



Universidad La Salle

Fundamentos de Lenguajes de Programación

Informe

Una Comparación entre C++, Go y Python

Karlo Emigdio Pacha Curimayhua

Quinto Semestre - Ingeniería de Software

2022

# 1 Introducción

Go es un lenguaje de programación compilado basado en C (en el aspecto sintáctico) y parecido a Python al momento del dinamismo que posee. El objetivo de este trabajo es poner a prueba los 3 lenguajes mencionados con los algoritmos de ordenamiento: Cocktail Sort y Counting Sort.

## 2 Algoritmos

### 2.1 Cocktail Sort

Es un algoritmo de ordenamiento con complejidad cuadrática  $O(n^2)$  como se puede apreciar en la Tabla 1; este es una variación del Bubble Sort que recorre la lista de datos en ambas direcciones de forma alternativa (al contrario que el Bubble Sort que sólo lo hace en una dirección).

Los pasos son:

Caso	Complejidad
Mejor caso	$O(n)$
Caso promedio	$O(n^2)$
Peor caso	$O(n^2)$

Tabla 1 : Complejidad Temporal

- I. El primer paso es recorrer la lista de datos de izquierda a derecha, se comparan los valores adyacentes, y si el valor de la izquierda es mayor, se intercambian los valores. El objetivo es poner el valor máximo al final.
- II. El segundo paso es recorrer la lista de datos de derecha a izquierda, se comparan los valores adyacentes, y si el valor de la derecha es menor, se intercambian los valores. El objetivo es poner el valor mínimo al inicio.

Este proceso continúa hasta que los elementos de la matriz no se ordenan.

### 2.2 Counting Sort

Es un algoritmo de ordenamiento con complejidad  $O(n+k)$ , esta aplica para todos los casos como se puede apreciar en la Tabla 2; este algoritmo basa su funcionamiento en los índices, no en comparaciones como los demás. Este algoritmo depende mucho del elemento con el valor máximo.

Caso	Complejidad
Mejor caso	$O(n + k)$
Caso promedio	$O(n + k)$
Peor caso	$O(n + k)$

Tabla 2 : Complejidad Temporal

Los pasos son:

- I. Busca el elemento máximo (max) y el mínimo (min).
- II. Crea una lista auxiliar de tamaño  $\text{max} + 1$ .
- III. Se subdivide en los siguientes pasos:
  - Se almacena el recuento de cada elemento de la lista en su índice correspondiente en la lista auxiliar.
  - Se suman los recuentos ( $a[i] + a[i-1]$ ), esto para colocar los elementos en el índice correcto de la lista ordenada.
- IV. Posiciona los elementos en la lista original o una de salida, para esto se utilizan los índices.

El punto débil de este algoritmo es la complejidad espacial  $O(\text{max})$ , si el elemento de mayor valor tiene un número de dígitos superior al tamaño de la lista de datos, el costo es casi cuadrático.

### 3 Implementación

Para poner a prueba los algoritmos de Cocktail y Counting Sort, se necesitan varios archivos CSV que contengan entre: 100, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000, 20000, 30000, 40000 y 50000 datos.

Una vez se implementen los algoritmos en los 3 lenguajes, se ejecutarán con cada lista de datos generada, por cada tamaño se ejecutará y medirá el tiempo de ejecución (en segundos) 5 veces y luego se obtendrá un promedio.

Finalmente, los datos obtenidos se exportarán en un CSV.

#### 3.1 Generador C++

```
1 #include <iostream>
2 #include <string>
3 #include <fstream>
4 #include <vector>
5 #include <climits>
6 #include <set>
7 #include <cstdlib>
8 #include <vector>
9 using namespace std;
10
11 int main() {
```

```

12
13     int unit = 99;
14
15     vector<int> sizes = { 100, 1000, 2000, 3000, 4000, 5000, 6000,
16                           7000, 8000, 9000, 10000, 20000, 30000, 40000, 50000 };
17
18     for(int s = 0; s < sizes.size(); s++) {
19         int UL = sizes[s]*1.8;
20         int i = 0;
21         set<int, greater<int> > Set;
22         ofstream myFile("data" + to_string(sizes[s]) + ".csv");
23
24         while(Set.size() <= sizes[s]) {
25             string cad = "";
26             for(int j = i; j <= i+unit; j++) {
27                 int number = rand() % (UL + 1);
28                 int prev = Set.size();
29                 Set.insert(number);
30                 if(prev < Set.size()) cad += to_string(number)+",";
31                 else j--;
32             }
33             myFile << cad+"\n";
34             if(Set.size() == sizes[s]) break;
35             i += unit;
36         }
37         cout<<(Set.size())<<endl;
38     }
39     return 0;
40 }

```

## 3.2 Cocktail Sort C++

```

1  #include <iostream>
2  #include <vector>
3  #include <chrono>
4  #include <ctime>
5  #include <sstream>
6  #include <fstream>
7  using namespace std;
8
9  void PrintVector(vector<int> list) {
10     cout<<" [ ";
11     for(int i = 0; i < list.size(); i++) cout<<list[i]<<" ";
12     cout<<"]"<<endl;
13 }
14
15 vector<int> cocktailSort(vector<int> list) {
16     int last = list.size() - 1;
17     while(true) {
18         bool swapped = false;
19         for (int i = 0; i < last; i++) {
20             if (list[i] > list[i+1]) {
21                 int a = list[i];
22                 list[i] = list[i+1];
23                 list[i+1] = a;
24                 swapped = true;
25             }

```

```

26     }
27     if(!swapped) return list;
28
29     swapped = false;
30     for (int i = last - 1; i >= 0; i--) {
31         if (list[i] > list[i+1]) {
32             int a = list[i];
33             list[i] = list[i+1];
34             list[i+1] = a;
35             swapped = true;
36         }
37     }
38     if (!swapped) return list;
39 }
40 return list;
41 }
42
43
44 vector<int> Read(string fileName) {
45     vector<int> list;
46     vector<vector<string>> content;
47     vector<string> row;
48     string line, word;
49     fstream file(fileName, ios::in);
50     if(file.is_open()) {
51         while(getline(file, line)) {
52             row.clear();
53             stringstream str(line);
54             while(getline(str, word, ',')) row.push_back(word);
55             content.push_back(row);
56         }
57     }
58     else {
59         cout<<"ERROR!!"<<endl;
60         return list;
61     }
62
63     for(int i = 0; i < content.size(); i++) {
64         for(int j = 0; j < content[i].size(); j++) {
65             list.push_back(stoi(content[i][j]));
66         }
67     }
68     return list;
69 }
70
71
72 int main() {
73     vector<int> sizes = { 100, 1000, 2000, 3000, 4000, 5000, 6000,
74         7000, 8000, 9000, 10000, 20000, 30000, 40000, 50000 };
75     vector<vector<double>> times;
76     ofstream myFile("cocktailSortTimesCpp.csv");
77
78     for(int k = 0; k < 5; k++) {
79         cout<<":::::::::"<<k<<endl;
80         vector<double> subTime;
81         times.push_back(subTime);
82         for(int i = 0; i < sizes.size(); i++) {
83             vector<int> list = Read("data" + to_string(sizes[i]) + ".csv"

```

```

    );
83     auto startTime = chrono::system_clock::now();
84     list = cocktailSort(list);
85     auto endTime = chrono::system_clock::now();
86     std::chrono::duration<double> elapsed_seconds = endTime -
    startTime;
87     times[k].push_back(elapsed_seconds.count());
88
89     cout<<"finished: "<< sizes[i]<<endl;
90 }
91
92 string cad = "";
93
94 for(int i = 0; i < sizes.size(); i++) cad += to_string(times[k
    ][i])+", ";
95 myFile << cad + "\n";
96 }
97
98 string cad = "";
99 vector<double> subTime;
100
101 times.push_back(subTime);
102 for(int i = 0; i < times.size(); i++) {
103     double avg = 0.0;
104     for(int j = 0; j < 5; j++) avg += times[j][i];
105     times[5].push_back(avg/5.0);
106 }
107
108 for(int i = 0; i < sizes.size(); i++) cad += to_string(times[5][i
    ]) + ", ";
109 myFile << cad;
110 myFile.close();
111
112 return 0;
113 }

```

### 3.3 Cocktail Sort Go

```

1 package main
2
3 import (
4     "encoding/csv"
5     "fmt"
6     "os"
7     "strconv"
8     "time"
9 )
10
11 func main() {
12     sizes := []int{100, 1000, 2000, 3000, 4000, 5000, 6000, 7000,
13         8000, 9000, 10000, 20000, 30000, 40000, 50000}
14     times := [][]float64{}
15
16     for k := 0; k < 5; k++ {
17         times = append(times, []float64{})
18         fmt.Println(":::::::::", k)
19         for i := 0; i < len(sizes); i++ {
20             list := []int{}

```

```

20     fileName := "data" + strconv.Itoa(sizes[i]) + ".csv"
21
22     list = Read(fileName)
23
24     start := time.Now()
25     cocktailSort(list)
26
27     times[k] = append(times[k], time.Since(start).Seconds())
28
29     fmt.Println("finished: ", sizes[i])
30 }
31 }
32 times = append(times, []float64{})
33 Write(times)
34 }
35
36 func Read(filePath string) []int {
37     list := []int{}
38     file, err := os.Open(filePath)
39
40     if err != nil {
41         fmt.Println("Unable to read input file "+filePath, err)
42     }
43
44     defer file.Close()
45     csvReader := csv.NewReader(file)
46     data, err := csvReader.ReadAll()
47
48     if err != nil {
49         fmt.Println("Unable to parse file as CSV for "+filePath, err)
50     }
51
52     for i := 0; i < len(data); i++ {
53         for j := 0; j < len(data[i]); j++ {
54             number, err := strconv.Atoi(data[i][j])
55             if err == nil {
56                 list = append(list, number)
57             }
58         }
59     }
60
61     return list
62 }
63
64 func Write(data [][]float64) {
65     records := [][]string{{}, {}}
66
67     file, err := os.Create("cocktailSortTimesGo.csv")
68     defer file.Close()
69
70     if err != nil {
71         fmt.Println("failed to open file", err)
72     }
73
74     writer := csv.NewWriter(file)
75     defer writer.Flush()
76
77     for i := 0; i < len(data[0]); i++ {

```

```

78     j := 0.0
79     for k := 0; k < 5; k++ {
80         j += data[k][i]
81     }
82     data[5] = append(data[5], j/5.0)
83 }
84
85 for i := 0; i < 6; i++ {
86     records = append(records, []string{})
87     for k := 0; k < len(data[i]); k++ {
88         j := data[i][k]
89         records[i] = append(records[i], strconv.FormatFloat(j, 'E',
90             -1, 64))
91     }
92 }
93
94 for _, record := range records {
95     if err := writer.Write(record); err != nil {
96         fmt.Println("error writing record to file", err)
97     }
98 }
99
100 func cocktailSort(list []int) {
101     last := len(list) - 1
102     for {
103         swapped := false
104         for i := 0; i < last; i++ {
105             if list[i] > list[i+1] {
106                 list[i], list[i+1] = list[i+1], list[i]
107                 swapped = true
108             }
109         }
110         if !swapped {
111             return
112         }
113         swapped = false
114         for i := last - 1; i >= 0; i-- {
115             if list[i] > list[i+1] {
116                 list[i], list[i+1] = list[i+1], list[i]
117                 swapped = true
118             }
119         }
120         if !swapped {
121             return
122         }
123     }
124 }

```

### 3.4 Cocktail Sort Python

```

1 import csv
2 import time
3
4 def cocktailSort(list):
5     last = len(list) - 1
6     while True:
7         swapped = False

```



```

8     for i in range (last-1,1,-1):
9         if list[i] > list[i+1]:
10             list[i], list[i+1] = list[i+1], list[i]
11             swapped = True
12
13     if swapped == False: return
14
15     swapped = False
16     for i in range(last-1,-1,-1):
17         if list[i] > list[i+1]:
18             list[i], list[i+1] = list[i+1], list[i]
19             swapped = True
20
21     if swapped == False: return
22
23
24 #main
25
26 sizes = [ 100, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000,
27           9000, 10000, 20000, 30000, 40000, 50000 ]
28 times = []
29 for k in range(0,5):
30     times.append([])
31     print(':::::::::',k)
32     for i in sizes:
33         list = []
34         with open('data' + str(i) + '.csv', newline='') as csvfile:
35             spamreader = csv.reader(csvfile, delimiter=',')
36             for row in spamreader:
37                 for number in row:
38                     if(number != ''):
39                         list.append(int(number))
40
41         startTime = time.time()
42         cocktailSort(list)
43         endTime = time.time()
44         times[k].append((endTime - startTime))
45
46     print("finished: ",i)
47
48 times.append([])
49 for i in range(0,len(sizes)):
50     print(i)
51     k = 0.0
52     for j in range(0,5):
53         print(j)
54         k += times[j][i]
55     times[5].append(k/5.0)
56
57
58 with open('cocktailSortTimesPy.csv', 'w', encoding='UTF8', newline=
59         '') as file:
60     writer = csv.writer(file)
61     for i in range(0,6): writer.writerow(times[i])

```

### 3.5 Counting Sort C++

```

1  #include <iostream>
2  #include <vector>
3  #include <chrono>
4  #include <ctime>
5  #include <sstream>
6  #include <fstream>
7  #include <algorithm>
8  using namespace std;
9
10 void PrintVector(vector<int> list) {
11     cout<<" ";
12     for(int i = 0; i < list.size(); i++) cout<<list[i]<<" ";
13     cout<<" "<<endl;
14 }
15
16 vector<int> CountingSort(vector<int> list) {
17     int minItem = *min_element(list.begin(), list.end());
18     vector<int> countList ((*max_element(list.begin(), list.end()) -
19                             minItem + 1), 0);
20     vector<int> outputList (list.size(),0);
21     for(int i = 0; i < list.size(); i++) countList[list[i] - minItem]
22         += 1;
23     for(int i = 1; i < countList.size(); i++) countList[i] +=
24         countList[i-1];
25     for (int i = list.size() - 1; i >= 0; i--) {
26         outputList[countList[list[i] - minItem] - 1] = list[i];
27         countList[list[i] - minItem] -= 1;
28     }
29     for(int i = 0; i < list.size(); i++) list[i] = outputList[i];
30     return list;
31 }
32
33
34
35
36 vector<int> Read(string fileName) {
37     vector<int> list;
38     vector<vector<string>> content;
39     vector<string> row;
40     string line, word;
41     fstream file(fileName, ios::in);
42     if(file.is_open()) {
43         while(getline(file, line)) {
44             row.clear();
45             stringstream str(line);
46             while(getline(str, word, ',')) row.push_back(word);
47             content.push_back(row);
48         }
49     }
50     else {
51         cout<<"ERROR!!"<<endl;
52         return list;
53     }
54     for(int i = 0; i < content.size(); i++) {

```

```

56     for(int j = 0; j < content[i].size(); j++) {
57         list.push_back(stoi(content[i][j]));
58     }
59 }
60 return list;
61 }
62
63
64 int main() {
65
66     vector<int> sizes = { 100, 1000, 2000, 3000, 4000, 5000, 6000,
67         7000, 8000, 9000, 10000, 20000, 30000, 40000, 50000 };
68     vector<vector<double>> times;
69     ofstream myFile("countingSortTimesCpp.csv");
70
71     for(int k = 0; k < 5; k++) {
72         cout<<":::::::::"<<k<<endl;
73         vector<double> subTime;
74         times.push_back(subTime);
75         for(int i = 0; i < sizes.size(); i++) {
76             vector<int> list = Read("data" + to_string(sizes[i]) + ".csv"
77 );
78             auto startTime = chrono::system_clock::now();
79             list = CountingSort(list);
80             auto endTime = chrono::system_clock::now();
81             std::chrono::duration<double> elapsed_seconds = endTime -
82                 startTime;
83             times[k].push_back(elapsed_seconds.count());
84
85             cout<<"finished: "<< sizes[i]<<endl;
86         }
87
88         string cad = "";
89
90         for(int i = 0; i < sizes.size(); i++) cad += to_string(times[k
91 ][i])+",";
92         myFile << cad + "\n";
93     }
94
95     string cad = "";
96     vector<double> subTime;
97
98     times.push_back(subTime);
99     for(int i = 0; i < times.size(); i++) {
100         double avg = 0.0;
101         for(int j = 0; j < 5; j++) avg += times[j][i];
102         times[5].push_back(avg/5.0);
103     }
104
105     for(int i = 0; i < sizes.size(); i++) cad += to_string(times[5][i
106 ][i])+",";
107     myFile << cad;
108     myFile.close();
109
110     return 0;
111 }

```

### 3.6 Counting Sort Go

```

1 package main
2
3 import (
4     "encoding/csv"
5     "fmt"
6     "os"
7     "strconv"
8     "time"
9 )
10
11 func main() {
12     sizes := []int{100, 1000, 2000, 3000, 4000, 5000, 6000, 7000,
13         8000, 9000, 10000, 20000, 30000, 40000, 50000}
14     times := [][]float64{}
15     for k := 0; k < 5; k++ {
16         times = append(times, []float64{})
17         fmt.Println(":::::::::", k)
18         for i := 0; i < len(sizes); i++ {
19             list := []int{}
20             fileName := "data" + strconv.Itoa(sizes[i]) + ".csv"
21
22             list = Read(fileName)
23
24             start := time.Now()
25             countingSort(list)
26
27             times[k] = append(times[k], time.Since(start).Seconds())
28
29             fmt.Println("finished: ", sizes[i])
30         }
31     }
32     times = append(times, []float64{})
33     Write(times)
34 }
35
36 func Read(filePath string) []int {
37     list := []int{}
38     file, err := os.Open(filePath)
39
40     if err != nil {
41         fmt.Println("Unable to read input file "+filePath, err)
42     }
43
44     defer file.Close()
45     csvReader := csv.NewReader(file)
46     data, err := csvReader.ReadAll()
47
48     if err != nil {
49         fmt.Println("Unable to parse file as CSV for "+filePath, err)
50     }
51
52     for i := 0; i < len(data); i++ {
53         for j := 0; j < len(data[i]); j++ {
54             number, err := strconv.Atoi(data[i][j])
55             if err == nil {
56                 list = append(list, number)
57             }
58         }
59     }
60 }

```

```

58     }
59 }
60
61 return list
62 }
63
64 func Write(data [][]float64) {
65     records := [][]string{{}, {}}
66
67     file, err := os.Create("countingSortTimeGo.csv")
68     defer file.Close()
69
70     if err != nil {
71         fmt.Println("failed to open file", err)
72     }
73
74     writer := csv.NewWriter(file)
75     defer writer.Flush()
76
77     for i := 0; i < len(data[0]); i++ {
78         j := 0.0
79         for k := 0; k < 5; k++ {
80             j += data[k][i]
81         }
82         data[5] = append(data[5], j/5.0)
83     }
84
85     for i := 0; i < 6; i++ {
86         records = append(records, []string{})
87         for k := 0; k < len(data[i]); k++ {
88             j := data[i][k]
89             records[i] = append(records[i], strconv.FormatFloat(j, 'E',
90                 -1, 64))
91         }
92     }
93
94     for _, record := range records {
95         if err := writer.Write(record); err != nil {
96             fmt.Println("error writing record to file", err)
97         }
98     }
99
100 func countingSort(list []int) {
101     minItem := Min(list)
102     countList := Zero(Max(list) - minItem + 1)
103     outputList := Zero(len(list))
104
105     for i := 0; i < len(list); i++ {
106         countList[list[i]-minItem] += 1
107     }
108
109     for i := 1; i < len(countList); i++ {
110         countList[i] += countList[i-1]
111     }
112
113     for i := len(list) - 1; i >= 0; i-- {
114         outputList[countList[list[i]-minItem]-1] = list[i]

```

```

115     countList[list[i]-minItem] -= 1
116 }
117
118 for i := 0; i < len(list); i++ {
119     list[i] = outputList[i]
120 }
121 }
122
123 func Max(list []int) int {
124     max := list[0]
125     for i := 0; i < len(list); i++ {
126         if max < list[i] {
127             max = list[i]
128         }
129     }
130     return max
131 }
132
133 func Min(list []int) int {
134     min := list[0]
135     for i := 0; i < len(list); i++ {
136         if min > list[i] {
137             min = list[i]
138         }
139     }
140     return min
141 }
142
143 func Zero(length int) []int {
144     list := []int{}
145     for i := 0; i < length; i++ {
146         list = append(list, 0)
147     }
148     return list
149 }

```

### 3.7 Counting Sort Python

```

1 import csv
2 import time
3
4 def CountingSort(list):
5     minItem = min(list)
6     countList = [0] * (max(list) - minItem + 1)
7     outputList = [0] * len(list)
8
9     for i in range(0, len(list)): countList[list[i] - minItem] += 1
10
11     for i in range(1, len(countList)): countList[i] += countList[i-1]
12
13     for i in range(len(list)-1, -1, -1):
14         outputList[countList[list[i] - minItem] - 1] = list[i]
15         countList[list[i] - minItem] -= 1
16
17     for i in range(0, len(list)): list[i] = outputList[i]
18
19
20 #main

```

```

21
22 sizes = [ 100, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000,
           9000, 10000, 20000, 30000, 40000, 50000 ]
23 times = []
24
25 for k in range(0,5):
26     times.append([])
27     print('::::::::::::',k)
28     for i in sizes:
29         list = []
30         with open('data' + str(i) + '.csv', newline='') as csvfile:
31             spamreader = csv.reader(csvfile, delimiter=',')
32             for row in spamreader:
33                 for number in row:
34                     if(number != ''):
35                         list.append(int(number))
36
37             startTime = time.time()
38             CountingSort(list)
39             endTime = time.time()
40             times[k].append((endTime - startTime))
41
42             print("finished: ",i)
43
44 times.append([])
45 for i in range(0,len(sizes)):
46     print(i)
47     k = 0.0
48     for j in range(0,5):
49         print(j)
50         k += times[j][i]
51     times[5].append(k/5.0)
52
53
54 with open('countingSortTimePy.csv', 'w', encoding='UTF8', newline=
55           ') as file:
56     writer = csv.writer(file)
57     for i in range(0,6): writer.writerow(times[i])

```

### 3.8 Graficador Python

```

1 import matplotlib.pyplot as plt
2
3 sizes = [
4     100, 1000, 2000, 3000, 4000, 5000, 6000, 7000,
5     8000, 9000, 10000, 20000, 30000, 40000, 50000
6 ]
7
8 times = [
9     [
10         0.000084,0.005125,0.019379,0.041610,0.074329,0.116676,
11         0.1687696,0.2257266,0.2976812,0.3724236,0.4619382,
12         1.8438654,4.1674776,7.4450562,11.5773972
13     ],
14     [
15         0E+00,7.0618E-04,2.46516E-03,6.0231E-03,1.04571E-02,
16         1.8584499999999997E-02,2.6566E-02,3.798084E-02,
17         5.2525619999999995E-02,6.684128E-02,8.331876E-02,

```

```

18         3.6075668E-01,8.398381E-01,1.49931916000000002E+00,
19         2.36201094E+00
20     ],
21     [
22         0.0006016731262207031,0.09107704162597656,0.380082368850708,
23         0.8619470119476318,1.5388020038604737,2.421519899368286,
24         3.522882890701294,4.79917278289795,6.264121246337891,
25         7.976968431472779,9.764828872680663,40.998408508300784,
26         92.16900811195373,165.19458498954774,258.4798514842987
27     ]
28 ]
29
30
31 plt.plot(sizes,list(times[0]),label="Cpp")
32 plt.plot(sizes,list(times[1]),label="Go")
33 plt.plot(sizes,list(times[2]),label="Python")
34 plt.xlabel("number data")
35 plt.ylabel("time (s)")
36 plt.legend(loc="lower right")
37 plt.title("Cocktail Sort")
38 plt.show()
39
40 plt.plot(sizes,list(times[0]),label="Cpp")
41 plt.plot(sizes,list(times[1]),label="Go")
42 plt.xlabel("number data")
43 plt.ylabel("time (s)")
44 plt.legend(loc="lower right")
45 plt.title("Cocktail Sort")
46 plt.show()
47
48
49 times = [
50     [
51         0.000017,0.000206,0.000250,0.000330,0.000415,0.000502,0.0006348,
52
53         0.0007302,0.0007406,0.0008184,0.0009192,0.001723,0.0024994,
54         0.0032802,0.0046394
55     ],
56     [
57         0E+00,0E+00,0E+00,4.0414000000000004E-04,9.42E-06,1.0958E
58         -04,4.255E-04,
59         2.0683999999999996E-04,3.1272E-04,2.3562000000000002E
60         -04,5.4902E-04,
61         8.4048E-04,1.36154E-03,2.00754E-03,2.421E-03
62     ],
63     [
64         0.0,0.0006076335906982422,0.0012085437774658203,0.0017999172210693359,
65
66         0.0025574684143066405,0.0029863834381103514,0.003389883041381836,
67
68         0.004581928253173828,0.005384492874145508,0.005778026580810547,
69         0.006590986251831054,0.01417369842529297,0.02115340232849121,

```



```

65         0.029258871078491212,0.036231613159179686
66     ]
67 ]
68
69
70 plt.plot(sizes,list(times[0]),label="Cpp")
71 plt.plot(sizes,list(times[1]),label="Go")
72 plt.plot(sizes,list(times[2]),label="Python")
73 plt.xlabel("number data")
74 plt.ylabel("time (s)")
75 plt.legend(loc="lower right")
76 plt.title("Counting Sort")
77 plt.show()
78
79 plt.plot(sizes,list(times[0]),label="Cpp")
80 plt.plot(sizes,list(times[1]),label="Go")
81 plt.xlabel("number data")
82 plt.ylabel("time (s)")
83 plt.legend(loc="lower right")
84 plt.title("Counting Sort")
85 plt.show()

```

## 4 Resultados

### 4.1 Tabla Comparativa: Cocktail Sort

Cocktail Sort C++							
N	1	2	3	4	5	AVG	SD
100	0.0001210	0.0000620	0.0000630	0.0000550	0.0001170	0.0000840	0.0000325
1000	0.0073240	0.0044920	0.0045210	0.0047140	0.0045730	0.0051250	0.0012323
2000	0.0209500	0.0192380	0.0186970	0.0185280	0.0194810	0.0193790	0.0009602
3000	0.0423200	0.0428840	0.0413290	0.0405690	0.0409470	0.0416100	0.0009656
4000	0.0748370	0.0759640	0.0751800	0.0726640	0.0730030	0.0743290	0.0014306
5000	0.1173170	0.1179270	0.1180260	0.1150370	0.1150720	0.1166760	0.0015048
6000	0.1728220	0.1711300	0.1703270	0.1653840	0.1641850	0.1687696	0.0037716
7000	0.2304690	0.2307990	0.2247500	0.2209520	0.2216630	0.2257266	0.0047033
8000	0.3001500	0.3033230	0.2994490	0.2949660	0.2905180	0.2976812	0.0049933
9000	0.3743590	0.3814760	0.3777690	0.3672540	0.3612600	0.3724236	0.0081489
10000	0.4705890	0.4750150	0.4666200	0.4507310	0.4467360	0.4619382	0.0124947
20000	1.9044310	1.8321160	1.8715210	1.8095730	1.8016860	1.8438654	0.0433767
30000	4.3267350	4.1452550	4.2491800	4.0529160	4.0633020	4.1674776	0.1188581
40000	7.6755100	7.4616550	7.5487580	7.2538390	7.2855190	7.4450562	0.1775913
50000	11.9991890	11.8070980	11.4353260	11.3283770	11.3169960	11.5773972	0.3084943

Tabla 3 : Tiempos de ejecución, promedio y desviación estándar

Cocktail Sort Go							
N	1	2	3	4	5	AVG	SD
100	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
1000	0.00054110	0.00051270	0.00050730	0.00051370	0.00067870	0.00055070	0.00007275
2000	0.00222630	0.00323820	0.00276340	0.00309620	0.00272060	0.00280894	0.00039247
3000	0.00577730	0.00573850	0.00566680	0.00570890	0.00553670	0.00568564	0.00009257
4000	0.01199740	0.01097010	0.01147850	0.01098950	0.01062730	0.01121256	0.00053333
5000	0.01860440	0.01965420	0.01931080	0.01820910	0.01946090	0.01904788	0.00061391
6000	0.02748400	0.06975730	0.02783230	0.05129350	0.02857450	0.04098832	0.01899606
7000	0.06312240	0.03826570	0.08436440	0.03873560	0.03842490	0.05258260	0.02072633
8000	0.05835550	0.05767750	0.05579640	0.05249800	0.09743330	0.06435214	0.01863193
9000	0.11227550	0.06887470	0.06880830	0.09164540	0.06863300	0.08204738	0.01958698
10000	0.08924940	0.08614490	0.09034580	0.08584710	0.08701550	0.08772054	0.00198235
20000	0.39686310	0.36994560	0.37265440	0.39173690	0.40576260	0.38739252	0.01555340
30000	0.86940450	0.89746080	0.86979670	0.89927570	0.85584450	0.87835644	0.01912319
40000	1.56087190	1.57692380	1.59631990	1.65159930	1.54598780	1.58634054	0.04099989
50000	2.50865950	2.49324650	2.49963360	2.49998280	2.51063210	2.50243090	0.00714539

Tabla 4 : Tiempos de ejecución, promedio y desviación estándar

Cocktail Sort Python							
N	1	2	3	4	5	AVG	SD
100	0.000000	0.001019	0.000978	0.001011	0.000000	0.000602	0.000549
1000	0.087769	0.093783	0.093749	0.088759	0.091326	0.091077	0.002777
2000	0.374890	0.395051	0.383602	0.376592	0.370276	0.380082	0.009640
3000	0.839196	0.899416	0.841379	0.870861	0.858883	0.861947	0.024665
4000	1.560754	1.504115	1.489200	1.562043	1.577898	1.538802	0.039415
5000	2.447124	2.475884	2.385824	2.350404	2.448363	2.421520	0.051632
6000	3.658965	3.473529	3.435544	3.535811	3.510566	3.522883	0.085001
7000	4.775979	4.861353	4.726935	4.733182	4.898416	4.799173	0.077179
8000	6.193019	6.377859	6.227132	6.256342	6.266255	6.264121	0.069687
9000	8.073006	8.070265	7.859922	7.860985	8.020664	7.976968	0.108384
10000	9.845040	9.880301	9.606630	9.653400	9.838773	9.764829	0.125179
20000	40.939095	44.672501	39.504922	39.570410	40.305115	40.998409	2.136166
30000	93.755402	93.770701	92.610282	90.417916	90.290740	92.169008	1.722716
40000	168.082986	168.496986	164.325686	162.420765	162.646501	165.194585	2.923633
50000	266.325962	260.812810	254.193686	255.022999	256.043802	258.479851	5.080835

Tabla 5 : Tiempos de ejecución, promedio y desviación estándar

## 4.2 Tabla Comparativa: Counting Sort

Counting Sort C++							
N	1	2	3	4	5	AVG	SD
100	0.00001300	0.00001800	0.00001800	0.00001700	0.00001700	0.00001700	0.00000207
1000	0.00006800	0.00024400	0.00024100	0.00024000	0.00023300	0.00020600	0.00007680
2000	0.00014100	0.00028200	0.00028100	0.00027500	0.00027300	0.00025000	0.00006128
3000	0.00020100	0.00037800	0.00036300	0.00035400	0.00035200	0.00033000	0.00007262
4000	0.00027500	0.00045600	0.00046500	0.00045000	0.00042800	0.00041500	0.00007933
5000	0.00041100	0.00050400	0.00061600	0.00049700	0.00048400	0.00050200	0.00007353
6000	0.00057200	0.00068200	0.00062900	0.00064400	0.00064700	0.00063480	0.00004012
7000	0.00057500	0.00068700	0.00085600	0.00086800	0.00066500	0.00073020	0.00012749
8000	0.00066300	0.00076600	0.00077900	0.00075800	0.00073700	0.00074060	0.00004598
9000	0.00070800	0.00085100	0.00084800	0.00084900	0.00083600	0.00081840	0.00006199
10000	0.00084300	0.00093000	0.00094900	0.00092200	0.00095200	0.00091920	0.00004443
20000	0.00176000	0.00173300	0.00172100	0.00172100	0.00168000	0.00172300	0.00002884
30000	0.00244100	0.00238500	0.00263700	0.00240300	0.00263100	0.00249940	0.00012454
40000	0.00335400	0.00315700	0.00344400	0.00300300	0.00344300	0.00328020	0.00019416
50000	0.00729100	0.00419600	0.00392200	0.00394200	0.00384600	0.00463940	0.00148813

Tabla 4 : Tiempos de ejecución, promedio y desviación estándar

Counting Sort Go							
N	1	2	3	4	5	AVG	SD
100	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
1000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
2000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
3000	0.00100780	0.00050640	0.00000000	0.00000000	0.00050650	0.00040414	0.000421900
4000	0.00004710	0.00000000	0.00000000	0.00000000	0.00000000	0.00000942	0.000021064
5000	0.00000000	0.00000000	0.00000000	0.00054790	0.00000000	0.00010958	0.000245028
6000	0.00053600	0.00107970	0.00051180	0.00000000	0.00000000	0.00042550	0.000449927
7000	0.00051750	0.00000000	0.00000000	0.00000000	0.00051670	0.00020684	0.000283227
8000	0.00000000	0.00053940	0.00050740	0.00000000	0.00051680	0.00031272	0.000285710
9000	0.00000000	0.00000000	0.00062060	0.00055750	0.00000000	0.00023562	0.000323406
10000	0.00051380	0.00055530	0.00013060	0.00050420	0.00104120	0.00054902	0.000324270
20000	0.00051370	0.00108970	0.00101970	0.00099720	0.00058210	0.00084048	0.000270341
30000	0.00158390	0.00161050	0.00157420	0.00099800	0.00104110	0.00136154	0.000312847
40000	0.00216950	0.00223260	0.00153590	0.00199770	0.00210200	0.00200754	0.000277645
50000	0.00322110	0.00267620	0.00206540	0.00199480	0.00214750	0.00242100	0.000521581

Tabla 5 : Tiempos de ejecución, promedio y desviación estándar

Counting Sort Python							
N	1	2	3	4	5	AVG	SD
100	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
1000	0.00000000	0.00102425	0.00000000	0.00102377	0.00099015	0.00060763	0.00055486
2000	0.00202298	0.00102568	0.00099635	0.00099683	0.00100088	0.00120854	0.00045545
3000	0.00201797	0.00199485	0.00099826	0.00199461	0.00199389	0.00179992	0.00044826
4000	0.00297618	0.00181246	0.00196314	0.00301504	0.00302052	0.00255747	0.00061388
5000	0.00296259	0.00299287	0.00298977	0.00299191	0.00299478	0.00298638	0.00001342
6000	0.00398588	0.00299215	0.00299168	0.00299191	0.00398779	0.00338988	0.00054494
7000	0.00495625	0.00398970	0.00398946	0.00498724	0.00498700	0.00458193	0.00054089
8000	0.00598526	0.00497961	0.00498676	0.00498652	0.00598431	0.00538449	0.00054800
9000	0.00498700	0.00595331	0.00598359	0.00698161	0.00498462	0.00577803	0.00083319
10000	0.00701094	0.00596571	0.00697303	0.00598407	0.00702119	0.00659099	0.00056274
20000	0.01396251	0.01499248	0.01296544	0.01396346	0.01498461	0.01417370	0.00084804
30000	0.02193761	0.02093816	0.02097392	0.01994753	0.02196980	0.02115340	0.00083877
40000	0.02860856	0.02895284	0.02992010	0.03091812	0.02789474	0.02925887	0.00117981
50000	0.03490758	0.03587413	0.03587437	0.03760242	0.03689957	0.03623161	0.00104089

Tabla 6 : Tiempos de ejecución, promedio y desviación estándar

### 4.3 Gráfico Comparativo: Cocktail Sort

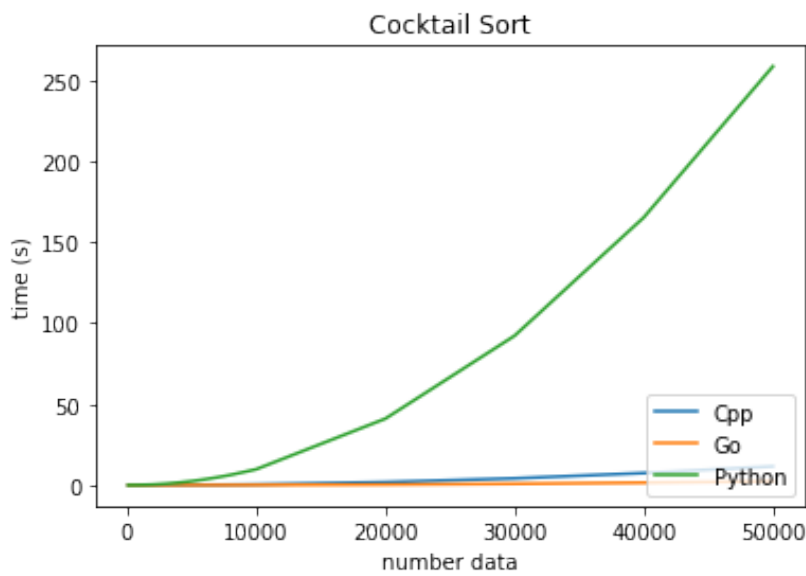


Figure 1: Cocktail Sort en C + +, Go y Python

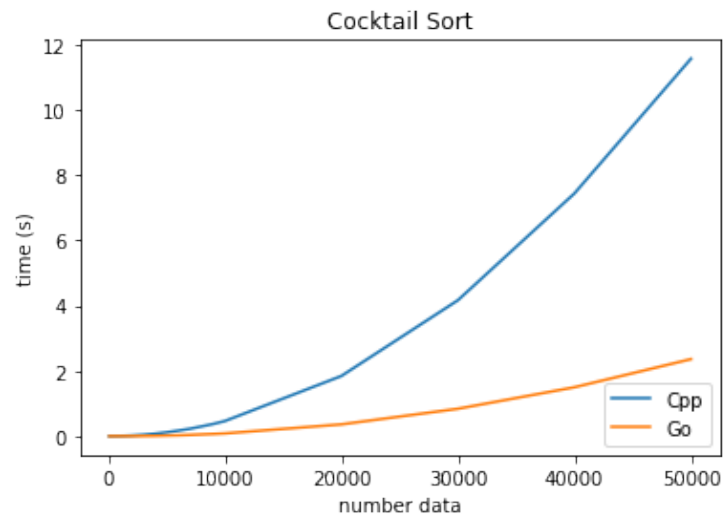


Figure 2: Cocktail Sort en C++ y Go

Como se ve en la Figura 1, Python es uno de los peores lenguajes en lo que rendimiento respecta (esto debido a su naturaleza [interpretado]). A pesar de que C++ suele ser el mejor en tiempo de ejecución, esta vez Go es superior.

Si sólo se examina la Figura 1, parece que la diferencia no es tan amplia, pero en la Figura 2, se ve que Go tuvo un tiempo de ejecución notablemente mejor, se toma casi la mitad del tiempo que se toma C++.

#### 4.4 Gráfico Comparativo: Counting Sort

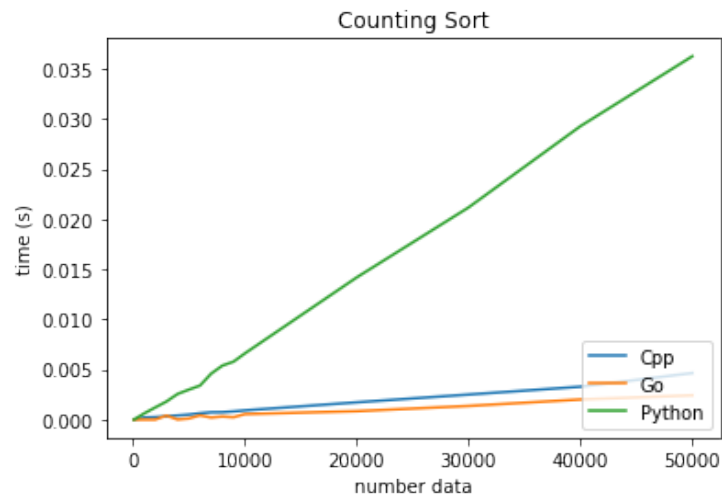


Figure 3: Counting Sort en C++, Go y Python

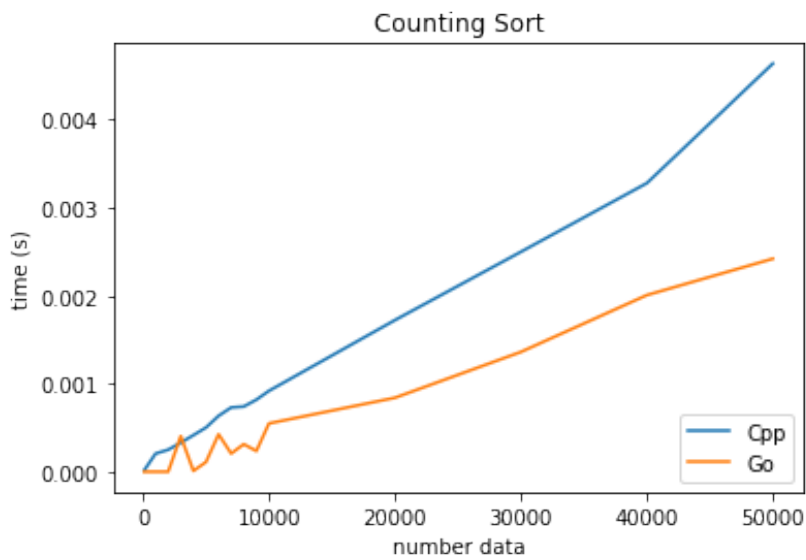


Figure 4: Counting Sort en C++ y Go

Si se examina la Figura 3, Python sigue mostrando un mal rendimiento, sigue teniendo una grna diferencia con Go y C++. Nuevamente Go se moetró superior a C++.

Al examinar la Figura 4, se puede notar que Go ya no fue tan superior a C++, incluso en un punto fue mejor. Esta vez la diferencia sería de un  $t - C$ , donde  $C$  es casi igual a 0.0005 segundos, aunque este se va incrementando.

## 5 Conclusiones

Realmente no se podría concluir algo claro con los resultados del experimento, a pesar de que parece que Go es mejor que C++, esto debido a las tablas de los tiempos obtenidos, se puede apreciar que Go y Python en los primeros casos dan como resultado 0, lo cual no puede ser posible incluso en el Counting Sort, C++ no retorna 0 en tiempos de ejecución. Una vez expuesto esto, se podría decir que Go y Python no capturaron el tiempo de ejecución de la mejor forma, podría ser esa la razón por la que C++ se mostró inferior a Go.

Una vez aclarado la "victoria" de Go sobre C++, se puede decir que los lenguajes de programación compilados son claramente superiores a los interpretados, o incluso a los que están en un punto intermedio como es el caso de Java. Esta es la principal razón por la que estos se usan para la base de sistemas grandes y con una gran cantidad de operaciones, es la razón de la existencia de TypeScript y Numba que tiene como objetivo mejorar JavaScript y Python respectivamente. Pero esto no quiere decir que los lenguajes interpretados no tengan uso (de hecho son los más utilizados), pero si se ven limitados para casos de exigencia máxima

GitHub: <https://github.com/KEPCU/FundamentalsOfProgrammingLanguages/tree/master/go/practice16>