



Análisis de Algoritmos e Iterador de Combinaciones

PROGRAMACIÓN III

AMUSQUÍVAR POPPE, JOSÉ MARÍA

JESWANI TEJWANI, PRASHANT

MARÍN REYES, IGNACIO JOSÉ

INDICE

Parte 1: Análisis de Algoritmos: Algoritmo 1 (Gauss-Jordan Elimination)	3
1. Nombre del algoritmo y enlace a la web del código	3
2. Breve descripción del algoritmo	3
3. Fragmento de código fuente con el algoritmo (solamente en uno de los 3 lenguajes elegidos)....	4
4. Análisis asintótico del algoritmo	4
5. Coste empírico de ejecución del algoritmo en los 3 lenguajes elegidos (sustituir en la siguiente tabla 'Lenguaje 1' .. 'Lenguaje 3' por los lenguajes de programación elegidos, y adaptar los valores de N para que el coste de ejecución del algoritmo esté en un rango entre 0 y 60 segundos).....	5
6. Copia de pantalla que muestre el uso del programa desde consola activando la opción que muestra el tiempo consumido en la ejecución del programa (en los tres lenguajes)	6
7. [Opcional] Gráfico que muestra el coste empírico de ejecución en los 3 lenguajes procesando un fichero con los datos de entrada.....	7
 Parte 1: Análisis de Algoritmos: Algoritmo 2 (Suma de Polinomios)	8
1. Nombre del algoritmo y enlace a la web del código	8
2. Breve descripción del algoritmo	8
3. Fragmento de código fuente con el algoritmo (solamente en uno de los 3 lenguajes elegidos).....	9
4. Análisis asintótico del algoritmo	9
5. Coste empírico de ejecución del algoritmo en los 3 lenguajes elegidos (sustituir en la siguiente tabla 'Lenguaje 1' .. 'Lenguaje 3' por los lenguajes de programación elegidos, y adaptar los valores de N para que el coste de ejecución del algoritmo esté en un rango entre 0 y 60 segundos).....	10
6. Copia de pantalla que muestre el uso del programa desde consola activando la opción que muestra el tiempo consumido en la ejecución del programa (en los tres lenguajes)	11
7. [Opcional] Gráfico que muestra el coste empírico de ejecución en los 3 lenguajes procesando un fichero con los datos de entrada.....	12
 Parte 2: Iterador de Combinaciones	13
1. Breve explicación del iterador de permutaciones o combinaciones utilizado en cada lenguaje (en caso de haber utilizado un iterador disponible en el lenguaje, referenciar la página Web de documentación del iterador utilizado; en caso de haberlo programado, referenciar la página Web que describe el algoritmo implementado, o explicar el algoritmo implementado; en caso de haber programado el algoritmo descrito en clase solamente hay que especificar 'Algoritmo visto en clase').....	13
2. Código fuente del iterador (en uno de los tres lenguajes)	13
3. Copia de pantalla que muestre el uso del iterador (en los tres lenguajes)	14

Programación III
Informe de la Práctica 2
(Análisis de Algoritmos / Iterador de Combinaciones)

1. Nombre y apellido de los miembros del grupo

José María Amusquívar Poppe

Prashant Jeswani Tejwani

Ignacio José Marín Reyes

2. Horario de laboratorio asignado al grupo para defender las prácticas (Martes 12:30-14:30, Miércoles 10:30-12:30, Jueves 8:30-10:30, etc)

La defensa del trabajo se realizará en la hora de prácticas del grupo 44, grupo al que pertenecemos (Jueves 10:30-12:30).

3. Lenguajes de programación elegidos.

Java, C++ y Python.

******Información adicional******

El archivo comprimido *.zip proporcionado contiene dos carpetas, una para la medición de los tiempos de los algoritmos, y otra carpeta con el desarrollo de los iteradores. Dentro de la carpeta de los algoritmos, se encuentra una carpeta por cada proyecto (Java, C++, y Python), además de un proyecto escrito en Java que se encarga de generar los vectores aleatorios.

Para hacer más sencilla la tarea de calcular los tiempos, se desarrolló un script “.bat” en Windows, que ejecuta un bucle generando cada vez un número mayor de vectores, los cuales son pasados a cada proyecto, con lo que se obtiene finalmente dos ficheros de texto en la carpeta creada llamada “Ejecutables”, uno contiene los tiempos del algoritmo Gauss-Jordan Elimination, y el otro de la Suma de Polinomios.

Para cualquier consulta adicional, se proporciona un fichero de texto “usage.txt” con las correspondientes instrucciones de compilación, ejecución y empaquetamiento “.jar”.

Parte 1: Análisis de Algoritmos: Algoritmo 1 (Gauss-Jordan Elimination)

1. Nombre del algoritmo y enlace a la web del código

El algoritmo principal que se ha implementado es Gauss-Jordan Elimination, método que se usa para resolver un sistema lineal de ecuaciones.

Su enlace es:

<https://www.geeksforgeeks.org/program-for-gauss-jordan-elimination-method/>

2. Breve descripción del algoritmo

Este algoritmo calcula todas las incógnitas de un sistema lineal de ecuaciones cuadrado, donde el número de vectores es igual al número de columnas más uno, ya que también se incluyen los términos independientes.

Para realizar este algoritmo se recibe una matriz de vectores, donde se dejan ceros por encima de la diagonal y por debajo de ella, de haber algún cero en la diagonal, se permuta el vector que la contiene con otro vector hasta conseguir evitar el cero en la diagonal, en caso de ser todos ceros, se produciría una indeterminación de tipo $(k/0)$ o $(0/0)$.

Finalmente, se realiza la división del término independiente de cada vector con la respectiva diagonal que le pertenece, consiguiendo un valor que será solución de su ecuación.

```
Input : 2y + z = 4
         x + y + 2z = 6
         2x + y + z = 7

Output :
Final Augumented Matrix is :
1 0 0 2.2
0 2 0 2.8
0 0 -2.5 -3

Result is : 2.2 1.4 1.2
```

A pesar de que la primera ecuación no tiene la incógnita "x", se considera como un 0 a la hora de pasarle un vector. De tal modo que queda: $0x + 2y + z = 4 \rightarrow \{0, 2, 1, 4\}$

3. Fragmento de código fuente con el algoritmo (solamente en uno de los 3 lenguajes elegidos)

```
// Performing elementary operations
long startTN = System.nanoTime();
long startTM = System.currentTimeMillis();
for (i = 0; i < orden; i++) {
    if (vector[i][i] == 0) {
        c = 1;

        while (vector[i + c][i] == 0 && (i + c) < orden) c++;

        if ((i + c) == orden) {
            flag = 1;
            break;
        }

        for (j = i, k = 0; k <= orden; k++) {
            float temp = vector[j][k];
            vector[j][k] = vector[j+c][k];
            vector[j+c][k] = temp;
        }
    }

    for (j = 0; j < orden; j++) {
        // Excluding all i == j
        if (i != j) {
            // Converting Matrix to reduced row
            // echelon form(diagonal matrix)
            float p = vector[j][i] / vector[i][i];

            for (k = 0; k <= orden; k++){
                vector[j][k] = vector[j][k] - (vector[i][k]) * p;
            }
        }
    }
}
long endTM = System.currentTimeMillis();
long endTN = System.nanoTime();

timeMillis = (endTM - startTM);
timeNano = (endTN - startTN);
return flag;
```

4. Análisis asintótico del algoritmo

Usamos el análisis abreviado para determinar el orden del algoritmo. Para ello identificamos la operación crítica de nuestro algoritmo, es decir, la operación elemental que se ejecuta el mayor número de veces, que es:

$\text{vector}[j][k] = \text{vector}[j][k] - (\text{vector}[i][k]) * p$, cuyo coste es $O(1)$.

Por tanto, como esta operación crítica tiene tres bucles anidados independientes, determinamos que el algoritmo es de $O(n^3)$.

```

// Performing elementary operations
long startTN = System.nanoTime();
long startTM = System.currentTimeMillis();
for (i = 0; i < orden; i++) {
    if (vector[i][i] == 0) {
        c = 1;

        while (vector[i + c][i] == 0 && (i + c) < orden) c++;

        if ((i + c) == orden) {
            flag = 1;
            break;
        }

        for (j = i, k = 0; k <= orden; k++) {
            float temp = vector[j][k];
            vector[j][k] = vector[j+c][k];
            vector[j+c][k] = temp;
        }
    }

    for (j = 0; j < orden; j++) {
        // Excluding all i == j
        if (i != j) {
            // Converting Matrix to reduced row
            // echelon form(diagonal matrix)
            float p = vector[j][i] / vector[i][i];

            for (k = 0; k <= orden; k++){
                vector[j][k] = vector[j][k] - (vector[i][k]) * p;
            }
        }
    }
}

long endTM = System.currentTimeMillis();
long endTN = System.nanoTime();

timeMillis = (endTM - startTM);
timeNano = (endTN - startTN);
return flag;

```

$O(n^3)$

$O(n^2)$

$O(n)$

Operación crítica

$\in O(n^3)$

5. Coste empírico de ejecución del algoritmo en los 3 lenguajes elegidos (sustituir en la siguiente tabla 'Lenguaje 1' .. 'Lenguaje 3' por los lenguajes de programación elegidos, y adaptar los valores de N para que el coste de ejecución del algoritmo esté en un rango entre 0 y 60 segundos).

*Nota: todos los tiempos están expresados en unidad de segundos, y N = Número de vectores.


N	Java	Python	C++
100	0,0250000000	0,2499935627	0,00500000000
200	0,0160000000	2,6443128586	0.03900000
300	0,0470000000	8,7213804722	0.15000001
400	0,1100000000	19,8767807484	0.32200000
500	0,1710000000	38,5774137974	0.62900001
600	0,3300000001	65,3023676872	1,06799996000

1000	1,5629999600		5,2979998600
1400	4,2690000500		14,4589996300
1800	9,1479997600		32,9280014000
2200	15,93000031		56,70500183000
2600	26,18899918		95,92600250000
3000	46,20700073		
3400	58,41899872		
3800	85,80899811		

***No se han calculado más valores para Python y C++ ya que se exceden del minuto.

6. Copia de pantalla que muestre el uso del programa desde consola activando la opción que muestra el tiempo consumido en la ejecución del programa (en los tres lenguajes)

JAVA

 Símbolo del sistema

```
C:\Users\YO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmJava\src>ls
algorithmjava  org


C:\Users\YO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmJava\src>javac -version
javac 1.8.0_162

C:\Users\YO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmJava\src>java -version
java version "1.8.0_231"
Java(TM) SE Runtime Environment (build 1.8.0_231-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.231-b11, mixed mode)

C:\Users\YO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmJava\src>javac algorithmjava/*.java org/apache/commons/cli/*.java
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

C:\Users\YO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmJava\src>java algorithmjava.AlgorithmJava -f ../../Vectores -dt
For 400 vectors -> Taken time: 0,09400000 s.
```

C++

 Símbolo del sistema

```
C:\Users\YO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmCpp>ls
AddPoly.cpp  AlgorithmCpp.cpp  ParserArgs.cpp  ReadFile.cpp  SolveEquation.cpp  build  longOptions.h
AddPoly.h    Makefile           ParserArgs.h    ReadFile.h    SolveEquation.h    dist  nbproject

C:\Users\YO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmCpp>g++ --version
g++ (GCC) 7.4.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

C:\Users\YO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmCpp>g++ -o output AlgorithmCpp.cpp AddPoly.cpp ParserArgs.cpp ReadFile.cpp SolveEquation.cpp

C:\Users\YO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmCpp>output.exe -f ../../Vectores -t
For 400 vectors -> Taken time: 0.31999999 s.
```

PYTHON

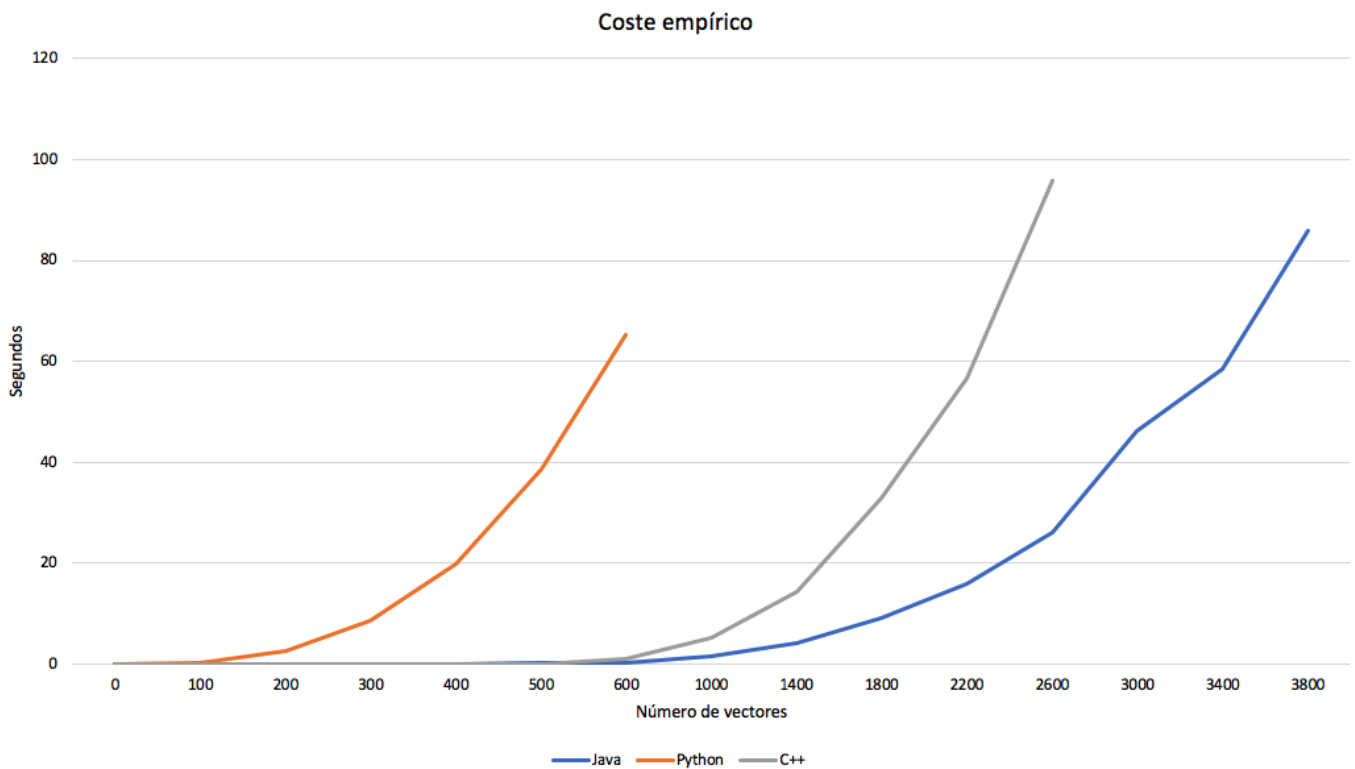
Simbolo del sistema

```
C:\Users\YO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmPython\AlgorithmPython>ls
AlgorithmPython.py  AlgorithmPython.pyproj

C:\Users\YO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmPython\AlgorithmPython>python --version
Python 3.8.0

C:\Users\YO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmPython\AlgorithmPython>python AlgorithmPython.py -f ../../Vectores -dt
For 400 vectors -> Taken time: 27.513564586639404 s
```

7. [Opcional] Gráfico que muestra el coste empírico de ejecución en los 3 lenguajes procesando un fichero con los datos de entrada.



Parte 1: Análisis de Algoritmos: Algoritmo 2 (Suma de Polinomios)

1. Nombre del algoritmo y enlace a la web del código

Se ha implementado, además, otro algoritmo, el cual suma todas las filas de la matriz tratándolos como vectores independientes (para ejecutar este algoritmo se usará la opción '-a').

Su enlace:

<https://www.geeksforgeeks.org/program-add-two-polynomials/>

2. Breve descripción del algoritmo

Este algoritmo se apoya en el anterior, ya que simplemente con añadir la opción "-a" a línea de comandos, se ejecutará una suma de todos los vectores de la matriz, en lugar de realizar el cálculo de las incógnitas mediante Gauss-Jordan.

Para realizar este algoritmo, se recibe una matriz (al igual que el algoritmo anterior), se va iterando por columnas, sumando todos los valores de cada columna y almacenándolo en un vector, que será el que se devolverá.

```
Input:  A[] = {5, 0, 10}
        B[] = {1, 2, 4}
Output: sum[] = {6, 2, 14}
```

```
The first input array represents "5x^2+0x^1+10"
The second array represents "1x^2+2x^1+4"
And Output is "6x^2+2x^1+14"
```

Donde el vector A y el vector B, conforman una matriz de 2x3, de orden 2.

3. Fragmento de código fuente con el algoritmo (solamente en uno de los 3 lenguajes elegidos)

```
public void add() {
    sum = new float[size];

    long startTN = System.nanoTime();
    long startTM = System.currentTimeMillis();
    for (int i = 0; i < size; i++) {
        for(int j = 0; j < size-1; j++){
            sum[i] += vector[j][i];
        }
    }
    long endTM = System.currentTimeMillis();
    long endTN = System.nanoTime();

    timeMillis = (endTM - startTM);
    timeNano = (endTN - startTN);
}
```

4. Análisis asintótico del algoritmo

Usamos el análisis abreviado para determinar el orden del algoritmo. Para ello identificamos la operación crítica de nuestro algoritmo, es decir, la operación elemental que se ejecuta el mayor número de veces, que es:

`sum[i] += vector[j][i]`, cuyo coste es $O(1)$.

Por tanto, como esta operación crítica tiene dos bucles anidados independientes, determinamos que el algoritmo es de $O(n^2)$.

```
public void add() {
    sum = new float[size];

    long startTN = System.nanoTime();
    long startTM = System.currentTimeMillis();
    for (int i = 0; i < size; i++) {
        for(int j = 0; j < size-1; j++){
            sum[i] += vector[j][i];
        }
    }
    long endTM = System.currentTimeMillis();
    long endTN = System.nanoTime();

    timeMillis = (endTM - startTM);
    timeNano = (endTN - startTN);
}
```

$O(n^2)$
 $O(n)$
Operación crítica

$\in O(n^2)$

5. Coste empírico de ejecución del algoritmo en los 3 lenguajes elegidos (sustituir en la siguiente tabla ‘Lenguaje 1’ .. ‘Lenguaje 3’ por los lenguajes de programación elegidos, y adaptar los valores de N para que el coste de ejecución del algoritmo esté en un rango entre 0 y 60 segundos).

*Nota: todos los tiempos están expresados en unidad de segundos, y N = Número de vectores.

N	Java	Python	C++
1000	0,0160000000	0,2343676090	0,0100000000
3000	0,2029999900	2,4694247246	0,1080000000
5000	0,5320000100	7,1523075104	0,3339999900
7000	1,1879999600	25,4874370098	0,6880000200
9000	2,1349999900	42,7354497910	1,5340000400
11000	2,8570001100	49,4665472507	1,9450000500
13000	4,6209998100	84,35600615	2,7939991000
15000	6,0260000200	104,8131249	5,2420001000
20000	15,7939967000		11,9180002200
25000	30,9190006300		26,7469997400
30000	<i>OutOffMemoryError</i>		37,1819992100
35000			46,3610000600
40000			62,2949981700
45000			78,7119979900

Se puede observar que el lenguaje más rápido en ejecutar este algoritmo es C++, al cual le sigue Java, y finalmente Python.

Cabe destacar que al intentar resolver el sistema de ecuaciones con una matriz de orden 30000, en Java salta un error de memoria de tipo *OutOffMemory: GC overhead limit exceeded*, el cual indica que el recolector de basura se ejecuta todo el tiempo y que el programa Java avanza muy lentamente. Después de una recolección de basura, el proceso de Java está gastando más del, aproximadamente, 98% de su tiempo haciendo recolección de basura y está recuperando menos del 2% del montón. Y en ocasiones, salta otro error de memoria, pero de tipo *OutOffMemory: heap space*, el cual indica que el algoritmo hace un uso excesivo de finalizadores y no tienen su espacio reclamado en el momento de la recolección de basura.

6. Copia de pantalla que muestre el uso del programa desde consola activando la opción que muestra el tiempo consumido en la ejecución del programa (en los tres lenguajes)

JAVA

```
cmd x + v
C:\Users\COCO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmJava\src>ls
algorithmjava  org

C:\Users\COCO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmJava\src>javac -version
javac 1.8.0_231

C:\Users\COCO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmJava\src>java -version
java version "1.8.0_231"
Java(TM) SE Runtime Environment (build 1.8.0_231-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.231-b11, mixed mode)

C:\Users\COCO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmJava\src>javac algorithmjava/*.java org/apache/commons/cli/*.java
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

C:\Users\COCO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmJava\src>java algorithmjava.AlgorithmJava -f ../../Vectores -dt -a
For 400 vectors -> Taken time: 0,00314330 s.
```

C++

```
cmd x + v
C:\Users\COCO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmCpp>ls
AddPoly.cpp AddPoly.h AlgorithmCpp.cpp Makefile ParserArgs.cpp ParserArgs.h ReadFile.cpp ReadFile.h SolveEquation.cpp SolveEquation.h build dist longOptions.h nbproject output.exe

C:\Users\COCO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmCpp>g++ --version
g++ (GCC) 7.4.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

C:\Users\COCO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmCpp>g++ -o output AlgorithmCpp.cpp AddPoly.cpp ParserArgs.cpp ReadFile.cpp SolveEquation.cpp

C:\Users\COCO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmCpp>output.exe -f ../../Vectores -t -a
For 400 vectors -> Taken time: 0.00066110 s.
```

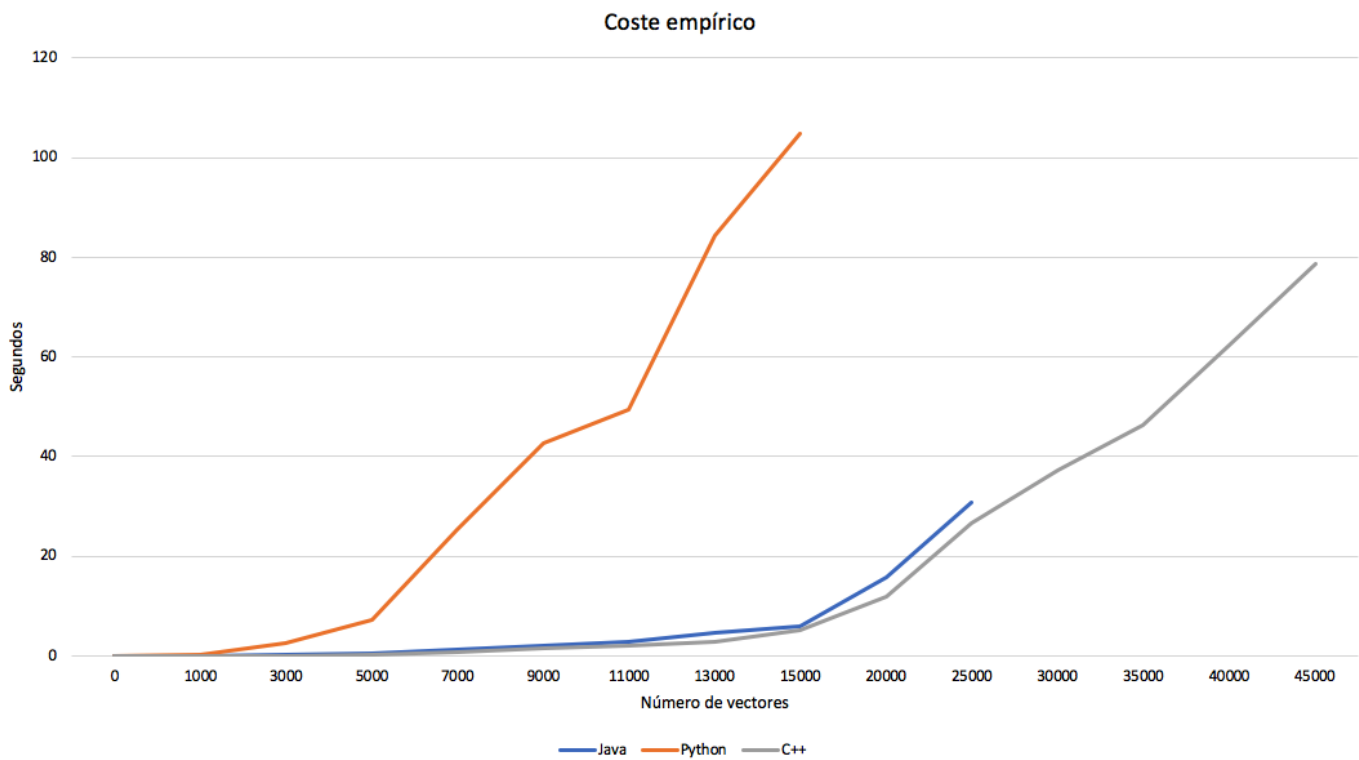
PYTHON

```
cmd x + v
C:\Users\COCO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmPython\AlgorithmPython>ls
AlgorithmPython.py  AlgorithmPython.pyproj

C:\Users\COCO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmPython\AlgorithmPython>python --version
Python 3.8.0

C:\Users\COCO\Desktop\PR3Practica2\AlgorithmAnalysis\AlgorithmPython\AlgorithmPython>python AlgorithmPython.py -f ../../Vectores -dt -a
For 400 vectors -> Taken time: 0.031248807907104492 s
```

7. [Opcional] Gráfico que muestra el coste empírico de ejecución en los 3 lenguajes procesando un fichero con los datos de entrada.



Parte 2: Iterador de Combinaciones

1. Breve explicación del iterador de permutaciones o combinaciones utilizado en cada lenguaje (en caso de haber utilizado un iterador disponible en el lenguaje, referenciar la página Web de documentación del iterador utilizado; en caso de haberlo programado, referenciar la página Web que describe el algoritmo implementado, o explicar el algoritmo implementado; en caso de haber programado el algoritmo descrito en clase solamente hay que especificar ‘Algoritmo visto en clase’).

Algoritmo visto en clase.

En los lenguajes utilizados, Java, C++ y Python hemos programado un iterador de combinaciones, para ello, hemos usado en todos los lenguajes una clase `IteratorOfCombination`, que almacena la combinación actual, obtiene la siguiente combinación a través del método `siguienteCombinacion()`, y verifica que hemos alcanzado la última combinación a través de `ultimaCombinacion()`. La combinación actual se basa en un array de contadores que van de 0 a 1.

Por tanto, el iterador nos permite obtener cada una de las combinaciones posibles de un conjunto de números, devolviéndonos en cada momento la siguiente combinación.

Existen diferencias en la programación de cada iterador debido a las particularidades de cada lenguaje, por ejemplo, en C++ hemos tenido que hacer uso de punteros. El resto de particularidades se pueden comprobar en el código fuente de cada lenguaje.

2. Código fuente del iterador (en uno de los tres lenguajes)

Main.py

```
from IteradorCombinaciones import *
import sys

if __name__ == "__main__":
    combinacion = []
    it = IteradorCombinaciones(len(sys.argv) - 1)
    while not it.ultimaCombinacion():
        combinacion = it.siguienteCombinacion()
        for i in range(0, len(sys.argv) - 1):
            if combinacion[i] == 1:
                print(sys.argv[i + 1] + " ", end="")
        print()
```

IteradorCombinaciones.py

```
class IteradorCombinaciones:

    def __init__(self, N):
        self.valorMaximo = 1
        self.valorMinimo = 0
        self.combinacion = []
        for i in range(N):
            self.combinacion.append(0)

    def ultimaCombinacion(self):
        for i in self.combinacion:
            if i == 0:
                return False
        return True

    def siguienteCombinacion(self):
        for i in range(0, len(self.combinacion)):
            if self.combinacion[i] == self.valorMaximo:
                self.combinacion[i] = self.valorMinimo
            else:
                self.combinacion[i] = self.combinacion[i] + 1
        return self.combinacion
```

3. Copia de pantalla que muestre el uso del iterador (en los tres lenguajes)

Para hacer uso del iterador le pasamos al programa por parámetro el conjunto de números, de los cuáles obtendremos todas las combinaciones posibles.

Python

```
[MacBook-Pro-de-ignacio-marin-reyes:IteradorPhyton ignaciomarinreyes$ python3 Main.py 10 20 30 40
10
20
10 20
30
10 30
20 30
10 20 30
40
10 40
20 40
10 20 40
30 40
10 30 40
20 30 40
10 20 30 40
```

C++

```
[MacBook-Pro-de-ignacio-marin-reyes:GNU-MacOSX ignaciomarinreyes$ ./iteradorcombinaciones__ 20 10 50
20
10
20 10
50
20 50
10 50
20 10 50
```

Java

```
[MacBook-Pro-de-ignacio-marin-reyes:dist ignaciomarinreyes$ java -jar IteradorCombinacionesJava.jar 40 50 60
40
50
40 50
60
40 60
50 60
40 50 60
```