UNIVERSIDAD NACIONAL DE MOQUEGUA

Facultad de Ingeniería y Arquitectura

Escuela Profesional de Ingeniería de Sistemas e Informática

Metodos de ordenamiento

Informe laboratorio N° 1

Estudiantes:
Andhree Shilo Chavez Gutierrez

Profesores:
Honorio Apaza Alanoca

25 de mayo de 2023

Índice

1. Introducción

1.1. Motivación sobre los algoritmos

En los tiempos que vivimos donde se generan muchos datos, los procedos diarios es abrumado hasta para una base de datos complicada.Necesitamos una verdadera gestion y en el ordanamiento de datos en valores numeros entran los metodos de ordenamiento que se han convertido en una necesidad vital.

1.2. Objetivos específicos

Un buen método de ordenamiento puede mejorar significativamente la eficiencia y el rendimiento de los programas informáticos. Al ordenar los datos, podemos acceder y buscar la información de manera más rápida y efectiva, lo que se traduce en ahorro de tiempo y recursos. Además, el ordenamiento adecuado es esencial para realizar operaciones como la búsqueda binaria o la eliminación de duplicados, entre otras. Existen numerosos métodos de ordenamiento, cada uno con sus propias características y complejidad. Algunos de los más conocidos son el método de burbuja, el método de inserción, el método de selección, el método de mezcla, el método de quicksort, entre otros. Cada algoritmo tiene ventajas y desventajas en términos de eficiencia, estabilidad y consumo de recursos, por lo que es fundamental conocerlos para elegir el más apropiado según las necesidades y limitaciones del problema que se esté abordando.

1.3. Numeros ramdon

Con la necesidad de generar archivos.txt con numeros aleatorios se creo este codigo

```
import random
rango_inicial = 1
rango_final = 100
num_elementos = 100000
numeros_aleatorios = random.choices(range(rango_inicial, rango_final +
    1), k=num_elementos)
ruta_archivo = "/Users/PC/Desktop/100000numeros.txt"
with open(ruta_archivo, "w") as archivo:
    archivo.write(" ".join(map(str, numeros_aleatorios)))
print("Archivo generado exitosamente.")
```

2. Ejercicios

2.1. METODO BURBUJA

Aqui le presentamos el METODO APLICADO EN PYTHON Y EN C++, este algoritmo genera la funcion del metodo y llama a un archivo.txt ya creado con numeros aleatorios del 0 al 100, mira el nombre del archivo y si no existen numeros que ordenar dira que no es el archivo correcto, tambien dice el tiempo de ejecucion del archivo con librerias

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <chrono>

using namespace std;
using namespace std::chrono;

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

int main() {
    string ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/
    NumerosTxT/Numeros ramdon/100000numeros.txt";

    ifstream archivo(ruta_archivo);
    if (!archivo) {
        cout << "Error al abrir el archivo." << endl;
        return 1;
    }

    vector<int> numeros;
    int numero;
    while (archivo >> numero) {
        numeros.push_back(numero);
    }
    archivo.close();
    auto inicio = high_resolution_clock::now();
    bubbleSort(numeros);
    auto fin = high_resolution_clock::now();
    auto duracion = duration_cast<duration<double>>(fin - inicio);
    cout << "N meros ordenados: ";
    for (int num : numeros) {
```

```
        cout << num << " ";
    }
    cout << endl;
    cout << "Tiempo de ejecuci n: " << duracion.count() << " segundos"
   << endl;

    return 0;
}
```

Para Python:

```
import time
def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(n - 1 - i):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros
   ramdon/100000numeros.txt"
with open(ruta_archivo, "r") as archivo:
    numeros = archivo.read().split()
numeros = list(map(int, numeros))
inicio = time.time()
bubble_sort(numeros)
fin = time.time()
tiempo_ejecucion = fin - inicio
print("N meros ordenados:", numeros)
print("Tiempo de ejecuci n:", tiempo_ejecucion, "segundos")
```

Ruta del archivo en ambos codigos es el lugar donde yo guardo mis codigos.

2.2. METODO COUNTING

Aqui le presentamos el METODO APLICADO EN PYTHON Y EN C++, este algoritmo genera la funcion del metodo y llama a un archivo.txt ya creado con numeros aleatorios del 0 al 100, mira el nombre del archivo y si no existen numeros que ordenar dira que no es el archivo correcto, tambien dice el tiempo de ejecucion del archivo con librerias

```
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <chrono>

using namespace std;
using namespace std::chrono;

void countingSort(vector<int>& arr) {
    int n = arr.size();

```

```
13      int max_value = *max_element(arr.begin(), arr.end());
14      int min_value = *min_element(arr.begin(), arr.end());
15
16      int range = max_value - min_value + 1;
17
18      vector<int> count(range, 0);
19
20      for (int i = 0; i < n; i++) {
21          count[arr[i] - min_value]++;
22      }
23
24      int index = 0;
25      for (int i = 0; i < range; i++) {
26          while (count[i] > 0) {
27              arr[index] = i + min_value;
28              count[i]--;
29              index++;
30          }
31      }
32  }
33
34  int main() {
35
36      string ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/
        NumerosTxT/Numeros ramdon/50000numeros.txt";
37
38      ifstream archivo(ruta_archivo);
39      if (!archivo) {
40          cout << "Error al abrir el archivo." << endl;
41          return 1;
42      }
43
44      vector<int> numeros;
45      int numero;
46      while (archivo >> numero) {
47          numeros.push_back(numero);
48      }
49      archivo.close();
50
51      auto inicio = high_resolution_clock::now();
52      countingSort(numeros);
53      auto fin = high_resolution_clock::now();
54      auto duracion = duration_cast<duration<double>>(fin - inicio);
55
56      cout << "N meros ordenados: ";
57      for (int num : numeros) {
58          cout << num << " ";
59      }
60      cout << endl;
```

```
    cout << "Tiempo de ejecuci n: " << duracion.count() << " segundos"
   << endl;

    return 0;
}
```

Para Python:

```
import time

def counting_sort(arr):
    max_value = max(arr)
    min_value = min(arr)
    range_value = max_value - min_value + 1

    count = [0] * range_value

    for num in arr:
        count[num - min_value] += 1

    sorted_arr = []
    for i in range(range_value):
        while count[i] > 0:
            sorted_arr.append(i + min_value)
            count[i] -= 1

    return sorted_arr

ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros
   ramdon/8000numeros.txt"

numeros = []
with open(ruta_archivo, 'r') as archivo:
    for linea in archivo:
        numeros.extend([int(num) for num in linea.split() if num.strip()
   .isdigit()])

if not numeros:
    print("El archivo no contiene n meros v lidos para ordenar.")
    exit()

inicio = time.time()
numeros_ordenados = counting_sort(numeros)
fin = time.time()
duracion = fin - inicio

print("N meros ordenados:", numeros_ordenados)
print("Tiempo de ejecuci n:", duracion, "segundos")
```

2.3. METODO HEAP SORT

Aqui le presentamos el METODO APLICADO EN PYTHON Y EN C++, este algoritmo genera la funcion del metodo y llama a un archivo.txt ya creado con numeros aleatorios del 0 al 100, mira el nombre del archivo y si no existen numeros que ordenar dira que no es el archivo correcto, tambien dice el tiempo de ejecucion del archivo con librerias

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>
#include <chrono>

void heapify(std::vector<int>& arr, int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[i] < arr[left])
        largest = left;

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

    if (largest != i) {
        std::swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(std::vector<int>& arr) {
    int n = arr.size();

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

    for (int i = n - 1; i >= 0; i--) {
        std::swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}

int main() {
    std::string ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/
    NumerosTxT/Numeros ramdon/100000numeros.txt";

    std::ifstream archivo(ruta_archivo);
    std::string linea;
    std::vector<int> numeros;
```

```
43
44      if (archivo.is_open()) {
45          std::getline(archivo, linea);
46          std::istringstream iss(linea);
47
48          int numero;
49          while (iss >> numero)
50              numeros.push_back(numero);
51
52          archivo.close();
53      }
54      else {
55          std::cout << "No se pudo abrir el archivo." << std::endl;
56          return 0;
57      }
58
59
60      if (numeros.empty()) {
61          std::cout << "El archivo no contiene n meros v lidos para
    ordenar." << std::endl;
62          return 0;
63      }
64
65      auto inicio = std::chrono::steady_clock::now();
66      heapSort(numeros);
67      auto fin = std::chrono::steady_clock::now();
68      double duracion = std::chrono::duration_cast<std::chrono::
    microseconds>(fin - inicio).count() / 1000000.0;
69
70      std::cout << "N meros ordenados: ";
71      for (int num : numeros)
72          std::cout << num << " ";
73      std::cout << std::endl;
74      std::cout << "Tiempo de ejecuci n: " << duracion << " segundos"<<
    ruta_archivo << std::endl;
75
76      return 0;
77  }
```

Para Python:

```
1  import time
2
3  def heapify(arr, n, i):
4      largest = i
5      left = 2 * i + 1
6      right = 2 * i + 2
7
8      if left < n and arr[i] < arr[left]:
9          largest = left
10
```

```
    if right < n and arr[largest] < arr[right]:
        largest = right

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

def heap_sort(arr):
    n = len(arr)

    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)

    return arr

ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros
    ramdon/100000numeros.txt"

numeros = []
with open(ruta_archivo, 'r') as archivo:
    linea = archivo.readline()
    numeros = [int(num) for num in linea.split() if num.strip().isdigit
    ()]

if not numeros:
    print("El archivo no contiene n meros v lidos para ordenar.")
    exit()

inicio = time.time()
numeros_ordenados = heap_sort(numeros)
fin = time.time()
duracion = fin - inicio

print("N meros ordenados:", numeros_ordenados)
print("Tiempo de ejecuci n:", duracion, "segundos")
```

2.4. METODO INSERITON SORT

Aqui le presentamos el METODO APLICADO EN PYTHON Y EN C++, este algoritmo genera la funcion del metodo y llama a un archivo.txt ya creado con numeros aleatorios del 0 al 100, mira el nombre del archivo y si no existen numeros que ordenar dira que no es el archivo correcto, tambien dice el tiempo de ejecucion del archivo con librerias

```
#include <iostream>
#include <fstream>
```

```cpp
#include <sstream>
#include <vector>
#include <algorithm>
#include <chrono>

std::vector<int> insertionSort(std::vector<int>& arr) {
    int n = arr.size();

    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }

        arr[j + 1] = key;
    }

    return arr;
}

int main() {
    std::string ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros ramdon/100000numeros.txt";;

    std::ifstream archivo(ruta_archivo);
    std::string linea;
    std::vector<int> numeros;

    if (archivo.is_open()) {
        std::getline(archivo, linea);
        std::istringstream iss(linea);

        int numero;
        while (iss >> numero)
            numeros.push_back(numero);

        archivo.close();
    }
    else {
        std::cout << "No se pudo abrir el archivo." << std::endl;
        return 0;
    }

    if (numeros.empty()) {
        std::cout << "El archivo no contiene n meros v lidos para ordenar." << std::endl;
        return 0;
```

```
    }

    auto inicio = std::chrono::steady_clock::now();
    std::vector<int> numeros_ordenados = insertionSort(numeros);
    auto fin = std::chrono::steady_clock::now();
    double duracion = std::chrono::duration_cast<std::chrono::
   microseconds>(fin - inicio).count() / 1000000.0;


    std::cout << "N meros ordenados: ";
    for (int num : numeros_ordenados)
        std::cout << num << " ";
    std::cout << std::endl;

    std::cout << "Tiempo de ejecuci n: " << duracion << " segundos" <<
   std::endl;

    return 0;
}
```

Para Python:

```
import time

def insertion_sort(arr):
    n = len(arr)

    for i in range(1, n):
        key = arr[i]
        j = i - 1

        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1

        arr[j + 1] = key

    return arr


ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros
   ramdon/100000numeros.txt"

with open(ruta_archivo, 'r') as archivo:
    numeros = [int(num) for num in archivo.read().split()]

if not numeros:
    print("El archivo no contiene n meros v lidos para ordenar.")
    exit()

inicio = time.time()
```

```
numeros_ordenados = insertion_sort(numeros)
fin = time.time()
duracion = fin - inicio

print("Tiempo de ejecuci n:", duracion, "segundos")
```

2.5. METODO QUICK SORT

Aqui le presentamos el METODO APLICADO EN PYTHON Y EN C++, este algoritmo genera la funcion del metodo y llama a un archivo.txt ya creado con numeros aleatorios del 0 al 100, mira el nombre del archivo y si no existen numeros que ordenar dira que no es el archivo correcto, tambien dice el tiempo de ejecucion del archivo con librerias

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <chrono>

void quickSort(std::vector<int>& arr, int left, int right) {
    if (left < right) {
        int pivot = arr[right];
        int i = left - 1;

        for (int j = left; j < right; j++) {
            if (arr[j] <= pivot) {
                i++;
                std::swap(arr[i], arr[j]);
            }
        }

        std::swap(arr[i + 1], arr[right]);

        int pivotIndex = i + 1;

        quickSort(arr, left, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, right);
    }
}

int main() {
    std::string ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/
   NumerosTxT/Numeros ramdon/500numeros.txt";

    std::ifstream archivo(ruta_archivo);
    std::string linea;
    std::vector<int> numeros;

    if (archivo.is_open()) {
```

```
        std::getline(archivo, linea);
        std::istringstream iss(linea);

        int numero;
        while (iss >> numero)
            numeros.push_back(numero);

        archivo.close();
    }
    else {
        std::cout << "No se pudo abrir el archivo." << std::endl;
        return 0;
    }

    if (numeros.empty()) {
        std::cout << "El archivo no contiene n meros v lidos para ordenar." << std::endl;
        return 0;
    }

    auto inicio = std::chrono::steady_clock::now();
    quickSort(numeros, 0, numeros.size() - 1);
    auto fin = std::chrono::steady_clock::now();
    double duracion = std::chrono::duration_cast<std::chrono::microseconds>(fin - inicio).count() / 1000000.0;


    std::cout << "N meros ordenados: ";
    for (int num : numeros)
        std::cout << num << " ";
    std::cout << std::endl;

    std::cout << "Tiempo de ejecuci n: " << duracion << " segundos" << std::endl;

    return 0;
}
```

Para Python:

```
import time

def quick_sort(arr):
    if len(arr) <= 1:
        return arr

    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]

```

```
    return quick_sort(left) + middle + quick_sort(right)

ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros
    ramdon/60000numeros.txt"


with open(ruta_archivo, 'r') as archivo:
    numeros = [int(num) for num in archivo.read().split()]


if not numeros:
    print("El archivo no contiene n meros v lidos para ordenar.")
    exit()

inicio = time.time()
numeros_ordenados = quick_sort(numeros)
fin = time.time()
duracion = fin - inicio


print("N meros ordenados:", numeros_ordenados)
print("Tiempo de ejecuci n:", duracion, "segundos")
```

2.6. METODO SELECTION SORT

Aqui le presentamos el METODO APLICADO EN PYTHON Y EN C++, este algoritmo genera la funcion del metodo y llama a un archivo.txt ya creado con numeros aleatorios del 0 al 100, mira el nombre del archivo y si no existen numeros que ordenar dira que no es el archivo correcto, tambien dice el tiempo de ejecucion del archivo con librerias

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <chrono>

void selectionSort(std::vector<int>& arr) {
    int n = arr.size();

    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;
        }

        std::swap(arr[i], arr[minIndex]);
    }
```

```
}

int main() {
    std::string ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros ramdon/80000numeros.txt";


    std::ifstream archivo(ruta_archivo);
    std::string linea;
    std::vector<int> numeros;

    if (archivo.is_open()) {
        std::getline(archivo, linea);
        std::istringstream iss(linea);

        int numero;
        while (iss >> numero)
            numeros.push_back(numero);

        archivo.close();
    }
    else {
        std::cout << "No se pudo abrir el archivo." << std::endl;
        return 0;
    }


    if (numeros.empty()) {
        std::cout << "El archivo no contiene n meros v lidos para ordenar." << std::endl;
        return 0;
    }


    auto inicio = std::chrono::steady_clock::now();
    selectionSort(numeros);
    auto fin = std::chrono::steady_clock::now();
    double duracion = std::chrono::duration_cast<std::chrono::microseconds>(fin - inicio).count() / 1000000.0;

    std::cout << "N meros ordenados: ";
    for (int num : numeros)
        std::cout << num << " ";
    std::cout << std::endl;

    std::cout << "Tiempo de ejecuci n: " << duracion << " segundos" << std::endl;

    return 0;
}
```

Para Python:

```python
import time

def selection_sort(arr):
    n = len(arr)

    for i in range(n - 1):
        min_index = i

        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j

        arr[i], arr[min_index] = arr[min_index], arr[i]

ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros ramdon/100000numeros.txt"


with open(ruta_archivo, 'r') as archivo:
    numeros = [int(numero) for numero in archivo.read().split()]


if not numeros:
    print("El archivo no contiene n meros v lidos para ordenar.")
    exit()


inicio = time.time()
selection_sort(numeros)
fin = time.time()
duracion = fin - inicio


print("N meros ordenados:", numeros)
print("Tiempo de ejecuci n:", duracion, "segundos")
```

2.7. METODO MERGE SORT

Aqui le presentamos el METODO APLICADO EN PYTHON Y EN C++, este algoritmo genera la funcion del metodo y llama a un archivo.txt ya creado con numeros aleatorios del 0 al 100, mira el nombre del archivo y si no existen numeros que ordenar dira que no es el archivo correcto, tambien dice el tiempo de ejecucion del archivo con librerias

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <chrono>
```

```
6
7  void merge(std::vector<int>& arr, int left, int middle, int right) {
8      int n1 = middle - left + 1;
9      int n2 = right - middle;
10
11     std::vector<int> leftArr(n1);
12     std::vector<int> rightArr(n2);
13
14     for (int i = 0; i < n1; i++)
15         leftArr[i] = arr[left + i];
16     for (int j = 0; j < n2; j++)
17         rightArr[j] = arr[middle + 1 + j];
18
19     int i = 0;
20     int j = 0;
21     int k = left;
22
23     while (i < n1 && j < n2) {
24         if (leftArr[i] <= rightArr[j]) {
25             arr[k] = leftArr[i];
26             i++;
27         }
28         else {
29             arr[k] = rightArr[j];
30             j++;
31         }
32         k++;
33     }
34
35     while (i < n1) {
36         arr[k] = leftArr[i];
37         i++;
38         k++;
39     }
40
41     while (j < n2) {
42         arr[k] = rightArr[j];
43         j++;
44         k++;
45     }
46 }
47
48 void mergeSort(std::vector<int>& arr, int left, int right) {
49     if (left < right) {
50         int middle = left + (right - left) / 2;
51
52         mergeSort(arr, left, middle);
53         mergeSort(arr, middle + 1, right);
54
55         merge(arr, left, middle, right);
```

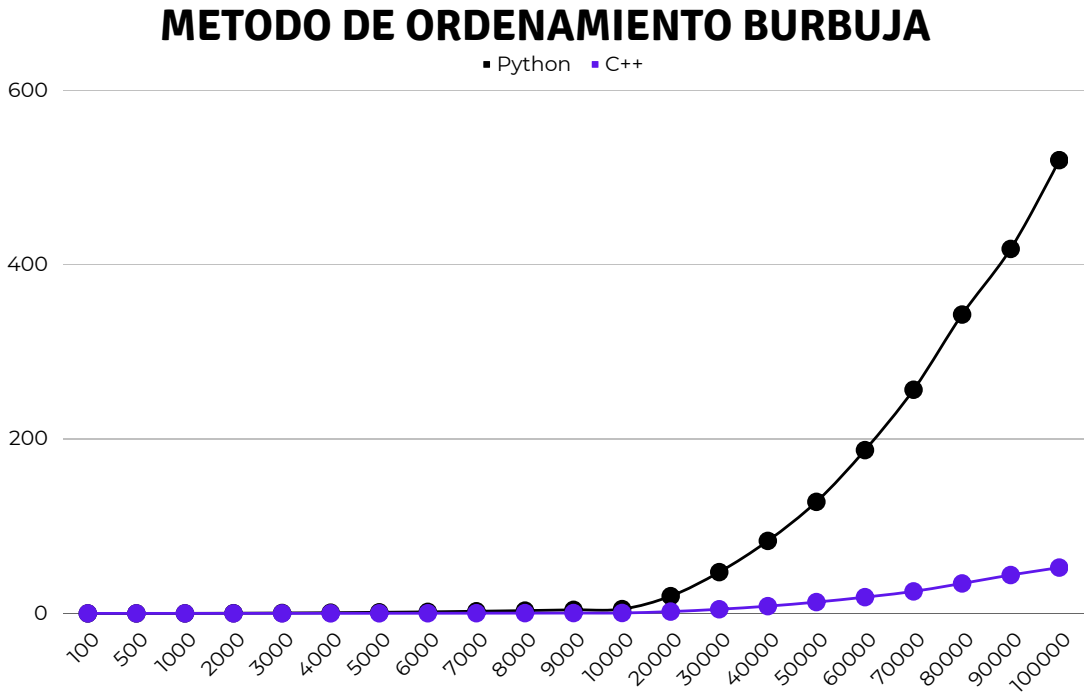
```
    }
}

int main() {
    std::string ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros ramdon/100000numeros.txt";

    std::ifstream archivo(ruta_archivo);
    std::string linea;
    std::vector<int> numeros;

    if (archivo.is_open()) {
        std::getline(archivo, linea);
        std::istringstream iss(linea);

        int numero;
        while (iss >> numero)
            numeros.push_back(numero);

        archivo.close();
    }
    else {
        std::cout << "No se pudo abrir el archivo." << std::endl;
        return 0;
    }

    if (numeros.empty()) {
        std::cout << "El archivo no contiene n meros v lidos para ordenar." << std::endl;
        return 0;
    }

    auto inicio = std::chrono::steady_clock::now();
    mergeSort(numeros, 0, numeros.size() - 1);
    auto fin = std::chrono::steady_clock::now();
    double duracion = std::chrono::duration_cast<std::chrono::microseconds>(fin - inicio).count() / 1000000.0;

    std::cout << "N meros ordenados: ";
    for (int num : numeros)
        std::cout << num << " ";
    std::cout << std::endl;

    std::cout << "Tiempo de ejecuci n: " << duracion << " segundos" << std::endl;

    return 0;
}
```

Para Python:

```
import time

def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    middle = len(arr) // 2
    left = merge_sort(arr[:middle])
    right = merge_sort(arr[middle:])

    return merge(left, right)

def merge(left, right):
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])
    return result

ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros
    ramdon/80000numeros.txt"

with open(ruta_archivo, 'r') as archivo:
    numeros = [int(num) for num in archivo.read().split()]

if not numeros:
    print("El archivo no contiene n meros  v lidos para ordenar.")
    exit()

inicio = time.time()
numeros_ordenados = merge_sort(numeros)
fin = time.time()
duracion = fin - inicio

print("N meros ordenados:", numeros_ordenados)
print("Tiempo de ejecuci n:", duracion, "segundos")
```

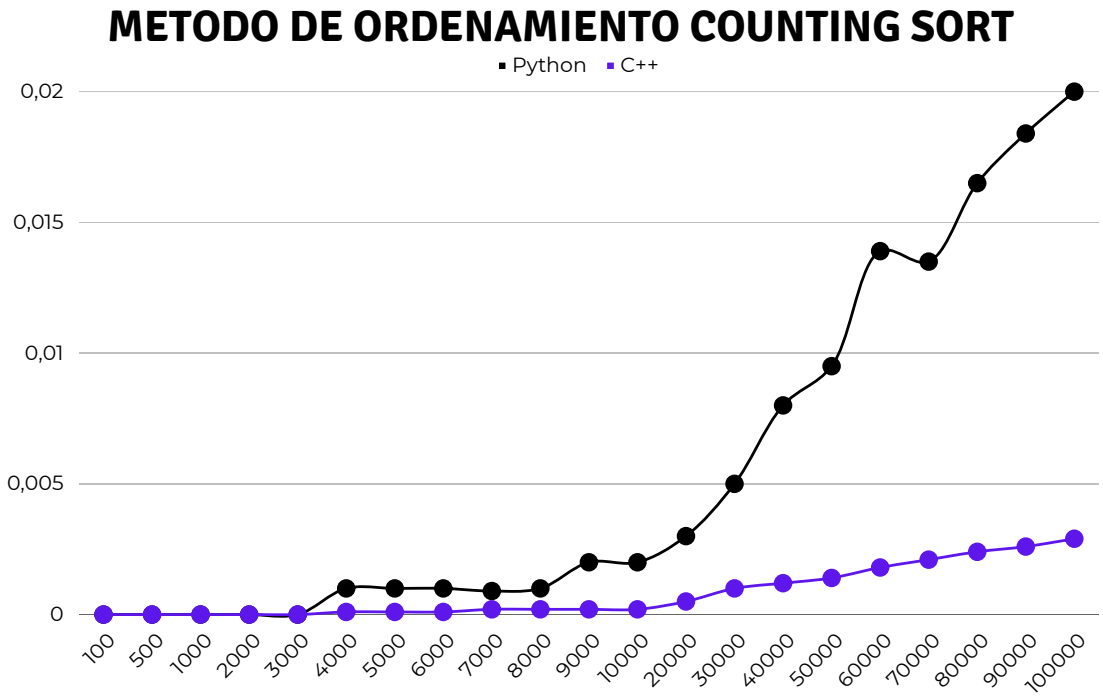
3. Resultados

3.1. Resultado del METODO BURBUJA



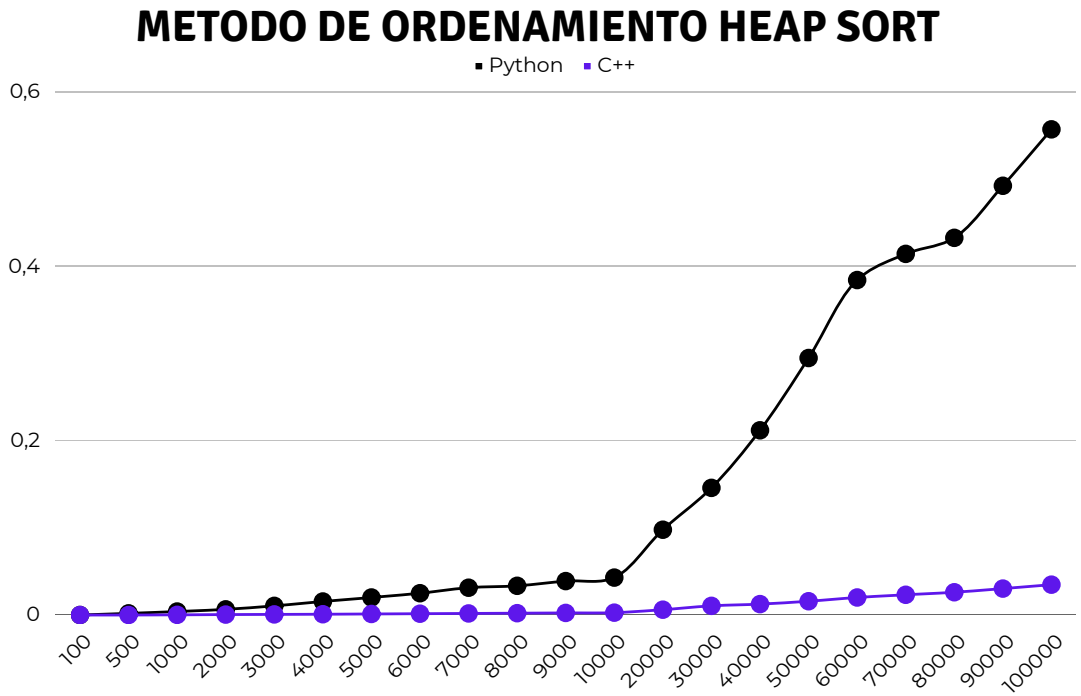
Metodo comparando numeros de (100,500,1000,2000...10000,20000....100000)

3.2. Resultado del METODO COUNTING SORT



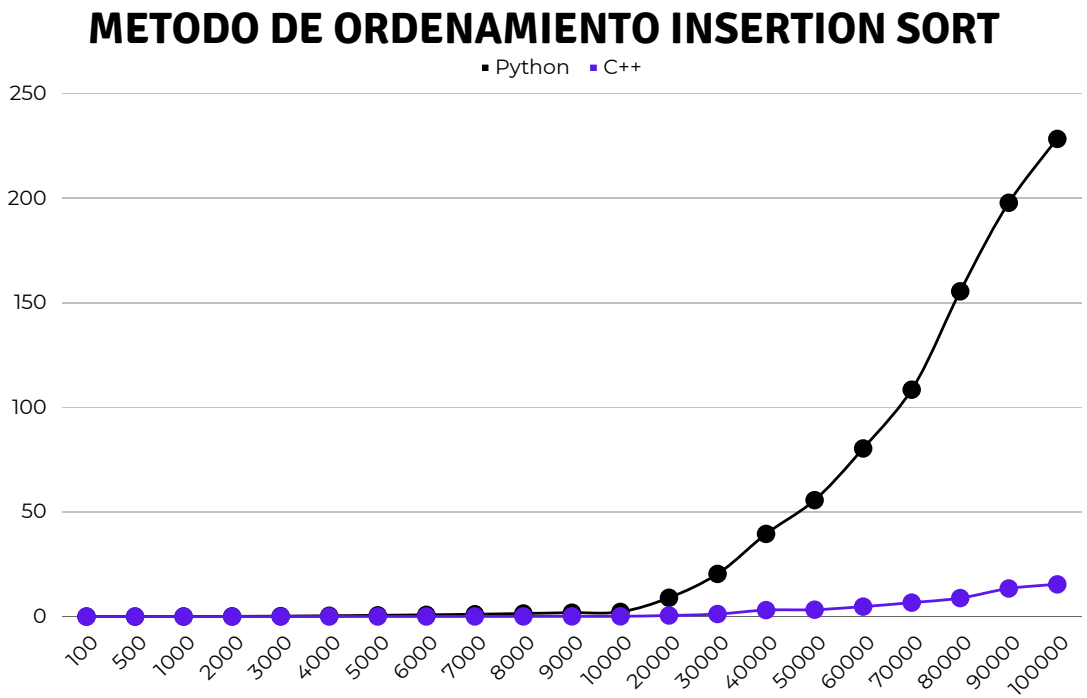
Metodo comparando numeros de (100,500,1000,2000...10000,20000....100000)

3.3. Resultado del METODO HEAP SORT



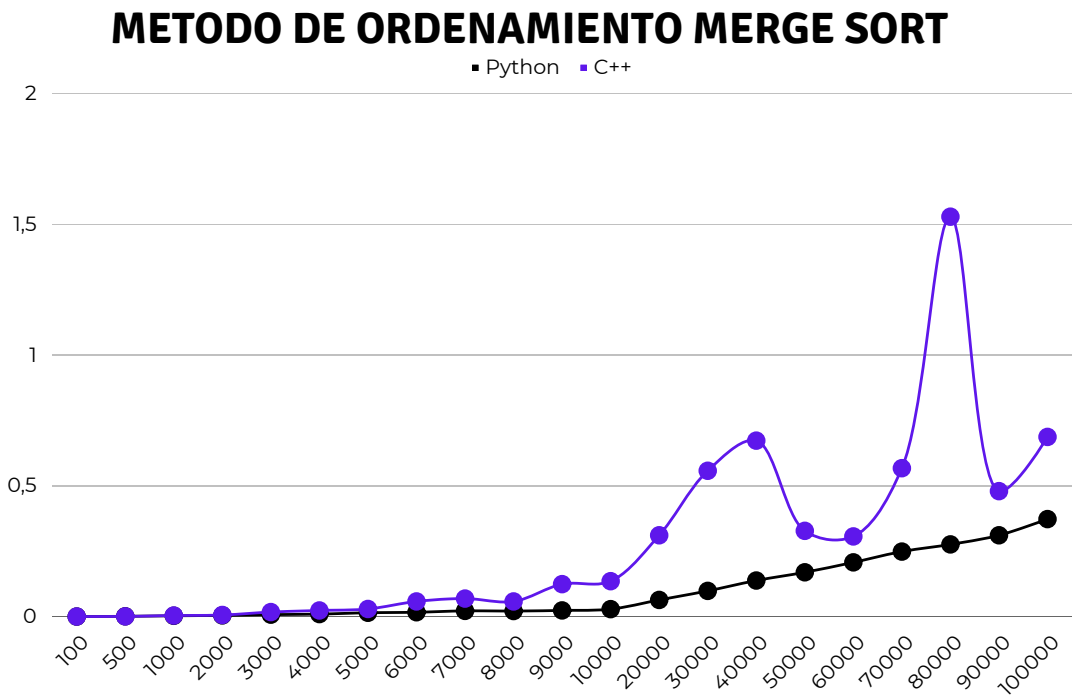
Metodo comparando numeros de (100,500,1000,2000...10000,20000....100000)

3.4. Resultado del METODO INSERTION SORT



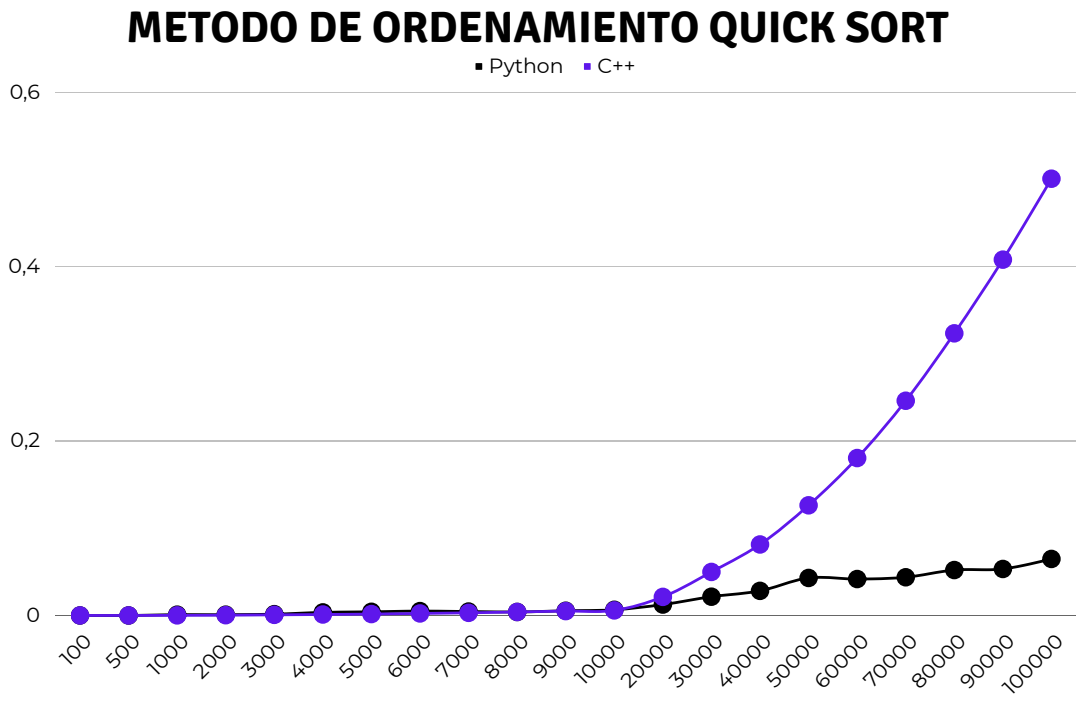
Metodo comparando numeros de (100,500,1000,2000...10000,20000....100000)

3.5. Resultado del METODO MERGE SORT



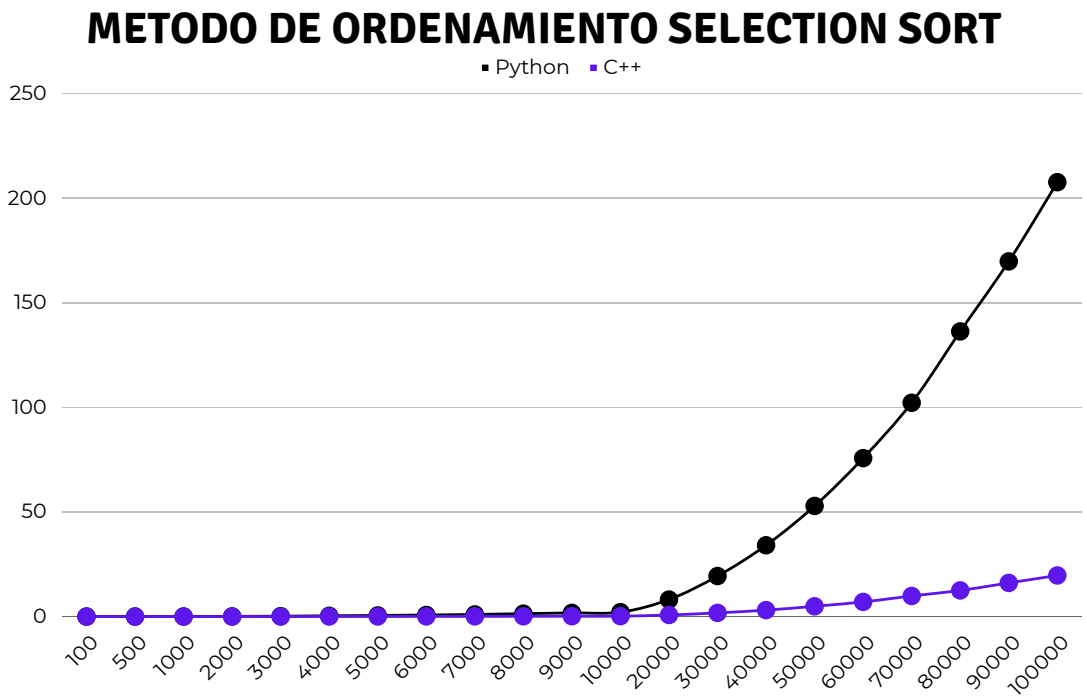
Metodo comparando numeros de (100,500,1000,2000...10000,20000....100000)

3.6. Resultado del METODO QUICK SORT



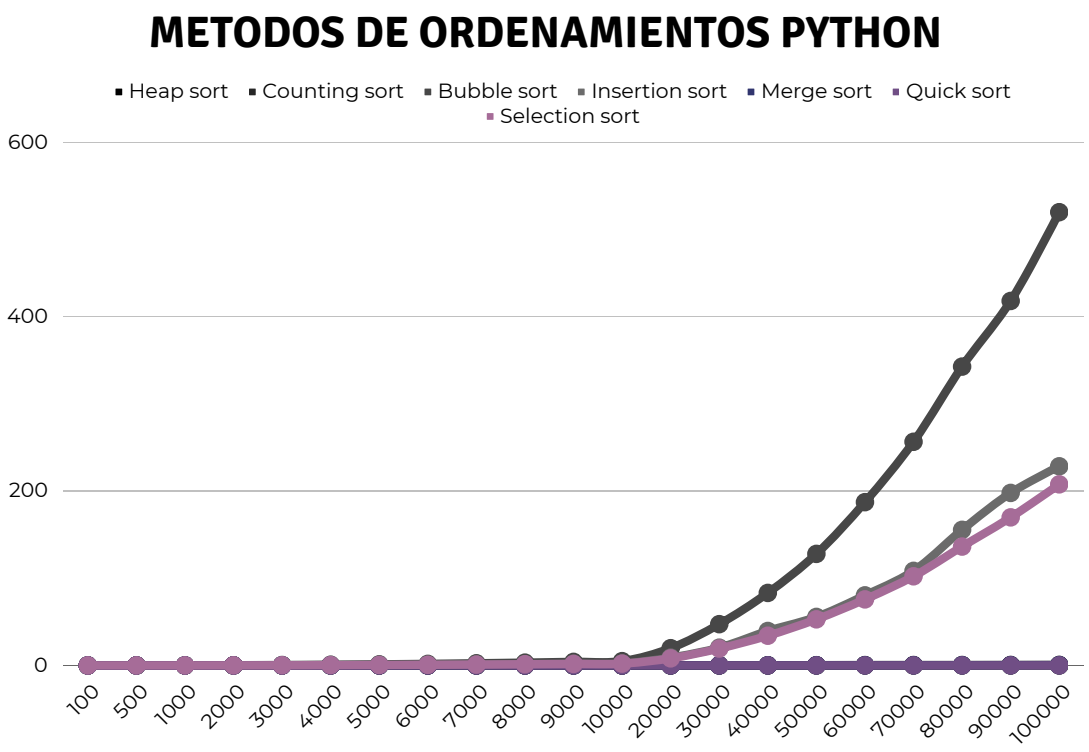
Metodo comparando numeros de (100,500,1000,2000...10000,20000....100000)

3.7. Resultado del METODO SELECTION SORT



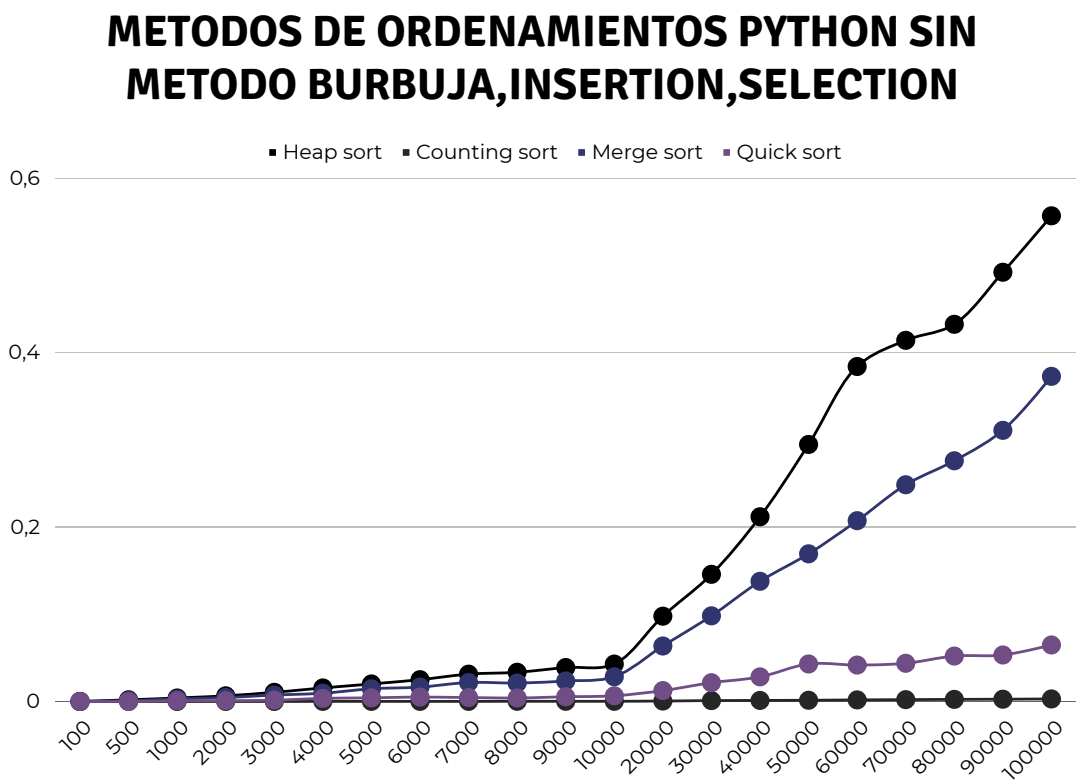
Metodo comparando numeros de (100,500,1000,2000...10000,20000....100000)

3.8. Resultado de todos los metodos en PYTHON



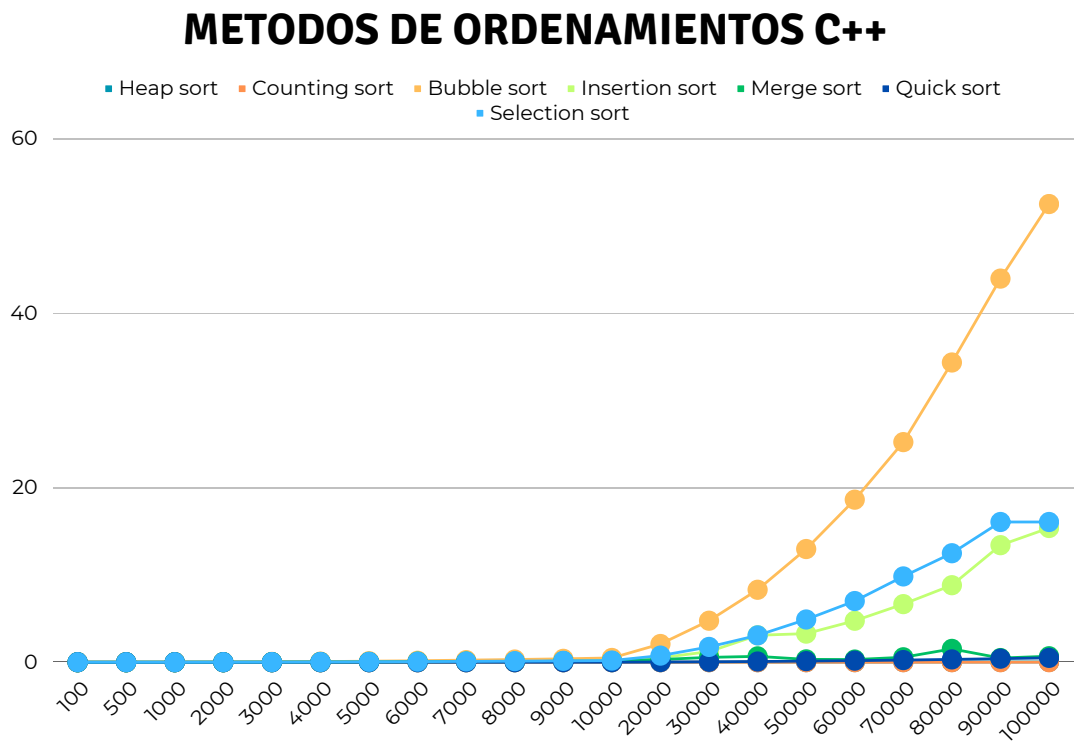
Comparando los metodos de ordenamiento

3.9. Resultado de metodos cortos de PYTHON



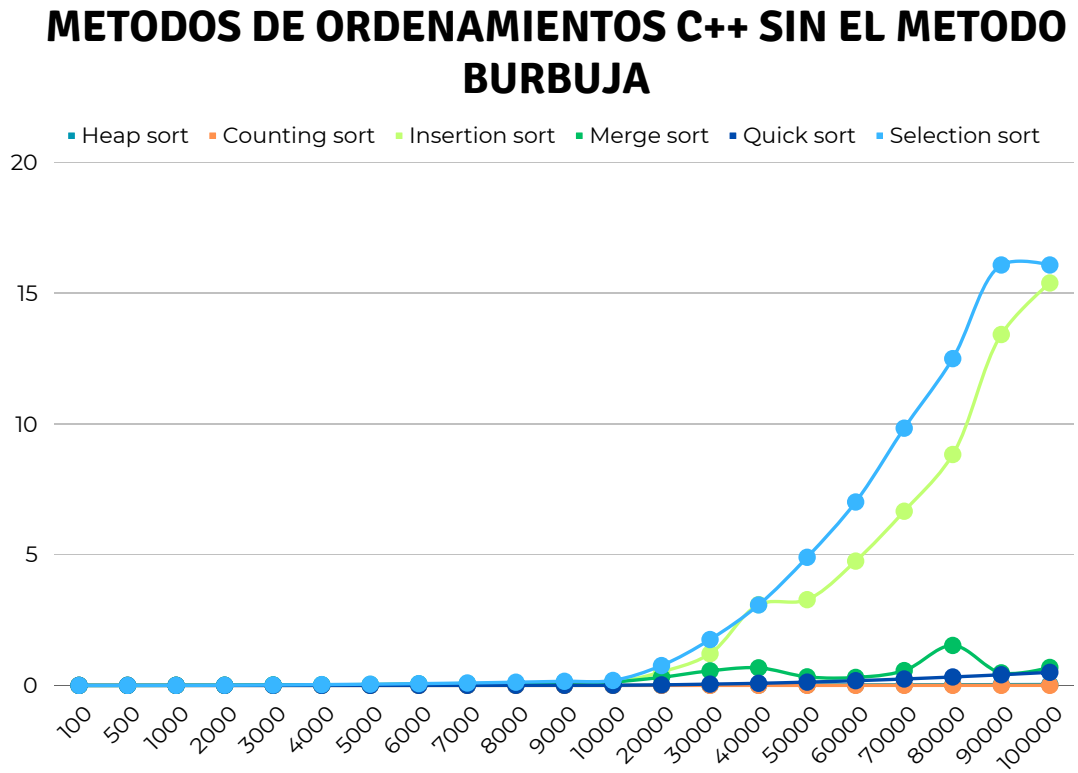
Descarta algunos metodos de ordenamiento muy grandes)

3.10. Resultado de todos los metodos en C++



Comparando los metodos de ordenamiento

3.11. Resultado de metodos cortos de C++



Descarta algunos metodos de ordenamiento muy grandes)

4. Conclusiones

Es beneficioso aprender y utilizar métodos de ordenamiento debido a su capacidad para organizar eficientemente grandes volúmenes de datos, mejorar la productividad, optimizar el uso de recursos, brindar una mejor experiencia al usuario y facilitar el mantenimiento del código. Dominar estos algoritmos nos permite gestionar y procesar información de manera efectiva, tanto a nivel personal como profesional, en un mundo donde la cantidad de datos generados es gigantesca . Link para la pagina GitHub donde se podran ver los codigos y entregas del proyecto.
Repositorio de GITHUB