Nombre: Gabriel Roberto Revelo Rodrigues **Fecha:** 05/06/2023

Asignatura: Estructura de Datos **NRC:** 9898

Tarea: Algoritmos de Ordenación Interna

Los métodos de clasificación interna son algoritmos empleados para organizar una serie de elementos dentro de la memoria principal de una computadora.

1. Intercambio

El algoritmo de intercambio, es una técnica que compara elementos consecutivos de una lista y los intercambia si están en el orden incorrecto. Este proceso se repite hasta que todos los elementos estén ordenados. Aunque es un método sencillo, no es eficiente cuando se trata de conjuntos de datos grandes.

C++:

TareaSegundoParcial > C Intercambio.c

```cpp
#include <iostream>
using namespace std;

void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                swap(arr[j], arr[j+1]);
            }
        }
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    cout << "Array ordenado: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;
}

```

Python:

TareaSegundoParcial > Intercambio.py > ...

```python
def bubbleSort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

arr = [64, 34, 25, 12, 22, 11, 90]
bubbleSort(arr)
print("Array ordenado:", arr)

```

Java:

TareaSegundoParcial > Intercambio.java

```java
class BubbleSort {
    void bubbleSort(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n-1; i++) {
            for (int j = 0; j < n-i-1; j++) {
                if (arr[j] > arr[j+1]) {
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }
    }

    public static void main(String args[]) {
        BubbleSort bubble = new BubbleSort();
        int arr[] = {64, 34, 25, 12, 22, 11, 90};
        bubble.bubbleSort(arr);
        System.out.println("Array ordenado:");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}

```

2. Quicksort

Quicksort es un algoritmo que emplea la técnica "divide y vencerás" para ordenar una lista. En cada paso, selecciona un elemento llamado pivote y reorganiza la lista de tal manera que los elementos más pequeños al pivote se ubiquen antes que él, mientras que los mayores se coloquen después. Este proceso se aplica de forma recursiva a las sublistas resultantes hasta que toda la lista esté ordenada.

C++:

TareaSegundoParcial > C quicksort.c

```c
#include <iostream>
using namespace std;

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1);
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    cout << "Array ordenado: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Python:

TareaSegundoParcial > quicksort.py

```python
def partition(arr, low, high):
    i = (low - 1)
    pivot = arr[high]
    for j in range(low, high):
        if arr[j] < pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return (i + 1)

def quickSort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        quickSort(arr, low, pi - 1)
        quickSort(arr, pi + 1, high)

arr = [64, 34, 25, 12, 22, 11, 90]
n = len(arr)
quickSort(arr, 0, n - 1)
print("Array ordenado:", arr)
```

Java:

TareaSegundoParcial > quicksort.java

```java
class QuickSort {
    int partition(int arr[], int low, int high) {
        int pivot = arr[high];
        int i = (low - 1);
        for (int j = low; j <= high - 1; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return (i + 1);
    }

    void quickSort(int arr[], int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

    public static void main(String args[]) {
        QuickSort quick = new QuickSort();
        int arr[] = {64, 34, 25, 12, 22, 11, 90};
        int n = arr.length;
        quick.quickSort(arr, 0, n - 1);
        System.out.println("Array ordenado:");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

3. ShellSort

ShellSort es una optimización del algoritmo de inserción directa que divide la lista en subgrupos más pequeños y los ordena individualmente. Luego, disminuye gradualmente la brecha entre los elementos de los subgrupos y repite este proceso hasta que la lista esté completamente ordenada.

C++:

TareaSegundoParcial > C shellsort.c

```cpp
#include <iostream>
using namespace std;

void shellSort(int arr[], int n) {
    for (int gap = n/2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    shellSort(arr, n);
    cout << "Array ordenado: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Python:

TareaSegundoParcial > shellsort.py > ...

```python
def shellSort(arr):
    n = len(arr)
    gap = n // 2
    while gap > 0:
        for i in range(gap, n):
            temp = arr[i]
            j = i
            while j >= gap and arr[j - gap] > temp:
                arr[j] = arr[j - gap]
                j -= gap
            arr[j] = temp
        gap //= 2

arr = [64, 34, 25, 12, 22, 11, 90]
shellSort(arr)
print("Array ordenado:", arr)
```

Java:

TareaSegundoParcial > shellsort.java

```java
class ShellSort {
    void shellSort(int arr[]) {
        int n = arr.length;
        for (int gap = n / 2; gap > 0; gap /= 2) {
            for (int i = gap; i < n; i++) {
                int temp = arr[i];
                int j;
                for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {
                    arr[j] = arr[j - gap];
                }
                arr[j] = temp;
            }
        }
    }

    public static void main(String args[]) {
        ShellSort shell = new ShellSort();
        int arr[] = {64, 34, 25, 12, 22, 11, 90};
        shell.shellSort(arr);
        System.out.println("Array ordenado:");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}

```

4. Ordenación por Distribución

La técnica de ordenación por distribución, también conocida como Counting Sort, organiza los elementos de una lista en función de sus valores y determina cuántos elementos tienen valores más pequeños que ellos. A continuación, utiliza esta información para ubicar cada elemento en la posición correcta dentro de la lista ordenada.

C++:

TareaSegundoParcial > C OD.c

```c
#include <iostream>
using namespace std;

void countingSort(int arr[], int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    int count[max + 1];
    int output[n];
    for (int i = 0; i <= max; i++) {
        count[i] = 0;
    }
    for (int i = 0; i < n; i++) {
        count[arr[i]]++;
    }
    for (int i = 1; i <= max; i++) {
        count[i] += count[i - 1];
    }
    for (int i = n - 1; i >= 0; i--) {
        output[count[arr[i]] - 1] = arr[i];
        count[arr[i]]--;
    }
    for (int i = 0; i < n; i++) {
        arr[i] = output[i];
    }
}

int main() {
    int arr[] = {4, 2, 2, 8, 3, 3, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    countingSort(arr, n);
    cout << "Array ordenado: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Python:

TareaSegundoParcial > OD.py > ...

```python
def countingSort(arr):
    max_val = max(arr)
    count = [0] * (max_val + 1)
    output = [0] * len(arr)
    for num in arr:
        count[num] += 1
    for i in range(1, max_val + 1):
        count[i] += count[i - 1]
    for num in arr:
        output[count[num] - 1] = num
        count[num] -= 1
    return output

arr = [4, 2, 2, 8, 3, 3, 1]
sorted_arr = countingSort(arr)
print("Array ordenado:", sorted_arr)
```

Java:

TareaSegundoParcial > OD.java

```java
class CountingSort {
    void countingSort(int arr[]) {
        int n = arr.length;
        int max = arr[0];
        for (int i = 1; i < n; i++) {
            if (arr[i] > max) {
                max = arr[i];
            }
        }
        int count[] = new int[max + 1];
        int output[] = new int[n];
        for (int i = 0; i <= max; i++) {
            count[i] = 0;
        }
        for (int i = 0; i < n; i++) {
            count[arr[i]]++;
        }
        for (int i = 1; i <= max; i++) {
            count[i] += count[i - 1];
        }
        for (int i = n - 1; i >= 0; i--) {
            output[count[arr[i]] - 1] = arr[i];
            count[arr[i]]--;
        }
        for (int i = 0; i < n; i++) {
            arr[i] = output[i];
        }
    }

    public static void main(String args[]) {
        CountingSort counting = new CountingSort();
        int arr[] = {4, 2, 2, 8, 3, 3, 1};
        counting.countingSort(arr);
        System.out.println("Array ordenado:");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

5. Ordenación por Radix

La ordenación por radix es un método de clasificación que no se basa en comparaciones directas entre elementos. En su lugar, ordena los elementos según sus dígitos, de forma similar a cómo se organizan los números en una agenda telefónica o en una biblioteca. Este algoritmo asegura una clasificación estable y puede basarse en el principio de ordenar primero los dígitos menos significativos (LSD) o los dígitos más significativos (MSD) para lograr el ordenamiento deseado.

C++:

TareaSegundoParcial > C OR.c

```cpp
#include <iostream>
using namespace std;

int getMax(int arr[], int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}

void countSort(int arr[], int n, int exp) {
    int output[n];
    int count[10] = {0};
    for (int i = 0; i < n; i++) {
        count[(arr[i] / exp) % 10]++;
    }
    for (int i = 1; i < 10; i++) {
        count[i] += count[i - 1];
    }
    for (int i = n - 1; i >= 0; i--) {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }
    for (int i = 0; i < n; i++) {
        arr[i] = output[i];
    }
}

void radixSort(int arr[], int n) {
    int max = getMax(arr, n);
    for (int exp = 1; max / exp > 0; exp *= 10) {
        countSort(arr, n, exp);
    }
}

int main() {
    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
    int n = sizeof(arr)/sizeof(arr[0]);
    radixSort(arr, n);
    cout << "Array ordenado: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Python:

TareaSegundoParcial > OR.py > ...

```python
def countingSort(arr, exp):
    n = len(arr)
    output = [0] * n
    count = [0] * 10
    for i in range(n):
        index = arr[i] // exp
        count[index % 10] += 1
    for i in range(1, 10):
        count[i] += count[i - 1]
    i = n - 1
    while i >= 0:
        index = arr[i] // exp
        output[count[index % 10] - 1] = arr[i]
        count[index % 10] -= 1
        i -= 1
    for i in range(n):
        arr[i] = output[i]

def radixSort(arr):
    max_val = max(arr)
    exp = 1
    while max_val // exp > 0:
        countingSort(arr, exp)
        exp *= 10

arr = [170, 45, 75, 90, 802, 24, 2, 66]
radixSort(arr)
print("Array ordenado:", arr)
```

Java:

TareaSegundoParcial > OR.java

```java
class RadixSort {
    int getMax(int arr[], int n) {
        int max = arr[0];
        for (int i = 1; i < n; i++) {
            if (arr[i] > max) {
                max = arr[i];
            }
        }
        return max;
    }

    void countSort(int arr[], int n, int exp) {
        int output[] = new int[n];
        int count[] = new int[10];
        for (int i = 0; i < n; i++) {
            count[(arr[i] / exp) % 10]++;
        }
        for (int i = 1; i < 10; i++) {
            count[i] += count[i - 1];
        }
        for (int i = n - 1; i >= 0; i--) {
            output[count[(arr[i] / exp) % 10] - 1] = arr[i];
            count[(arr[i] / exp) % 10]--;
        }
        for (int i = 0; i < n; i++) {
            arr[i] = output[i];
        }
    }

    void radixSort(int arr[]) {
        int n = arr.length;
        int max = getMax(arr, n);
        for (int exp = 1; max / exp > 0; exp *= 10) {
            countSort(arr, n, exp);
        }
    }

    public static void main(String args[]) {
        RadixSort radix = new RadixSort();
        int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
        radix.radixSort(arr);
        System.out.println("Array ordenado:");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

6. Burbuja

El algoritmo de ordenación por burbuja compara elementos adyacentes en pares y realiza intercambios si se encuentran en un orden incorrecto. Este proceso se repite de manera iterativa hasta que la lista esté completamente ordenada.

C++:

TareaSegundoParcial > C burbuja.c

```cpp
#include <iostream>
using namespace std;

void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    cout << "Array ordenado: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Python:

TareaSegundoParcial > burbuja.py > ...

```python
def bubbleSort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

arr = [64, 34, 25, 12, 22, 11, 90]
bubbleSort(arr)
print("Array ordenado:", arr)
```

Java:

TareaSegundoParcial > burbuja.java

```java
class BubbleSort {
    void bubbleSort(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    public static void main(String args[]) {
        BubbleSort bubble = new BubbleSort();
        int arr[] = {64, 34, 25, 12, 22, 11, 90};
        bubble.bubbleSort(arr);
        System.out.println("Array ordenado:");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Fuentes de consulta:

1. http://mapaches.itz.edu.mx/~mbarajas/edinf/Ordenamiento.pdf

2. https://juncotic.com/ordenamiento-de-burbuja-algoritmos-de-ordenamiento/

3. http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro9/mtodo_quick_sort.html

4. https://www.ecured.cu/Algoritmo_de_Ordenamiento_Shell

5. https://www.uv.mx/personal/ermeneses/files/2021/08/Clase9-Ordenamiento.pdf

6. https://www.tdx.cat/bitstream/handle/10803/5982/05Djg05de14.pdf;jsessionid=40BF77D9BC372471F0C100B904240699?sequence=5