

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

<b>1. Introducción</b>	<b>2</b>
1.1. Motivación sobre los algoritmos . . . . .	2
1.2. Objetivos específicos . . . . .	2
1.3. Numeros ramdon . . . . .	2
<b>2. Ejercicios</b>	<b>3</b>
2.1. METODO BURBUJA . . . . .	3
2.2. METODO COUNTING . . . . .	4
2.3. METODO HEAP SORT . . . . .	7
2.4. METODO INSERTION SORT . . . . .	9
2.5. METODO QUICK SORT . . . . .	12
2.6. METODO SELECTION SORT . . . . .	14
2.7. METODO MERGE SORT . . . . .	16
<b>3. Resultados</b>	<b>20</b>
3.1. Resultado del METODO BURBUJA . . . . .	20
3.2. Resultado del METODO COUNTING SORT . . . . .	21
3.3. Resultado del METODO HEAP SORT . . . . .	22
3.4. Resultado del METODO INSERTION SORT . . . . .	23
3.5. Resultado del METODO MERGE SORT . . . . .	24
3.6. Resultado del METODO QUICK SORT . . . . .	25
3.7. Resultado del METODO SELECTION SORT . . . . .	26
3.8. Resultado de todos los metodos en PYTHON . . . . .	27
3.9. Resultado de metodos cortos de PYTHON . . . . .	28
3.10. Resultado de todos los metodos en C++ . . . . .	29
3.11. Resultado de metodos cortos de C++ . . . . .	30
<b>4. Conclusiones</b>	<b>31</b>

# 1. Introducción

## 1.1. Motivación sobre los algoritmos

*En los tiempos que vivimos donde se generan muchos datos, los procesos diarios se abrumado hasta para una base de datos complicada. Necesitamos una verdadera gestión y en el ordenamiento de datos en valores numéricos entran los métodos 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. Números aleatorios

Con la necesidad de generar archivos.txt con números aleatorios se creó este código

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

## 2. Ejercicios

### 2.1. METODO BURBUJA

Aquí le presentamos el METODO APLICADO EN PYTHON Y EN C++, este algoritmo genera la función del método y llama a un archivo.txt ya creado con números aleatorios del 0 al 100, mira el nombre del archivo y si no existen números que ordenar dirá que no es el archivo correcto, también dice el tiempo de ejecución del archivo con librerías

```
1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <chrono>
5
6 using namespace std;
7 using namespace std::chrono;
8
9 void bubbleSort(vector<int>& arr) {
10     int n = arr.size();
11     for (int i = 0; i < n - 1; i++) {
12         for (int j = 0; j < n - 1 - i; j++) {
13             if (arr[j] > arr[j + 1]) {
14                 swap(arr[j], arr[j + 1]);
15             }
16         }
17     }
18 }
19
20 int main() {
21     string ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/
NumerosTxt/Numeros ramdon/100000numeros.txt";
22
23     ifstream archivo(ruta_archivo);
24     if (!archivo) {
25         cout << "Error al abrir el archivo." << endl;
26         return 1;
27     }
28
29     vector<int> numeros;
30     int numero;
31     while (archivo >> numero) {
32         numeros.push_back(numero);
33     }
34     archivo.close();
35     auto inicio = high_resolution_clock::now();
36     bubbleSort(numeros);
37     auto fin = high_resolution_clock::now();
38     auto duracion = duration_cast<duration<double>>(fin - inicio);
39     cout << "N meros ordenados: ";
40     for (int num : numeros) {
```

```
41     cout << num << " ";
42 }
43 cout << endl;
44 cout << "Tiempo de ejecuci n: " << duracion.count() << " segundos"
45 << endl;
46
47 return 0;
48 }
```

Para Python:

```
1 import time
2 def bubble_sort(arr):
3     n = len(arr)
4     for i in range(n - 1):
5         for j in range(n - 1 - i):
6             if arr[j] > arr[j + 1]:
7                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
8 ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros
9 ramdon/100000numeros.txt"
10 with open(ruta_archivo, "r") as archivo:
11     numeros = archivo.read().split()
12 numeros = list(map(int, numeros))
13 inicio = time.time()
14 bubble_sort(numeros)
15 fin = time.time()
16 tiempo_ejecucion = fin - inicio
17 print("N meros ordenados:", numeros)
18 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

```
1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <chrono>
6
7 using namespace std;
8 using namespace std::chrono;
9
10 void countingSort(vector<int>& arr) {
11     int n = arr.size();
12 }
```

```
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;
```

```
61     cout << "Tiempo de ejecuci n: " << duracion.count() << " segundos"
    << endl;
62
63     return 0;
64 }
```

Para Python:

```
1 import time
2
3 def counting_sort(arr):
4     max_value = max(arr)
5     min_value = min(arr)
6     range_value = max_value - min_value + 1
7
8     count = [0] * range_value
9
10    for num in arr:
11        count[num - min_value] += 1
12
13    sorted_arr = []
14    for i in range(range_value):
15        while count[i] > 0:
16            sorted_arr.append(i + min_value)
17            count[i] -= 1
18
19    return sorted_arr
20
21 ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros
    ramdon/8000numeros.txt"
22
23 numeros = []
24 with open(ruta_archivo, 'r') as archivo:
25     for linea in archivo:
26         numeros.extend([int(num) for num in linea.split() if num.strip()
            .isdigit()])
27
28 if not numeros:
29     print("El archivo no contiene n meros v lidos para ordenar.")
30     exit()
31
32 inicio = time.time()
33 numeros_ordenados = counting_sort(numeros)
34 fin = time.time()
35 duracion = fin - inicio
36
37 print("N meros ordenados:", numeros_ordenados)
38 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

```
1 #include <iostream>
2 #include <fstream>
3 #include <sstream>
4 #include <vector>
5 #include <algorithm>
6 #include <chrono>
7
8 void heapify(std::vector<int>& arr, int n, int i) {
9     int largest = i;
10    int left = 2 * i + 1;
11    int right = 2 * i + 2;
12
13    if (left < n && arr[i] < arr[left])
14        largest = left;
15
16    if (right < n && arr[largest] < arr[right])
17        largest = right;
18
19    if (largest != i) {
20        std::swap(arr[i], arr[largest]);
21        heapify(arr, n, largest);
22    }
23 }
24
25 void heapSort(std::vector<int>& arr) {
26     int n = arr.size();
27
28     for (int i = n / 2 - 1; i >= 0; i--)
29         heapify(arr, n, i);
30
31     for (int i = n - 1; i >= 0; i--) {
32         std::swap(arr[0], arr[i]);
33         heapify(arr, i, 0);
34     }
35 }
36
37 int main() {
38     std::string ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/
39     NumerosTxT/Numeros ramdon/100000numeros.txt";
40
41     std::ifstream archivo(ruta_archivo);
42     std::string linea;
43     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
```

```
11     if right < n and arr[largest] < arr[right]:
12         largest = right
13
14     if largest != i:
15         arr[i], arr[largest] = arr[largest], arr[i]
16         heapify(arr, n, largest)
17
18 def heap_sort(arr):
19     n = len(arr)
20
21     for i in range(n // 2 - 1, -1, -1):
22         heapify(arr, n, i)
23
24     for i in range(n - 1, 0, -1):
25         arr[i], arr[0] = arr[0], arr[i]
26         heapify(arr, i, 0)
27
28     return arr
29
30 ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxt/Numeros
31     ramdon/100000numeros.txt"
32
33 numeros = []
34 with open(ruta_archivo, 'r') as archivo:
35     linea = archivo.readline()
36     numeros = [int(num) for num in linea.split() if num.strip().isdigit
37     ()]
38
39 if not numeros:
40     print("El archivo no contiene n meros v lidos para ordenar.")
41     exit()
42
43 inicio = time.time()
44 numeros_ordenados = heap_sort(numeros)
45 fin = time.time()
46 duracion = fin - inicio
47
48 print("N meros ordenados:", numeros_ordenados)
49 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

```
1 #include <iostream>
2 #include <fstream>
```

```
3 #include <sstream>
4 #include <vector>
5 #include <algorithm>
6 #include <chrono>
7
8 std::vector<int> insertionSort(std::vector<int>& arr) {
9     int n = arr.size();
10
11     for (int i = 1; i < n; i++) {
12         int key = arr[i];
13         int j = i - 1;
14
15         while (j >= 0 && arr[j] > key) {
16             arr[j + 1] = arr[j];
17             j--;
18         }
19
20         arr[j + 1] = key;
21     }
22
23     return arr;
24 }
25
26 int main() {
27     std::string ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/
NumerosTxT/Numeros ramdon/100000numeros.txt";
28
29     std::ifstream archivo(ruta_archivo);
30     std::string linea;
31     std::vector<int> numeros;
32
33     if (archivo.is_open()) {
34         std::getline(archivo, linea);
35         std::istringstream iss(linea);
36
37         int numero;
38         while (iss >> numero)
39             numeros.push_back(numero);
40
41         archivo.close();
42     }
43     else {
44         std::cout << "No se pudo abrir el archivo." << std::endl;
45         return 0;
46     }
47
48     if (numeros.empty()) {
49         std::cout << "El archivo no contiene n meros v lidos para
ordenar." << std::endl;
50         return 0;
```

```
51     }
52
53     auto inicio = std::chrono::steady_clock::now();
54     std::vector<int> numeros_ordenados = insertionSort(numeros);
55     auto fin = std::chrono::steady_clock::now();
56     double duracion = std::chrono::duration_cast<std::chrono::
microseconds>(fin - inicio).count() / 1000000.0;
57
58
59     std::cout << "N meros ordenados: ";
60     for (int num : numeros_ordenados)
61         std::cout << num << " ";
62     std::cout << std::endl;
63
64     std::cout << "Tiempo de ejecuci n: " << duracion << " segundos" <<
std::endl;
65
66     return 0;
67 }
```

Para Python:

```
1 import time
2
3 def insertion_sort(arr):
4     n = len(arr)
5
6     for i in range(1, n):
7         key = arr[i]
8         j = i - 1
9
10        while j >= 0 and arr[j] > key:
11            arr[j + 1] = arr[j]
12            j -= 1
13
14        arr[j + 1] = key
15
16    return arr
17
18
19 ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros
ramdon/100000numeros.txt"
20
21 with open(ruta_archivo, 'r') as archivo:
22     numeros = [int(num) for num in archivo.read().split()]
23
24 if not numeros:
25     print("El archivo no contiene n meros v lidos para ordenar.")
26     exit()
27
28 inicio = time.time()
```

```
29 numeros_ordenados = insertion_sort(numeros)
30 fin = time.time()
31 duracion = fin - inicio
32
33 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

```
1 #include <iostream>
2 #include <fstream>
3 #include <sstream>
4 #include <vector>
5 #include <chrono>
6
7 void quickSort(std::vector<int>& arr, int left, int right) {
8     if (left < right) {
9         int pivot = arr[right];
10        int i = left - 1;
11
12        for (int j = left; j < right; j++) {
13            if (arr[j] <= pivot) {
14                i++;
15                std::swap(arr[i], arr[j]);
16            }
17        }
18
19        std::swap(arr[i + 1], arr[right]);
20
21        int pivotIndex = i + 1;
22
23        quickSort(arr, left, pivotIndex - 1);
24        quickSort(arr, pivotIndex + 1, right);
25    }
26 }
27
28 int main() {
29     std::string ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/
NumerosTxT/Numeros ramdon/500numeros.txt";
30
31     std::ifstream archivo(ruta_archivo);
32     std::string linea;
33     std::vector<int> numeros;
34
35     if (archivo.is_open()) {
```

```
36     std::getline(archivo, linea);
37     std::istringstream iss(linea);
38
39     int numero;
40     while (iss >> numero)
41         numeros.push_back(numero);
42
43     archivo.close();
44 }
45 else {
46     std::cout << "No se pudo abrir el archivo." << std::endl;
47     return 0;
48 }
49
50 if (numeros.empty()) {
51     std::cout << "El archivo no contiene n meros v lidos para
ordenar." << std::endl;
52     return 0;
53 }
54
55 auto inicio = std::chrono::steady_clock::now();
56 quickSort(numeros, 0, numeros.size() - 1);
57 auto fin = std::chrono::steady_clock::now();
58 double duracion = std::chrono::duration_cast<std::chrono::
microseconds>(fin - inicio).count() / 1000000.0;
59
60
61 std::cout << "N meros ordenados: ";
62 for (int num : numeros)
63     std::cout << num << " ";
64 std::cout << std::endl;
65
66 std::cout << "Tiempo de ejecuci n: " << duracion << " segundos" <<
std::endl;
67
68 return 0;
69 }
```

Para Python:

```
1 import time
2
3 def quick_sort(arr):
4     if len(arr) <= 1:
5         return arr
6
7     pivot = arr[len(arr) // 2]
8     left = [x for x in arr if x < pivot]
9     middle = [x for x in arr if x == pivot]
10    right = [x for x in arr if x > pivot]
11
```

```
12     return quick_sort(left) + middle + quick_sort(right)
13
14 ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros
    ramdon/60000numeros.txt"
15
16
17 with open(ruta_archivo, 'r') as archivo:
18     numeros = [int(num) for num in archivo.read().split()]
19
20
21 if not numeros:
22     print("El archivo no contiene n meros v lidos para ordenar.")
23     exit()
24
25 inicio = time.time()
26 numeros_ordenados = quick_sort(numeros)
27 fin = time.time()
28 duracion = fin - inicio
29
30
31 print("N meros ordenados:", numeros_ordenados)
32 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

```
1 #include <iostream>
2 #include <fstream>
3 #include <sstream>
4 #include <vector>
5 #include <chrono>
6
7 void selectionSort(std::vector<int>& arr) {
8     int n = arr.size();
9
10    for (int i = 0; i < n - 1; i++) {
11        int minIndex = i;
12
13        for (int j = i + 1; j < n; j++) {
14            if (arr[j] < arr[minIndex])
15                minIndex = j;
16        }
17
18        std::swap(arr[i], arr[minIndex]);
19    }
```

```
20 }
21
22 int main() {
23     std::string ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/
NumerosTxT/Numeros ramdon/80000numeros.txt";
24
25
26     std::ifstream archivo(ruta_archivo);
27     std::string linea;
28     std::vector<int> numeros;
29
30     if (archivo.is_open()) {
31         std::getline(archivo, linea);
32         std::istringstream iss(linea);
33
34         int numero;
35         while (iss >> numero)
36             numeros.push_back(numero);
37
38         archivo.close();
39     }
40     else {
41         std::cout << "No se pudo abrir el archivo." << std::endl;
42         return 0;
43     }
44
45
46     if (numeros.empty()) {
47         std::cout << "El archivo no contiene n meros v lidos para
ordenar." << std::endl;
48         return 0;
49     }
50
51
52     auto inicio = std::chrono::steady_clock::now();
53     selectionSort(numeros);
54     auto fin = std::chrono::steady_clock::now();
55     double duracion = std::chrono::duration_cast<std::chrono::
microseconds>(fin - inicio).count() / 1000000.0;
56
57     std::cout << "N meros ordenados: ";
58     for (int num : numeros)
59         std::cout << num << " ";
60     std::cout << std::endl;
61
62     std::cout << "Tiempo de ejecuci n: " << duracion << " segundos" <<
std::endl;
63
64     return 0;
65 }
```



Para Python:

```
1 import time
2
3 def selection_sort(arr):
4     n = len(arr)
5
6     for i in range(n - 1):
7         min_index = i
8
9         for j in range(i + 1, n):
10             if arr[j] < arr[min_index]:
11                 min_index = j
12
13         arr[i], arr[min_index] = arr[min_index], arr[i]
14
15 ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxT/Numeros
16             ramdon/100000numeros.txt"
17
18 with open(ruta_archivo, 'r') as archivo:
19     numeros = [int(numero) for numero in archivo.read().split()]
20
21
22 if not numeros:
23     print("El archivo no contiene n meros v lidos para ordenar.")
24     exit()
25
26
27 inicio = time.time()
28 selection_sort(numeros)
29 fin = time.time()
30 duracion = fin - inicio
31
32
33 print("N meros ordenados:", numeros)
34 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

```
1 #include <iostream>
2 #include <fstream>
3 #include <sstream>
4 #include <vector>
5 #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);
56     }
```

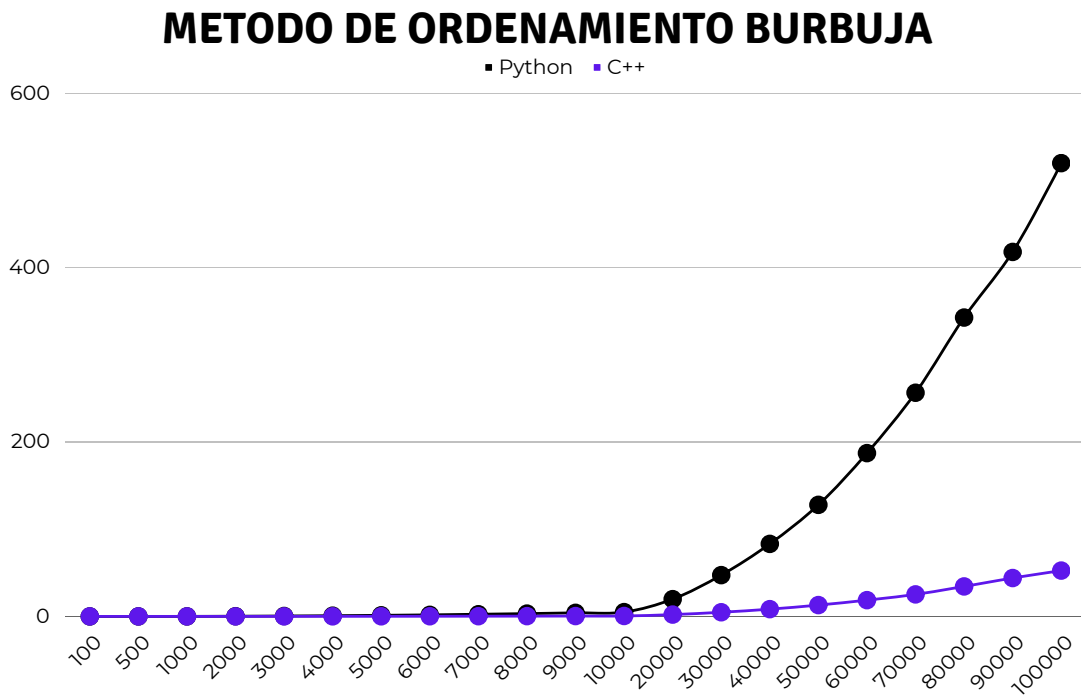
```
56     }
57 }
58
59 int main() {
60     std::string ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/
NumerosTxT/Numeros ramdon/100000numeros.txt";
61
62     std::ifstream archivo(ruta_archivo);
63     std::string linea;
64     std::vector<int> numeros;
65
66     if (archivo.is_open()) {
67         std::getline(archivo, linea);
68         std::istringstream iss(linea);
69
70         int numero;
71         while (iss >> numero)
72             numeros.push_back(numero);
73
74         archivo.close();
75     }
76     else {
77         std::cout << "No se pudo abrir el archivo." << std::endl;
78         return 0;
79     }
80
81     if (numeros.empty()) {
82         std::cout << "El archivo no contiene n meros v lidos para
ordenar." << std::endl;
83         return 0;
84     }
85
86     auto inicio = std::chrono::steady_clock::now();
87     mergeSort(numeros, 0, numeros.size() - 1);
88     auto fin = std::chrono::steady_clock::now();
89     double duracion = std::chrono::duration_cast<std::chrono::
microseconds>(fin - inicio).count() / 1000000.0;
90
91     std::cout << "N meros ordenados: ";
92     for (int num : numeros)
93         std::cout << num << " ";
94     std::cout << std::endl;
95
96     std::cout << "Tiempo de ejecuci n: " << duracion << " segundos" <<
std::endl;
97
98     return 0;
99 }
```

Para Python:

```
1 import time
2
3 def merge_sort(arr):
4     if len(arr) <= 1:
5         return arr
6
7     middle = len(arr) // 2
8     left = merge_sort(arr[:middle])
9     right = merge_sort(arr[middle:])
10
11     return merge(left, right)
12
13 def merge(left, right):
14     result = []
15     i = j = 0
16
17     while i < len(left) and j < len(right):
18         if left[i] <= right[j]:
19             result.append(left[i])
20             i += 1
21         else:
22             result.append(right[j])
23             j += 1
24
25     result.extend(left[i:])
26     result.extend(right[j:])
27     return result
28
29 ruta_archivo = "/Users/PC/Documents/TRABAJOPRACTICA/NumerosTxt/Numeros
    ramdon/80000numeros.txt"
30
31 with open(ruta_archivo, 'r') as archivo:
32     numeros = [int(num) for num in archivo.read().split()]
33
34 if not numeros:
35     print("El archivo no contiene n meros v lidos para ordenar.")
36     exit()
37
38 inicio = time.time()
39 numeros_ordenados = merge_sort(numeros)
40 fin = time.time()
41 duracion = fin - inicio
42
43 print("N meros ordenados:", numeros_ordenados)
44 print("Tiempo de ejecuci n:", duracion, "segundos")
```

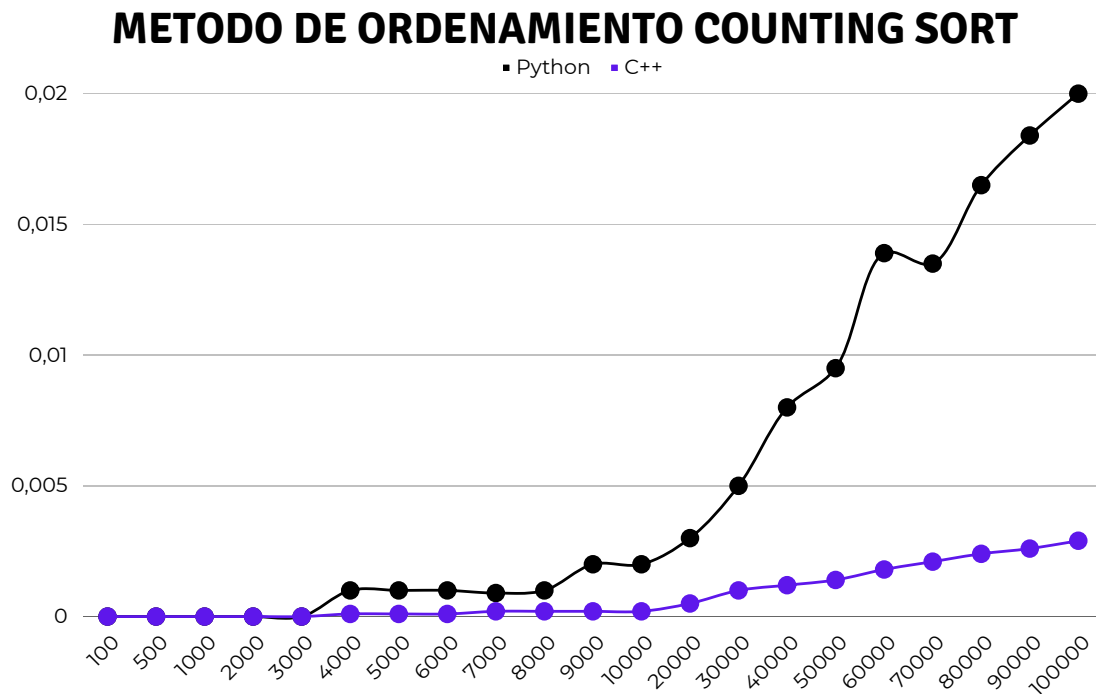
### 3. Resultados

#### 3.1. Resultado del METODO BURBUJA



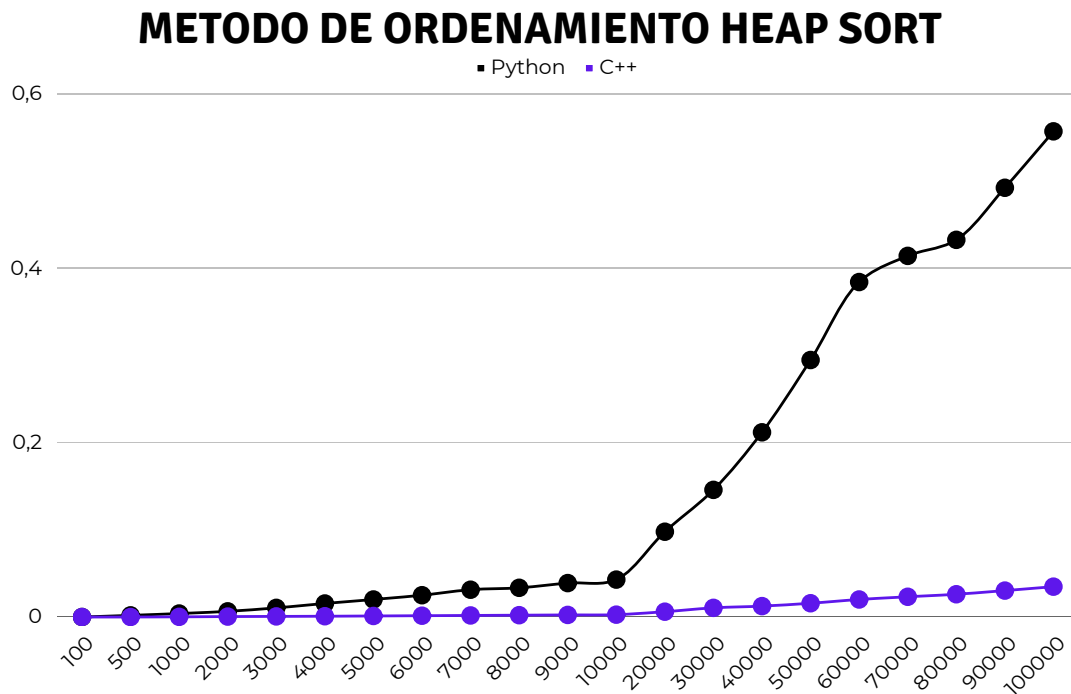
Metodo comparando numeros de (100,500,1000,2000...10000,20000....100000)

### 3.2. Resultado del METODO COUNTING SORT



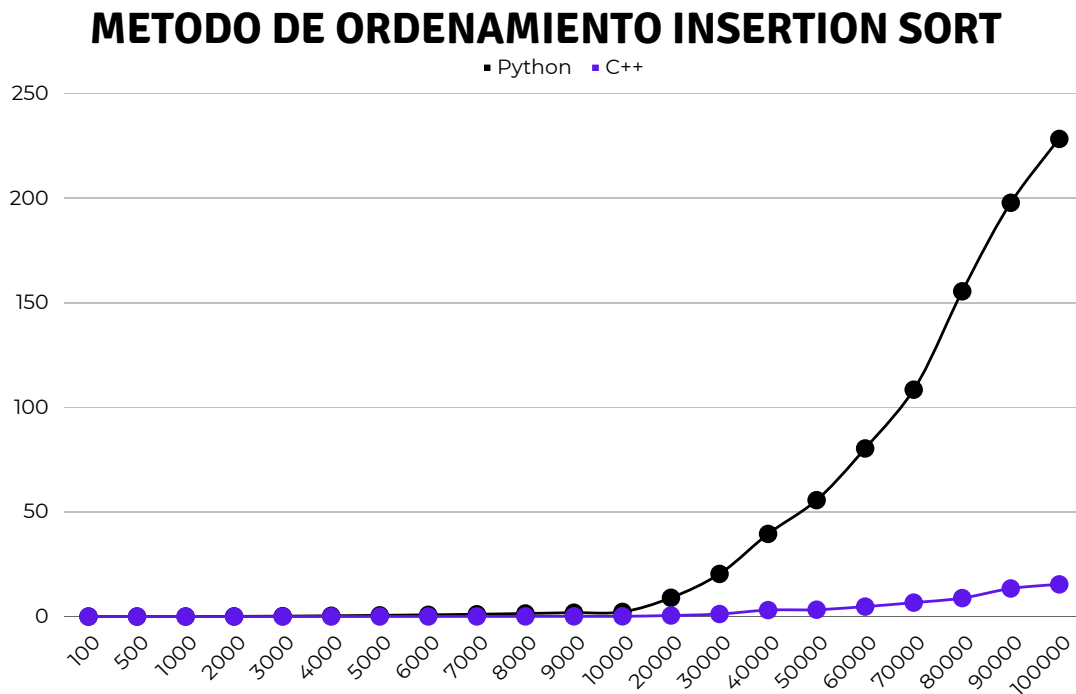
Metodo comparando numeros de (100,500,1000,2000...10000,20000....100000)

### 3.3. Resultado del METODO HEAP SORT



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

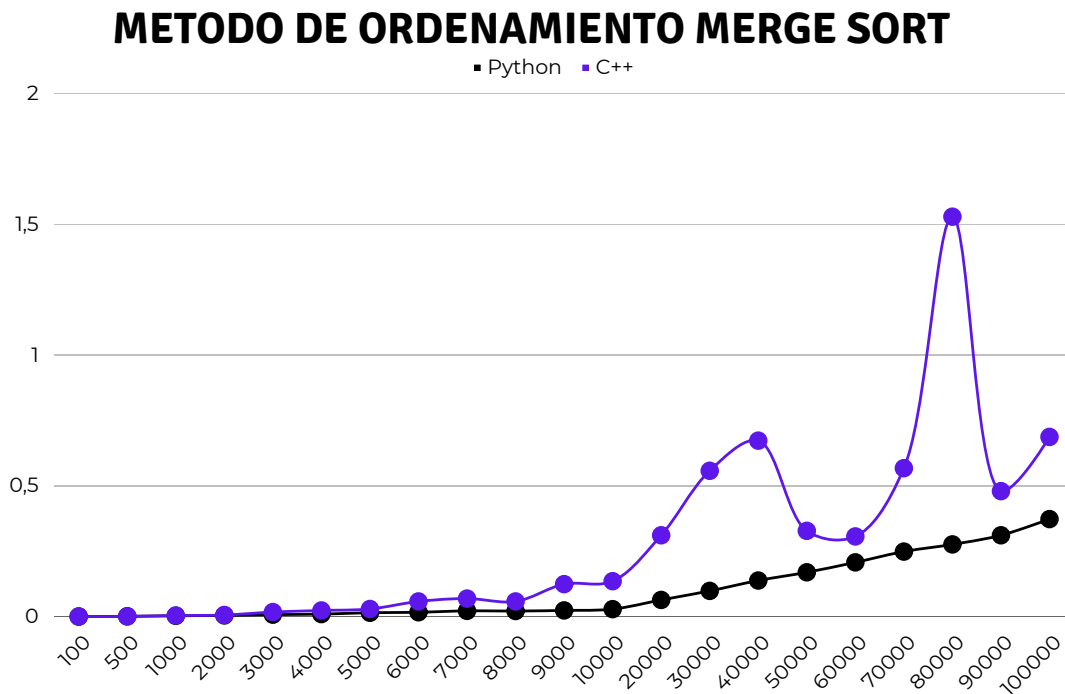
### 3.4. Resultado del METODO INSERTION SORT



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

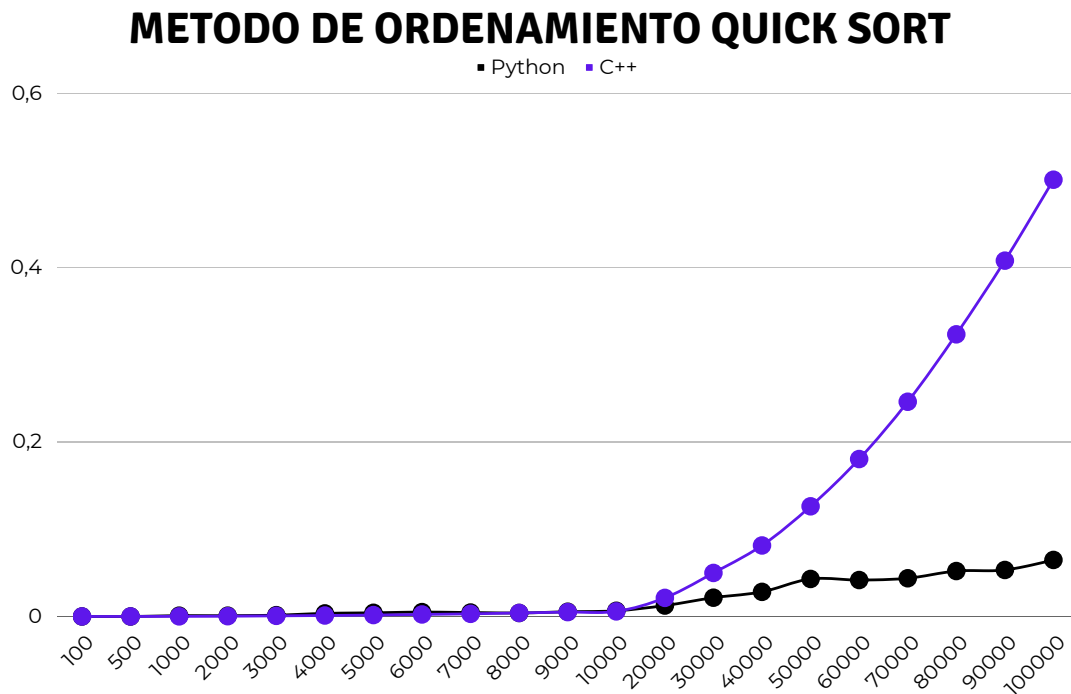


### 3.5. Resultado del METODO MERGE SORT



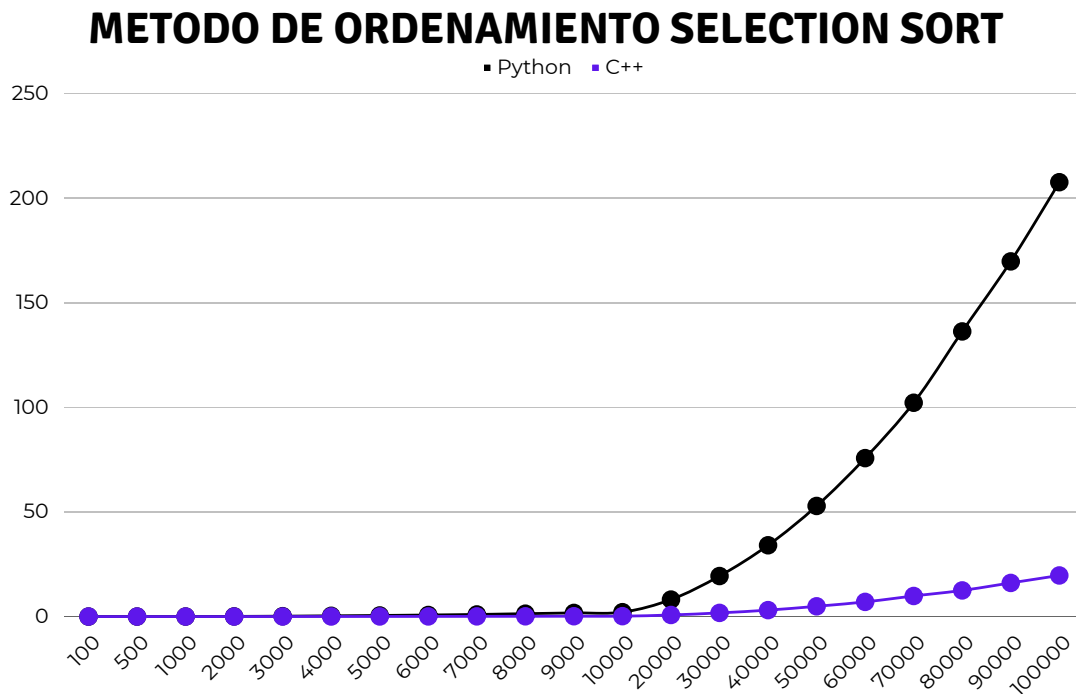
Metodo comparando numeros de (100,500,1000,2000...10000,20000....100000)

### 3.6. Resultado del METODO QUICK SORT



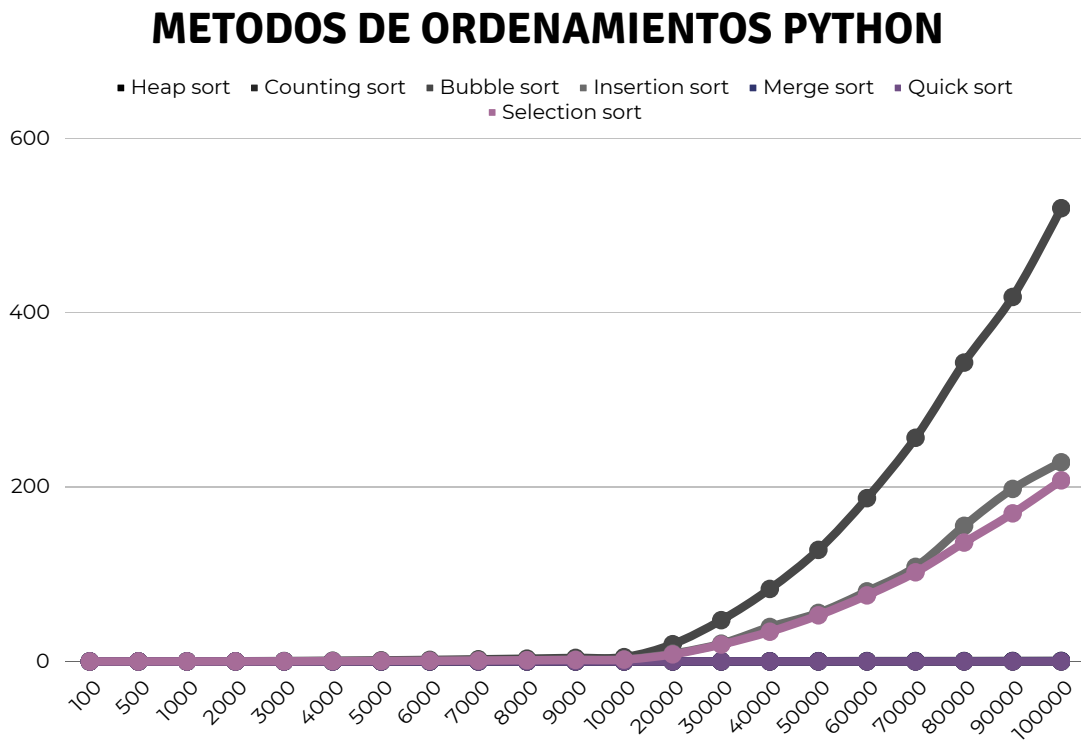
Metodo comparando numeros de (100,500,1000,2000...10000,20000....100000)

### 3.7. Resultado del METODO SELECTION SORT



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

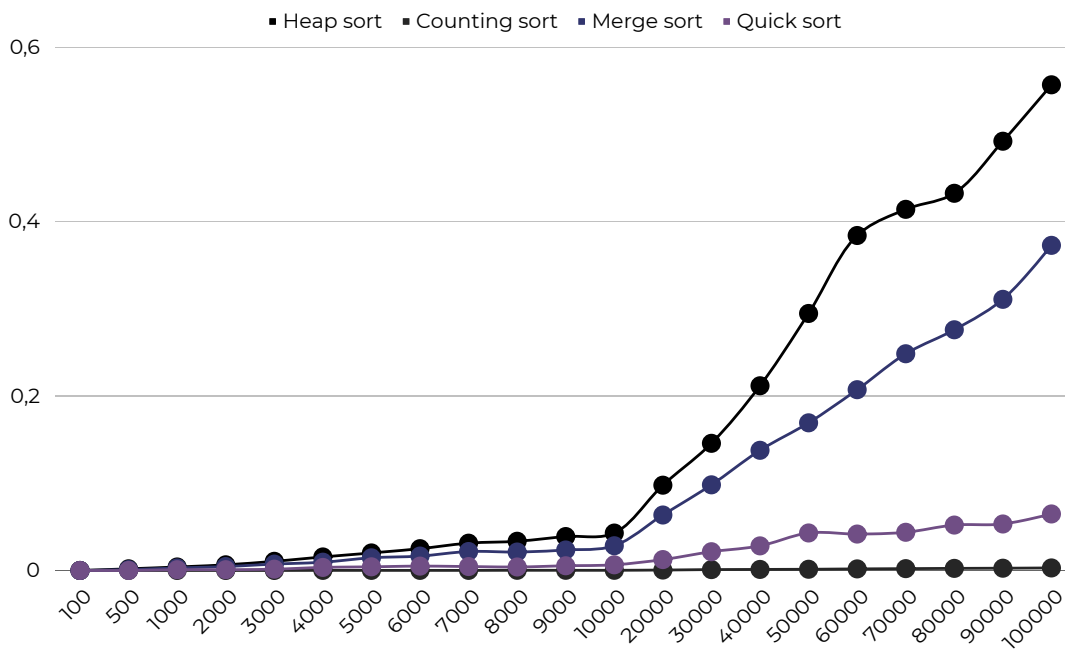
### 3.8. Resultado de todos los metodos en PYTHON



Comparando los metodos de ordenamiento

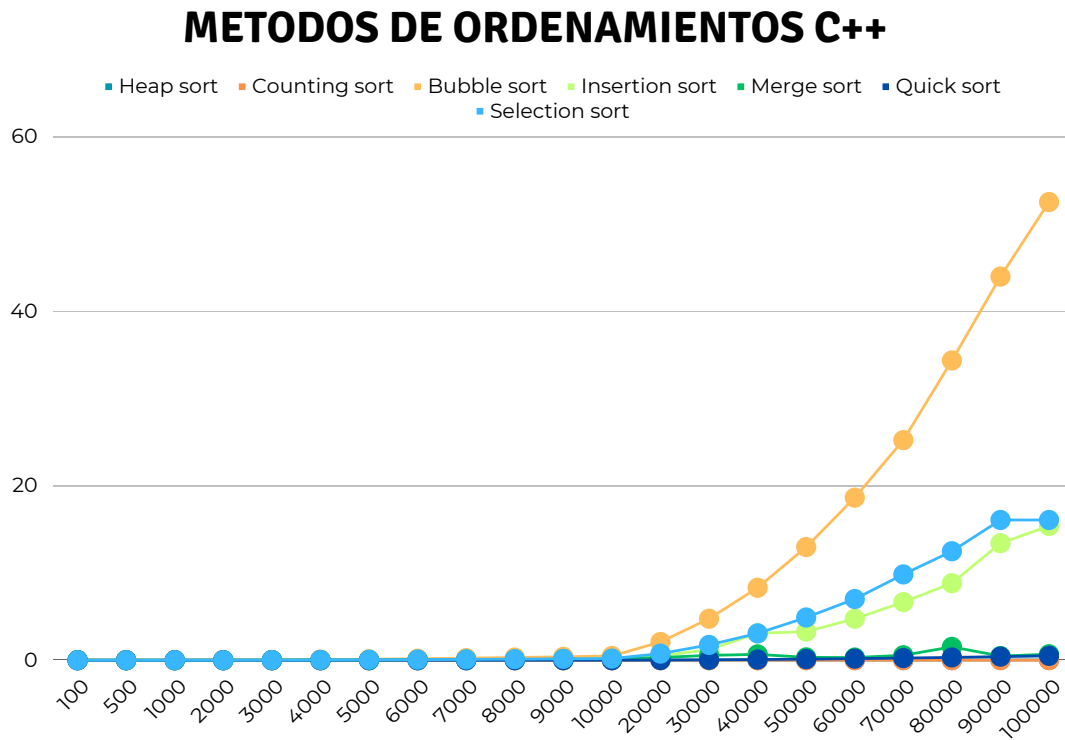
### 3.9. Resultado de metodos cortos de PYTHON

#### METODOS DE ORDENAMIENTOS PYTHON SIN METODO BURBUJA,INSERTION,SELECTION



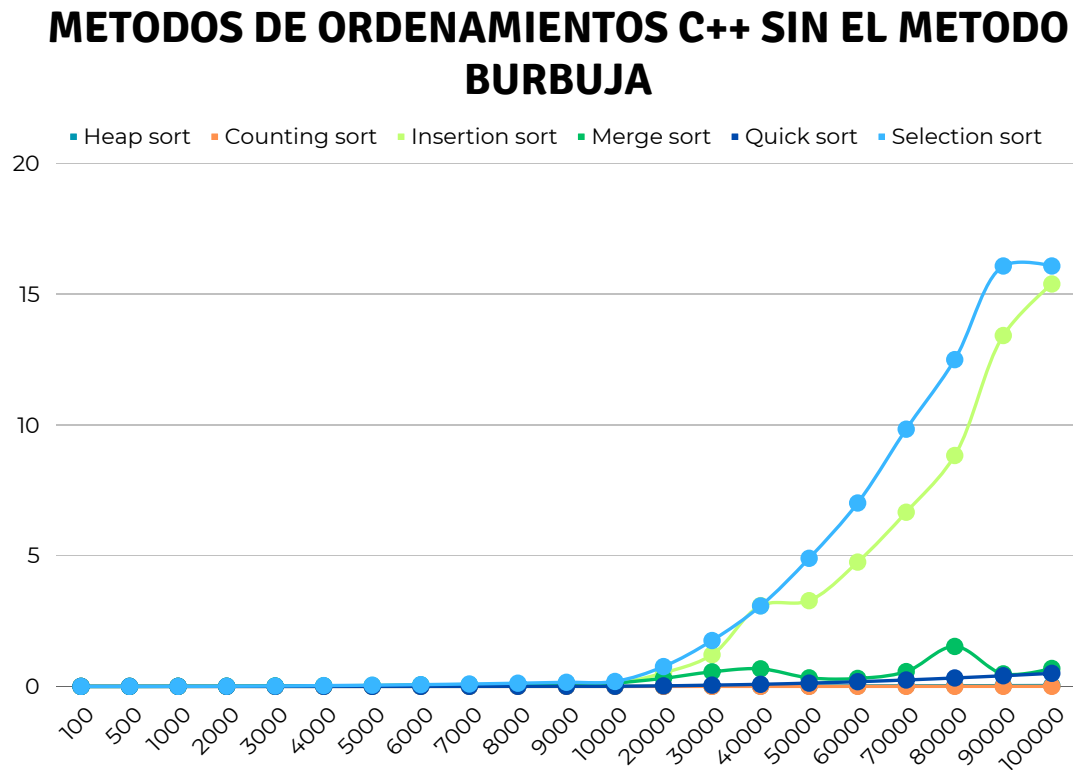
Descarta algunos metodos de ordenamiento muy grandes)

### 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*