UNIVERSIDAD NACIONAL DE MOQUEGUA

Facultad de Ingeniería y Arquitectura

Escuela Profesional de Ingeniería de Sistemas e Informática

Implementa y analiza algoritmos basicos de ordenamiento

Informe de Practica Nro 1

Estudiante:
Andree Cristian Cotrado Zapana

Profesor:
Honorio Apaza Alanoca

22 de octubre de 2024

Índice

1. Introducción

Los algoritmos de ordenacion son procedimientos o conjuntos de instrucciones que se utilizan para organizar un conjunto de elementos en un orden especifico. Estos algoritmos son ampliamente utilizados en ciencias de la computacion y programacion debido a su importancia para la eficiencia y optimizacion de procesos. Existen numerosos algoritmos de ordenacion, cada uno con sus propias caracterısticas y complejidades.

1.1. Motivación y Contexto

Un algoritmo de ordenacion es un algoritmo compuesto por una serie de instrucciones que toma una matriz como entrada, realiza operaciones especıficas en la matriz, a veces llamada lista, y genera como salida una matriz ordenada. El ordenamiento de datos es una tarea fundamental en la informática que mejora la eficiencia de muchos algoritmos. Este trabajo presenta un análisis de los siete algoritmos de ordenamiento más conocidos.

1.2. Objetivo general

El objetivo principal de este documento es comparar los algoritmos de ordenamiento más comunes y analizar su rendimiento en diferentes escenarios.

1.3. Objetivos específicos

- Describir los 7 algoritmos de ordenamiento más importantes.
- Implementar cada uno de los algoritmos en diferentes lenguajes de programación.
- Comparar la eficiencia de cada algoritmo en términos de complejidad temporal y espacial.

1.4. Justificación

La correcta elección de un algoritmo de ordenamiento puede impactar significativamente en el rendimiento de aplicaciones y sistemas que manejan grandes volúmenes de datos.

2. Marco teórico

2.1. Algoritmos de Ordenamiento

Los algoritmos de ordenamiento son procedimientos que organizan un conjunto de elementos en un orden predefinido (ascendente o descendente).

2.2. Comparación de métodos de ordenamiento

2.2.1. Ordenamiento por comparación

Los algoritmos por comparación trabajan con la premisa de comparar pares de elementos para decidir su orden.

2.2.2. Ordenamiento por intercambio

Estos algoritmos intercambian posiciones de los elementos hasta que todos están en el orden correcto.

2.3. Origenes Historicos

Desde tiempos antiguos, las civilizaciones han necesitado ordenar datos, aunque de manera informal. Por ejemplo, los babilonios y egipcios organizaban informacion en tablillas de arcilla y papi. En la Grecia antigua, matematicos como Euclides introdujeron principios de organizacion que influyeron en el desarrollo mismo.

2.4. Antecedentes

Los algoritmos de ordenamiento han evolucionado significativamente desde sus primeras versiones, con un enfoque en mejorar la eficiencia y la adaptabilidad a diferentes tipos de datos. La investigación y el desarrollo en este campo continúan, y se están explorando nuevos enfoques como el aprendizaje automático para optimizar el rendimiento de los algoritmos de ordenamiento.

3. Marco conceptual

El marco conceptual de los algoritmos de ordenamiento se basa en la comprensión de:
Complejidad Temporal y Espacial: Los dos principales factores que determinan la eficacia de un algoritmo de ordenamiento.
Estabilidad: La propiedad de un algoritmo que garantiza que los elementos con valores iguales mantendrán su orden relativo después de la ordenación.
Adaptabilidad: Algunos algoritmos son más efectivos con listas que ya están parcialmente ordenadas.

3.0.1. Bubble sort

Bubble Sort es un algoritmo de ordenamiento simple que funciona comparando pares de elementos adyacentes y cambiándolos si están en el orden incorrecto. Este proceso se repite hasta que no se necesiten más intercambios, lo que indica que la lista está ordenada. El nombre proviene de la forma en que los elementos "grandes"tienden a "flotar"hacia la parte superior de la lista, mientras que los más pequeños se hunden hacia la parte inferior.

```
1  package com.mycompany.mavenproject1;
2  import java.util.Scanner;
3  public class Mavenproject1 {
4
5      public static void main(int[] arr) {
6
7          int n = arr.length;
8          for (int i = 0; i < n - 1; i++) {
9              for (int j = 0; j < n - i - 1; j++) {
10                 if (arr[j] > arr[j + 1]) {
11                     int temp = arr[j];
12                     arr[j] = arr[j + 1];
13                     arr[j + 1] = temp;
14                 }
15             }
16         }
17     }
18
19     public static void main(String[] args) {
20         Scanner scanner = new Scanner(System.in);
21         System.out.print("Ingrese el numero de elementos a ordenar: ");
22         int n = scanner.nextInt();
23         int[] arr = new int[n];
24
25         System.out.println("Ingrese los elementos:");
26         for (int i = 0; i < n; i++) {
27             arr[i] = scanner.nextInt();
28         }
29
30         System.out.println("Original array:");
31         printArray(arr);
32
33         main(arr);
34
35         System.out.println("Con el metodo de ordenamiento de Buble sort
       :");
36         printArray(arr);
37         scanner.close();
38     }
39
40     public static void printArray(int[] arr) {
41         for (int num : arr) {
```

```
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

3.0.2. Couting Sort

Counting Sort es un algoritmo de ordenamiento eficiente para listas de enteros con valores en un rango conocido y limitado. A diferencia de otros algoritmos de comparación, Counting Sort no compara elementos directamente. En su lugar, cuenta la cantidad de veces que cada valor aparece en la lista y luego utiliza esa información para posicionar los elementos en su lugar correcto.

```
import java.util.Scanner;
public class Mavenproject1 {


       public static void countingSort(int[] arr) {
        int max = findMax(arr);
        int[] count = new int[max + 1];


        for (int num : arr) {
            count[num]++;
        }

        int index = 0;
        for (int i = 0; i < count.length; i++) {
            while (count[i] > 0) {
                arr[index++] = i;
                count[i]--;
            }
        }
    }

    private static int findMax(int[] arr) {
        int max = arr[0];
        for (int i : arr) {
            if (i > max) {
                max = i;
            }
        }
        return max;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Ingrese el numero de elementos a ordenar: ");
```

```
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Ingrese los elementos:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.println("Original array:");
        printArray(arr);

        countingSort(arr);

        System.out.println("Con ordenamiento de Counting sort:");
        printArray(arr);
        scanner.close();
    }

    public static void printArray(int[] arr) {
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

3.0.3. Heap sort

Heap Sort es un algoritmo de ordenamiento eficiente basado en una estructura de datos llamada árbol heap (o montículo). El algoritmo convierte el arreglo en un montículo máximo, una estructura en la que cada nodo padre es mayor o igual a sus hijos. Luego, el elemento más grande (la raíz del heap) se intercambia con el último elemento del arreglo, y se reduce el tamaño del heap. Este proceso se repite hasta que todos los elementos estén ordenados.

```
public class Mavenproject1 {


        public static void heapSort(int[] arr) {
        int n = arr.length;

        for (int i = n / 2 - 1; i >= 0; i--) {
            heapify(arr, n, i);
        }


        for (int i = n - 1; i > 0; i--) {

            int temp = arr[0];
```

```
            arr[0] = arr[i];
            arr[i] = temp;

            heapify(arr, i, 0);
        }
    }

    private static void heapify(int[] arr, int n, int i) {
        int largest = i;
        int left = 2 * i + 1;
        int right = 2 * i + 2;

        if (left < n && arr[left] > arr[largest]) {
            largest = left;
        }

        if (right < n && arr[right] > arr[largest]) {
            largest = right;
        }

        if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;

            heapify(arr, n, largest);
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Ingrese el numero de elementos a ordenar: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Ingrese los elementos:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.println("Original array:");
        printArray(arr);

        heapSort(arr);

        System.out.println("Por metodo de Heap Sort:");
        printArray(arr);
        scanner.close();
    }

```

```
65      public static void printArray(int[] arr) {
66          for (int num : arr) {
67              System.out.print(num + " ");
68          }
69          System.out.println();
70      }
71  }
```

3.0.4. Insertion sort

Insertion Sort es un algoritmo de ordenamiento simple que construye la lista ordenada de manera gradual, al insertar un elemento a la vez en la posición correcta. Funciona de manera similar a cómo ordenarías las cartas en tu mano.

```
1  public class InsertionSort {
2      public static void insertionSort(int[] arr) {
3          int n = arr.length;
4          for (int i = 1; i < n; ++i) {
5              int key = arr[i];
6              int j = i - 1;
7
8              adelante
9              while (j >= 0 && arr[j] > key) {
10                 arr[j + 1] = arr[j];
11                 j = j - 1;
12             }
13             arr[j + 1] = key;
14         }
15     }
16
17     public static void main(String[] args) {
18         Scanner scanner = new Scanner(System.in);
19         System.out.print("Ingrese el numero de elementos a ordenar: ");
20         int n = scanner.nextInt();
21         int[] arr = new int[n];
22
23         System.out.println("Ingrese los elementos:");
24         for (int i = 0; i < n; i++) {
25             arr[i] = scanner.nextInt();
26         }
27
28         System.out.println("Original array:");
29         printArray(arr);
30
31         insertionSort(arr);
32
33         System.out.println("Por metodo de Insertion Sort:");
34         printArray(arr);
35         scanner.close();
36     }
```

```
37
38      public static void printArray(int[] arr) {
39          for (int num : arr) {
40              System.out.print(num + " ");
41          }
42          System.out.println();
43      }
44  }
```

3.0.5. Merge sort

Merge Sort es un algoritmo de ordenamiento eficiente basado en la técnica de divide y vencerás. Divide recursivamente el arreglo en dos mitades hasta que cada subarreglo tiene un solo elemento (que por definición está ordenado). Luego, las mitades se combinan (o "fusionan") de manera ordenada para formar el arreglo completo y ordenado.

```
1  public class MergeSort {
2      public static void mergeSort(int[] arr) {
3          if (arr.length < 2) {
4              return;
5          }
6
7          int mid = arr.length / 2;
8          int[] left = new int[mid];
9          int[] right = new int[arr.length - mid];
10
11
12         for (int i = 0; i < mid; i++) {
13             left[i] = arr[i];
14         }
15         for (int i = mid; i < arr.length; i++) {
16             right[i - mid] = arr[i];
17         }
18
19
20         mergeSort(left);
21         mergeSort(right);
22
23
24         merge(arr, left, right);
25     }
26
27     private static void merge(int[] arr, int[] left, int[] right) {
28         int i = 0, j = 0, k = 0;
29
30
31         while (i < left.length && j < right.length) {
32             if (left[i] <= right[j]) {
33                 arr[k++] = left[i++];
```

```
            } else {
                arr[k++] = right[j++];
            }
        }

        while (i < left.length) {
            arr[k++] = left[i++];
        }

        while (j < right.length) {
            arr[k++] = right[j++];
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Ingrese el numero de elementos a ordenar: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Ingrese los elementos:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.println("Original array:");
        printArray(arr);

        mergeSort(arr);

        System.out.println("Usando el metodo de Merge Sort:");
        printArray(arr);
        scanner.close();
    }

    public static void printArray(int[] arr) {
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

3.0.6. Quick sort

Quick Sort es un algoritmo de ordenamiento eficiente que también sigue el paradigma de divide y vencerás. Selecciona un elemento del arreglo como pivote y reorganiza los demás elementos de tal manera que todos los elementos menores que el pivote queden a su izquierda y los mayores a su derecha. Este proceso se repite recursivamente en las sublistas

generadas hasta que toda la lista esté ordenada.

```
public class QuickSort {
    public static 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);
        }
    }

    private static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = (low - 1);

        for (int j = low; j < high; 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;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Ingrese el numero de elementos a ordenar: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Ingrese los elementos:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.println("Original array:");
        printArray(arr);

        quickSort(arr, 0, arr.length - 1);

        System.out.println("Usando el metodo de Quick Sort:");
        printArray(arr);
```

```
        scanner.close();
    }

    public static void printArray(int[] arr) {
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

3.0.7. Selection sort

Selection Sort es un algoritmo de ordenamiento simple que divide el arreglo en dos partes: la parte ordenada y la parte desordenada. A medida que el algoritmo avanza, selecciona el elemento más pequeño de la parte desordenada y lo intercambia con el primer elemento de esa parte, ampliando así la parte ordenada del arreglo.

```
public class SelectionSort {
    public static void selectionSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }

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

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Ingrese el numero de elementos a ordenar: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Ingrese los elementos:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.println("Original array:");
        printArray(arr);

```

```
32          selectionSort(arr);
33
34          System.out.println("Usando el metodo de Selection Sort:");
35          printArray(arr);
36          scanner.close();
37      }
38
39      public static void printArray(int[] arr) {
40          for (int num : arr) {
41              System.out.print(num + " ");
42          }
43          System.out.println();
44      }
45  }
```

4. Algoritmo

4.1. Codigo de Java

```
package com.mycompany.proyecto;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;

public class Proyecto {

    public static void main(String[] args) {
      // Obt n la ruta del escritorio del usuario actual
        String userHome = System.getProperty("user.home");
        String folderPath = userHome + "\\Desktop\\numeros desordenados
    \\"; // Ruta a la carpeta

        List<String> files = getTextFiles(folderPath);

        for (String fileName : files) {
            System.out.println("Ordenando archivo: " + fileName);
            List<Integer> numbers = readNumbersFromFile(folderPath +
    fileName);

            if (numbers != null) {
                measureSortTime("Bubble Sort", numbers.toArray(Integer
    []::new), Proyecto::bubbleSort);
                measureSortTime("Counting Sort", numbers.toArray(Integer
    []::new), Proyecto::countingSort);
                measureSortTime("Heap Sort", numbers.toArray(Integer[]::
    new), Proyecto::heapSort);
                measureSortTime("Insertion Sort", numbers.toArray(
    Integer[]::new), Proyecto::insertionSort);
                measureSortTime("Merge Sort", numbers.toArray(Integer
    []::new), Proyecto::mergeSort);
                measureSortTime("Quick Sort", numbers.toArray(Integer
    []::new), Proyecto::quickSort);
                measureSortTime("Selection Sort", numbers.toArray(
    Integer[]::new), Proyecto::selectionSort);
            }
        }
    }

    private static List<String> getTextFiles(String folderPath) {
        List<String> fileNames = new ArrayList<>();
```

```
38          try {
39              Files.list(Paths.get(folderPath))
40                   .filter(path -> path.toString().endsWith(".txt"))
41                   .forEach(path -> fileNames.add(path.getFileName().
    toString()));
42          } catch (IOException e) {
43              e.printStackTrace();
44          }
45          return fileNames;
46      }
47
48      private static List<Integer> readNumbersFromFile(String filePath) {
49      List<Integer> numbers = new ArrayList<>();
50      try (BufferedReader br = new BufferedReader(new FileReader(filePath)
    )) {
51          String line = br.readLine();
52          if (line != null) {
53              // Limpiar y dividir los n meros
54              String[] nums = line.replaceAll("[\\[\\]\\s]", "").split
    (","); // Elimina corchetes y espacios
55              for (String num : nums) {
56                  try {
57                      numbers.add(Integer.parseInt(num.trim()));
58                  } catch (NumberFormatException e) {
59                      System.out.println("N mero no v lido: " + num);
60                  }
61              }
62          }
63      } catch (IOException e) {
64          e.printStackTrace();
65      }
66      return numbers;
67  }
68
69      private static void measureSortTime(String algorithmName, Integer[]
    array, SortingAlgorithm algorithm) {
70          long startTime = System.nanoTime();
71          algorithm.sort(array);
72          long endTime = System.nanoTime();
73          System.out.println(algorithmName + " took " + (endTime -
    startTime) / 1_000_000.0 + " ms");
74      }
75
76      interface SortingAlgorithm {
77          void sort(Integer[] array);
78      }
79
80      // Bubble Sort
81      private static void bubbleSort(Integer[] array) {
82          int n = array.length;
```

```
83          for (int i = 0; i < n - 1; i++) {
84              for (int j = 0; j < n - i - 1; j++) {
85                  if (array[j] > array[j + 1]) {
86                      int temp = array[j];
87                      array[j] = array[j + 1];
88                      array[j + 1] = temp;
89                  }
90              }
91          }
92      }
93
94      // Counting Sort
95      private static void countingSort(Integer[] array) {
96          int max = Integer.MIN_VALUE;
97          for (int num : array) {
98              if (num > max) max = num;
99          }
100
101         int[] count = new int[max + 1];
102         for (int num : array) {
103             count[num]++;
104         }
105
106         int index = 0;
107         for (int i = 0; i < count.length; i++) {
108             while (count[i]-- > 0) {
109                 array[index++] = i;
110             }
111         }
112     }
113
114     // Heap Sort
115     private static void heapSort(Integer[] array) {
116         int n = array.length;
117         for (int i = n / 2 - 1; i >= 0; i--) {
118             heapify(array, n, i);
119         }
120         for (int i = n - 1; i > 0; i--) {
121             int temp = array[0];
122             array[0] = array[i];
123             array[i] = temp;
124             heapify(array, i, 0);
125         }
126     }
127
128     private static void heapify(Integer[] array, int n, int i) {
129         int largest = i;
130         int left = 2 * i + 1;
131         int right = 2 * i + 2;
132         if (left < n && array[left] > array[largest]) {
```

```
133             largest = left;
134         }
135         if (right < n && array[right] > array[largest]) {
136             largest = right;
137         }
138         if (largest != i) {
139             int swap = array[i];
140             array[i] = array[largest];
141             array[largest] = swap;
142             heapify(array, n, largest);
143         }
144     }
145 
146     // Insertion Sort
147     private static void insertionSort(Integer[] array) {
148         int n = array.length;
149         for (int i = 1; i < n; i++) {
150             int key = array[i];
151             int j = i - 1;
152             while (j >= 0 && array[j] > key) {
153                 array[j + 1] = array[j];
154                 j--;
155             }
156             array[j + 1] = key;
157         }
158     }
159 
160     // Merge Sort
161     private static void mergeSort(Integer[] array) {
162         if (array.length < 2) {
163             return;
164         }
165         int mid = array.length / 2;
166         Integer[] left = new Integer[mid];
167         Integer[] right = new Integer[array.length - mid];
168         System.arraycopy(array, 0, left, 0, mid);
169         System.arraycopy(array, mid, right, 0, array.length - mid);
170         mergeSort(left);
171         mergeSort(right);
172         merge(array, left, right);
173     }
174 
175     private static void merge(Integer[] array, Integer[] left, Integer[]
    right) {
176         int i = 0, j = 0, k = 0;
177         while (i < left.length && j < right.length) {
178             if (left[i] <= right[j]) {
179                 array[k++] = left[i++];
180             } else {
181                 array[k++] = right[j++];
```

```
182             }
183         }
184         while (i < left.length) {
185             array[k++] = left[i++];
186         }
187         while (j < right.length) {
188             array[k++] = right[j++];
189         }
190     }
191 
192     // Quick Sort
193     private static void quickSort(Integer[] array) {
194         quickSort(array, 0, array.length - 1);
195     }
196 
197     private static void quickSort(Integer[] array, int low, int high) {
198         if (low < high) {
199             int pi = partition(array, low, high);
200             quickSort(array, low, pi - 1);
201             quickSort(array, pi + 1, high);
202         }
203     }
204 
205     private static int partition(Integer[] array, int low, int high) {
206         int pivot = array[high];
207         int i = (low - 1);
208         for (int j = low; j < high; j++) {
209             if (array[j] <= pivot) {
210                 i++;
211                 int temp = array[i];
212                 array[i] = array[j];
213                 array[j] = temp;
214             }
215         }
216         int temp = array[i + 1];
217         array[i + 1] = array[high];
218         array[high] = temp;
219         return i + 1;
220     }
221 
222     // Selection Sort
223     private static void selectionSort(Integer[] array) {
224         int n = array.length;
225         for (int i = 0; i < n - 1; i++) {
226             int minIdx = i;
227             for (int j = i + 1; j < n; j++) {
228                 if (array[j] < array[minIdx]) {
229                     minIdx = j;
230                 }
231             }
```

```
232                 int temp = array[minIdx];
233                 array[minIdx] = array[i];
234                 array[i] = temp;
235             }
236         }
237 }
```

5. Graficas

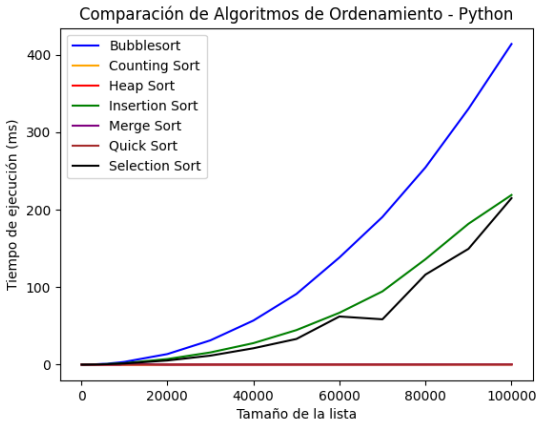

Figura 1: Gráfica de Python

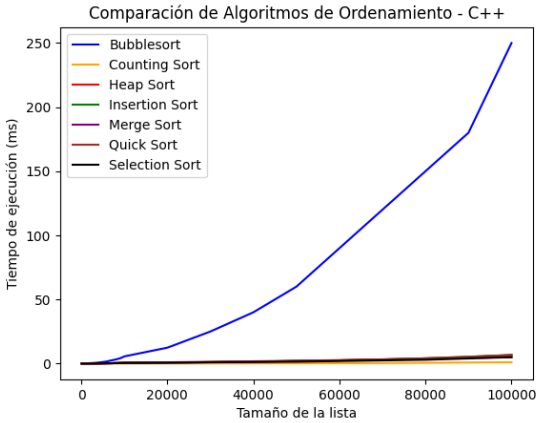

Figura 2: Grafica de Java

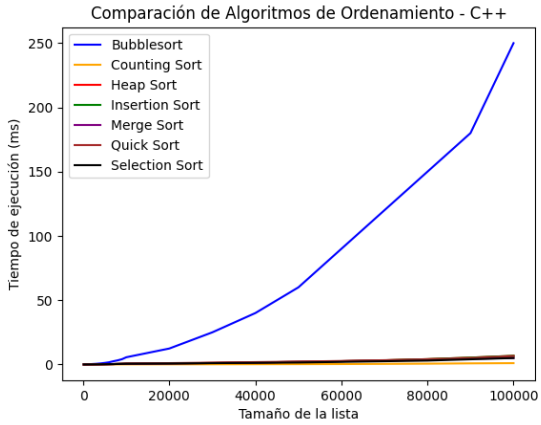

Figura 3: Grafica de C++

6. Graficas de cada metodo de ordenamiento:

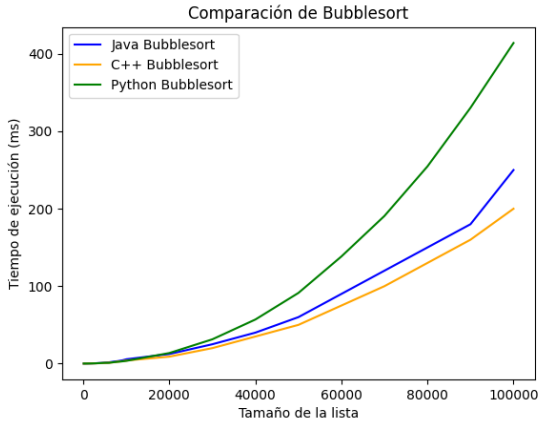

Figura 4: Bubble Sort

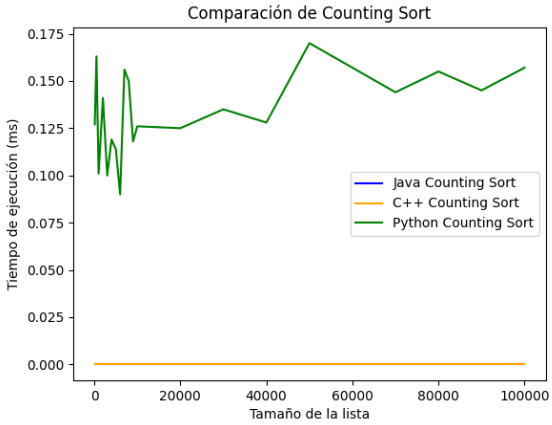

Figura 5: Couting Sort

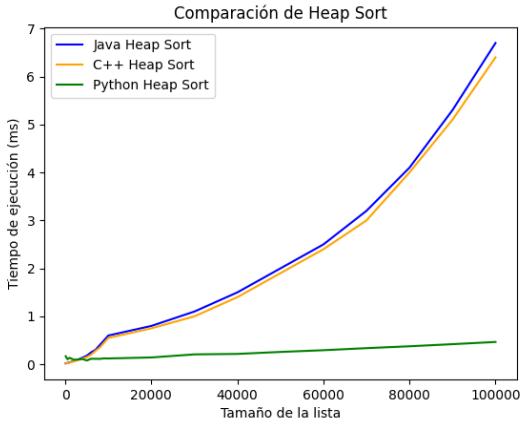

Figura 6: Heap Sort

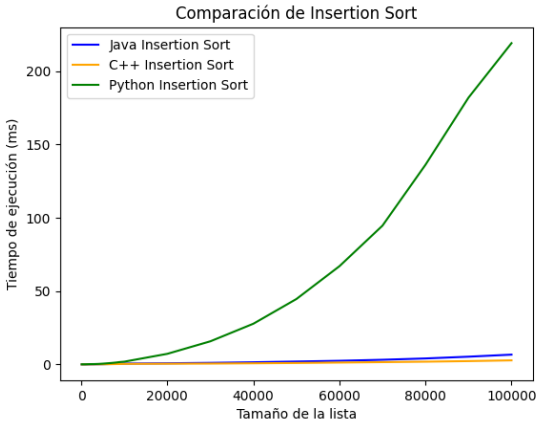

Figura 7: Insertion Sort

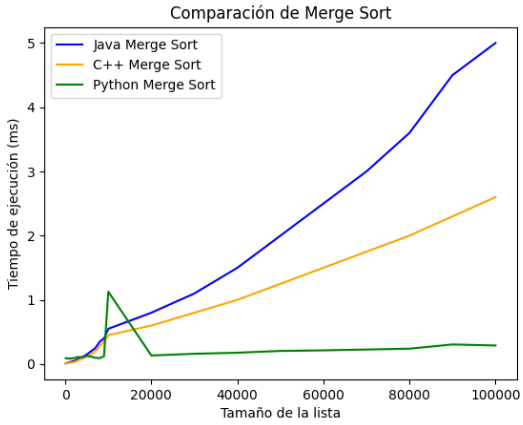

Figura 8: Merge Sort

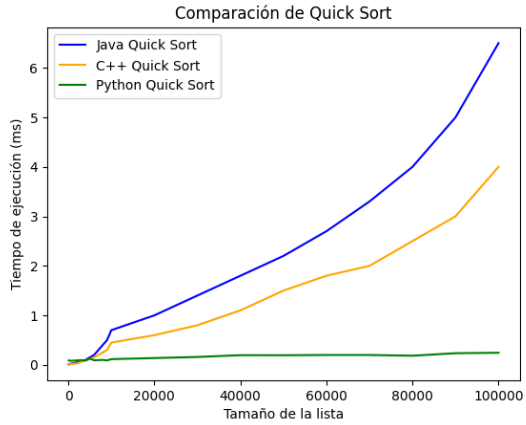

Figura 9: Quick Sort

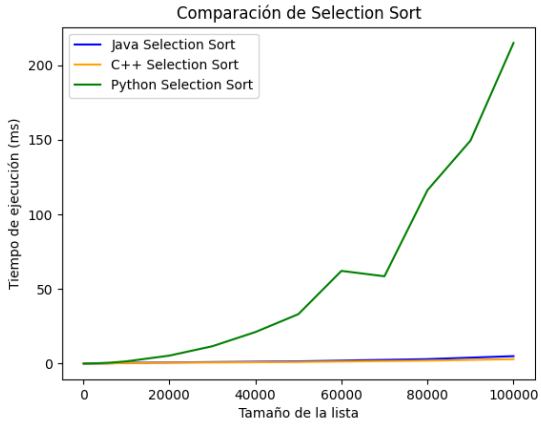

Figura 10: Selection Sort

7. Pseudocodigos

7.1. Pseudocodigo de Python

```
INICIO

    // Importar m dulos necesarios
    IMPORTAR os
    IMPORTAR time
    IMPORTAR ast

    // Definir funci n de ordenamiento: Bubble Sort
    FUNCI N bubble_sort(arr)
        n = longitud(arr)
        PARA i EN rango(n)
            PARA j EN rango(0, n-i-1)
                SI arr[j] > arr[j+1]
                    intercambiar arr[j] y arr[j+1]
        RETORNAR arr

    // Definir funci n de ordenamiento: Counting Sort
    FUNCI N counting_sort(arr)
        max_val = m ximo(arr)
        count = lista de ceros de tama o (max_val + 1)
        PARA cada num EN arr
            count[num] += 1

        sorted_arr = lista vac a
        PARA i EN rango(longitud(count))
            agregar i repetido count[i] veces a sorted_arr
        RETORNAR sorted_arr

    // Definir funci n de ordenamiento: Heap Sort
    FUNCI N heap_sort(arr)
        FUNCI N heapify(arr, n, i)
            definir largest = i
            left = 2 * i + 1
            right = 2 * i + 2

            SI left < n Y arr[left] > arr[largest]
                largest = left
            SI right < n Y arr[right] > arr[largest]
                largest = right

            SI largest     i
                intercambiar arr[i] y arr[largest]
                Llamar a heapify(arr, n, largest)

        n = longitud(arr)
        PARA i EN rango(n//2 - 1, -1, -1)
```

```
            Llamar a heapify(arr, n, i)
        PARA i EN rango(n-1, 0, -1)
            intercambiar arr[i] y arr[0]
            Llamar a heapify(arr, i, 0)
        RETORNAR arr

    // Definir funci n de ordenamiento: Insertion Sort
    FUNCI N insertion_sort(arr)
        PARA i EN rango(1, longitud(arr))
            key = arr[i]
            j = i - 1
            MIENTRAS j >= 0 Y key < arr[j]
                arr[j + 1] = arr[j]
                j -= 1
            arr[j + 1] = key
        RETORNAR arr

    // Definir funci n de ordenamiento: Merge Sort
    FUNCI N merge_sort(arr)
        SI longitud(arr) > 1
            mid = longitud(arr) // 2
            L = arr[:mid]
            R = arr[mid:]

            Llamar a merge_sort(L)
            Llamar a merge_sort(R)

            i = j = k = 0

            MIENTRAS i < longitud(L) Y j < longitud(R)
                SI L[i] < R[j]
                    arr[k] = L[i]
                    i += 1
                SINO
                    arr[k] = R[j]
                    j += 1
                k += 1

            MIENTRAS i < longitud(L)
                arr[k] = L[i]
                i += 1
                k += 1

            MIENTRAS j < longitud(R)
                arr[k] = R[j]
                j += 1
                k += 1

        RETORNAR arr

```

```
97      // Definir funci n de ordenamiento: Quick Sort
98      FUNCI N quick_sort(arr)
99          SI longitud(arr) <= 1
100             RETORNAR arr
101         SINO
102             pivot = arr[longitud(arr) // 2]
103             left = [x PARA x EN arr SI x < pivot]
104             middle = [x PARA x EN arr SI x == pivot]
105             right = [x PARA x EN arr SI x > pivot]
106             RETORNAR quick_sort(left) + middle + quick_sort(right)
107
108     // Definir funci n de ordenamiento: Selection Sort
109     FUNCI N selection_sort(arr)
110         PARA i EN rango(longitud(arr))
111             min_idx = i
112             PARA j EN rango(i+1, longitud(arr))
113                 SI arr[j] < arr[min_idx]
114                     min_idx = j
115             intercambiar arr[i] y arr[min_idx]
116         RETORNAR arr
117
118     // Definir funci n para leer n meros desde un archivo
119     FUNCI N read_numbers_from_file(file_path)
120         ABRIR archivo en file_path
121         leer contenido del archivo
122         INTENTAR
123             numbers = evaluar el contenido como una lista
124             RETORNAR numbers
125         CAPTURAR error
126             mostrar mensaje de error
127             RETORNAR lista vac a
128
129     // Funci n principal para ejecutar todos los m todos y medir el
    tiempo
130     FUNCI N sort_numbers_from_files()
131         desktop_path = ruta al escritorio
132         folder_path = unir desktop_path con "numeros desordenados"
133
134         // Obtener todos los archivos .txt en la carpeta
135         file_names = lista de archivos en folder_path que terminan en '.
    txt'
136
137         PARA cada file_name EN file_names
138             file_path = unir folder_path con file_name
139             INTENTAR
140                 numbers = llamar a read_numbers_from_file(file_path)
141                 SI numbers est  vac a
142                     CONTINUAR
143
144                 mostrar mensaje de inicio de ordenamiento
```

```
145
146                 // Medir el tiempo para cada m todo de ordenamiento
147                 PARA cada m todo de ordenamiento
148                     start_time = tiempo actual
149                     llamar a m todo de ordenamiento con numbers.copy()
150                     mostrar tiempo de ejecuci n
151
152             CAPTURAR FileNotFoundError
153                 mostrar mensaje de archivo no encontrado
154                 CONTINUAR
155
156     SI __nombre__ ES "__main__"
157         llamar a sort_numbers_from_files()
158
159 FIN
```

7.2. Pseudocodigo de Java

```
1  INICIO
2
3      DEFINIR clase Proyecto
4
5      M TODO principal
6          // Obtiene la ruta del escritorio del usuario actual
7          userHome = obtener ruta del usuario
8          folderPath = userHome + "\Desktop\numeros desordenados\"
9
10         files = obtener archivos de texto en folderPath
11
12         PARA cada fileName en files HACER
13             IMPRIMIR "Ordenando archivo: " + fileName
14             numbers = leer n meros desde el archivo folderPath +
       fileName
15
16             SI numbers NO ES NULL ENTONCES
17                 medir y mostrar tiempo para Bubble Sort
18                 medir y mostrar tiempo para Counting Sort
19                 medir y mostrar tiempo para Heap Sort
20                 medir y mostrar tiempo para Insertion Sort
21                 medir y mostrar tiempo para Merge Sort
22                 medir y mostrar tiempo para Quick Sort
23                 medir y mostrar tiempo para Selection Sort
24             FIN SI
25         FIN PARA
26     FIN M TODO
27
28     FUNCI N obtener archivos de texto(folderPath)
29         INICIALIZAR lista vac a fileNames
30         INTENTAR
```

```
            LISTAR archivos en folderPath
            FILTRAR archivos que terminan en ".txt"
            PARA cada path en archivos FILTRADOS HACER
                AGREGAR nombre de archivo a fileNames
            FIN PARA
        CAPTURAR IOException e HACER
            IMPRIMIR e
        FIN INTENTAR
        RETORNAR fileNames
    FIN FUNCI N

    FUNCI N leer n meros desde archivo(filePath)
        INICIALIZAR lista vac a numbers
        INTENTAR
            ABRIR archivo en filePath
            LEER primera l nea
            SI l nea NO ES NULL ENTONCES
                // Limpiar y dividir los n meros
                nums = limpiar l nea y dividir en ","
                PARA cada num en nums HACER
                    INTENTAR
                        AGREGAR n mero parseado a numbers
                    CAPTURAR NumberFormatException e HACER
                        IMPRIMIR "N mero no v lido: " + num
                    FIN INTENTAR
                FIN PARA
            FIN SI
        CAPTURAR IOException e HACER
            IMPRIMIR e
        FIN INTENTAR
        RETORNAR numbers
    FIN FUNCI N

    FUNCI N medir y mostrar tiempo(algorithmName, array, algorithm)
        INICIO TIEMPO
        ALGORITMO.sort(array)
        FIN TIEMPO
        IMPRIMIR algorithmName + " took " + tiempo en ms
    FIN FUNCI N

    INTERFAZ SortingAlgorithm
        M TODO sort(array)
    FIN INTERFAZ

    // M todos de ordenamiento
    FUNCI N bubbleSort(array)
        n = longitud de array
        PARA i desde 0 HASTA n-1 HACER
            PARA j desde 0 HASTA n-i-1 HACER
                SI array[j] > array[j+1] ENTONCES
```

```
                    intercambiar array[j] y array[j+1]
                FIN SI
            FIN PARA
        FIN PARA
    FIN FUNCI N

    FUNCI N countingSort(array)
        max = encontrar valor m ximo en array
        INICIALIZAR array count de tama o max+1 con ceros
        PARA num en array HACER
            incrementar count[num]
        FIN PARA

        index = 0
        PARA i desde 0 HASTA count.length HACER
            MIENTRAS count[i] > 0 HACER
                array[index++] = i
                decrementa count[i]
            FIN MIENTRAS
        FIN PARA
    FIN FUNCI N

    FUNCI N heapSort(array)
        n = longitud de array
        PARA i desde n/2-1 HASTA 0 HACER
            heapify(array, n, i)
        FIN PARA
        PARA i desde n-1 HASTA 1 HACER
            intercambiar array[0] y array[i]
            heapify(array, i, 0)
        FIN PARA
    FIN FUNCI N

    FUNCI N heapify(array, n, i)
        largest = i
        left = 2*i + 1
        right = 2*i + 2
        SI left < n Y array[left] > array[largest] ENTONCES
            largest = left
        FIN SI
        SI right < n Y array[right] > array[largest] ENTONCES
            largest = right
        FIN SI
        SI largest != i ENTONCES
            intercambiar array[i] y array[largest]
            heapify(array, n, largest)
        FIN SI
    FIN FUNCI N

    FUNCI N insertionSort(array)
```

```
131         n = longitud de array
132         PARA i desde 1 HASTA n HACER
133             key = array[i]
134             j = i - 1
135             MIENTRAS j >= 0 Y array[j] > key HACER
136                 array[j + 1] = array[j]
137                 j--
138             FIN MIENTRAS
139             array[j + 1] = key
140         FIN PARA
141     FIN FUNCI N
142
143     FUNCI N mergeSort(array)
144         SI longitud de array < 2 ENTONCES
145             RETORNAR
146         FIN SI
147         mid = longitud de array / 2
148         left = crear array nuevo de tama o mid
149         right = crear array nuevo de tama o array.length - mid
150         COPIAR elementos de array a left y right
151         mergeSort(left)
152         mergeSort(right)
153         merge(array, left, right)
154     FIN FUNCI N
155
156     FUNCI N merge(array, left, right)
157         i = 0, j = 0, k = 0
158         MIENTRAS i < longitud de left Y j < longitud de right HACER
159             SI left[i] <= right[j] ENTONCES
160                 array[k++] = left[i++]
161             SINO
162                 array[k++] = right[j++]
163             FIN SI
164         FIN MIENTRAS
165         MIENTRAS i < longitud de left HACER
166             array[k++] = left[i++]
167         FIN MIENTRAS
168         MIENTRAS j < longitud de right HACER
169             array[k++] = right[j++]
170         FIN MIENTRAS
171     FIN FUNCI N
172
173     FUNCI N quickSort(array)
174         quickSort(array, 0, longitud de array - 1)
175     FIN FUNCI N
176
177     FUNCI N quickSort(array, low, high)
178         SI low < high ENTONCES
179             pi = partition(array, low, high)
180             quickSort(array, low, pi - 1)
```

```
            quickSort(array, pi + 1, high)
        FIN SI
    FIN FUNCI N

    FUNCI N partition(array, low, high)
        pivot = array[high]
        i = (low - 1)
        PARA j desde low HASTA high HACER
            SI array[j] <= pivot ENTONCES
                i++
                intercambiar array[i] y array[j]
            FIN SI
        FIN PARA
        intercambiar array[i + 1] y array[high]
        RETORNAR i + 1
    FIN FUNCI N

    FUNCI N selectionSort(array)
        n = longitud de array
        PARA i desde 0 HASTA n-1 HACER
            minIdx = i
            PARA j desde i + 1 HASTA n HACER
                SI array[j] < array[minIdx] ENTONCES
                    minIdx = j
                FIN SI
            FIN PARA
            intercambiar array[minIdx] y array[i]
        FIN PARA
    FIN FUNCI N

FIN
```

7.3. Pseudocodigo de C++

```
INICIO

    DEFINIR espacio de nombres fs = std::filesystem
    USAR std y chrono

    FUNCI N filtrarCadena(str)
        INICIALIZAR string resultado vac o
        PARA cada car cter c en str HACER
            SI c es un d gito O c es un espacio ENTONCES
                agregar c a resultado
            FIN SI
        FIN PARA
        RETORNAR resultado
    FIN FUNCI N

```

```
16    FUNCI N leerArchivo(filePath)
17        INICIALIZAR vector de enteros numeros vac o
18        ABRIR archivo en filePath
19        SI archivo NO se puede abrir ENTONCES
20            IMPRIMIR "No se pudo abrir el archivo: " + filePath
21            RETORNAR numeros
22        FIN SI
23
24        INICIALIZAR string linea
25        MIENTRAS LEER linea del archivo HACER
26            INICIALIZAR stringstream ss con linea
27            INICIALIZAR string temp
28            MIENTRAS LEER temp desde ss separado por comas HACER
29                temp = filtrarCadena(temp) // Filtrar caracteres no
    deseados
30                eliminar espacios de temp
31
32                SI temp est  vac o ENTONCES continuar
33
34                INTENTAR
35                    agregar stoi(temp) a numeros
36                CAPTURAR invalid_argument e HACER
37                    IMPRIMIR "N mero inv lido: " + temp
38                CAPTURAR out_of_range e HACER
39                    IMPRIMIR "N mero fuera de rango: " + temp
40                FIN INTENTAR
41            FIN MIENTRAS
42        FIN MIENTRAS
43
44        CERRAR archivo
45        RETORNAR numeros
46    FIN FUNCI N
47
48    FUNCI N medirTiempoOrdenamiento(nombreAlgoritmo, numeros, algoritmo
    )
49        INICIAR cron metro
50        ejecutar algoritmo(numeros)
51        DETENER cron metro
52        DURACION = calcular duraci n en milisegundos
53        IMPRIMIR nombreAlgoritmo + " tom  " + DURACION + " ms"
54    FIN FUNCI N
55
56    // Algoritmos de ordenamiento
57    FUNCI N bubbleSort(array)
58        n = tama o de array
59        PARA i desde 0 HASTA n-1 HACER
60            PARA j desde 0 HASTA n-i-1 HACER
61                SI array[j] > array[j+1] ENTONCES
62                    intercambiar array[j] y array[j+1]
63                FIN SI
```

```
64              FIN PARA
65          FIN PARA
66      FIN FUNCI N
67
68      FUNCI N countingSort(array)
69          max = encontrar elemento m ximo en array
70          INICIALIZAR vector count de tama o max+1 con ceros
71
72          PARA num en array HACER
73              incrementar count[num]
74          FIN PARA
75
76          index = 0
77          PARA i desde 0 HASTA max HACER
78              MIENTRAS count[i] > 0 HACER
79                  array[index++] = i
80                  decrementar count[i]
81              FIN MIENTRAS
82          FIN PARA
83      FIN FUNCI N
84
85      FUNCI N heapify(array, n, i)
86          largest = i
87          left = 2*i + 1
88          right = 2*i + 2
89
90          SI left < n Y array[left] > array[largest] ENTONCES
91              largest = left
92          FIN SI
93          SI right < n Y array[right] > array[largest] ENTONCES
94              largest = right
95          FIN SI
96          SI largest != i ENTONCES
97              intercambiar array[i] y array[largest]
98              heapify(array, n, largest)
99          FIN SI
100     FIN FUNCI N
101
102     FUNCI N heapSort(array)
103         n = tama o de array
104         PARA i desde n/2-1 HASTA 0 HACER
105             heapify(array, n, i)
106         FIN PARA
107         PARA i desde n-1 HASTA 1 HACER
108             intercambiar array[0] y array[i]
109             heapify(array, i, 0)
110         FIN PARA
111     FIN FUNCI N
112
113     FUNCI N insertionSort(array)
```

```
114         n = tama o de array
115         PARA i desde 1 HASTA n HACER
116             key = array[i]
117             j = i - 1
118             MIENTRAS j >= 0 Y array[j] > key HACER
119                 array[j + 1] = array[j]
120                 j--
121             FIN MIENTRAS
122             array[j + 1] = key
123         FIN PARA
124     FIN FUNCI N
125
126     FUNCI N merge(array, left, right)
127         i = 0, j = 0, k = 0
128         MIENTRAS i < tama o de left Y j < tama o de right HACER
129             SI left[i] <= right[j] ENTONCES
130                 array[k++] = left[i++]
131             SINO
132                 array[k++] = right[j++]
133             FIN SI
134         FIN MIENTRAS
135         MIENTRAS i < tama o de left HACER
136             array[k++] = left[i++]
137         FIN MIENTRAS
138         MIENTRAS j < tama o de right HACER
139             array[k++] = right[j++]
140         FIN MIENTRAS
141     FIN FUNCI N
142
143     FUNCI N mergeSort(array)
144         SI tama o de array < 2 ENTONCES RETORNAR
145
146         mid = tama o de array / 2
147         left = crear un nuevo vector desde array[0] hasta mid
148         right = crear un nuevo vector desde array[mid] hasta final
149
150         mergeSort(left)
151         mergeSort(right)
152         merge(array, left, right)
153     FIN FUNCI N
154
155     FUNCI N partition(array, low, high)
156         pivot = array[high]
157         i = low - 1
158
159         PARA j desde low HASTA high HACER
160             SI array[j] <= pivot ENTONCES
161                 i++
162                 intercambiar array[i] y array[j]
163             FIN SI
```

```
164         FIN PARA
165         intercambiar array[i + 1] y array[high]
166         RETORNAR i + 1
167     FIN FUNCI N
168 
169     FUNCI N quickSort(array, low, high)
170         SI low < high ENTONCES
171             pi = partition(array, low, high)
172             quickSort(array, low, pi - 1)
173             quickSort(array, pi + 1, high)
174         FIN SI
175     FIN FUNCI N
176 
177     FUNCI N selectionSort(array)
178         n = tama o de array
179         PARA i desde 0 HASTA n-1 HACER
180             minIdx = i
181             PARA j desde i + 1 HASTA n HACER
182                 SI array[j] < array[minIdx] ENTONCES
183                     minIdx = j
184                 FIN SI
185             FIN PARA
186             intercambiar array[minIdx] y array[i]
187         FIN PARA
188     FIN FUNCI N
189 
190     FUNCI N principal()
191         folderPath = "C:\\Users\\ALIENWARE\\Desktop\\c++\\numeros
    desordenados\\"
192 
193         SI la carpeta NO existe ENTONCES
194             IMPRIMIR "La ruta especificada no existe: " + folderPath
195             RETORNAR 1
196         FIN SI
197 
198         PARA cada entry en el directorio folderPath HACER
199             IMPRIMIR "Archivo encontrado: " + entry.path()
200 
201             SI entry es un archivo regular Y entry tiene extensi n .txt
     ENTONCES
202                 filePath = entry.path().string()
203                 IMPRIMIR "Ordenando archivo: " + filePath
204 
205                 numeros = leerArchivo(filePath)
206                 SI numeros est  vac o ENTONCES continuar
207 
208                 // Medir tiempo de cada algoritmo
209                 medirTiempoOrdenamiento("Bubble Sort", numeros,
    bubbleSort)
210                 medirTiempoOrdenamiento("Counting Sort", numeros,
```

```
    countingSort)
211                 medirTiempoOrdenamiento("Heap Sort", numeros, heapSort)
212                 medirTiempoOrdenamiento("Insertion Sort", numeros,
    insertionSort)
213                 medirTiempoOrdenamiento("Merge Sort", numeros, mergeSort
    )
214                 medirTiempoOrdenamiento("Quick Sort", numeros, quickSort
    )
215                 medirTiempoOrdenamiento("Selection Sort", numeros,
    selectionSort)
216             FIN SI
217         FIN PARA
218      FIN FUNCI N
219
220 FIN
```

8. Resultados

Counting Sort destacó por su rapidez, especialmente en lenguajes como Python y Java, donde el tiempo se mantuvo bajo incluso para grandes conjuntos de datos. En contraste, C++ también mostró un rendimiento competitivo, pero con ligeras variaciones en el rendimiento. Este algoritmo es particularmente efectivo para conjuntos de datos con valores enteros en un rango limitado.
Los resultados muestran que C++ tiende a ser más rápido en la mayoría de los algoritmos, seguido de Java y Python. Los métodos como Counting Sort y Merge Sort son preferibles para conjuntos de datos grandes, mientras que algoritmos como Bubble Sort y Selection Sort son menos adecuados debido a su ineficiencia inherente. Estos hallazgos resaltan la importancia de elegir el algoritmo de ordenamiento adecuado en función del contexto y los requisitos de rendimiento específicos.

9. Conclusiones

[1]

1. Coclusion numero uno : Eficiencia de Algoritmos: Los algoritmos de ordenamiento como Counting Sort y Merge Sort demostraron ser significativamente más eficientes que métodos más simples como Bubble Sort y Selection Sort. Esto resalta la importancia de seleccionar algoritmos apropiados que se adapten a las características de los datos y a los requisitos de rendimiento del problema específico.

2. Coclusion numero dos: Variabilidad entre Lenguajes: Aunque C++ mostró un rendimiento superior en la mayoría de los algoritmos de ordenamiento, Java y Python también ofrecieron tiempos competitivos. Esto sugiere que, dependiendo de la aplicación y el contexto, la elección del lenguaje puede influir en la eficiencia del algoritmo, pero no es el único factor determinante.

3. Coclusion numero tres : Importancia del Tamaño de los Datos: A medida que el tamaño de los datos aumenta, los tiempos de ejecución de los algoritmos se vuelven más críticos. Los algoritmos ineficientes, como Bubble Sort y Selection Sort, mostraron tiempos de ejecución exponencialmente mayores con volúmenes de datos más grandes, lo que sugiere que su uso debe evitarse en situaciones donde el rendimiento sea una preocupación importante.

Referencias

[1] George D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993.

10. Ejemplo de uso de latex

10.1. Como inserter codigo a latex

Use el paquete `listings` para escribir el código, por ejemplo para C++:

```cpp
#include <iostream>

int main() {
  std::cout << "Hello, world!\n";
  return 0;
}
```

Para Python:

```python
print("Hello, world!")
```

10.2. Como crear tablas en latex

Cuadro 1: Cronograma de actividades del proyectos de investigación.

Meses		
Actividades	Primer mes	Segundo mes
Planteamiento de preguntas de investigación	X	
Generación de cadenas de búsqueda	X	
Búsqueda y análisis de resultados	X	
Selección de artículos científicos	X	
Elaboración de una revista científica	X	
Elaboración de informe final		X

Nota. Sujeto a ampliación de plazo en caso de ser necesario.

10.3. Como citar o referenciar en latex

El archivo de referencia de BibTeX se encuentra en la ruta `ref/ref.bib`. Editando el nombre del archivo `.bib` tendrá que editar el contenido del archivo `ref.tex`.
Este es un ejemplo citar un documento[1].

10.4. Como crear pseudocódigo en latex

Use los paquetes `algorithm` y `algpseudocode` para escribir el algoritmo 1

Algorithm 1 Algoritmo para contar si hay más pollos o más perros

```
function NOMBREFUNCION(ga, cho)
    soGa ← 0
    soCho ← 0
    for i ∈ [0, |soGa| − 1] do
        soGa ← soGa + 1
    end for
    for i ∈ [0, |soCho| − 1] do
        soCho ← soCho + 1
    end for
    if soGa > soCho then
        return soGa
    end if
    return soCho
end function
```

10.5. Como crear ecuaciones en látex

Fórmulas matemáticas escritas en 1 línea, luego use el signo de dólar dos veces: $f(x) = x^2 + 2x + 1$. Con la fórmula en una línea separada, escriba 2 pares de signos de dólar:

$$ReLU(x) = \text{máx}(0, x)$$

Más trascendente, el sistema de escritura de ecuaciones debe usar la etiqueta `equation`

$$\begin{aligned} a_1x_1 + a_2x_2 + .. + a_nx_n &= u \\ b_1x_1 + b_2x_2 + .. + b_nx_n &= v \\ c_1x_1 + c_2x_2 + .. + c_nx_n &= w \end{aligned}$$

¡Consulte la ecuación escribiendo en Overleaf!

10.6. Como insertar un gráfico

Puede insertar figura con el siguiente bloque de codigo y puede referenciar usando el código **ref**, ejemplo: Figura 11

Figura 11: Aquí puede agregar una decepción de la figura