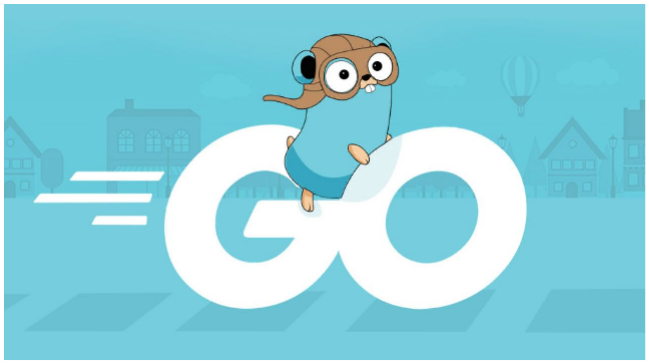
Ingeniería de Software
- Fundamentos de Lenguaje de Programación -

Evidencia 04 - Práctica 16

Estudiante: Prado Apaza Felix David
Profesor: Machaca Arceda Vicente Enrique
Fecha: 23 de junio del 2022

Introducción

La necesidad de ser un genio de las matemáticas para aprender código es cosa del pasado. Más lenguajes de programación de alto nivel ofrecen una alternativa al código de máquina de bajo nivel, lo que hace que la codificación sea más accesible que nunca. Pero con docenas de idiomas disponibles, ¿cuál es el mejor para desarrollar un proyecto? Independientemente de lo que se planee plasmar, lo mejor es aprender cuando y quienes usan un determinado lenguaje. En este documento se comparan tres lenguajes de programación muy famosos los cuales son: **Python, C++ y Golang**, con algoritmos de ordenamiento (se podría decir costosos) con entradas mas grandes cada vez, siendo el tiempo de ejecución la variable a analizar.



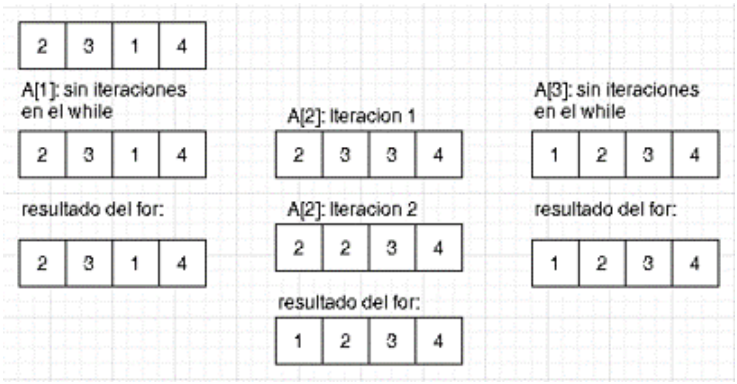
python™

Índice

1. Algoritmos

1.1. Insertion Sort

El Insertion Sort es un algoritmo de ordenamiento simple basado en comparaciones, en cual va recorriendo un array haciendo comparaciones por cada posición que recorra. Empieza desde la posición 1, y la compara con todos los elementos anteriores a él, haciendo comparaciones e intercambiando posiciones, hasta que encuentre un elemento que no sea mayor a el, insertando así cada elemento en su posición correcta.



Además que este algoritmo se considera estable, (Cuando elementos iguales indistinguibles entre sí, como números enteros, o más generalmente, cualquier tipo de dato en donde el elemento entero es la clave, la estabilidad no es un problema). Siendo que el caso promedio tiene una complejidad de:

$$\mathbf{O}(n^2)$$

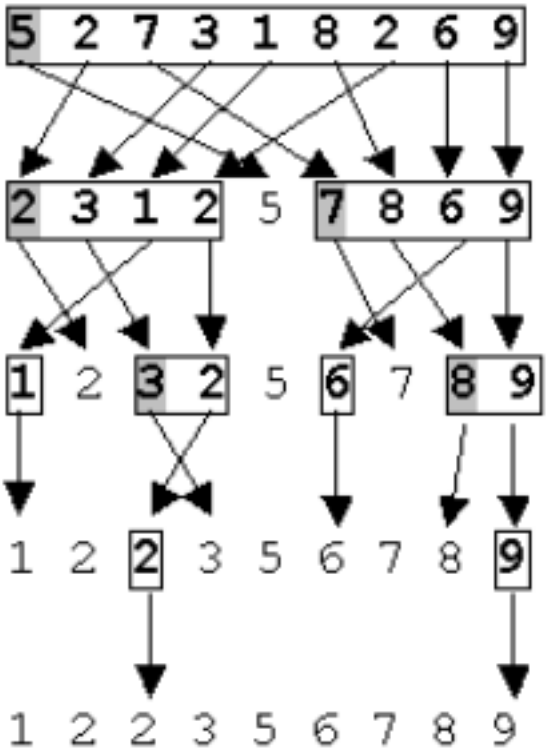
Algo si miliar a la del peor caso.

Pseudocódigo:

InsertionSort(A)
1. for $j \leftarrow 2$ to $length[A]$
2. $\quad key \leftarrow A[j]$
3. $\quad i \leftarrow j - 1$
4. $\quad$ while $i > 0$ and $A[i] > key$
5. $\quad\quad A[i + 1] \leftarrow A[i]$
6. $\quad\quad i \leftarrow i - 1$
7. $\quad A[i + 1] \leftarrow key$

1.2. Quicksort

Quicksort es un algoritmo de clasificación basado en el enfoque divide y vencerás donde Una matriz se divide en subarreglos seleccionando un elemento pivote (elemento seleccionado de la matriz). Al dividir la matriz, el elemento pivote debe colocarse de tal manera que los elementos menores que el pivote se mantengan en el lado izquierdo y los elementos mayores que el pivote estén en el lado derecho del pivote. Los subarreglos izquierdo y derecho también se dividen utilizando el mismo enfoque. Este proceso continúa hasta que cada subarreglo contiene un solo elemento. En este punto, los elementos ya están ordenados. Finalmente, los elementos se combinan para formar una matriz ordenada.



El caso promedio tiene una complejidad de:

$$\mathbf{O}(nlogn)$$

Algo parecida a la del mejor caso, siendo el peor caso una complejidad cuadrática.

Pseudocódigo:

```
quickSort(array, leftmostIndex, rightmostIndex)
  if (leftmostIndex < rightmostIndex)
    pivotIndex <- partition(array,leftmostIndex, rightmostIndex)
    quickSort(array, leftmostIndex, pivotIndex - 1)
    quickSort(array, pivotIndex, rightmostIndex)

partition(array, leftmostIndex, rightmostIndex)
  set rightmostIndex as pivotIndex
  storeIndex <- leftmostIndex - 1
  for i <- leftmostIndex + 1 to rightmostIndex
  if element[i] < pivotElement
    swap element[i] and element[storeIndex]
    storeIndex++
  swap pivotElement and element[storeIndex+1]
return storeIndex + 1
```

2. Implementación

La comparación se basa en entradas de arrays almacenadas en ficheros, para que sea la misma entrada en cada implementación de cada lenguaje, además de ello se itera 5 veces cada entrada y se halla el promedio y desviación estándar (para medir la dispersión de una distribución de los tiempos de ejecución).

2.1. Insertion Sort - Implementación

C++:

```
#include<iostream>
#include<stdlib.h>
#include<string.h>
#include <ctime>
#include<fstream>
#include<string>
#include <math.h>
using namespace std;

void insertionSort(int arr[], int n){
    int i, key, j;
    for (i = 1; i < n; i++){
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key){
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

void printArray(int arr[], int n){
    int i;
    cout << "[ ";
    for (i = 0; i < n; i++){
        cout << arr[i] << " ";
    }
    cout <<"]"<< endl;
}

int main(){
    string arrN[]={"test1.txt","test2.txt", "test3.txt", "test4.txt", "test5.txt",
        "test6.txt", "test7.txt", "test8.txt", "test9.txt", "test10.txt", "test11.txt",
        "test12.txt", "test13.txt", "test14.txt","test15.txt"};
        int arr2[]={100,1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000,
        9000, 10000, 20000, 30000, 40000,50000};
```

```
    for(int j=0;j<15;j++){
        double ltiempos[5];
        for(int m=0;m<5;m++){
            unsigned t0, t1;
            int n=arr2[j];
            int arr[n];

            ifstream archivo(arrN[j]);
            string texto;

            int i=0;
            while(getline(archivo,texto)){
                arr[i]=stoi(texto);
                i=i+1;
            }

            archivo.close();
            //printArray(arr,n);
            t0 = clock();
            insertionSort(arr, n);
            t1 = clock();
            double time = (double(t1-t0)/CLOCKS_PER_SEC);
            ltiempos[m]=time;
            //printArray(arr,n);
        }
        double promf = 0;
        for(int k=0;k<5;k++){
            promf += ltiempos[k];
        }
        promf = promf / 5;
        double dvs = 0;
        for(int p=0;p<5;p++){
            dvs += pow((ltiempos[p]-promf),2);
        }
        dvs=sqrt(dvs/4);
        cout<<"[Array Tamaño:"<<arr2[j]<<"][Promedio de Tiempo de ejecucion:"<<promf<<"]
        [Desviacion Estandar:"<<dvs<<"]"<<endl;
    }

        return 0;
}
```

Golang:

```
package main

import (
        "bufio"
        "fmt"
        "log"
        "math"
        "os"
        "strconv"
        "time"
)

func insertionSort(arr []int) {
        for j := 1; j < len(arr); j++ {
                key := arr[j]
                i := j - 1
                for i >= 0 && arr[i] > key {
                        arr[i+1] = arr[i]
                        i -= 1
                }
                arr[i+1] = key
        }
}

func printArray(arr []int) {
        fmt.Print("[")
        for i := 0; i < len(arr); i++ {
                fmt.Print(arr[i], " ")
        }
        fmt.Println("]")
}

func main() {
        nombres := []string{"test1.txt", "test2.txt", "test3.txt", "test4.txt",
        "test5.txt", "test6.txt", "test7.txt", "test8.txt", "test9.txt","test10.txt",
        "test11.txt", "test12.txt", "test13.txt", "test14.txt", "test15.txt"}
        tamaños := []int{100, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000,
        10000, 20000, 30000, 40000, 50000}
        // open the file
        for i := 0; i < 15; i++ {
                var ltiempos []float64
                for j := 0; j < 5; j++ {
                        var arr []int
                        file, err := os.Open(nombres[i])

                        //handle errors while opening
```

```
                        if err != nil {
                                log.Fatalf("Error when opening file: %s", err)
                        }

                        fileScanner := bufio.NewScanner(file)
                        for fileScanner.Scan() {
                                sv := fileScanner.Text()
                                if sv, err := strconv.Atoi(sv); err == nil {
                                        arr = append(arr, sv)
                                }
                        }
                        file.Close()
                        insertionSort(arr)
                        start := time.Now()
                        if j == 4 {
                                printArray(arr)
                        }
                        duration := time.Since(start).Seconds()
                        ltiempos = append(ltiempos, float64(duration))

                }
                var promf float64 = 0
                for k := 0; k < 5; k++ {
                        promf += ltiempos[k]
                }
                promf = promf / 5
                var dvs float64 = 0
                for p := 0; p < 5; p++ {
                        dvs += math.Pow(ltiempos[p]-promf, 2)
                }
                dvs = math.Sqrt(dvs / 4)
                fmt.Println("[Array Tamaño:", tamaños[i], "][Promedio de Tiempo de
                ejecucion:", promf, "][Desviacion Estandar:", dvs, "]")
        }

}
```

Python:

```
from tabulate import tabulate
import time
import math

def insertionSort(arr):
  for i in range(1, len(arr)):
    key = arr[i]
    j = i-1
    while j >= 0 and arr[j]>key :
      arr[j + 1] = arr[j]
      j -= 1
    arr[j + 1] = key

def printArray(arr):
  for i in range(len(arr)):
    print(arr[i])


ns=[100,1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000, 20000,
30000, 40000,50000]
prom=[]
dvs=[]

arr2=["test1.txt","test2.txt", "test3.txt", "test4.txt", "test5.txt", "test6.txt",
"test7.txt", "test8.txt", "test9.txt", "test10.txt", "test11.txt", "test12.txt",
"test13.txt", "test14.txt","test15.txt"]
for i in range(0,15):
  lprom=[]
  for k in range(0,5):
    arr = []
    archivo = open(arr2[i], "r", encoding='utf-8')
    for linea in archivo:
      arr.append(int(linea.strip()))
    archivo.close
    inicio = time.time()
    insertionSort(arr)
    fin = time.time()
    lprom.append(fin-inicio)
  promf = 0
  for j in range(0,5):
    promf += lprom[j]
  promf = (promf/5)
  prom.append(promf)
  devst=0
  for m in range(0,5):
    devst += pow(lprom[m]-promf,2)
```

```
  devst=math.sqrt(devst)
  dvs.append(devst)

print(tabulate({'n': ns, 'Promedio': prom, 'Desviacion Estandar': dvs}, headers="keys",
tablefmt='fancy_grid'))
```

2.2. Quicksort - Implementación

C++:

```
#include<iostream>
#include<stdlib.h>
#include<string.h>
#include <ctime>
#include<fstream>
#include<string>
#include <math.h>
#include <algorithm>
using namespace std;

void swap(int* a, int* b){
    int t = *a;
    *a = *b;
    *b = t;
}

void printArray(int arr[], int n){
    int i;
    cout << "[ ";
    for (i = 0; i < n; i++){
        cout << arr[i] << " ";
    }
    cout <<"]"<< endl;
}

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

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

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

```
}

int main(){
    string arrN[]={"test1.txt","test2.txt", "test3.txt", "test4.txt",
    "test5.txt", "test6.txt", "test7.txt", "test8.txt", "test9.txt",
    "test10.txt", "test11.txt", "test12.txt", "test13.txt", "test14.txt",
    "test15.txt"};
        int arr2[]={100,1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000,
    10000, 20000, 30000, 40000,50000};
    for(int j=0;j<15;j++){
        double ltiempos[5];
        for(int m=0;m<5;m++){
            unsigned t0, t1;
            int n=arr2[j];
            int arr[n];

            ifstream archivo(arrN[j]);
            string texto;

            int i=0;
            while(getline(archivo,texto)){
                arr[i]=stoi(texto);
                i=i+1;
            }

            archivo.close();
            //if(m==4){
                //printArray(arr,n);
            //}
            t0 = clock();
            quickSort(arr, 0, n - 1);
            t1 = clock();
            double time = (double(t1-t0)/CLOCKS_PER_SEC);
            ltiempos[m]=time;
            //if(m==4){
                //printArray(arr,n);
            //}
        }
        double promf = 0;
        for(int k=0;k<5;k++){
            promf += ltiempos[k];
        }
        promf = promf / 5;
        double dvs = 0;
        for(int p=0;p<5;p++){
            dvs += pow((ltiempos[p]-promf),2);
        }
        dvs=sqrt(dvs/4);
```

```
        cout<<"[Array Tamaño:"<<arr2[j]<<"][Promedio de Tiempo de
        ejecucion:"<<promf<<"][Desviacion Estandar:"<<dvs<<"]"<<endl;
    }
        return 0;
}
```

Golang:

```
package main

import (
        "bufio"
        "fmt"
        "log"
        "math"
        "os"
        "strconv"
        "time"
)

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

func quickSort(arr []int, low, high int) []int {
        if low < high {
                var p int
                arr, p = partition(arr, low, high)
                arr = quickSort(arr, low, p-1)
                arr = quickSort(arr, p+1, high)
        }
        return arr
}

func printArray(arr []int) {
        fmt.Print("[")
        for i := 0; i < len(arr); i++ {
                fmt.Print(arr[i], " ")
        }
        fmt.Println("]")
}

func main() {
        nombres := []string{"test1.txt", "test2.txt", "test3.txt", "test4.txt"
        , "test5.txt", "test6.txt", "test7.txt", "test8.txt", "test9.txt",
```

```
        "test10.txt", "test11.txt", "test12.txt", "test13.txt", "test14.txt",
        "test15.txt"}
        tamaños := []int{100, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000,
        9000, 10000, 20000, 30000, 40000, 50000}
        // open the file
        for i := 0; i < 15; i++ {
                var ltiempos []float64
                for j := 0; j < 5; j++ {
                        var arr []int
                        file, err := os.Open(nombres[i])

                        //handle errors while opening
                        if err != nil {
                                log.Fatalf("Error when opening file: %s", err)
                        }

                        fileScanner := bufio.NewScanner(file)
                        for fileScanner.Scan() {
                                sv := fileScanner.Text()
                                if sv, err := strconv.Atoi(sv); err == nil {
                                        arr = append(arr, sv)
                                }
                        }
                        file.Close()

                        start := time.Now()
                        quickSort(arr, 0, len(arr)-1)
                        duration := time.Since(start).Seconds()
                        ltiempos = append(ltiempos, float64(duration))

                }
                var promf float64 = 0
                for k := 0; k < 5; k++ {
                        promf += ltiempos[k]
                }
                promf = promf / 5
                var dvs float64 = 0
                for p := 0; p < 5; p++ {
                        dvs += math.Pow(ltiempos[p]-promf, 2)
                }
                dvs = math.Sqrt(dvs / 4)
                fmt.Println("[Array Tamaño:", tamaños[i], "][Promedio de Tiempo de
                ejecucion:", promf, "][Desviacion Estandar:", dvs, "]")

        }
}
```

Python:

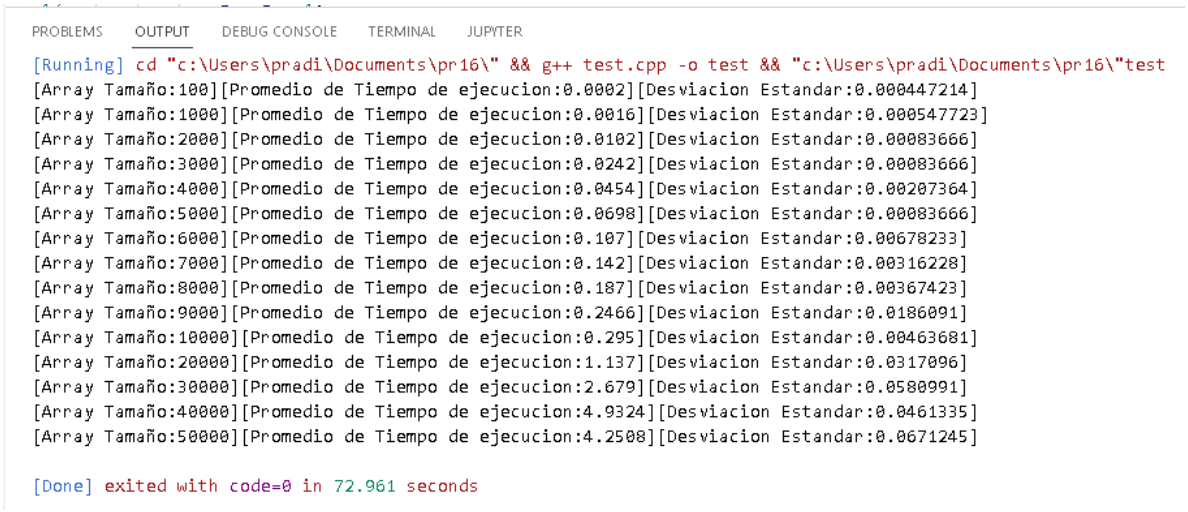
```
def partition(array, low, high):
  pivot = array[high]
  i = low - 1
  for j in range(low, high):
    if array[j] <= pivot:
      i = i + 1
      (array[i], array[j]) = (array[j], array[i])
  (array[i + 1], array[high]) = (array[high], array[i + 1])
  return i + 1


def quick_sort(array, low, high):
  if low < high:
    pi = partition(array, low, high)
    quick_sort(array, low, pi - 1)
    quick_sort(array, pi + 1, high)

ns=[100,1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000, 20000,
30000, 40000,50000]
prom=[]
dvs=[]
arr2=["test1.txt","test2.txt", "test3.txt", "test4.txt", "test5.txt", "test6.txt"
, "test7.txt", "test8.txt", "test9.txt", "test10.txt", "test11.txt", "test12.txt"
, "test13.txt", "test14.txt","test15.txt"]
for i in range(0,15):
    lprom=[]
    for k in range(0,5):
        arr = []
        archivo = open(arr2[i], "r", encoding='utf-8')
        for linea in archivo:
            arr.append(int(linea.strip()))
        archivo.close
        inicio = time.time()
        quick_sort(arr, 0, len(arr) - 1)
        fin = time.time()
        lprom.append(fin-inicio)
    promf = 0
    for j in range(0,5):
        promf += lprom[j]
    promf = (promf/5)
    prom.append(promf)
    devst=0
    for m in range(0,5):
        devst += pow(lprom[m]-promf,2)
    devst=math.sqrt(devst/4)
    dvs.append(devst)
```

```
print(tabulate({'n': ns, 'Promedio': prom, 'Desviacion Estandar': dvs}, headers="keys"
, tablefmt='fancy_grid'))
```

3. Resultados

3.1. Insertion Sort - Resultados

C++:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER
[Running] cd "c:\Users\pradi\Documents\pr16\" && g++ test.cpp -o test && "c:\Users\pradi\Documents\pr16\"test
[Array Tamaño:100][Promedio de Tiempo de ejecucion:0.0002][Desviacion Estandar:0.000447214]
[Array Tamaño:1000][Promedio de Tiempo de ejecucion:0.0016][Desviacion Estandar:0.000547723]
[Array Tamaño:2000][Promedio de Tiempo de ejecucion:0.0102][Desviacion Estandar:0.00083666]
[Array Tamaño:3000][Promedio de Tiempo de ejecucion:0.0242][Desviacion Estandar:0.00083666]
[Array Tamaño:4000][Promedio de Tiempo de ejecucion:0.0454][Desviacion Estandar:0.00207364]
[Array Tamaño:5000][Promedio de Tiempo de ejecucion:0.0698][Desviacion Estandar:0.00083666]
[Array Tamaño:6000][Promedio de Tiempo de ejecucion:0.107][Desviacion Estandar:0.00678233]
[Array Tamaño:7000][Promedio de Tiempo de ejecucion:0.142][Desviacion Estandar:0.00316228]
[Array Tamaño:8000][Promedio de Tiempo de ejecucion:0.187][Desviacion Estandar:0.00367423]
[Array Tamaño:9000][Promedio de Tiempo de ejecucion:0.2466][Desviacion Estandar:0.0186091]
[Array Tamaño:10000][Promedio de Tiempo de ejecucion:0.295][Desviacion Estandar:0.00463681]
[Array Tamaño:20000][Promedio de Tiempo de ejecucion:1.137][Desviacion Estandar:0.0317096]
[Array Tamaño:30000][Promedio de Tiempo de ejecucion:2.679][Desviacion Estandar:0.0580991]
[Array Tamaño:40000][Promedio de Tiempo de ejecucion:4.9324][Desviacion Estandar:0.0461335]
[Array Tamaño:50000][Promedio de Tiempo de ejecucion:4.2508][Desviacion Estandar:0.0671245]

[Done] exited with code=0 in 72.961 seconds
```

Golang:

```
[Running] go run "c:\Users\pradi\go\src\practica16\main.go"
[Array Tamaño: 100 ][Promedio de Tiempo de ejecucion: 0 ][Desviacion Estandar: 0 ]
[Array Tamaño: 1000 ][Promedio de Tiempo de ejecucion: 0.0004465 ][Desviacion Estandar: 0.0009984043519536562 ]
[Array Tamaño: 2000 ][Promedio de Tiempo de ejecucion: 0.00076118 ][Desviacion Estandar: 0.0017020502231132899 ]
[Array Tamaño: 3000 ][Promedio de Tiempo de ejecucion: 0.00293846 ][Desviacion Estandar: 0.006570596309164032 ]
[Array Tamaño: 4000 ][Promedio de Tiempo de ejecucion: 0.0015073 ][Desviacion Estandar: 0.0033704252624854334 ]
[Array Tamaño: 5000 ][Promedio de Tiempo de ejecucion: 0.0019066 ][Desviacion Estandar: 0.004263287205901099 ]
[Array Tamaño: 6000 ][Promedio de Tiempo de ejecucion: 0.0027046 ][Desviacion Estandar: 0.0060476694519459315 ]
[Array Tamaño: 7000 ][Promedio de Tiempo de ejecucion: 0.0035054599999999997 ][Desviacion Estandar: 0.007838446852406412 ]
[Array Tamaño: 8000 ][Promedio de Tiempo de ejecucion: 0.00313346 ][Desviacion Estandar: 0.007006629564776491 ]
[Array Tamaño: 9000 ][Promedio de Tiempo de ejecucion: 0.00379822 ][Desviacion Estandar: 0.008493078113499251 ]
[Array Tamaño: 10000 ][Promedio de Tiempo de ejecucion: 0.00824912 ][Desviacion Estandar: 0.018445593074553065 ]
[Array Tamaño: 20000 ][Promedio de Tiempo de ejecucion: 0.0085959 ][Desviacion Estandar: 0.019221016727790443 ]
[Array Tamaño: 30000 ][Promedio de Tiempo de ejecucion: 0.01200128 ][Desviacion Estandar: 0.026835677897008674 ]
[Array Tamaño: 40000 ][Promedio de Tiempo de ejecucion: 0.02119812 ][Desviacion Estandar: 0.047400437315197844 ]
[Array Tamaño: 50000 ][Promedio de Tiempo de ejecucion: 0.019599079999999998 ][Desviacion Estandar: 0.04382487517645658 ]

[Done] exited with code=0 in 17.722 seconds
```

Python:

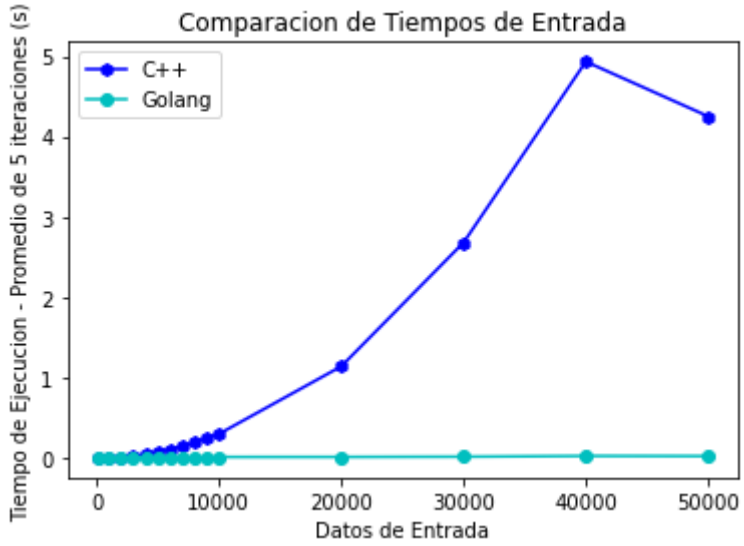
n	Promedio	Desviacion Estandar
100	0.00100408	1.05034e-05
1000	0.0766014	0.00461269
2000	0.408625	0.00815552
3000	1.06069	0.0371665
4000	1.83653	0.0235117
5000	2.92558	0.0462259
6000	4.23121	0.104465
7000	5.7844	0.160853
8000	7.59435	0.0836083
9000	9.81185	0.350118
10000	12.5024	0.618652
20000	46.8308	0.746774
30000	114.431	3.64621
40000	206.628	1.96436
50000	184.463	2.56335

Comparación de tiempo promedio			
n	C++	Golang	Python
100	0.0002	0.005856	0.00100408
1000	0.0016	0.000404208	0.0766014
2000	0.0102	0.0125483826	0.408625
3000	0.0242	0.0042027216	1.06069
4000	0.0454	0.0177015843	1.83653
5000	0.0698	0.0195619374	2.92558
6000	0.107	0.026065085	4.23121
7000	0.142	0.0420151652	5.7844
8000	0.187	0.059864852	7.59435
9000	0.2466	0.0678044218	9.81185
10000	0.295	0.0969972826	12.5024
20000	1.137	0.3137747325	46.8308
30000	2.679	0.7502539888	114.431
40000	4.9324	1.6356149908	206.628
50000	4.2508	1.45911639	184.463

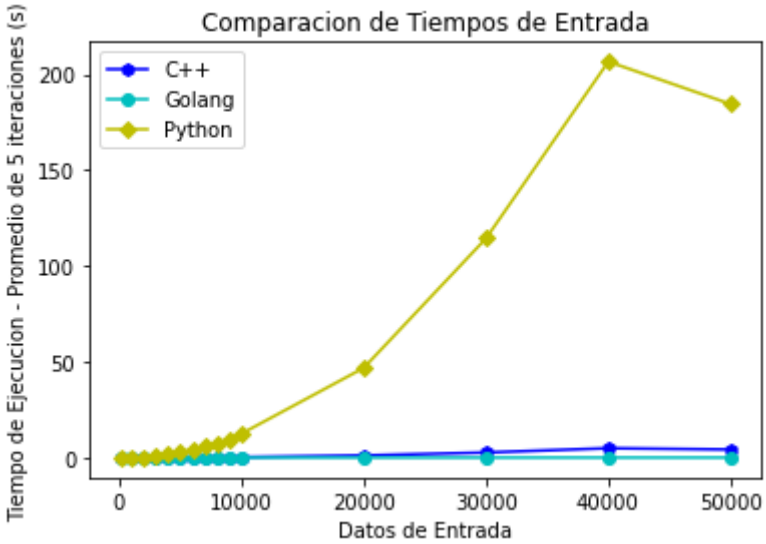
Comparación de desviaciones estándar de las pruebas			
n	C++	Golang	Python
100	0.000447214	0.00000474552	0.0000105034
1000	0.000547723	0.0000327922	0.00461269
2000	0.00083666	0.00013310592	0.00815552
3000	0.00083666	0.000693278	0.0371665
4000	0.00207364	0.001901152	0.0235117
5000	0.00083666	0.0286178306	0.0462259
6000	0.00678233	0.032125558	0.104465
7000	0.00316228	0.03160070	0.160853
8000	0.00367423	0.028834410	0.0836083
9000	0.0186091	0.03248487	0.350118
10000	0.00463681	0.03144509	0.618652
20000	0.0317096	0.11182786	0.746774
30000	0.0580991	0.10584663	3.64621
40000	0.0461335	0.118808819	1.96436
50000	0.0671245	0.1198302	2.56335

3.2. Insertion Sort - Gráficos

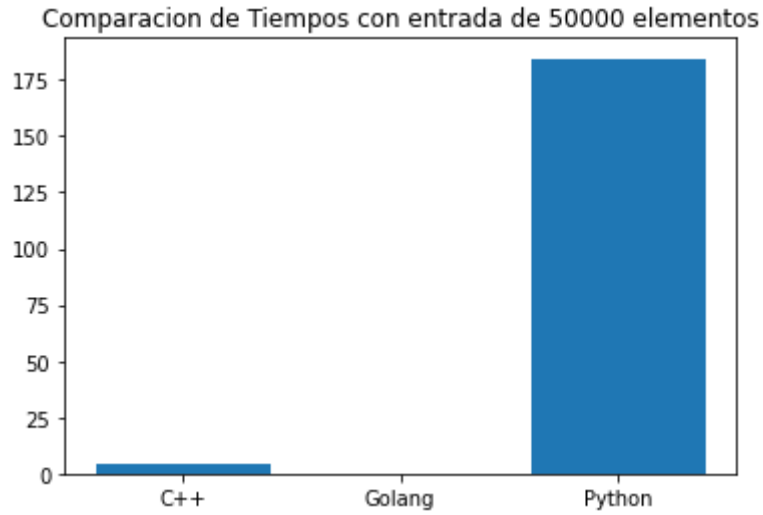
C++ vs Golang: Primero se realiza esta comparación, ya que los tiempos obtenidos en estos dos lenguajes son menores comparados con la cantidad de tiempo de ejecución exorbitante que se obtiene en Python.



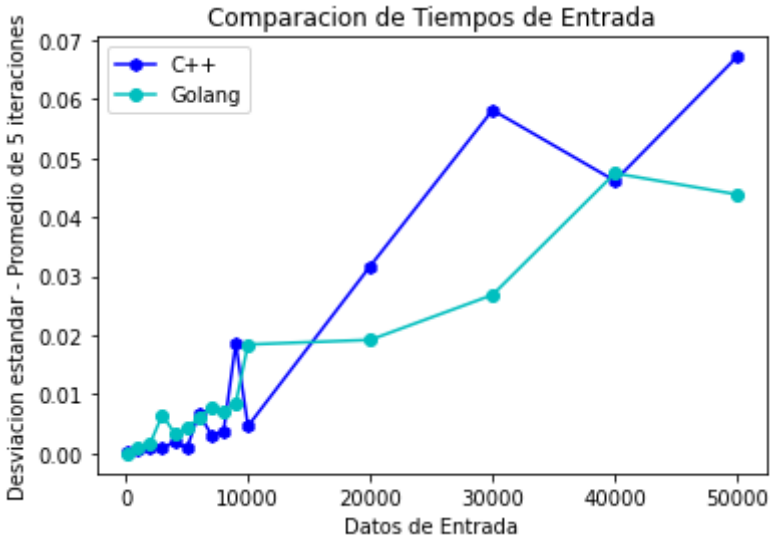
C++ vs Golang vs Python: Ahora se aprecia como Python tiene un mayor tiempo de ejecución con respecto al crecimiento de datos como entrada en el algoritmo, ya que al tener un costo cuadrático con su caso promedio, no se recomienda para entradas enormes.



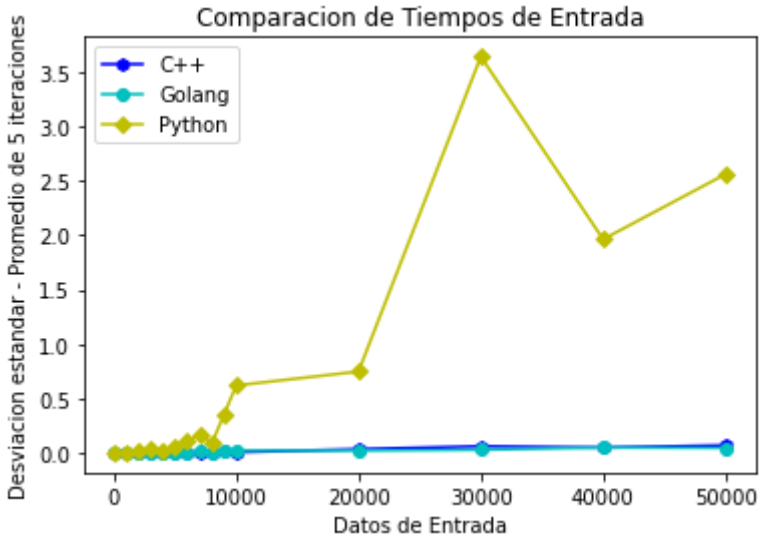
Una comparacion que demuestra aun mas esto, es con la cantidad de datos de 50000, ya que la diferencia de Python con C++ o Golang es demasiada.



Desviación Estándar:



La desviación estándar en C++ y Golang demuestra ser muy baja, en lo cual podemos intuir que su dispersión es baja y los tiempos de ejecución en cada uno de estos son muy parecidos en si.



Pero otra vez en Python la dispersión es muy alta en entradas grandes como de 10000 a 50000, llegando a tener poca exactitud y precisión en el calculo de estas.

3.3. Quicksort - Resultados

C++:

```
[Running] cd "c:\Users\pradi\Documents\pr16\" && g++ test.cpp -o test && "c:\Users\pradi\Documents\pr16\"test
[Array Tamaño:100][Promedio de Tiempo de ejecucion:0][Desviacion Estandar:0]
[Array Tamaño:1000][Promedio de Tiempo de ejecucion:0][Desviacion Estandar:0]
[Array Tamaño:2000][Promedio de Tiempo de ejecucion:0.0004][Desviacion Estandar:0.000547723]
[Array Tamaño:3000][Promedio de Tiempo de ejecucion:0.0004][Desviacion Estandar:0.000547723]
[Array Tamaño:4000][Promedio de Tiempo de ejecucion:0.0008][Desviacion Estandar:0.000447214]
[Array Tamaño:5000][Promedio de Tiempo de ejecucion:0.001][Desviacion Estandar:0]
[Array Tamaño:6000][Promedio de Tiempo de ejecucion:0.001][Desviacion Estandar:0]
[Array Tamaño:7000][Promedio de Tiempo de ejecucion:0.002][Desviacion Estandar:0]
[Array Tamaño:8000][Promedio de Tiempo de ejecucion:0.0022][Desviacion Estandar:0.000447214]
[Array Tamaño:9000][Promedio de Tiempo de ejecucion:0.0026][Desviacion Estandar:0.000547723]
[Array Tamaño:10000][Promedio de Tiempo de ejecucion:0.0028][Desviacion Estandar:0.000447214]
[Array Tamaño:20000][Promedio de Tiempo de ejecucion:0.0046][Desviacion Estandar:0.000547723]
[Array Tamaño:30000][Promedio de Tiempo de ejecucion:0.0072][Desviacion Estandar:0.000447214]
[Array Tamaño:40000][Promedio de Tiempo de ejecucion:0.011][Desviacion Estandar:0]
[Array Tamaño:50000][Promedio de Tiempo de ejecucion:0.0108][Desviacion Estandar:0.000447214]
```

Golang:

```
[Running] go run "c:\Users\pradi\go\src\practica16\main.go"
Quick Sort
[Array Tamaño: 100 ][Promedio de Tiempo de ejecucion: 0 ][Desviacion Estandar: 0 ]
[Array Tamaño: 1000 ][Promedio de Tiempo de ejecucion: 0.0001967 ][Desviacion Estandar: 0.0004398345711742087 ]
[Array Tamaño: 2000 ][Promedio de Tiempo de ejecucion: 0 ][Desviacion Estandar: 0 ]
[Array Tamaño: 3000 ][Promedio de Tiempo de ejecucion: 0.00019742000000000002 ][Desviacion Estandar: 0.0004414445401180085 ]
[Array Tamaño: 4000 ][Promedio de Tiempo de ejecucion: 0.00040008000000000003 ][Desviacion Estandar: 0.0005478326541928657 ]
[Array Tamaño: 5000 ][Promedio de Tiempo de ejecucion: 0.0009999000000000002 ][Desviacion Estandar: 3.52703841770967e-06 ]
[Array Tamaño: 6000 ][Promedio de Tiempo de ejecucion: 0.0004018 ][Desviacion Estandar: 0.0005501889493619442 ]
[Array Tamaño: 7000 ][Promedio de Tiempo de ejecucion: 0.00060324 ][Desviacion Estandar: 0.000550763568148802 ]
[Array Tamaño: 8000 ][Promedio de Tiempo de ejecucion: 0.00040178 ][Desviacion Estandar: 0.0005501719749314754 ]
[Array Tamaño: 9000 ][Promedio de Tiempo de ejecucion: 0.0005999800000000001 ][Desviacion Estandar: 0.0005477473979856042 ]
[Array Tamaño: 10000 ][Promedio de Tiempo de ejecucion: 0.00060158 ][Desviacion Estandar: 0.0005491754837936595 ]
[Array Tamaño: 20000 ][Promedio de Tiempo de ejecucion: 0.0015106999999999998 ][Desviacion Estandar: 0.0004999190884533216 ]
[Array Tamaño: 30000 ][Promedio de Tiempo de ejecucion: 0.0018048799999999996 ][Desviacion Estandar: 0.00044448697056269273 ]
[Array Tamaño: 40000 ][Promedio de Tiempo de ejecucion: 0.00240008 ][Desviacion Estandar: 0.0005394693476000282 ]
[Array Tamaño: 50000 ][Promedio de Tiempo de ejecucion: 0.00320328 ][Desviacion Estandar: 0.0004484163322181743 ]
```

Python:

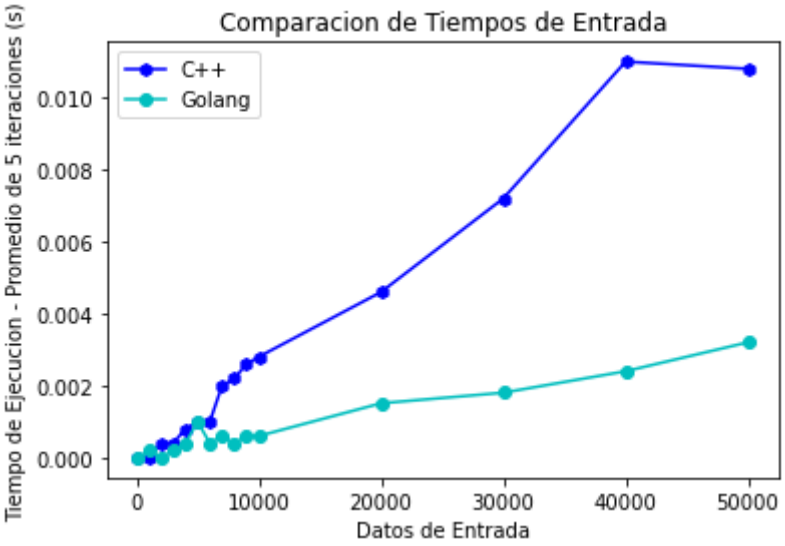
n	Promedio	Desviacion Estandar
100	0.000200129	0.000447501
1000	0.00320306	0.000448823
2000	0.00720301	0.00109116
3000	0.018599	0.0175587
4000	0.0203972	0.0043398
5000	0.0209984	4.65863e-06
6000	0.0325986	0.000889512
7000	0.0412038	0.000838069
8000	0.0498005	0.00110208
9000	0.0533999	0.00219091
10000	0.0679967	0.00157531
20000	0.104403	0.00336384
30000	0.159396	0.00167632
40000	0.2506	0.00669269
50000	0.245797	0.00535653

Comparación de tiempo promedio			
n	C++	Golang	Python
100	0.000000	0	0.000200128555
1000	0.000000	0.0001967	0.0032030582
2000	0.000400	0	0.00720300674
3000	0.000400	0.00019742	0.01859903
4000	0.000800	0.00040008	0.02039723
5000	0.001000	0.0009999	0.020998383
6000	0.001000	0.0004018	0.03259859
7000	0.002000	0.00060324	0.04120383
8000	0.002200	0.00040178	0.049800539
9000	0.002600	0.00059998	0.05339989662
10000	0.002800	0.00060158	0.06799669
20000	0.004600	0.0015107	0.1044034004
30000	0.007200	0.00180488	0.159396
40000	0.011000	0.00240008	0.25059995651
50000	0.010800	0.00320328	0.2457973

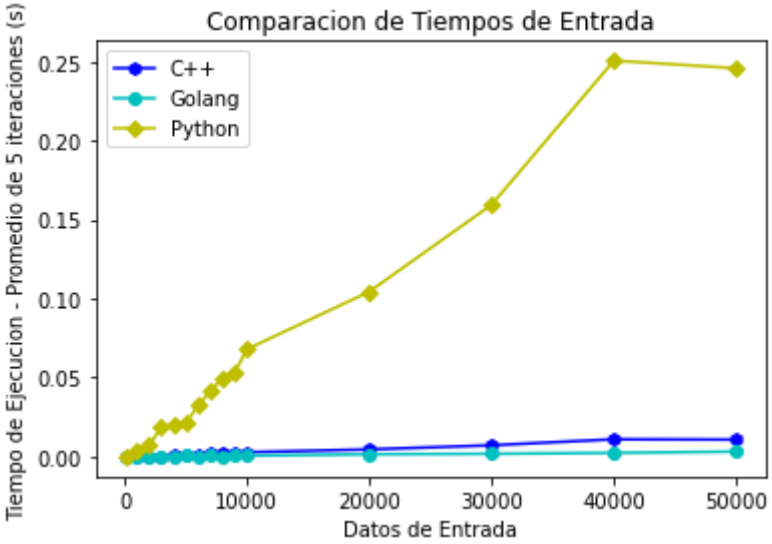
Comparación de desviaciones estándar de las pruebas			
n	C++	Golang	Python
100	0.000000	0	0.000447501
1000	0.000000	0.000439835	0.000448823
2000	0.000548	0	0.0010911628
3000	0.000548	0.0004414445401	0.01755869999
4000	0.000447	0.000547832654	0.004339802
5000	0.000000	0.00035270	0.0000046586
6000	0.000000	0.00055019	0.00088951
7000	0.000000	0.00055076	0.000838069
8000	0.000447	0.00055017	0.00110208
9000	0.000548	0.000547747	0.00219090558
10000	0.000447	0.000549175	0.0015753073
20000	0.000548	0.000499919	0.0033638437
30000	0.000447	0.00044448697	0.0016763173
40000	0.000000	0.000539469	0.006692692
50000	0.000447	0.000448416	0.0053565269

3.4. Quicksort - Gráficos

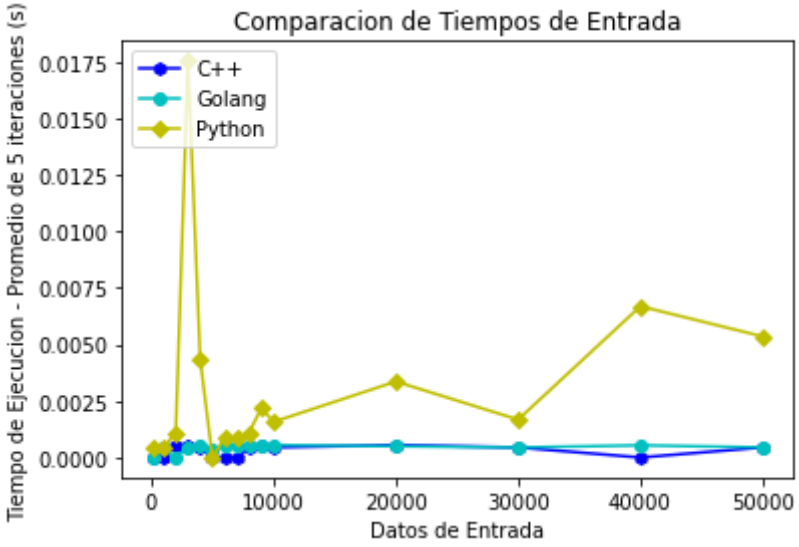
C++ vs Golang: Como en el anterior algoritmo, otra vez Golang demuestra un tiempo de ejecución bajo, y C++ uno casi igual, siendo Quicksort uno de los algoritmos de ordenamientos de mas baja complejidad.



C++ vs Golang vs Python: Bueno a comparación de las pruebas en el otro algoritmo, Python tiene un tiempo de ejecución relativamente bajo (obviamente por el Quicksort), pero sigue siendo el mas alto con respecto a C++ y Golang.



Desviación Estándar:



Y bueno otra vez la dispersión en C++ y Golang es muy baja, pero ahora en Python esta dispersión es muy alta en casi todas las entradas de datos, como aleatoriamente.

4. Conclusiones

Bueno, con el resultado de las pruebas, se puede decir que Golang es el mas rápido en comparación con los otros 2 (Python quedando como ultima opción), pero esto se debe a algunas características de Golang, ya que su lenguaje es de alto nivel (C++ es lenguaje de bajo nivel), apunta a un compromiso entre velocidad y simplicidad, logra concurrencia a través de goroutines (a comparación de C++ que tiene simultaneidad lograda a través de subprocesos del sistema operativo), en general, Golang supera a C++ en lo que respecta a la velocidad de codificación, pero esto no quiere decir que Golang sea el lenguaje predeterminado para cualquier proyecto o que Python no sea nada favorable, ya que Python es dinámico, tiene un intuitivo lenguaje, la gestión de memoria es automática, y otros, lo que importa esta en ver las utilidades de cada lenguaje: Python con el Machine Learning, AI, Back-end; C++ para codificación a nivel de hardware en sistemas embebidos y Golang para Backend, APIs RESTful, scripts para sistemas, Sockets. Todo lenguaje siempre tendrá cualidades fuertes en diferentes ámbitos, y por ello es necesario ser variado a la hora de programar.

5. Notas

Las pruebas del Insertion Sort se realizaron en Visual Studio Code de una misma maquina para la igualdad en la comparación.

Todos siguen la misma lógica del pseudocódigo, sin usar librerías para el ordenamiento o algún proceso del algoritmo; ya en el main de cada uno se realiza la lectura de los ficheros que contienen los elementos diferentes y repartidos aleatoriamente para generar el caso promedio, se lee cada uno de estos, se itera 5 veces, se calcula el tiempo promedio y desviación estándar y se imprime, siendo ese el output.

6. Enlace Github

FLP - Repositorio de Prado Apaza Felix David