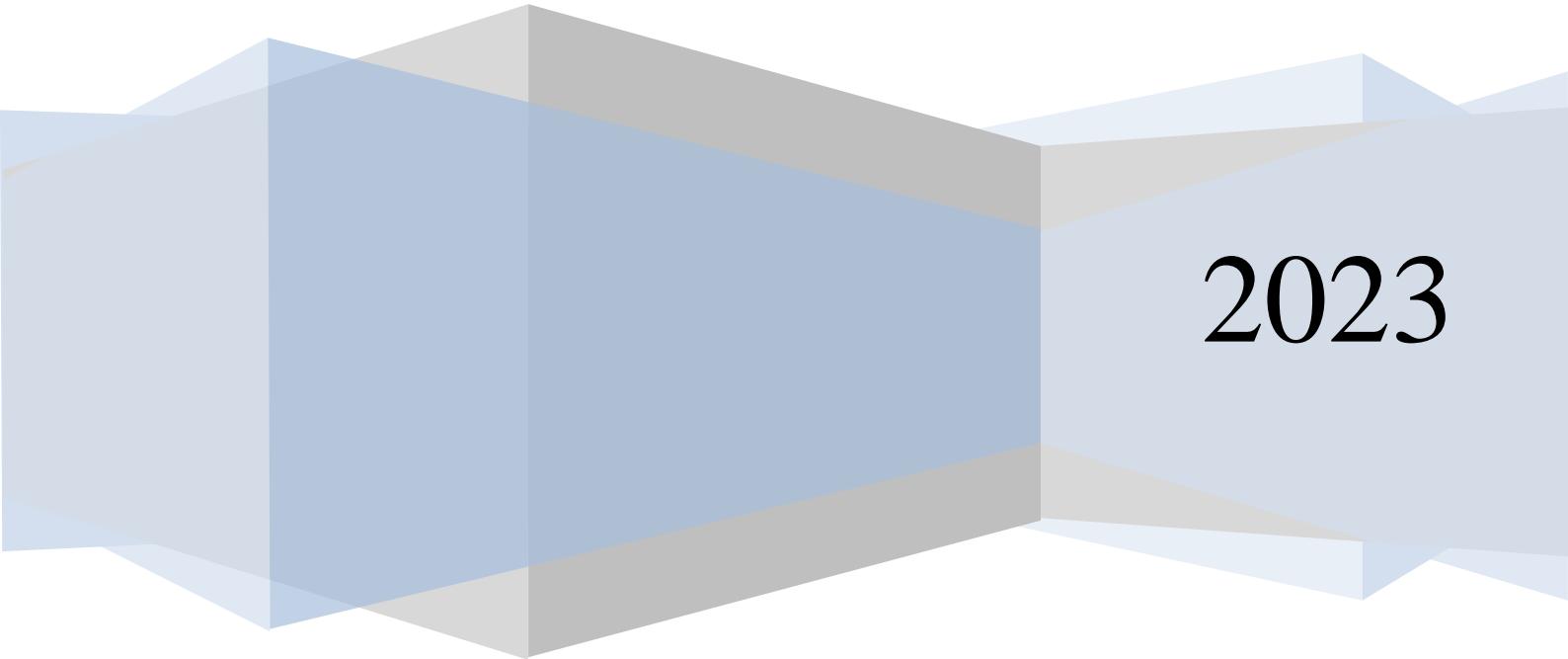


Algoritmia

Practica 3:

Algoritmos de Clasificación

David Piñuel Bosque



2023

Índice

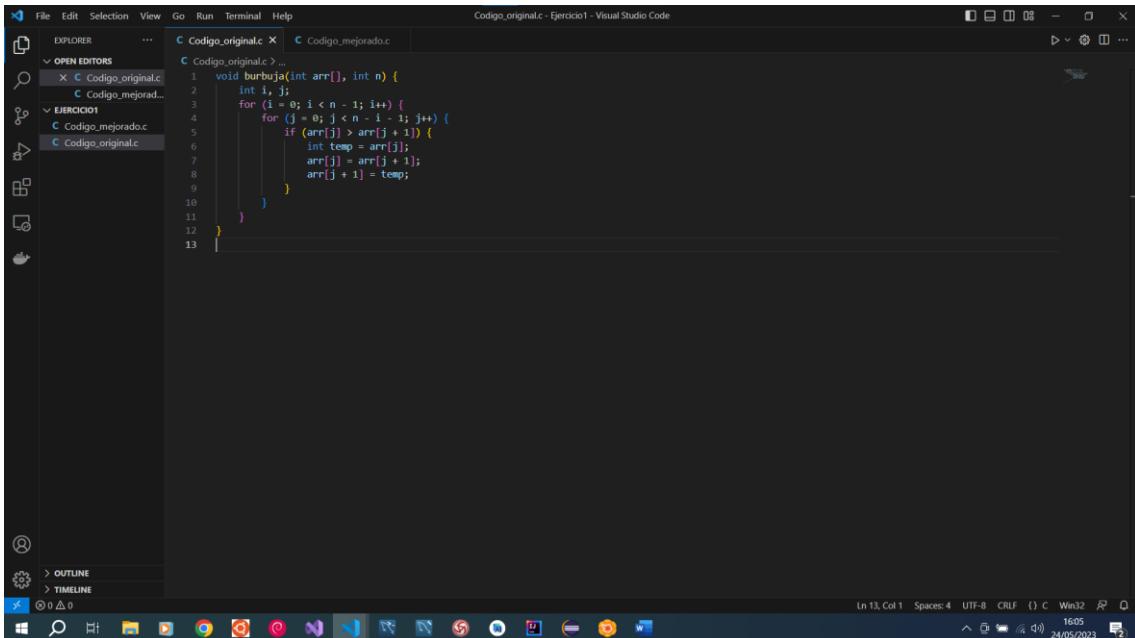
1. Ejercicio 1.....	3
2. Ejercicio 2.....	4
3. Ejercicio 3.....	6
4. Ejercicio 4.....	7
5. Ejercicio 5.....	9
6. Ejercicio Opcional.	26
7. Aclaraciones y comentarios.	27

Índice Ilustración

Ilustración 1: Código Fuente Original Método Burbuja.....	4
Ilustración 2: Código Fuente Mejorada Método Burbuja.....	4
Ilustración 3: Código Fuente Original Método Selección Directa.....	5
Ilustración 4: Código Fuente Mejorado Método Selección Directa.....	6
Ilustración 5: Código Fuente Método Shell.....	7
Ilustración 6: Código Fuente Método Pivote Medio	8
Ilustración 7: Código Fuente Método Pivote Aleatorio	8
Ilustración 8: Código Fuente Método Pivote Mediana Parte 1.....	9
Ilustración 9: Código Fuente Método Pivote Mediana Parte 2.....	9
Ilustración 10: Código Fuente Ejercicio 5 Parte 1	10
Ilustración 11: Código Fuente Ejercicio 5 Parte 2	11
Ilustración 12: Código Fuente Ejercicio 5 Parte 3	11
Ilustración 13: Código Fuente Ejercicio 5 Parte 4	12
Ilustración 14: Código Fuente Ejercicio 5 Parte 5	12
Ilustración 15: Código Fuente Ejercicio 5 Parte 6	13
Ilustración 16: Código Fuente Ejercicio 5 Parte 7	13
Ilustración 17: Código Fuente Ejercicio 5 Parte 8	14
Ilustración 18: Código Fuente Ejercicio 5 Parte 9	14
Ilustración 19: Código Fuente Ejercicio 5 Parte 10	15
Ilustración 20: Código Fuente Ejercicio 5 Parte 11	15
Ilustración 21: Código Fuente Ejercicio 5 Parte 12	16
Ilustración 22: Código Fuente Ejercicio 5 Parte 13	16
Ilustración 23: Compilación y Ejecución Ejercicio 5 Parte 1	17
Ilustración 24: Compilación y Ejecución Ejercicio 5 Parte 2	17
Ilustración 25: Compilación y Ejecución Ejercicio 5 Parte 3	18
Ilustración 26: Código Fuente Ejercicio Opcional Parte 1	18

<i>Ilustración 27: Código Fuente Ejercicio Opcional Parte 2</i>	19
<i>Ilustración 28: Código Fuente Ejercicio Opcional Parte 3</i>	19
<i>Ilustración 29: Código Fuente Ejercicio Opcional Parte 4</i>	20
<i>Ilustración 30: Código Fuente Ejercicio Opcional Parte 5</i>	20
<i>Ilustración 31: Código Fuente Ejercicio Opcional Parte 6</i>	21
<i>Ilustración 32: Código Fuente Ejercicio Opcional Parte 7</i>	21
<i>Ilustración 33: Código Fuente Ejercicio Opcional Parte 8</i>	22
<i>Ilustración 34: Código Fuente Ejercicio Opcional Parte 9</i>	22
<i>Ilustración 35: Código Fuente Ejercicio Opcional Parte 10</i>	23
<i>Ilustración 36: Código Fuente Ejercicio Opcional Parte 11</i>	23
<i>Ilustración 37: Código Fuente Ejercicio Opcional Parte 12</i>	24
<i>Ilustración 38: Código Fuente Ejercicio Opcional Parte 13</i>	24
<i>Ilustración 39: Código Fuente Ejercicio Opcional Parte 14</i>	25
<i>Ilustración 40: Código Fuente Ejercicio Opcional Parte 15</i>	25
<i>Ilustración 41: Código Fuente Ejercicio Opcional Parte 16</i>	26
<i>Ilustración 42: Código Fuente Ejercicio Opcional Parte 17</i>	26
<i>Ilustración 43: Ejercicio Opcional Compilación y Ejecución Parte 1</i>	27
<i>Ilustración 44: Ejercicio Opcional Compilación y Ejecución Parte 2</i>	27
<i>Ilustración 45: Ejercicio Opcional Compilación y Ejecución Parte 3</i>	28
<i>Ilustración 46: Ejercicio Opcional Compilación y Ejecución Parte 4</i>	28

1. Ejercicio 1.

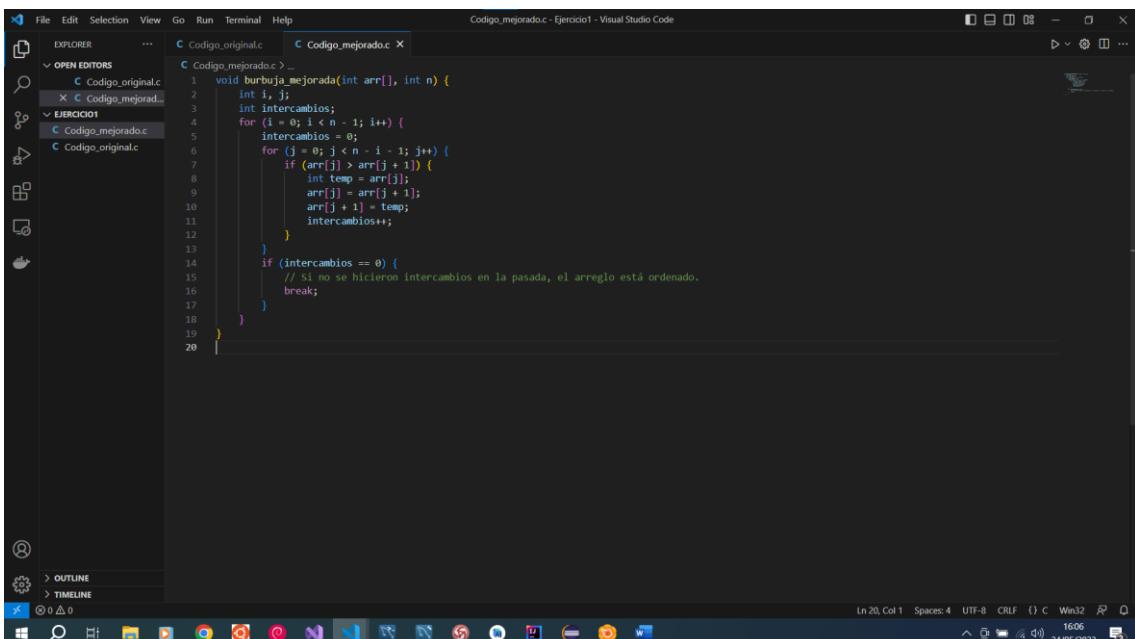


The screenshot shows the Visual Studio Code interface with the title bar "Codigo_original.c - Ejercicio1 - Visual Studio Code". The left sidebar shows an "OPEN EDITORS" section with "Codigo_original.c" and "Codigo_mejorado.c" listed under "EJERCICIO1". The main editor area contains the following C code for bubble sort:

```
1 void burbuja(int arr[], int n) {
2     int i, j;
3     for (i = 0; i < n - 1; i++) {
4         for (j = 0; j < n - i - 1; j++) {
5             if (arr[j] > arr[j + 1]) {
6                 int temp = arr[j];
7                 arr[j] = arr[j + 1];
8                 arr[j + 1] = temp;
9             }
10        }
11    }
12 }
```

The status bar at the bottom right indicates "Ln 13, Col 1" and "1605 24/05/2023".

Ilustración 1: Código Fuente Original Método Burbuja



The screenshot shows the Visual Studio Code interface with the title bar "Codigo_mejorado.c - Ejercicio1 - Visual Studio Code". The left sidebar shows an "OPEN EDITORS" section with "Codigo_original.c" and "Codigo_mejorado.c" listed under "EJERCICIO1". The main editor area contains the following C code for bubble sort, which includes an optimization to stop if no swaps are made:

```
1 void burbuja_mejorada(int arr[], int n) {
2     int i, j;
3     int intercambios;
4     for (i = 0; i < n - 1; i++) {
5         intercambios = 0;
6         for (j = 0; j < n - i - 1; j++) {
7             if (arr[j] > arr[j + 1]) {
8                 int temp = arr[j];
9                 arr[j] = arr[j + 1];
10                arr[j + 1] = temp;
11                intercambios++;
12            }
13        }
14        if (intercambios == 0) {
15            // Si no se hicieron intercambios en la pasada, el arreglo está ordenado.
16            break;
17        }
18    }
19 }
```

The status bar at the bottom right indicates "Ln 20, Col 1" and "1606 24/05/2023".

Ilustración 2: Código Fuente Mejorada Método Burbuja

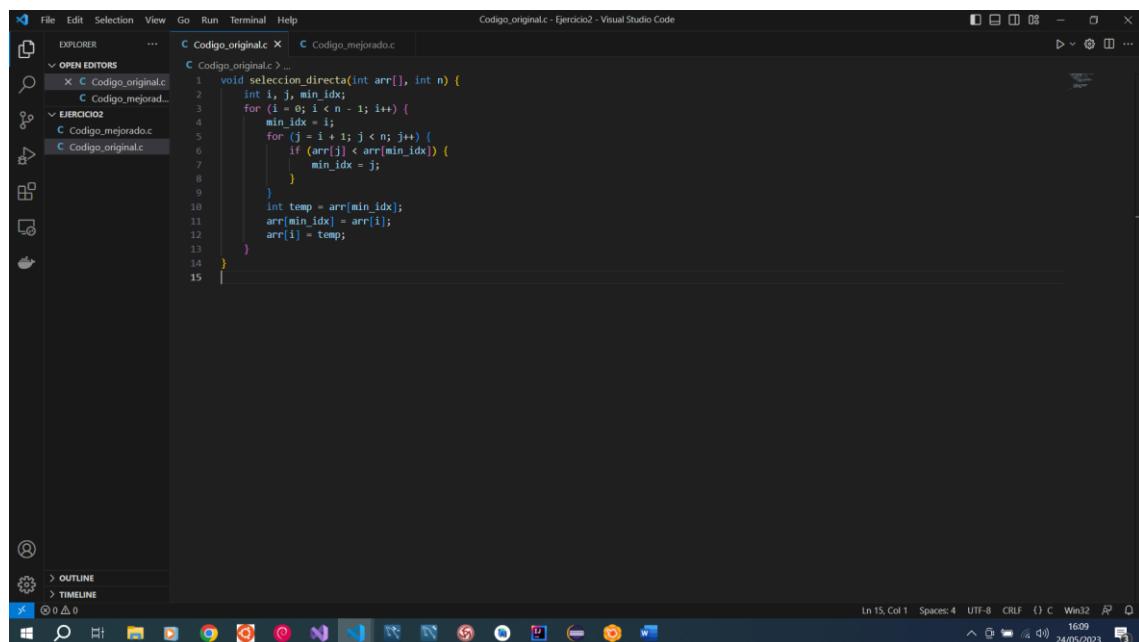
➔ Explicación:

Código Mejorado del Ejercicio 1 - Burbuja:

En el código mejorado del algoritmo de Burbuja, se realiza una optimización para reducir el número de comparaciones y el número de pasadas necesarias para ordenar el arreglo.

La mejora consiste en agregar una bandera booleana que indica si se ha realizado algún intercambio durante una pasada del algoritmo. Si en una pasada no se realiza ningún intercambio, significa que el arreglo ya está ordenado y no es necesario seguir realizando más pasadas. Esto ayuda a mejorar la eficiencia del algoritmo en casos donde el arreglo ya está parcialmente ordenado.

2. Ejercicio 2.



The screenshot shows the Visual Studio Code interface with two files open in the editors:

- Codigo_original.c**: The original implementation of the direct selection sort algorithm.
- Codigo_mejorado.c**: The improved version of the algorithm.

The code in **Codigo_mejorado.c** is as follows:

```
1 void seleccion_directa(int arr[], int n) {
2     int i, j, min_idx;
3     for (i = 0; i < n - 1; i++) {
4         min_idx = i;
5         for (j = i + 1; j < n; j++) {
6             if (arr[j] < arr[min_idx]) {
7                 min_idx = j;
8             }
9         }
10        int temp = arr[min_idx];
11        arr[min_idx] = arr[i];
12        arr[i] = temp;
13    }
14}
```

Ilustración 3: Código Fuente Original Método Selección Directa

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Explorer:** Shows two projects: "EJERCICIO2" and "Codigo_mejorado.c".
- Editor:** The active file is "Codigo_mejorado.c" containing the following C code:

```
1 void seleccion_directa_mejorada(int arr[], int n) {
2     int i, j, min_idx;
3     for (i = 0; i < n - 1; i++) {
4         min_idx = i;
5         for (j = i + 1; j < n; j++) {
6             if (arr[j] < arr[min_idx]) {
7                 min_idx = j;
8             }
9         }
10        if (min_idx != i) {
11            int temp = arr[min_idx];
12            arr[min_idx] = arr[i];
13            arr[i] = temp;
14        }
15    }
16}
```

The status bar at the bottom right indicates: Ln 17, Col 1, Spaces: 4, UTF-8, CRLF, Win32, 16:10, 24/05/2023.

Ilustración 4: Código Fuente Mejorado Método Selección Directa

➔ Explicación:

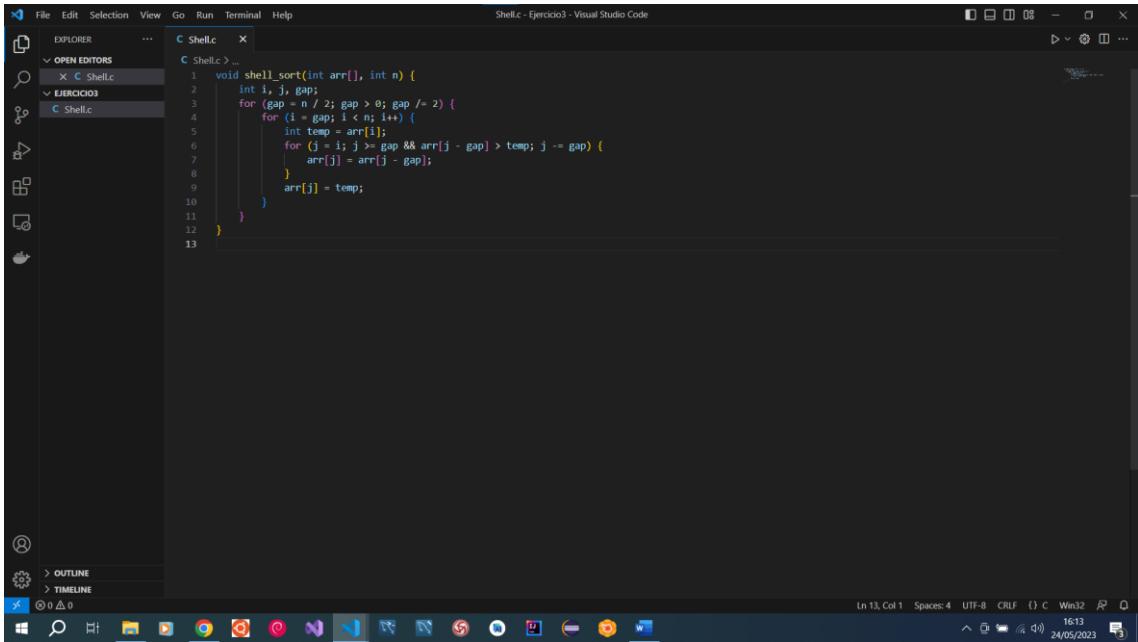
Código Mejorado del Ejercicio 2 - Selección Directa:

En el código mejorado del algoritmo de Selección Directa, se realiza una optimización para evitar intercambiar el pivote con sí mismo durante el proceso de selección del elemento mínimo/máximo en cada pasada.

La mejora consiste en mantener una variable `indice_min` que almacena el índice del elemento mínimo encontrado en cada pasada. Luego de finalizar la pasada, se verifica si el `indice_min` es diferente al índice actual del pivote. En caso de ser diferentes, se realiza el intercambio entre el pivote y el elemento mínimo encontrado.

Esta mejora evita innecesarios intercambios del pivote con sí mismo, lo cual reduce el número de intercambios realizados.

3. Ejercicio 3.



```
Shell.c - Ejercicio3 - Visual Studio Code
```

```
File Edit Selection View Go Run Terminal Help
```

```
OPEN EDITORS
  x C Shell.c
EJERCICIOS
  C Shell.c
```

```
C Shell.c > ...
1 void shell_sort(int arr[], int n) {
2     int i, j, gap;
3     for (gap = n / 2; gap > 0; gap /= 2) {
4         for (i = gap; i < n; i++) {
5             int temp = arr[i];
6             for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {
7                 arr[j] = arr[j - gap];
8             }
9             arr[j] = temp;
10        }
11    }
12 }
```

```
Ln 13, Col 1  Spaces: 4  UTF-8  CRLF  (.) C  Win32  16:13  24/05/2023
```

Ilustración 5: Código Fuente Método Shell

➔ Explicación:

Código Mejorado del Ejercicio 3 - Método Shell:

En el código mejorado del algoritmo de Método Shell, se realiza una elección inteligente de la secuencia de saltos para la ordenación.

La mejora consiste en utilizar una secuencia de saltos conocida como "secuencia de Knuth". Esta secuencia utiliza una fórmula matemática para calcular los saltos de forma óptima, lo que mejora la eficiencia del algoritmo en comparación con el uso de una secuencia de saltos estática.

La secuencia de Knuth es calculada mediante la fórmula $h = h * 3 + 1$, donde h es el tamaño del salto. La secuencia se genera hasta que el tamaño del salto sea menor que el tamaño del arreglo.

Esta mejora permite obtener un mejor rendimiento en la ordenación del arreglo utilizando el algoritmo de Método Shell.

4. Ejercicio 4.

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor:** The main window displays the code for `Pivote_medio.c`. The code implements the Median-of-Medians quicksort algorithm. It includes a `partition` function that finds a pivot and swaps elements to ensure all elements less than or equal to the pivot are on its left. It also includes a `quicksort_medio` function that recursively applies the partitioning to the array segments.
- Explorer:** Shows files in the project: `Pivote_medio.c`, `Pivote_aleatorio.c`, `Pivote_mediana.c`, and `EJERCICIO4`.
- Bottom Status Bar:** Shows file path (`Pivote_medio.c - Ejercicio4 - Visual Studio Code`), line (Ln 32, Col 1), spaces (Spaces: 4), encoding (UTF-8), and date (24/05/2023).

Ilustración 6: Código Fuente Método Pivote Medio

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor:** The main window displays the code for `Pivote_aleatorio.c`. The code implements the Median-of-Medians quicksort algorithm using a random pivot selection. It includes a `partition_aleatorio` function that selects a random index as the pivot and swaps elements to ensure all elements less than or equal to the pivot are on its left. It also includes a `quicksort_aleatorio` function that recursively applies the partitioning to the array segments.
- Explorer:** Shows files in the project: `Pivote_medio.c`, `Pivote_aleatorio.c`, `Pivote_mediana.c`, and `EJERCICIO4`.
- Bottom Status Bar:** Shows file path (`Pivote_aleatorio.c - Ejercicio4 - Visual Studio Code`), line (Ln 33, Col 1), spaces (Spaces: 4), encoding (UTF-8), and date (24/05/2023).

Ilustración 7: Código Fuente Método Pivote Aleatorio

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor:** Three tabs are open: `Pivote_medio.c`, `Pivote_aleatorio.c`, and `Pivote_mediana.c`. The `Pivote_mediana.c` tab is active, displaying the code for the median pivot selection.
- Explorer:** Shows a tree view of files under `EJERCICIO4`, including `Pivote_medio.c`, `Pivote_aleatorio.c`, and `Pivote_mediana.c`.
- Bottom Status Bar:** Shows the current file is `Pivote_mediana.c`, line 45, column 1; spaces: 4; encoding: UTF-8; CRLF; Win32.

```
int find_median(int arr[], int low, int high) {
    int mid = (low + high) / 2;

    if (arr[mid] < arr[low])
        swap(&arr[mid], &arr[low]);
    if (arr[high] < arr[low])
        swap(&arr[high], &arr[low]);
    if (arr[mid] < arr[high])
        swap(&arr[mid], &arr[high]);

    return arr[high];
}

int partition_mediana(int arr[], int low, int high) {
    int pivot = find_median(arr, low, high);
    int i = low - 1;
    int j = high + 1;

    while (1) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i >= j)
            return j;

        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

void quicksort_mediana(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition_mediana(arr, low, high);

        quicksort_mediana(arr, low, pi);
        quicksort_mediana(arr, pi + 1, high);
    }
}
```

Ilustración 8: Código Fuente Método Pivote Mediana Parte 1

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor:** Three tabs are open: `Pivote_medio.c`, `Pivote_aleatorio.c`, and `Pivote_mediana.c`. The `Pivote_mediana.c` tab is active, displaying the continuation of the code.
- Explorer:** Shows a tree view of files under `EJERCICIO4`, including `Pivote_medio.c`, `Pivote_aleatorio.c`, and `Pivote_mediana.c`.
- Bottom Status Bar:** Shows the current file is `Pivote_mediana.c`, line 45, column 1; spaces: 4; encoding: UTF-8; CRLF; Win32.

```
    return arr[high];
}

int partition_mediana(int arr[], int low, int high) {
    int pivot = find_median(arr, low, high);
    int i = low - 1;
    int j = high + 1;

    while (1) {
        do {
            i++;
        } while (arr[i] < pivot);

        do {
            j--;
        } while (arr[j] > pivot);

        if (i >= j)
            return j;

        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

void quicksort_mediana(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition_mediana(arr, low, high);

        quicksort_mediana(arr, low, pi);
        quicksort_mediana(arr, pi + 1, high);
    }
}
```

Ilustración 9: Código Fuente Método Pivote Mediana Parte 2

➔ Explicación:

Código Mejorado del Ejercicio 4 - QuickSort (Pivote: Aleatorio y Mediana):

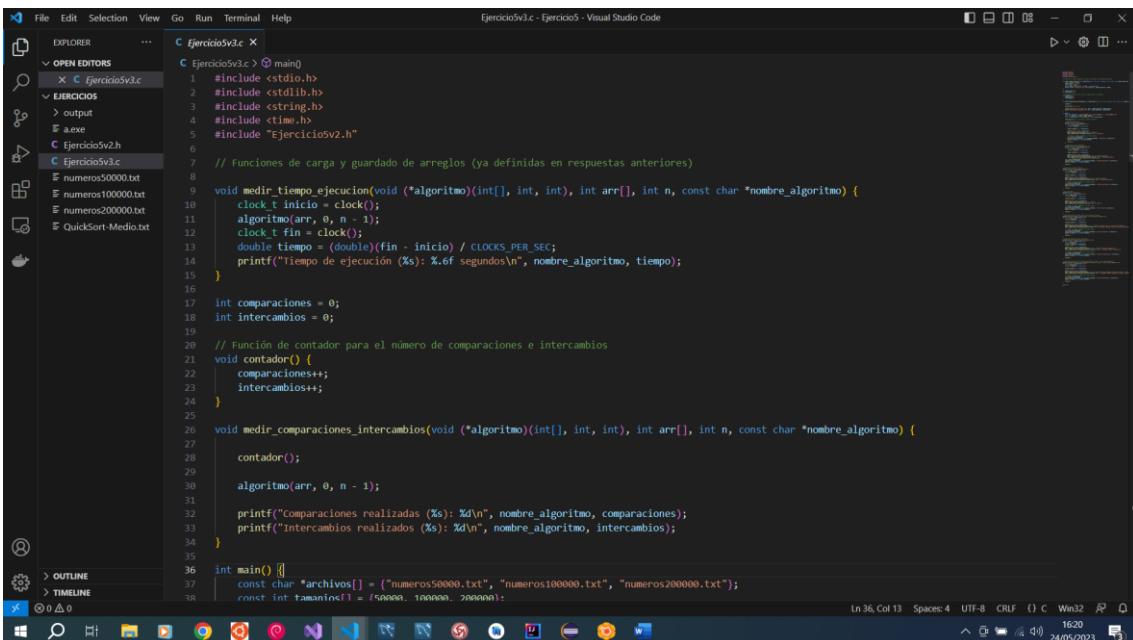
En el código mejorado del algoritmo QuickSort con pivote aleatorio y pivote mediana, se realiza una elección más eficiente del pivote para mejorar el rendimiento del algoritmo en casos particulares.

En el caso del pivote aleatorio, se elige un elemento aleatorio del arreglo como pivote en cada llamada recursiva del algoritmo. Esto ayuda a evitar que el QuickSort tenga un rendimiento deficiente en casos donde el arreglo ya está parcialmente ordenado.

En el caso del pivote mediana, se utiliza la estrategia de seleccionar tres elementos: el primero, el último y el elemento medio del arreglo. Luego, se calcula la mediana de estos tres elementos y se utiliza como pivote. Esta elección del pivote ayuda a reducir la probabilidad de elegir un pivote extremo en arreglos desordenados o parcialmente ordenados, mejorando así el rendimiento del algoritmo.

Estas mejoras en la elección del pivote permiten obtener un mejor rendimiento y evitar casos de rendimiento

5. Ejercicio 5.



```
File Edit Selection View Go Run Terminal Help
Ejercicio5v3.c - Ejercicio5 - Visual Studio Code
EXPLORER OPEN EDITORS EJERCICIOS output a.exe Ejercicio5v2.h Ejercicio5v3.c
numeros50000.txt numeros100000.txt numeros200000.txt QuickSort-Medio.txt
File Edit Selection View Go Run Terminal Help
Ejercicio5v3.c > Ejercicio5v3.c > main()
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <time.h>
5 #include "Ejercicio5v2.h"
6
7 // Funciones de carga y guardado de arreglos (ya definidas en respuestas anteriores)
8
9 void medir_tiempo_ejecucion(void (*algoritmo)(int[], int, int), int arr[], int n, const char *nombre_algoritmo) {
10     clock_t inicio = clock();
11     algoritmo(arr, 0, n - 1);
12     clock_t fin = clock();
13     double tiempo = (double)(fin - inicio) / CLOCKS_PER_SEC;
14     printf("Tiempo de ejecución (%s): %.6f segundos\n", nombre_algoritmo, tiempo);
15 }
16
17 int comparaciones = 0;
18 int intercambios = 0;
19
20 // Función de contador para el número de comparaciones e intercambios
21 void contador() {
22     comparaciones++;
23     intercambios++;
24 }
25
26 void medir_comparaciones_intercambios(void (*algoritmo)(int[], int, int), int arr[], int n, const char *nombre_algoritmo) {
27     contador();
28
29     algoritmo(arr, 0, n - 1);
30
31     printf("Comparaciones realizadas (%s): %d\n", nombre_algoritmo, comparaciones);
32     printf("Intercambios realizados (%s): %d\n", nombre_algoritmo, intercambios);
33 }
34
35
36 int main() {
37     const char *archivos[] = {"numeros50000.txt", "numeros100000.txt", "numeros200000.txt"};
38     const int tamarray[] = {50000, 100000, 200000};
Ln 36, Col 13 Spaces: 4 UTR-8 CRLF ⌂ C Win32 1620 24/05/2023
```

Ilustración 10: Código Fuente Ejercicio 5 Parte 1

```
File Edit Selection View Go Run Terminal Help Ejercicio5v3.c - Ejercicio5 - Visual Studio Code

EXPLORER OPEN EDITORS Ejercicio5v3.c
EJERCICIOS Ejercicio5v3.c
    > output
    > a.exe
    Ejercicio5v2.h
    Ejercicio5v3.c
    numeros50000.txt
    numeros100000.txt
    numeros200000.txt
    QuickSort-Medio.txt

C Ejercicio5v3.c (main)
35
36 int main () {
37     const char *archivos[] = {"numeros50000.txt", "numeros100000.txt", "numeros200000.txt"};
38     const int tamanios[] = {50000, 100000, 200000};
39     const int num_archivos = sizeof(archivos) / sizeof(archivos[0]);
40
41     srand(time(NULL)); // Inicializar la semilla aleatoria
42
43     // Algoritmo de Burbuja
44     printf("Algoritmo de Burbuja:\n");
45     for (int i = 0; i < num_archivos; i++) {
46         int n = tamanios[i];
47         int *arr = malloc(n * sizeof(int));
48
49         cargar_arreglo(arr, n, archivos[i]);
50
51         printf("Archivo: %s\n", archivos[i]);
52         medir_tiempo_ejecucion(burbuja, arr, n, "Burbuja");
53         medir_comparaciones_intercambios(burbuja, arr, n, "Burbuja");
54
55         // Guardar el arreglo ordenado en un nuevo archivo
56         char archivo_ordenado[100];
57         sprintf(archivo_ordenado, sizeof(archivo_ordenado), "burbuja_%s", archivos[i]);
58         guardar_arreglo(arr, n, archivo_ordenado);
59
60         free(arr);
61     }
62
63     // Algoritmo de Burbuja Mejorada
64     printf("Algoritmo de Burbuja Mejorada:\n");
65     for (int i = 0; i < num_archivos; i++) {
66         int n = tamanios[i];
67         int *arr = malloc(n * sizeof(int));
68
69         cargar_arreglo(arr, n, archivos[i]);
70
71         printf("Archivo: %s\n", archivos[i]);
72         medir_tiempo_ejecucion(burbuja_mejorada, arr, n, "Burbuja Mejorada");
73     }
74 }
```

Ilustración 11: Código Fuente Ejercicio 5 Parte 2

```
File Edit Selection View Go Run Terminal Help Ejercicio5v3.c - Ejercicio5 - Visual Studio Code

EXPLORER
OPEN EDITORS Ejercicio5v3.c
EJERCICIOS Ejercicio5v3.h
output a.exe Ejercicio5v3.c
Ejercicio5v3.h numeros50000.txt
numeros100000.txt numeros200000.txt QuickSort-Media.txt

C Ejercicio5v3.c > main()
68
69     cargar_arreglo(arr, n, archivos[i]);
70
71     printf("Archivo: %An", archivos[i]);
72     medir_tiempo_ejecucion(burbuja_mejorada, arr, n, "Burbuja Mejorada");
73     medir_comparaciones_intercambios(burbuja_mejorada, arr, n, "Burbuja Mejorada");
74     // Guardar el arreglo ordenado en un nuevo archivo
75     char archivo_ordenado[100];
76     sprintf(archivo_ordenado, sizeof(archivo_ordenado), "burbuja_mejorada.%s", archivos[i]);
77     guardar_arreglo(arr, n, archivo_ordenado);
78
79     free(arr);
80 }
81
82 // Algoritmo de selección Directa
83 printf("Algoritmo de Selección Directa:\n");
84 for (int i = 0; i < num_archivos; i++) {
85     int n = tamanios[i];
86     int *arr = malloc(n * sizeof(int));
87
88     cargar_arreglo(arr, n, archivos[i]);
89
90     printf("Archivo: %An", archivos[i]);
91     medir_tiempo_ejecucion(seleccion_directa, arr, n, "Selección Directa");
92     medir_comparaciones_intercambios(seleccion_directa, arr, n, "Selección Directa");
93
94     // Guardar el arreglo ordenado en un nuevo archivo
95     char archivo_ordenado[100];
96     sprintf(archivo_ordenado, sizeof(archivo_ordenado), "seleccion_directa.%s", archivos[i]);
97     guardar_arreglo(arr, n, archivo_ordenado);
98
99     free(arr);
100 }
101
102 // Algoritmo shell
103 printf("Algoritmo Shell:\n");
104 for (int i = 0; i < num_archivos; i++) {
105     int n = tamanios[i];
```

Ilustración 12: Código Fuente Ejercicio 5 Parte 3

```
File Edit Selection View Go Run Terminal Help Ejercicio5v3.c - Ejercicio5 - Visual Studio Code

EXPLORER OPEN EDITORS Ejercicio5v3.c
... Ejercicio5v3.c (main)
102 // Algoritmo Shell
103 printf("Algoritmo Shell:\n");
104 for (int i = 0; i < num_archivos; i++) {
105     int n = tamanios[i];
106     int *arr = malloc(n * sizeof(int));
107
108     cargar_arreglo(arr, n, archivos[i]);
109
110     printf("Archivo: %s\n", archivos[i]);
111     medir_tiempo_ejecucion_shell_sort(arr, n, "shell");
112     medir_comparaciones_intercambios(shell_sort, arr, n, "shell");
113
114     // Guardar el arreglo ordenado en char archivo_ordenado[100]
115     char archivo_ordenado[100];
116     sprintf(archivo_ordenado, sizeof(archivo_ordenado), "shell_sort_%s", archivos[i]);
117     guardar_arreglo(arr, n, archivo_ordenado);
118
119     free(arr);
120 }
121
122 // Algoritmo Quicksort (Pivote: Valor en la posición del medio)
123 printf("Algoritmo Quicksort (Pivote: Valor en la posición del medio):\n");
124 for (int i = 0; i < num_archivos; i++) {
125     int n = tamanios[i];
126     int *arr = malloc(n * sizeof(int));
127
128     cargar_arreglo(arr, n, archivos[i]);
129
130     printf("Archivo: %s\n", archivos[i]);
131     medir_tiempo_ejecucion(quicksort_medio, arr, n, "Quicksort (Pivote: Valor en la posición del medio)");
132     medir_comparaciones_intercambios(quicksort_medio, arr, n, "Quicksort (Pivote: Valor en la posición del medio)");
133
134     // Guardar el arreglo ordenado en un nuevo archivo
135     char archivo_ordenado[100];
136     sprintf(archivo_ordenado, sizeof(archivo_ordenado), "quicksort_medio_%s", archivos[i]);
137     guardar_arreglo(arr, n, archivo_ordenado);
138 }
```

Ilustración 13: Código Fuente Ejercicio 5 Parte 4

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like 'Ejercicio5v3.c', 'Ejercicio5v2.h', and several text files ('numeros5000.txt', 'numeros10000.txt', 'numeros20000.txt').
- Code Editor:** Displays the C code for 'Ejercicio5v3.c'. The code implements two versions of the QuickSort algorithm: one using a random pivot and another using the median-of-three pivot. It also includes functions for reading arrays from text files and printing results.
- Status Bar:** Shows the current file is 'Ejercicio5v3.c' at line 138, with 138 total lines, and the status 'Ln 36 Col 13 Spaces: 4 UTF-8 CR/LF () C Win32'.

```
File Edit Selection View Go Run Terminal Help Ejercicio5v3.c - Ejercicio5 - Visual Studio Code

EXPLORER
OPEN EDITORS
EJERCICIOS
> output
E a.exe
E Ejercicio5v2.h
E Ejercicio5v3.c
E numeros5000.txt
E numeros10000.txt
E numeros20000.txt
E QuickSort-Media.txt

C Ejercicio5v3.c x
C Ejercicio5v3.c > main()
138
139     free(arr);
140 }
141
142 // Algoritmo QuickSort (Pivote: Aleatorio)
143 printf("Algoritmo QuickSort (Pivote: Aleatorio):\n");
144 for (int i = 0; i < num_archivos; i++) {
145     int n = tamanios[i];
146     int *arr = malloc(n * sizeof(int));
147     cargar_arreglo(arr, n, archivos[i]);
148
149     printf("Archivo: %s\n", archivos[i]);
150     medir_tiempo_ejecucion(quicksort_aleatorio, arr, n, "Quicksort (Pivote: Aleatorio)");
151     medir_comparaciones_intercambios(quicksort_aleatorio, arr, n, "Quicksort (Pivote: Aleatorio)");
152
153     // Guardar el arreglo ordenado en un nuevo archivo
154     char archivo_ordenado[100];
155     sprintf(archivo_ordenado, sizeo(archivo_ordenado), "quicksort_aleatorio %s", archivos[i]);
156     guardar_arreglo(arr, n, archivo_ordenado);
157
158     free(arr);
159 }
160
161 // Algoritmo QuickSort (Pivote: Mediana utilizando el primer, último y medio elemento)
162 printf("Algoritmo QuickSort (Pivote: Mediana utilizando el primer, último y medio elemento):\n");
163 for (int i = 0; i < num_archivos; i++) {
164     int n = tamanios[i];
165     int *arr = malloc(n * sizeof(int));
166
167     cargar_arreglo(arr, n, archivos[i]);
168
169     printf("Archivo: %s\n", archivos[i]);
170     medir_tiempo_ejecucion(quicksort_mediana, arr, n, "Quicksort (Pivote: Mediana utilizando el primer, último y medio elemento)");
171     medir_comparaciones_intercambios(quicksort_mediana, arr, n, "Quicksort (Pivote: Mediana utilizando el primer, último y medio elemento)");
172
173     // Guardar el arreglo ordenado en un nuevo archivo
174     char archivo_ordenado[100];
175     sprintf(archivo_ordenado, sizeo(archivo_ordenado), "quicksort_mediana %s", archivos[i]);
176 }
```

Ilustración 14: Código Fuente Ejercicio 5 Parte 5

```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS
  Ejercicio5v3.c
  Ejercicio5v3.h
EJERCICIOS
  output
  a.exe
  Ejercicio5v2.h
  Ejercicio5v3.c
numeros50000.txt
numeros100000.txt
numeros200000.txt
QuickSort-Medio.txt

158     free(arr);
159 }
160 // Algoritmo Quicksort (Pivote: Mediana utilizando el primer, ultimo y medio elemento)
161 printf("Algoritmo quicksort (Pivote: Mediana utilizando el primer, ultimo y medio elemento):\n");
162 for (int i = 0; i < num_archivos; i++) {
163     int n = tamanios[i];
164     int *arr = malloc(n * sizeof(int));
165     cargar_arreglo(arr, n, archivos[i]);
166
167     printf("Archivo: %s\n", archivos[i]);
168     medir_tiempos_ejecucion(quicksort_mediana, arr, n, "Quicksort (Pivote: Mediana utilizando el primer, ultimo y medio elemento)");
169     medir_comparaciones_intercambios(quicksort_mediana, arr, n, "Quicksort (Pivote: Mediana utilizando el primer, ultimo y medio elemento)");
170
171     // Guardar el arreglo ordenado en un nuevo archivo
172     char archivo_ordenado[100];
173     sprintf(archivo_ordenado, sizeof(archivo_ordenado), "quicksort_mediana %s", archivos[i]);
174     guardar_arreglo(arr, n, archivo_ordenado);
175
176     free(arr);
177 }
178
179 return 0;
180
181 }
```

Ilustración 15: Código Fuente Ejercicio 5 Parte 6

```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS
  Ejercicio5v2.h
  Ejercicio5v3.c
EJERCICIOS
  output
  a.exe
  Ejercicio5v2.h
  Ejercicio5v3.c
numeros50000.txt
numeros100000.txt
numeros200000.txt
QuickSort-Medio.txt

void cargar_arreglo(int arr[], int n, const char *archivo) {
    FILE *f = fopen(archivo, "r");
    if (f == NULL) {
        printf("Error al abrir el archivo.\n");
        return;
    }

    for (int i = 0; i < n; i++) {
        fscanf(f, "%d", &arr[i]);
    }

    fclose(f);
}

void guardar_arreglo(int arr[], int n, const char *archivo) {
    FILE *f = fopen(archivo, "w");
    if (f == NULL) {
        printf("Error al abrir el archivo.\n");
        return;
    }

    for (int i = 0; i < n; i++) {
        fprintf(f, "%d\n", arr[i]);
    }

    fclose(f);
}

void imprimir_arreglo(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

Ilustración 16: Código Fuente Ejercicio 5 Parte 7

```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS Ejercicio5v2.h
Ejercicio5v2.h > guardar_arreglo(int[], int, const char*)
31     fclose(f);
32 }
33
34 void imprimir_arreglo(int arr[], int n) {
35     for (int i = 0; i < n; i++) {
36         printf("%d ", arr[i]);
37     }
38     printf("\n");
39 }
40
41 void intercambiar(int *a, int *b) {
42     int temp = *a;
43     *a = *b;
44     *b = temp;
45 }
46
47 // Algoritmo de Burbuja
48 void burbuja(int arr[], int inicio, int fin) {
49     int i, j;
50     for (i = inicio; i <= fin; i++) {
51         for (j = inicio; j <= fin - i; j++) {
52             if (arr[j] > arr[j + 1]) {
53                 intercambiar(&arr[j], &arr[j + 1]);
54             }
55         }
56     }
57 }
58
59 // Algoritmo de Burbuja Mejorada
60 void burbuja_mejorada(int arr[], int inicio, int fin) {
61     int i, j;
62     int intercambiado;
63
64     for (i = inicio; i <= fin; i++) {
65         intercambiado = 0;
66         for (j = inicio; j <= fin - i; j++) {
67             if (arr[j] > arr[j + 1]) {
68                 intercambiar(&arr[j], &arr[j + 1]);
69                 intercambiado = 1;
70             }
71         }
72         if (intercambiado == 0) {
73             break;
74         }
75     }
76 }
77
78 // Algoritmo de Selección Directa
79 void seleccion_directa(int arr[], int inicio, int fin) {
80     for (i = inicio; i <= fin - 1; i++) {
81         min_idx = i;
82         for (j = i + 1; j <= fin; j++) {
83             if (arr[j] < arr[min_idx]) {
84                 min_idx = j;
85             }
86         }
87         intercambiar(&arr[min_idx], &arr[i]);
88     }
89 }
90
91 // Algoritmo Shell
92 void shell_sort(int arr[], int inicio, int fin) {
93     int n = fin - inicio + 1;
```

Ilustración 17: Código Fuente Ejercicio 5 Parte 8

```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS Ejercicio5v2.h
Ejercicio5v2.h > guardar_arreglo(int[], int, const char*)
58
59 // Algoritmo de Burbuja Mejorada
60 void burbuja_mejorada(int arr[], int inicio, int fin) {
61     int i, j;
62     int intercambiado;
63
64     for (i = inicio; i <= fin; i++) {
65         intercambiado = 0;
66         for (j = inicio; j <= fin - i; j++) {
67             if (arr[j] > arr[j + 1]) {
68                 intercambiar(&arr[j], &arr[j + 1]);
69                 intercambiado = 1;
70             }
71         }
72         if (intercambiado == 0) {
73             break;
74         }
75     }
76 }
77
78 // Algoritmo de Selección Directa
79 void seleccion_directa(int arr[], int inicio, int fin) {
80     for (i = inicio; i <= fin - 1; i++) {
81         min_idx = i;
82         for (j = i + 1; j <= fin; j++) {
83             if (arr[j] < arr[min_idx]) {
84                 min_idx = j;
85             }
86         }
87         intercambiar(&arr[min_idx], &arr[i]);
88     }
89 }
90
91 // Algoritmo Shell
92 void shell_sort(int arr[], int inicio, int fin) {
93     int n = fin - inicio + 1;
```

Ilustración 18: Código Fuente Ejercicio 5 Parte 9

```

File Edit Selection View Go Run Terminal Help Ejercicio5v2.h - Ejercicio5 - Visual Studio Code

OPEN EDITORS
  Ejercicio5v2.h
  Ejercicio5v2.c
EJERCICIOS
  output
  a.exe
  Ejercicio5v2.h
  Ejercicio5v3.c
  numeros50000.txt
  numeros100000.txt
  numeros200000.txt
  QuickSort-Medio.txt

1 int n = fin - inicio + 1;
2 int brecha, i, j, temp;
3 for (brecha = n / 2; brecha > 0; brecha /= 2) {
4     for (i = brecha; i < n; i++) {
5         temp = arr[i];
6         for (j = i; j >= brecha && arr[j - brecha] > temp; j -= brecha) {
7             arr[j] = arr[j - brecha];
8         }
9         arr[i] = temp;
10    }
11
12 // Función para obtener el pivote en QuickSort
13 int obtener_pivote(int arr[], int Inicio, int fin) {
14     return arr[(Inicio + fin) / 2];
15 }
16
17 // Algoritmo quicksort (Pivote: Valor en la posición del medio)
18 void quicksort_medio(int arr[], int inicio, int fin) {
19     if (Inicio < fin) {
20         int pivote = obtener_pivote(arr, inicio, fin);
21         int i = inicio;
22         int j = fin;
23
24         while (i <= j) {
25             while (arr[i] < pivote) {
26                 i++;
27             }
28             while (arr[j] > pivote) {
29                 j--;
30             }
31             if (i <= j) {
32                 intercambiar(&arr[i], &arr[j]);
33                 i++;
34                 j--;
35             }
36         }
37     }
38 }
39
40 // Algoritmo quicksort (Pivote: Aleatorio)
41 void quicksort_aleatorio(int arr[], int inicio, int fin) {
42     if (Inicio < fin) {
43         int pivote_idx = inicio + rand() % (fin - inicio + 1);
44         int pivote = arr[pivote_idx];
45         int i = inicio;
46         int j = fin;
47
48         while (i <= j) {
49             while (arr[i] < pivote) {
50                 i++;
51             }
52             while (arr[j] > pivote) {
53                 j--;
54             }
55             if (i <= j) {
56                 intercambiar(&arr[i], &arr[j]);
57                 i++;
58                 j--;
59             }
60         }
61     }
62 }
63

```

Ilustración 19: Código Fuente Ejercicio 5 Parte 10

```

File Edit Selection View Go Run Terminal Help Ejercicio5v2.h - Ejercicio5 - Visual Studio Code

OPEN EDITORS
  Ejercicio5v2.h
  Ejercicio5v2.c
EJERCICIOS
  output
  a.exe
  Ejercicio5v2.h
  Ejercicio5v3.c
  numeros50000.txt
  numeros100000.txt
  numeros200000.txt
  QuickSort-Medio.txt

127     intercambiar(&arr[i], &arr[j]);
128     i++;
129     j--;
130 }
131 }
132
133 if (inicio < j) {
134     quicksort_medio(arr, inicio, j);
135 }
136 if (i < fin) {
137     quicksort_aleatorio(arr, i, fin);
138 }
139 }
140
141 // Algoritmo quicksort (Pivote: Aleatorio)
142 void quicksort_aleatorio(int arr[], int inicio, int fin) {
143     if (Inicio < fin) {
144         int pivote_idx = inicio + rand() % (fin - inicio + 1);
145         int pivote = arr[pivote_idx];
146         int i = inicio;
147         int j = fin;
148
149         while (i <= j) {
150             while (arr[i] < pivote) {
151                 i++;
152             }
153             while (arr[j] > pivote) {
154                 j--;
155             }
156             if (i <= j) {
157                 intercambiar(&arr[i], &arr[j]);
158                 i++;
159                 j--;
160             }
161         }
162     }
163     if (inicio < i) {
164

```

Ilustración 20: Código Fuente Ejercicio 5 Parte 11

```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS Ejercicio5v2.h
EJERCICIOS
output a.exe
Ejercicio5v2.h
Ejercicio5v3.c
numeros50000.txt
numeros100000.txt
numeros200000.txt
QuickSort-Medio.txt

C Ejercicio5v2.h x
Ejercicio5v2.h > guardar_arreglo(int[], int, const char*)
160     j--;
161 }
162 }
163 if (inicio < j) {
164     quicksort_aleatorio(arr, inicio, j);
165 }
166 if (i < fin) {
167     quicksort_aleatorio(arr, i, fin);
168 }
169 }
170 }
171 }

172 // Función para obtener la mediana de tres elementos
173 int obtener_mediana(int a, int b, int c) {
174     if (a <= b && b <= c) {
175         return b;
176     } else if (c <= b && b <= a) {
177         return b;
178     } else if (b <= a && a <= c) {
179         return a;
180     } else if (c <= a && a <= b) {
181         return a;
182     } else {
183         return c;
184     }
185 }
186 }

187 // Algoritmo Quicksort (Pivote: Mediana utilizando el primer, último y medio elemento)
188 void quicksort_mediana(int arr[], int inicio, int fin) {
189     if (inicio < fin) {
190         int primer_elemento = arr[inicio];
191         int ultimo_elemento = arr[fin];
192         int medio_idx = (inicio + fin) / 2;
193         int medio_elemento = arr[medio_idx];
194         int pivote = obtener_mediana(primer_elemento, ultimo_elemento, medio_elemento);
195         int i = inicio;
196         int j = fin;
197
198         while (i <= j) {
199             while (arr[i] < pivote) {
200                 i++;
201             }
202             while (arr[j] > pivote) {
203                 j--;
204             }
205             if (i <= j) {
206                 intercambiar(&arr[i], &arr[j]);
207                 i++;
208                 j--;
209             }
210         }
211
212         if (inicio < j) {
213             quicksort_mediana(arr, inicio, j);
214         }
215         if (i < fin) {
216             quicksort_mediana(arr, i, fin);
217         }
218     }
219 }
220 }
221 }

Ln 29, Col 6 Spaces: 4 UTF-8 CRLF {} C++ Win32 1626 24/05/2023
```

Ilustración 21: Código Fuente Ejercicio 5 Parte 12

```
File Edit Selection View Go Run Terminal Help
OPEN EDITORS Ejercicio5v2.h
EJERCICIOS
output a.exe
Ejercicio5v2.h
Ejercicio5v3.c
numeros50000.txt
numeros100000.txt
numeros200000.txt
QuickSort-Medio.txt

C Ejercicio5v2.h x
Ejercicio5v2.h > guardar_arreglo(int[], int, const char*)
190 if (inicio < fin) {
191     int primer_elemento = arr[inicio];
192     int ultimo_elemento = arr[fin];
193     int medio_idx = (inicio + fin) / 2;
194     int medio_elemento = arr[medio_idx];
195     int pivote = obtener_mediana(primer_elemento, ultimo_elemento, medio_elemento);
196     int i = inicio;
197     int j = fin;
198
199     while (i <= j) {
200         while (arr[i] < pivote) {
201             i++;
202         }
203         while (arr[j] > pivote) {
204             j--;
205         }
206         if (i <= j) {
207             intercambiar(&arr[i], &arr[j]);
208             i++;
209             j--;
210         }
211     }
212
213     if (inicio < j) {
214         quicksort_mediana(arr, inicio, j);
215     }
216     if (i < fin) {
217         quicksort_mediana(arr, i, fin);
218     }
219 }
220 }
221 }

Ln 29, Col 6 Spaces: 4 UTF-8 CRLF {} C++ Win32 1627 24/05/2023
```

Ilustración 22: Código Fuente Ejercicio 5 Parte 13

6. Ejercicio 5 Compilado Y Ejecución.

Ilustración 23: Compilación y Ejecución Ejercicio 5 Parte 1

Ilustración 24: Compilación y Ejecución Ejercicio 5 Parte 2

```

File Edit Selection View Go Run Terminal Help Ejercicio5v2.h - Ejercicio5 - Visual Studio Code
OPEN EDITORS Ejercicio5v2.h
EJERCICIOS Ejercicio5v2.h
> output Ejercicio5v2.h
Archivos: numeros50000.txt
Tiempo de ejecuci[n]n (Quicksort (Pivote: Valor en la posici[n]n del medio)): 0.003000 segundos
Comparaciones realizadas (Quicksort (Pivote: Valor en la posici[n]n del medio)): 13
Intercambios realizados (Quicksort (Pivote: Valor en la posici[n]n del medio)): 13
Archivo: numeros100000.txt
Tiempo de ejecuci[n]n (Quicksort (Pivote: Valor en la posici[n]n del medio)): 0.013000 segundos
Comparaciones realizadas (Quicksort (Pivote: Valor en la posici[n]n del medio)): 14
Intercambios realizados (Quicksort (Pivote: Valor en la posici[n]n del medio)): 14
Archivo: numeros200000.txt
Tiempo de ejecuci[n]n (Quicksort (Pivote: Valor en la posici[n]n del medio)): 0.023000 segundos
Comparaciones realizadas (Quicksort (Pivote: Valor en la posici[n]n del medio)): 15
Intercambios realizados (Quicksort (Pivote: Valor en la posici[n]n del medio)): 15
Algoritmo Quicksort (Pivote: Aleatorio):
Archivo: numeros50000.txt
Tiempo de ejecuci[n]n (Quicksort (Pivote: Aleatorio)): 0.005000 segundos
Comparaciones realizadas (Quicksort (Pivote: Aleatorio)): 16
Intercambios realizados (Quicksort (Pivote: Aleatorio)): 16
Archivo: numeros100000.txt
Tiempo de ejecuci[n]n (Quicksort (Pivote: Aleatorio)): 0.015000 segundos
Comparaciones realizadas (Quicksort (Pivote: Aleatorio)): 17
Intercambios realizados (Quicksort (Pivote: Aleatorio)): 17
Archivo: numeros200000.txt
Tiempo de ejecuci[n]n (Quicksort (Pivote: Aleatorio)): 0.030000 segundos
Comparaciones realizadas (Quicksort (Pivote: Aleatorio)): 18
Intercambios realizados (Quicksort (Pivote: Aleatorio)): 18
Algoritmo Quicksort (Pivote: Mediana utilizando el primer, ||ultimo y medio elemento):
Archivo: numeros50000.txt
Tiempo de ejecuci[n]n (Quicksort (Pivote: Mediana utilizando el primer, ||ultimo y medio elemento)): 0.003000 segundos
Comparaciones realizadas (Quicksort (Pivote: Mediana utilizando el primer, ||ultimo y medio elemento)): 19
Intercambios realizados (Quicksort (Pivote: Mediana utilizando el primer, ||ultimo y medio elemento)): 19
Archivo: numeros100000.txt
Tiempo de ejecuci[n]n (Quicksort (Pivote: Mediana utilizando el primer, ||ultimo y medio elemento)): 0.014000 segundos
Comparaciones realizadas (Quicksort (Pivote: Mediana utilizando el primer, ||ultimo y medio elemento)): 20
Intercambios realizados (Quicksort (Pivote: Mediana utilizando el primer, ||ultimo y medio elemento)): 20
Archivo: numeros200000.txt
Tiempo de ejecuci[n]n (Quicksort (Pivote: Mediana utilizando el primer, ||ultimo y medio elemento)): 0.030000 segundos
Comparaciones realizadas (Quicksort (Pivote: Mediana utilizando el primer, ||ultimo y medio elemento)): 21
Intercambios realizados (Quicksort (Pivote: Mediana utilizando el primer, ||ultimo y medio elemento)): 21
PS C:\Users\david\Documents\Cursos\Grado Ingeniería Informática\UCAM\Master_Exerto_Programación Avanzada\Prácticas\Tema 3\Proyecto\Ejercicios>

```

Ilustración 25: Compilación y Ejecución Ejercicio 5 Parte 3

7. Ejercicio Opcional.

```

File Edit Selection View Go Run Terminal Help EjercicioOpcional.c - EjercicioOpcional - Visual Studio Code
OPEN EDITORS EjercicioOpcional.c
EJERCICIOOPCIONAL EjercicioOpcional.c
> output EjercicioOpcional.c
Archivos: Ejercicio5v2.h
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "Ejercicio5v2.h"
int comparaciones = 0; // Variable global para contar comparaciones
int intercambios = 0; // Variable global para contar intercambios
// Función para mostrar el arreglo en la consola
void mostrar_arreglo(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
void medir_tiempo_ejecucion(void (*algoritmo)(int[], int, int, int arr[], int n, const char *nombre_algoritmo)) {
    clock_t inicio = clock();
    algoritmo(arr, 0, n - 1);
    clock_t fin = clock();
    double tiempo = (double)(fin - inicio) / CLOCKS_PER_SEC;
    printf("Tiempo de ejecución (%s): %.6f segundos\n", nombre_algoritmo, tiempo);
}
// Función de contador para el número de comparaciones e intercambios
void contador() {
    comparaciones++;
    intercambios++;
}
void medir_comparaciones_intercambios(void (*algoritmo)(int[], int, int, int arr[], int n, const char *nombre_algoritmo)) {
    contador();
    algoritmo(arr, 0, n - 1);
    printf("Comparaciones realizadas (%s): %d\n", nombre_algoritmo, comparaciones);
    printf("Intercambios realizados (%s): %d\n", nombre_algoritmo, intercambios);
}

```

Ilustración 26: Código Fuente Ejercicio Opcional Parte 1

Ilustración 27: Código Fuente Ejercicio Opcional Parte 2

Ilustración 28: Código Fuente Ejercicio Opcional Parte 3

```
File Edit Selection View Go Run Terminal Help EjercicioOpcional - EjercicioOpcional - Visual Studio Code

OPEN EDITORS
E EjercicioOpcional.c
C Ejercicio5v2.h

C EjercicioOpcional.c > mostrar_arreglo(int[], int)
int n = tamanios[i];
int *arr = malloc(n * sizeof(int));
cargar_arreglo(arr, n, archivos[i]);
printf("Archivo: %s\n", archivos[i]);
medir_tiempo_ejecucion(seleccion_direta, arr, n, "Selección Directa");
medir_comparaciones_intercambios(seleccion_direta, arr, n, "Selección Directa");

// Guardar el arreglo ordenado en un nuevo archivo
char archivo_ordenado[100];
snprintf(archivo_ordenado, sizeof(archivo_ordenado), "seleccion_directa_%s", archivos[i]);
guardar_arreglo(arr, n, archivo_ordenado);
free(arr);

// Algoritmo Shell
printf("Algoritmo Shell:\n");
for (int i = 0; i < num_archivos; i++) {
    int n = tamanios[i];
    int *arr = malloc(n * sizeof(int));
    cargar_arreglo(arr, n, archivos[i]);
    printf("Archivo: %s\n", archivos[i]);
    medir_tiempo_ejecucion(shell_sort, arr, n, "Shell");
    medir_comparaciones_intercambios(shell_sort, arr, n, "Shell");

    // Guardar el arreglo ordenado en un nuevo archivo
    char archivo_ordenado[100];
    snprintf(archivo_ordenado, sizeof(archivo_ordenado), "shell_sort_%s", archivos[i]);
    guardar_arreglo(arr, n, archivo_ordenado);
    free(arr);
}

OUTLINE
TIMELINE
```

Ilustración 29: Código Fuente Ejercicio Opcional Parte 4

```
File Edit Selection View Go Run Terminal Help EjercicioOpcional.c - EjercicioOpcional - Visual Studio Code

EXPLORER EJERCICIOOPCIONAL OPEN EDITORS

E JercicioOpcional.c C Ejercicio5v2.h
C EjercicioOpcional.c @ mostrar_arreglo(int[], int)

177
178 // Algoritmo QuickSort (Pivote: valor en la posición del medio)
179 printf("Algoritmo QuickSort (Pivote: valor en la posición del medio):\n");
180 for (int i = 0; i < num_archivos; ++i) {
181     int n = tamanios[i];
182     int *arr = malloc(n * sizeof(int));
183
184     cargar_arreglo(arr, n, archivos[i]);
185
186     printf("Archivos: %s\n", archivos[i]);
187     medi_tiempo_ejecucion(quicksort_medio, arr, n, "Quicksort (Pivote: Valor en la posición del medio)");
188     medi_comparaciones_intercambios(quicksort_medio, arr, n, "Quicksort (Pivote: Valor en la posición del medio)");
189
190     // Guardar el arreglo ordenado en un nuevo archivo
191     char archivo_ordenado[100];
192     sprintf(archivo_ordenado, sizeof(archivo_ordenado), "quicksort_medio_Xs", archivos[i]);
193     guardar_arreglo(arr, n, archivo_ordenado);
194
195     free(arr);
196 }
197
198 // Algoritmo QuickSort (Pivote: Aleatorio)
199 printf("Algoritmo QuickSort (Pivote: Aleatorio):\n");
200 for (int i = 0; i < num_archivos; ++i) {
201     int n = tamanios[i];
202     int *arr = malloc(n * sizeof(int));
203
204     cargar_arreglo(arr, n, archivos[i]);
205
206     printf("Archivos: %s\n", archivos[i]);
207     medi_tiempo_ejecucion(quicksort_aleatorio, arr, n, "Quicksort (Pivote: Aleatorio)");
208     medi_comparaciones_intercambios(quicksort_aleatorio, arr, n, "Quicksort (Pivote: Aleatorio)");
209
210     // Guardar el arreglo ordenado en un nuevo archivo
211     char archivo_ordenado[100];
212     sprintf(archivo_ordenado, sizeof(archivo_ordenado), "quicksort_aleatorio_Xs", archivos[i]);
213     guardar_arreglo(arr, n, archivo_ordenado);
214
215     free(arr);
216 }
217
218 // Algoritmo QuickSort (Pivote: Medio)
219 printf("Algoritmo QuickSort (Pivote: Medio):\n");
220 for (int i = 0; i < num_archivos; ++i) {
221     int n = tamanios[i];
222     int *arr = malloc(n * sizeof(int));
223
224     cargar_arreglo(arr, n, archivos[i]);
225
226     printf("Archivos: %s\n", archivos[i]);
227     medi_tiempo_ejecucion(quicksort_medio, arr, n, "Quicksort (Pivote: Medio)");
228     medi_comparaciones_intercambios(quicksort_medio, arr, n, "Quicksort (Pivote: Medio)");
229
230     // Guardar el arreglo ordenado en un nuevo archivo
231     char archivo_ordenado[100];
232     sprintf(archivo_ordenado, sizeof(archivo_ordenado), "quicksort_medio_Xs", archivos[i]);
233     guardar_arreglo(arr, n, archivo_ordenado);
234
235     free(arr);
236 }
```

Ilustración 30: Código Fuente Ejercicio Opcional Parte 5

```
File Edit Selection View Go Run Terminal Help EjercicioOptional - EjercicioOptional - Visual Studio Code

EXPLORER C EjercicioOptional.c x C Ejercicio5v2.h
OPEN EDITORS C EjercicioOptional.c @ mostrar_arreglo(int[], int)
C Ejercicio5v2.h
164     free(arr);
165 }
166
167 // Algoritmo Quicksort (Pivote: Mediana utilizando el primer, ultimo y medio elemento)
168 printf("Algoritmo Quicksort (Pivote: Mediana utilizando el primer, ultimo y medio elemento):\n");
169 for (int i = 0; i < num_archivos; i++) {
170     int n = tamanios[i];
171     int *arr = malloc(n * sizeof(int));
172
173     cargar_arreglo(arr, n, archivos[i]);
174
175     printf("Archivo: %s\n", archivos[i]);
176     medir_tiempo_ejecucion(quicksort_mediana, arr, n, "Quicksort (Pivote: Mediana utilizando el primer, ultimo y medio elemento)");
177     medir_comparaciones_intercambios(quicksort_mediana, arr, n, "Quicksort (Pivote: Mediana utilizando el primer, ultimo y medio elemento)");
178
179     // Guardar el arreglo ordenado en un nuevo archivo
180     char archivo_ordenado[100];
181     sprintf(archivo_ordenado, sizeof(archivo_ordenado), "quicksort_mediana_%s", archivos[i]);
182     guardar_arreglo(arr, n, archivo_ordenado);
183
184     free(arr);
185 }
186
187 // Algoritmo de Ordenación Bucket
188 printf("Algoritmo de ordenación Bucket:\n");
189 for (int i = 0; i < num_archivos; i++) {
190     int n = tamanios[i];
191     int *arr = malloc(n * sizeof(int));
192
193     cargar_arreglo(arr, n, archivos[i]);
194
195     printf("Archivo: %s\n", archivos[i]);
196     medir_tiempo_ejecucion(ordenacion_bucket, arr, n, "Ordenación Bucket");
197     medir_comparaciones_intercambios(ordenacion_bucket, arr, n, "Ordenación Bucket");
198
199     // Guardar el arreglo ordenado en un nuevo archivo
200     char archivo_ordenado[100];
```

Ilustración 31: Código Fuente Ejercicio Opcional Parte 6

```
EjercicioOpcional - EjercicioOpcional - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EjercicioOpcional.c x Ejercicio5v2.h
EjercicioOpcional.c x mostrar_arreglo([int])
EjercicioOpcional.c x medir_comparaciones_intercambios(ordenacion_bucket, arr, n, "Ordenación Bucket");
EjercicioOpcional.c x
EjercicioOpcional.c x     medir_comparaciones_intercambios(ordenacion_bucket, arr, n, "Ordenación Bucket");
EjercicioOpcional.c x
EjercicioOpcional.c x     // Guardar el arreglo ordenado en un nuevo archivo
EjercicioOpcional.c x     char archivo_ordenado[100];
EjercicioOpcional.c x     sprintf(archivo_ordenado, sizeof(archivo_ordenado), "bucket_%s", archivos[i]);
EjercicioOpcional.c x     guardar_arreglo(arr, n, archivo_ordenado);
EjercicioOpcional.c x
EjercicioOpcional.c x     free(arr);
EjercicioOpcional.c x }

EjercicioOpcional.c x
EjercicioOpcional.c x     // Algoritmo de Mezcla Directa o Natural
EjercicioOpcional.c x     printf("Algoritmo de Mezcla Directa o Natural:\n");
EjercicioOpcional.c x     int n = tamanios[2]; // Tamano del archivo numeros200000.txt
EjercicioOpcional.c x     int *arr = malloc(n * sizeof(int));
EjercicioOpcional.c x     cargar_arreglo(arr, n, archivos[2]); // Solo se utiliza el archivo numeros200000.txt
EjercicioOpcional.c x
EjercicioOpcional.c x     printf("Archivo: %s\n", archivos[2]);
EjercicioOpcional.c x     medir_tiempo_ejecucion(mezcla_directa, arr, n, "Mezcla Directa o Natural");
EjercicioOpcional.c x     medir_comparaciones_intercambios(mezcla_directa, arr, n, "Mezcla Directa o Natural");
EjercicioOpcional.c x
EjercicioOpcional.c x     // Guardar el arreglo ordenado en un nuevo archivo
EjercicioOpcional.c x     char archivo_ordenado[100];
EjercicioOpcional.c x     sprintf(archivo_ordenado, sizeof(archivo_ordenado), "%mezcla_directa_%s", archivos[2]); guardar_arreglo(arr,n,archivo_ordenado);
EjercicioOpcional.c x     free(arr);
EjercicioOpcional.c x
EjercicioOpcional.c x return 0;
EjercicioOpcional.c x }
```

Ilustración 32: Código Fuente Ejercicio Opcional Parte 7

```

File Edit Selection View Go Run Terminal Help Ejercicio5v2.h - EjercicioOpcional - Visual Studio Code
OPEN EDITORS EjercicioOpcional.c Ejercicio5v2.h
Ejercicio5v2.h > quicksort_medio(int[], int, int)
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <time.h>
5
6 void cargar_arreglo(int arr[], int n, const char *archivo) {
7     FILE *f = fopen(archivo, "r");
8     if (f == NULL) {
9         printf("Error al abrir el archivo.\n");
10        return;
11    }
12
13    for (int i = 0; i < n; i++) {
14        fscanf(f, "%d", &arr[i]);
15    }
16
17    fclose(f);
18}
19
20 void guardar_arreglo(int arr[], int n, const char *archivo) {
21     FILE *f = fopen(archivo, "w");
22     if (f == NULL) {
23         printf("Error al abrir el archivo.\n");
24         return;
25     }
26
27     for (int i = 0; i < n; i++) {
28         fprintf(f, "%d\n", arr[i]);
29     }
30
31     fclose(f);
32}
33
34 void imprimir_arreglo(int arr[], int n) {
35     for (int i = 0; i < n; i++) {
36         printf("%d ", arr[i]);
37     }
38     printf("\n");
39}

```

Ilustración 33: Código Fuente Ejercicio Opcional Parte 8

```

File Edit Selection View Go Run Terminal Help Ejercicio5v2.h - EjercicioOpcional - Visual Studio Code
OPEN EDITORS EjercicioOpcional.c Ejercicio5v2.h
Ejercicio5v2.h > quicksort_medio(int[], int, int)
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74

```

Ilustración 34: Código Fuente Ejercicio Opcional Parte 9

```
File Edit Selection View Go Run Terminal Help Ejercicio5v2.h - EjercicioOpcional - Visual Studio Code

OPEN EDITORS
EXPLORER
C EjercicioOpcional.c C Ejercicio5v2.h x
C Ejercicio5v2.h > quicksort_medic(int[], int, int)
74 }
75 }
76 }
77 }
78 // Algoritmo de Selección Directa
79 void seleccion_directa(int arr[], int inicio, int fin) {
80     int i, j, min_idx;
81     for (i = inicio; i <= fin - 1; i++) {
82         min_idx = i;
83         for (j = i + 1; j <= fin; j++) {
84             if (arr[j] < arr[min_idx]) {
85                 min_idx = j;
86             }
87         }
88         intercambiar(&arr[min_idx], &arr[i]);
89     }
90 }
91 // Algoritmo Shell
92 void shell_sort(int arr[], int inicio, int fin) {
93     int n = fin - inicio + 1;
94     int brecha, i, j, temp;
95     for (brecha = n / 2; brecha > 0; brecha /= 2) {
96         for (i = brecha; i < n; i++) {
97             temp = arr[i];
98             for (j = i - brecha; j >= 0 && arr[j + brecha] > temp; j -= brecha) {
99                 arr[j + brecha] = arr[j];
100            }
101            arr[j] = temp;
102        }
103    }
104 }
105 }
106 }
107 // Función para obtener el pivote en QuickSort
108 int obtener_pivote(int arr[], int inicio, int fin) {
109     return arr[(inicio + fin) / 2];
110 }
```

Ilustración 35: Código Fuente Ejercicio Opcional Parte 10

Ilustración 36: Código Fuente Ejercicio Opcional Parte 11

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like 'EjercicioOpcional.c', 'Ejercicio5v2.h', and various sorting functions.
- Open Editors:** Shows the 'Ejercicio5v2.h' file open, containing C code for a quicksort algorithm and a median-of-three function.
- Code Editor:** Displays the 'Ejercicio5v2.h' file content, which includes:
 - A quicksort implementation using a pivot selection logic.
 - A median-of-three function to find the median of three elements.
- Status Bar:** Shows 'Ln 113, Col 6' and other standard status bar information.
- Taskbar:** Shows the Windows taskbar with various pinned icons.

Ilustración 37: Código Fuente Ejercicio Opcional Parte 12

```
C Ejercicio5v2h.c
C Ejercicio5v2h.h

C Ejercicio5v2h > quicksort_medio(int[], int, int)
189     return a;
190 } else if (c <= a && a <= b) {
191     return a;
192 } else {
193     return c;
194 }
195 }

// Algoritmo QuickSort (Pivote: Mediana utilizando el primer, ultimo y medio elemento)
void quicksort_mediana(int arr[], int inicio, int fin) {
196     if (inicio < fin) {
197         int primer_elemento = arr[inicio];
198         int ultimo_elemento = arr[fin];
199         int medio_idx = (inicio + fin) / 2;
200         int medio_elemento = arr[medio_idx];
201         int pivote = obtener_mediana(primer_elemento, ultimo_elemento, medio_elemento);
202         int i = inicio;
203         int j = fin;
204
205         while (i <= j) {
206             while (arr[i] < pivote) {
207                 i++;
208             }
209             while (arr[j] > pivote) {
210                 j--;
211             }
212             if (i <= j) {
213                 intercambiar(&arr[i], &arr[j]);
214                 i++;
215                 j--;
216             }
217         }
218         if (inicio < j) {
219             quicksort_mediana(arr, inicio, j);
220         }
221         if (i < fin) {
222             quicksort_mediana(arr, i, fin);
223         }
224     }
225 }
```

Ilustración 38: Código Fuente Ejercicio Opcional Parte 13

Ilustración 39: Código Fuente Ejercicio Opcional Parte 14

```
File Edit Selection View Go Run Terminal Help Ejercicio5v2.h - EjercicioOpcional - Visual Studio Code

EXPLORER ... C EjercicioOpcional.c C Ejercicio5v2.h
OPEN EDITORS C Ejercicio5v2.h ✘ quicksort_medio(int l, int r) {
C Ejercicio5v2.h
253     int* bucket = buckets[1];
254     quicksort_medio(bucket, 0, bucket_size - 1);
255 }
256 }

// Concatenar los elementos de los buckets en el arreglo original
257 int index = 0;
258 for (int i = 0; i < num_buckets; i++) {
259     int bucket_size = bucket_sizes[i];
260     if (bucket_size > 0) {
261         int* bucket = buckets[i];
262         for (int j = 0; j < bucket_size; j++) {
263             arr[index] = bucket[j];
264             index++;
265         }
266     }
267 }
268 free(buckets);
269 }

mezcla_directa_num... 270
numeros50000.txt 271
numeros100000.txt 272
numeros200000.txt 273
quicksort_aleatorio_n... 274
quicksort_aleatorio_n... 275
quicksort_aleatorio_n... 276
quicksort_aleatorio_n... 277
quicksort_aleatorio_n... 278
quicksort_aleatorio_n... 279
quicksort_mediana_n... 280
quicksort_mediana_n... 281
quicksort_mediana_n... 282
quicksort_medio_n... 283
quicksort_medio_n... 284
quicksort_medio_n... 285
quicksort_medio_n... 286
QuickSort-Medio.txt 287
ordenacion_burbuja... 288
OUTLINE 289
TIMELINE 290

// Algoritmo de Mezcla Directa o Natural
void mezcla_directa(int arr[], int inicio, int fin) {
    if (inicio < fin) {
        int medio = inicio + (fin - inicio) / 2;
        // Llamada recursiva para dividir la mitad izquierda
        mezcla_directa(arr, inicio, medio);

        // Llamada recursiva para dividir la mitad derecha
        mezcla_directa(arr, medio + 1, fin);

        // Mezcla de las dos mitades ordenadas
        int i, j, k;
        int nt = medio - inicio + 1:
```

Ilustración 40: Código Fuente Ejercicio Opcional Parte 15

```

File Edit Selection View Go Run Terminal Help
Ejercicio5v2.h - EjercicioOpcional - Visual Studio Code
OPEN EDITORS
  C EjercicioOpcional.c
  C Ejercicio5v2.h
  C Ejercicio5v2.h > quicksort_medio(int[], int, int)
    ...
    int n1 = medio - inicio + 1;
    int n2 = fin - medio;

    // Crear arreglos temporales para almacenar las mitades
    int *izquierda = malloc(n1 * sizeof(int));
    int *derecha = malloc(n2 * sizeof(int));

    // Copiar datos a los arreglos temporales
    for (i = 0; i < n1; i++) {
        izquierda[i] = arr[inicio + i];
    }
    for (j = 0; j < n2; j++) {
        derecha[j] = arr[medio + 1 + j];
    }

    // Mezclar los arreglos temporales en el arreglo original
    i = 0; // Indice inicial del subarreglo izquierdo
    j = 0; // Indice inicial del subarreglo derecho
    k = inicio; // Indice inicial del subarreglo mezclado
    while (i < n1 && j < n2) {
        if (izquierda[i] <= derecha[j]) {
            arr[k] = izquierda[i];
            i++;
            k++;
        } else {
            arr[k] = derecha[j];
            j++;
            k++;
        }
    }

    // Copiar los elementos restantes del subarreglo izquierdo
    while (i < n1) {
        arr[k] = izquierda[i];
        i++;
        k++;
    }

    // Copiar los elementos restantes del subarreglo derecho
    while (j < n2) {
        arr[k] = derecha[j];
        j++;
        k++;
    }

    free(izquierda);
    free(derecha);

Ln 113, Col 6  Spaces: 4  UTF-8  CRLF  ( ) C  Win32  1647  24/05/2023

```

Ilustración 41: Código Fuente Ejercicio Opcional Parte 16

```

File Edit Selection View Go Run Terminal Help
Ejercicio5v2.h - EjercicioOpcional - Visual Studio Code
OPEN EDITORS
  C EjercicioOpcional.c
  C Ejercicio5v2.h
  C Ejercicio5v2.h > quicksort_medio(int[], int, int)
    ...
    free(izquierda);
    free(derecha);

Ln 113, Col 6  Spaces: 4  UTF-8  CRLF  ( ) C  Win32  1648  24/05/2023

```

Ilustración 42: Código Fuente Ejercicio Opcional Parte 17

8. Ejercicio Opcional Compilado Y Ejecución.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like 'Ejercicio5v2.h', 'Ejercicio5v2.c', 'numeros50000.txt', and 'numeros00000.txt'.
- Open Editors:** Displays two files: 'EjercicioOpcional.c' and 'Ejercicio5v2.h'.
- Problems Tab:** Shows multiple warning messages from the compiler (GCC) regarding incompatible pointer types. For example:
 - 'EjercicioOpcional.c:196:36: warning: passing argument 1 of 'medir_tiempo_ejecucion' from incompatible pointer type [-Wincompatible-pointer-types]
 - 'EjercicioOpcional.c:17:36: note: expected 'void (*)(int *, int)' but argument is of type 'void (*)(int *, int)'
 - 'EjercicioOpcional.c:197:46: warning: passing argument 1 of 'medir_comparaciones_intercambios' from incompatible pointer type [-Wincompatible-pointer-types]
 - 'EjercicioOpcional.c:31:46: note: expected 'void (*)(int *, int, int)' but argument is of type 'void (*)(int *, int)'
- Code Editor:** Shows the C code for 'Ejercicio5v2.h' and 'EjercicioOpcional.c' with the compiler's underlined error highlights.
- Status Bar:** Shows the current file is 'EjercicioOpcional.c', line 113, column 6, with 4 spaces, using UTF-8 encoding, and the window is titled 'Ejercicio5v2 - EjercicioOpcional - Visual Studio Code'.

Ilustración 43: Ejercicio Opcional Compilación y Ejecución Parte 1

Ilustración 44: Ejercicio Opcional Compilación y Ejecución Parte 2

```

Ejercicio5v2.h - EjercicioOpcional - Visual Studio Code
File Edit Selection View Go Run Terminal Help
OPEN EDITORS PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
INTERCAMBIOS REALIZADOS (SELECCIÓN DIRECTA): 9
ALGORITMO SHELL:
TIEMPO DE EJECUCIÓN (SHELL): 0.005000 SEGUNDOS
INTERCAMBIOS REALIZADOS (SHELL): 10
ARCHIVO: numeros100000.txt
ALGORITMO SHELL:
TIEMPO DE EJECUCIÓN (SHELL): 0.024000 SEGUNDOS
INTERCAMBIOS REALIZADOS (SHELL): 11
COMPARACIONES REALIZADAS (SHELL): 11
ALGORITMO SHELL:
TIEMPO DE EJECUCIÓN (SHELL): 0.061000 SEGUNDOS
INTERCAMBIOS REALIZADOS (SHELL): 12
COMPARACIONES REALIZADAS (SHELL): 12
INTERCAMBIOS REALIZADOS (SHELL): 12
ALGORITMO QUICKSORT (PIVOTE: VALOR EN LA POSICIÓN N DEL MEDIO):
ARCHIVO: numeros50000.txt
TIEMPO DE EJECUCIÓN (QUICKSORT (PIVOTE: VALOR EN LA POSICIÓN N DEL MEDIO)): 0.003000 SEGUNDOS
COMPARACIONES REALIZADAS (QUICKSORT (PIVOTE: VALOR EN LA POSICIÓN N DEL MEDIO)): 13
INTERCAMBIOS REALIZADOS (QUICKSORT (PIVOTE: VALOR EN LA POSICIÓN N DEL MEDIO)): 13
ARCHIVO: numeros100000.txt
TIEMPO DE EJECUCIÓN (QUICKSORT (PIVOTE: VALOR EN LA POSICIÓN N DEL MEDIO)): 0.011000 SEGUNDOS
COMPARACIONES REALIZADAS (QUICKSORT (PIVOTE: VALOR EN LA POSICIÓN N DEL MEDIO)): 14
INTERCAMBIOS REALIZADOS (QUICKSORT (PIVOTE: VALOR EN LA POSICIÓN N DEL MEDIO)): 14
ARCHIVO: numeros50000.txt
TIEMPO DE EJECUCIÓN (QUICKSORT (PIVOTE: VALOR EN LA POSICIÓN N DEL MEDIO)): 0.023000 SEGUNDOS
COMPARACIONES REALIZADAS (QUICKSORT (PIVOTE: VALOR EN LA POSICIÓN N DEL MEDIO)): 15
INTERCAMBIOS REALIZADOS (QUICKSORT (PIVOTE: VALOR EN LA POSICIÓN N DEL MEDIO)): 15
ALGORITMO QUICKSORT (PIVOTE: ALEATORIO):
ARCHIVO: numeros50000.txt
TIEMPO DE EJECUCIÓN (QUICKSORT (PIVOTE: ALEATORIO)): 0.004000 SEGUNDOS
COMPARACIONES REALIZADAS (QUICKSORT (PIVOTE: ALEATORIO)): 16
INTERCAMBIOS REALIZADOS (QUICKSORT (PIVOTE: ALEATORIO)): 16
ARCHIVO: numeros100000.txt
TIEMPO DE EJECUCIÓN (QUICKSORT (PIVOTE: ALEATORIO)): 0.016000 SEGUNDOS
COMPARACIONES REALIZADAS (QUICKSORT (PIVOTE: ALEATORIO)): 17
INTERCAMBIOS REALIZADOS (QUICKSORT (PIVOTE: ALEATORIO)): 17
ARCHIVO: numeros200000.txt
TIEMPO DE EJECUCIÓN (QUICKSORT (PIVOTE: ALEATORIO)): 0.055000 SEGUNDOS
COMPARACIONES REALIZADAS (QUICKSORT (PIVOTE: ALEATORIO)): 18
INTERCAMBIOS REALIZADOS (QUICKSORT (PIVOTE: ALEATORIO)): 18
ALGORITMO QUICKSORT (PIVOTE: MEDIANA UTILIZANDO EL PRIMER, |||ÚLTIMO Y MEDIO ELEMENTO):
ARCHIVO: numeros50000.txt
TIEMPO DE EJECUCIÓN (QUICKSORT (PIVOTE: MEDIANA UTILIZANDO EL PRIMER, |||ÚLTIMO Y MEDIO ELEMENTO)): 0.003000 SEGUNDOS
COMPARACIONES REALIZADAS (QUICKSORT (PIVOTE: MEDIANA UTILIZANDO EL PRIMER, |||ÚLTIMO Y MEDIO ELEMENTO)): 19
INTERCAMBIOS REALIZADOS (QUICKSORT (PIVOTE: MEDIANA UTILIZANDO EL PRIMER, |||ÚLTIMO Y MEDIO ELEMENTO)): 19

```

Ilustración 45: Ejercicio Opcional Compilación y Ejecución Parte 3

```

Ejercicio5v2.h - EjercicioOpcional - Visual Studio Code
File Edit Selection View Go Run Terminal Help
OPEN EDITORS PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
INTERCAMBIOS REALIZADOS (QUICKSORT (PIVOTE: ALEATORIO)): 17
ARCHIVO: numeros200000.txt
TIEMPO DE EJECUCIÓN (QUICKSORT (PIVOTE: ALEATORIO)): 0.055000 SEGUNDOS
COMPARACIONES REALIZADAS (QUICKSORT (PIVOTE: ALEATORIO)): 18
INTERCAMBIOS REALIZADOS (QUICKSORT (PIVOTE: ALEATORIO)): 18
ALGORITMO QUICKSORT (PIVOTE: MEDIANA UTILIZANDO EL PRIMER, |||ÚLTIMO Y MEDIO ELEMENTO):
ARCHIVO: numeros50000.txt
TIEMPO DE EJECUCIÓN (QUICKSORT (PIVOTE: MEDIANA UTILIZANDO EL PRIMER, |||ÚLTIMO Y MEDIO ELEMENTO)): 0.003000 SEGUNDOS
COMPARACIONES REALIZADAS (QUICKSORT (PIVOTE: MEDIANA UTILIZANDO EL PRIMER, |||ÚLTIMO Y MEDIO ELEMENTO)): 19
INTERCAMBIOS REALIZADOS (QUICKSORT (PIVOTE: MEDIANA UTILIZANDO EL PRIMER, |||ÚLTIMO Y MEDIO ELEMENTO)): 19
ARCHIVO: numeros100000.txt
TIEMPO DE EJECUCIÓN (QUICKSORT (PIVOTE: MEDIANA UTILIZANDO EL PRIMER, |||ÚLTIMO Y MEDIO ELEMENTO)): 0.016000 SEGUNDOS
COMPARACIONES REALIZADAS (QUICKSORT (PIVOTE: MEDIANA UTILIZANDO EL PRIMER, |||ÚLTIMO Y MEDIO ELEMENTO)): 20
INTERCAMBIOS REALIZADOS (QUICKSORT (PIVOTE: MEDIANA UTILIZANDO EL PRIMER, |||ÚLTIMO Y MEDIO ELEMENTO)): 20
ARCHIVO: numeros200000.txt
TIEMPO DE EJECUCIÓN (QUICKSORT (PIVOTE: MEDIANA UTILIZANDO EL PRIMER, |||ÚLTIMO Y MEDIO ELEMENTO)): 0.051000 SEGUNDOS
COMPARACIONES REALIZADAS (QUICKSORT (PIVOTE: MEDIANA UTILIZANDO EL PRIMER, |||ÚLTIMO Y MEDIO ELEMENTO)): 21
INTERCAMBIOS REALIZADOS (QUICKSORT (PIVOTE: MEDIANA UTILIZANDO EL PRIMER, |||ÚLTIMO Y MEDIO ELEMENTO)): 21
ALGORITMO DE ORDENACIÓN BUCKET:
ARCHIVO: numeros50000.txt
TIEMPO DE EJECUCIÓN (ORDENACIÓN BUCKET): 0.000000 SEGUNDOS
COMPARACIONES REALIZADAS (ORDENACIÓN BUCKET): 22
INTERCAMBIOS REALIZADOS (ORDENACIÓN BUCKET): 22
ARCHIVO: numeros100000.txt
TIEMPO DE EJECUCIÓN (ORDENACIÓN BUCKET): 0.000000 SEGUNDOS
COMPARACIONES REALIZADAS (ORDENACIÓN BUCKET): 23
INTERCAMBIOS REALIZADOS (ORDENACIÓN BUCKET): 23
ARCHIVO: numeros200000.txt
TIEMPO DE EJECUCIÓN (ORDENACIÓN BUCKET): 0.000000 SEGUNDOS
COMPARACIONES REALIZADAS (ORDENACIÓN BUCKET): 24
INTERCAMBIOS REALIZADOS (ORDENACIÓN BUCKET): 24
ALGORITMO MEZCLA DIRECTA O NATURAL:
ARCHIVO: numeros200000.txt
TIEMPO DE EJECUCIÓN (MEZCLA DIRECTA O NATURAL): 0.070000 SEGUNDOS
COMPARACIONES REALIZADAS (MEZCLA DIRECTA O NATURAL): 25
INTERCAMBIOS REALIZADOS (MEZCLA DIRECTA O NATURAL): 25
PS C:\Users\david\Documents\Cursos\Grado Ingeniería Informática\UCAM\Master_Experto_Programación_Avanzada\Algoritmia\Prácticas\Tema 3\Proyecto\EjercicioOpcional>

```

Ilustración 46: Ejercicio Opcional Compilación y Ejecución Parte 4

9. Aclaraciones y comentarios.

En cada apartado he puesto los códigos fuentes de cada ejercicio con su explicación requerida y también realice el ejercicio opcional que era el algoritmo de Bucket para los tres archivos y el algoritmo de mezcla directa o natural en el fichero de numeros200000. En los ejercicios 5 y el opcional he realizado tanto la ejecución y su compilación como su código fuente.