio con ordenacion por QuickSort: 2203
Clocks de fin con ordenacion por QuickSort: 2207
CLOCK_PER_SEC: 1000
Con ordenacion por QuickSort he tardado 0.004 segundos en ordenarlo.
El vector resultado es:
78 7182 30021 30155 31231

Presione una tecla para continuar . . .
```

Para la prueba con 30000 componentes, el resultado sería:

```
C:\Windows\system32\cmd.exe
Introduce el tamaño del vector (numero mayor o igual que 2): 30000
El vector original es el siguiente:

Clocks de inicio con ordenacion por seleccion: 2977
Clocks de fin con ordenacion por seleccion: 5293
CLOCK_PER_SEC: 1000
Con ordenacion por seleccion he tardado 2.316 segundos en ordenarlo.
El vector resultado es:

Clocks de inicio con ordenacion por burbuja: 5300
Clocks de fin con ordenacion por burbuja: 12043
CLOCK_PER_SEC: 1000
Con ordenacion por Burbuja he tardado 6.743 segundos en ordenarlo.
El vector resultado es:

Clocks de inicio con ordenacion por QuickSort: 12051
Clocks de fin con ordenacion por QuickSort: 12079
CLOCK_PER_SEC: 1000
Con ordenacion por QuickSort he tardado 0.028 segundos en ordenarlo.
El vector resultado es:

Presione una tecla para continuar . . .
```

### **Actividad adicional 2.5**

¿Cuál es el  $T(n)$  de la función "hacerTarea"?

```
//  $T(a,b) = a^2 - b + 1$ 
int comprobar(int a, int b) {...}

// Función hacerTarea
void hacerTarea(int n) {
    int pos=0;
    while (pos<n-1) {
        pos = pos+1;
        r = comprobar(n,pos+1);
        r--;
    }
}
```



### **Actividad adicional 2.6**

¿Cuál es el  $T(n)$  de la función "hacerTarea"?

```
// T(a,b) =  $a^2 - b + 1$ 
int comprobar(int a, int b) {...}

// Función hacerTarea
void hacerTarea(int n) {
    int r;
    int i=10;
    while (i>=0 && comprobar(n,0)==1) {
        int pos=0;
        // T(n) de funcionCompleja es  $n^3 + 1,5n^2 + 2,5n + 1$ 
        pos=funcionCompleja(n)-1;
        i--;
    }
}
```

### **Actividad adicional 2.7**

¿Cuál es el  $T(n)$  de la función "hacerTarea"?

```
// T(a,b) = a^2-b+1
int comprobar(int a, int b) {...}

// Función hacerTarea
void hacerTarea(int n) {
    int r;
    int i=n-1;
    while (i>=0 && comprobar(n,1)==1) {
        int pos=0;
        // T(n) de funcionCompleja es 2n^3+5n^2+2n+1
        pos=n*funcionCompleja(n)-1;
        i--;
    }
}
```