Análisis comparativo de eficiencia de lenguajes de programación

Tymoschuk, Jorge Pablo

Universidad Tecnológica Nacional, Facultad Regional Córdoba

Abstract

El objetivo de este estudio es realizar un análisis comparativo de la eficiencia de diferentes lenguajes para ejecutar un algoritmo determinado. La tarea se ejecutará en la misma plataforma (equipo, sistema operativo) variándose los lenguajes. Asimismo evaluarán también variantes por el uso de distintas máquinas virtuales y/o versiones (interpretado/compilado) para un mismo lenguaje. El equipo utilizado es una PC con placa madre AMD Athlon™ II X2 240 con procesador 2.81 GHz, 1.75 GB de RAM. El sistema operativo es Microsoft Windows XP, Profesional, versión 2002, Service Pack 3.

Los lenguajes/entornos/variantes utilizados los siguientes:

Java, NetBeans 7.3.1, JDK 1.6, JVM jrockit

Java, NetBeans 7.3.1, JDK 1.7, JVM default

C++, Microsoft Visual Studio .NET 2003

C++, NetBeans 3.7.1

Python , PyCharm 2.7.1, python.exe

Python , PyCharm 2.7.1, pypy.exe

Smalltalk, Squeak-4.3

Palabras Clave

Eficiencia, algoritmo, Java, C++, Smalltalk, Python

Introducción

A continuación transcribo un párrafo de la Unidad 1 del programa actualmente vigente en la asignatura Paradigmas de Programación, UTN FRC. (Cursiva)

4. Lenguajes de programación

Para obtener una solución utilizando un determinado paradigma se necesita de un Lenguaje de Programación. Pero con demasiada frecuencia, los lenguajes son seleccionados por razones equivocadas tales como: el fanatismo ("... es genial"), los prejuicios ("... es una basura"), la inercia ("... Es demasiado problema para aprender"), el miedo al cambio ("... es lo que mejor sabemos hacer, con todos sus defectos"), la moda ("... es lo que todos están utilizando ahora "), las presiones comerciales (" ... con el apoyo de MM. "), conformismo (" nadie fue despedido por la elección de ... "). Tales influencias sociales y emocionales son a menudo decisivas y reflejan un triste estado del software.

En el punto: *4.2 Criterios de evaluación de los lenguajes de programación* se describen 14 (catorce) criterios a tener en cuenta a la hora de decidir que lenguaje usar*: 1 – Legibilidad / 2 – Codificación / 3 – Fiabilidad / 4 – Escala / 5 – Modularidad / 6 – Reutilización / 7 – Portabilidad / 8 – Nivel / 9 – Eficiencia / 10 – Modelado de datos / 11 – Modelado de procesos / 12 – Disponibilidad de compiladores y herramientas / 13 – Familiaridad / 14 – Costo.*

El punto ***9 -*** **Eficiencia** es el que abordo en el presente estudio. Que este punto sea 1 de 14 a considerar en los criterios de evaluación deja en claro lo acotado del presente trabajo.

Es interesante transcribir aquí lo que dice el párrafo del material de la cátedra PPR respecto al punto *9 – eficiencia*, sobre todo porque las conclusiones de este trabajo no avalan lo allí expuesto:

9 – Eficiencia ¿Es el lenguaje capaz de ser aplicado de manera eficiente? Algunos aspectos de la programación orientada a objetos implican overheads en tiempo de

ejecución, tales como las etiquetas de clase y distribución dinámica. Los controles en tiempo de ejecución son costosos (aunque algunos compiladores están dispuestos a suprimir, por cuenta y riesgo del programador). La recolección de basura también es costosa. Además se debe tener en cuenta que el código interpretativo es aproximadamente diez veces más lento que el código de máquina nativo. Si las partes críticas del programa deben ser altamente eficientes, el lenguaje les permite estar sintonizados para recurrir a la codificación de bajo nivel, o mediante llamadas a procedimientos escritos en un lenguaje de bajo nivel.

El algoritmo que trataremos para medir la eficiencia de los diferentes lenguajes fue expuesto por el Ing. Valerio Fritelli en el curso **Fundamentos del Lenguaje Phyton – Nivel I**, 48 hs reloj, dictado entre el 19 de febrero y 7 de marzo del corriente año. El algoritmo, llamado **Prob2_SUM** debía ser resuelto en forma individual por c/u de los participantes.
El nombre del algoritmo **Prob2_SUM** hace referencia a la suma de 2 términos. Sucintamente se trata de determinar si un número cualquiera está constituido por la suma de 2 términos, siendo ambos elementos de un conjunto acotado de valores.

Elementos del trabajo y metodología.

Los elementos utilizados son los descriptos en el Abstract. Utilizando los lenguajes allí citados(Java, C++, Python, Smalltalk) y sus entornos, se programa el algoritmo **Prob2_SUM** en c/u de estos lenguajes ellos siguiendo la metodología de 5 etapas que describo a continuación:

1) Carga de un arreglo numérico de 100000 elementos (arregloV) a partir de la lectura de un archivo de texto (lote01.txt). Cada línea contiene un número (arreglo de caracteres, sólo dígitos), único, sin repeticiones.

2) Recorrer el arregloV e insertar cada número en un diccionario (tabla hash). El número en cuestión era la clave, su valor asociado no tenía importancia.

3) Informar cuales de los términos de una lista de 9 elementos enteros satisfacían el requisito 2_SUM, esto es, si se cumplía que:

   a) términoLista = termino1Arreglo + termino2Arreglo, siendo que
   b) termino2Arreglo = terminoLista – termino1Arreglo
   c) Dado que termino1Arreglo necesariamente existe en la tabla por su carga desde el arregloV, resta comprobar que termino2arreglo también existía en dicha tabla.

Normalmente esta estrategia se implementa mediante un par de ciclos anidados. El ciclo externo recorre la lista de los 9 números, para c/u de ellos el ciclo interno va tomando sucesivos términos de arregloV, y efectúa los pasos b) y c) arriba descriptos.

El ciclo interno se interrumpe si termino2Arreglo existe en la tabla (Pueden existir varios pares de términos de arregloV que satisfagan el requisito 2_SUM). Si ningún par satisface el requisito, el ciclo interno recorre el arregloV hasta el fin.

El resultado del cumplimiento de esta condición se expresa mediante:

‘1’ - requisito cumplido
‘0’ - requisito no cumplido.

Como la lista es de 9 elementos a verificar, el resultado se almacena en un arreglo de 9 elementos tipo

char: el casillero correspondiente a la posición del número en la lista contendrá '1' por requisito cumplido, '0' en caso contrario. Para el caso concreto con el que trabajó esa lista de 9 elementos el resultado correcto es "101001110".

Hasta aquí es el trabajo pedido en el curso **Fundamentos del Lenguaje Phyton.** Sin embargo, esta no es la única metodología o estrategia para resolver el problema. Una alternativa es buscar el termino2Arreglo en el propio arregloV. Como el arreglo no fue cargado en ningún orden de secuencia deberíamos utilizar búsqueda lineal, y hacer esto en un arreglo de 100000 elementos, recorriéndolo muchas veces, tantas como sucesivos candidatos termino2Arreglo vamos obteniendo, es sin mucho análisis una pésima opción. Una alternativa mucho mejor es:

4) Ordenar arregloV utilizando quickSort, por ejemplo.
5) Buscar el termino2Arreglo en arregloV usando búsqueda binaria.

Hasta ahora he descrito la metodología o estrategia para llegar al resultado pretendido, pero no he dicho nada de lo relacionado a la medición de la eficiencia, que en definitiva es el resultado pretendido. Es hora de hacerlo:
En cada uno de los lenguajes utilizados existen bibliotecas o clases que con comportamiento relacionado con tiempo y/o fecha. Esto hemos usado, no fue necesario un observador externo provisto de un buen cronómetro. De cualquier manera, los tiempos a computar necesitaban precisión de milisegundos, un observador humano serviría de poco. La metodología utilizada o secuencia de pasos constaba de las siguientes etapas:

a) Registro del tiempo de inicio de la etapa.
b) Ejecución de la etapa.
c) Registro del tiempo de fin de la etapa.
d) Cómputo del tiempo neto (fin – inicio).
e) Exhibición del tiempo neto.
f) Acumulación del tiempo neto de la etapa para obtener el tiempo total.

Resultados

Los resultados obtenidos por los procesamientos realizados están contenidos en las 5 captures de pantalla siguientes**.**

Capture 1 - Lenguaje Python, entorno Pycharm 2.7

Se realizan 2 procesamientos:

a) Utilizando plataforma Python 3.3.3, modalidad interpretado. El tiempo total registrado es de 7283,1513 milisegundos

b) Utilizando plataforma PyPy 1.9.0, modalidad compilado. El tiempo total registrado es de 1497,4249 milisegundos

Capture 2 – Lenguaje Smalltalk, entorno Squeak 4.3, modalidad interpretado.

Se realiza un único procesamiento. El tiempo total registrado es de 16584 milisegs.

Capture 3 – Lenguaje C++, entorno Microsoft Visual Studio (Compilado)

Se realiza un único procesamiento. El tiempo total registrado es de 655 milisegundos

Capture 4 – Lenguaje Java, entorno Netbeans 3.7.1

Se realizan 2 procesamientos:

a) Utilizando plataforma JDK 1.7 con máquina virtual default. El tiempo total registrado es de 312 milisegundos

b) Utilizando plataforma JDK 1.6, con máquina virtual jrockit. El tiempo total registrado es de 219 milisegundos

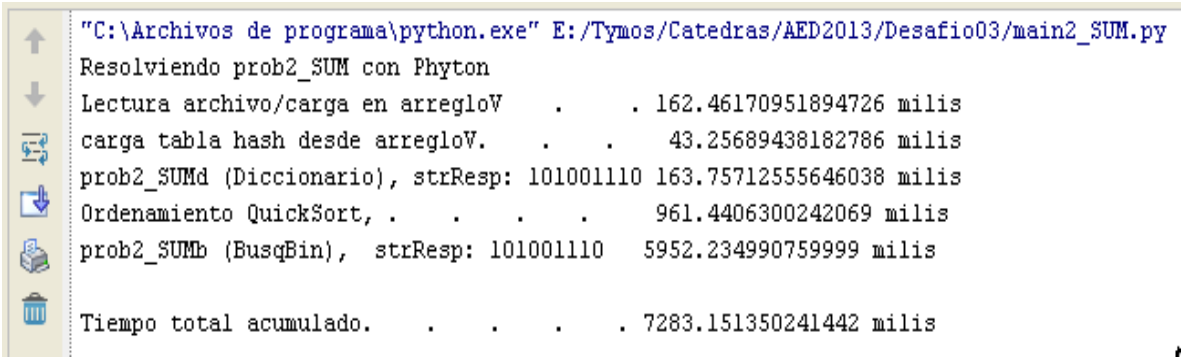
Capture 5 – Lenguaje C++, entorno Netbeans (Compilado)

Se realiza un único procesamiento. El tiempo total registrado es de 335 milisegundos

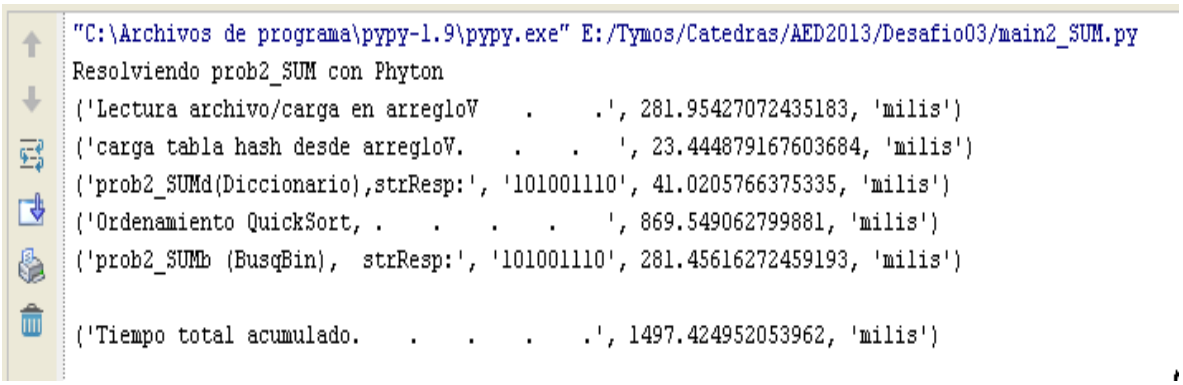
A continuación siguen los captures de pantalla citados

Capture 1 – Lenguaje Python – Entorno PyCharm

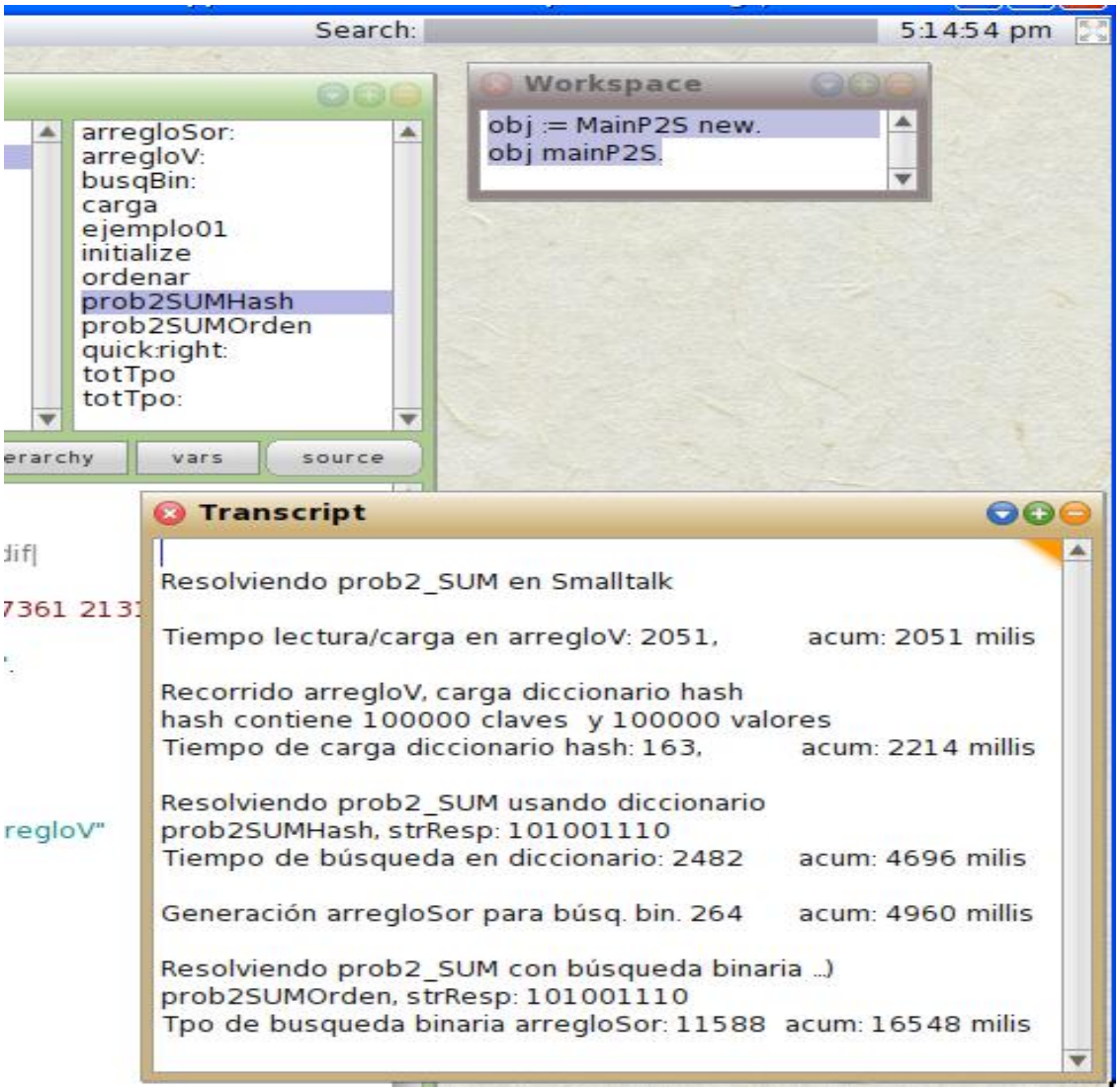
1 - Plataforma phyton 3.3.3 (**interpretado**)



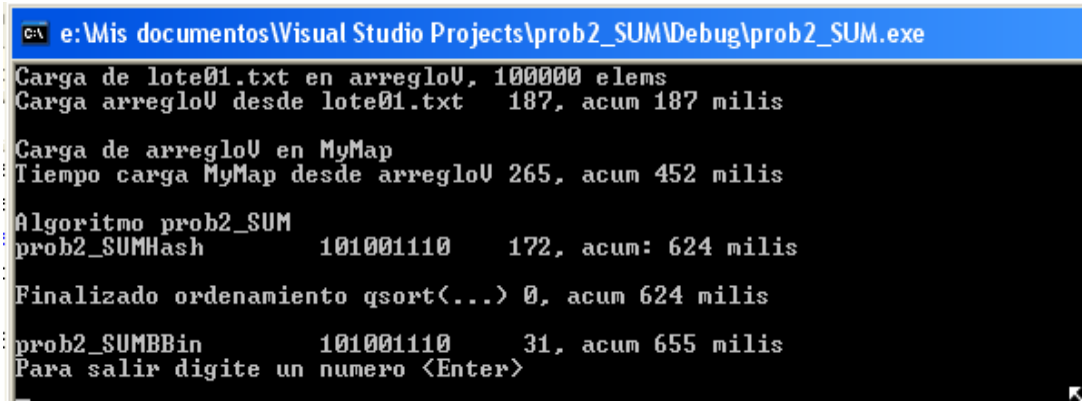
2 – Plataforma PyPy 1.9.0 (**Compilado**)



Capture 2 – Lenguaje Smalltalk – Entorno Squeak

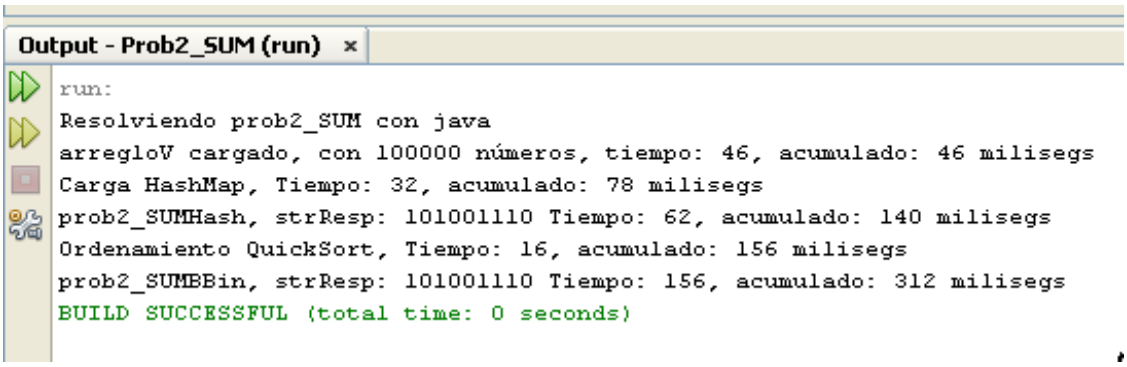


Capture 3 - C++ – Entorno Microsoft Visual Studio

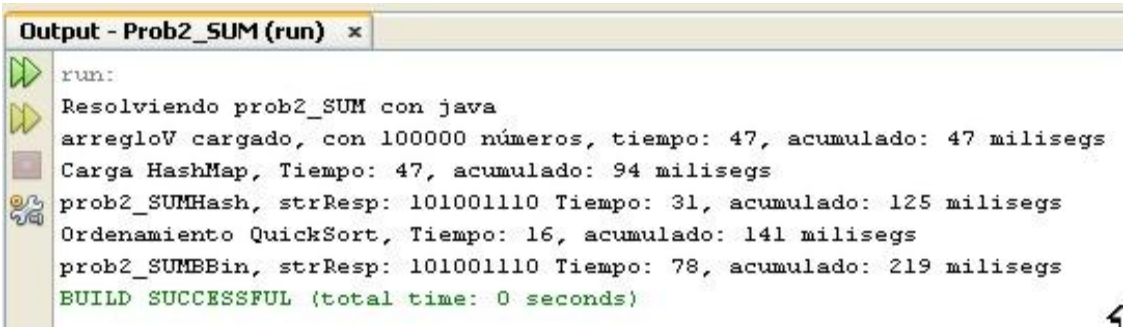


Capture 4 - Java – Entorno Netbeans 3.7.1

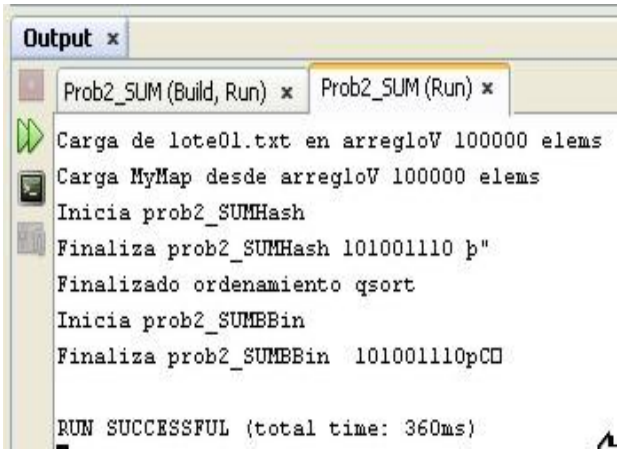
1) - Plataforma JDK 1.7 (Default), máquina virtual default



2) – Plataforma JDK 1.6, con máquina virtual jrockit (Oracle)



Capture 5 – C++ - Entorno NetBeans



Considerando 25 mseg como tiempo medio de carga, Acumulado total 360 - 25 = **335 miliseg**.

Discusión En este trabajo hubo algunos resultados inesperados/sorpresas.

1) Existe la idea generalizada de que un lenguaje que se ejecuta utilizando lenguaje de máquina es mucho más rápido en que cualquier otra opción.
En el presente trabajo hemos utilizado lenguajes con tres opciones.
   - Lenguaje de máquina
     - C++ (Microsoft Visual Studio, compilado)
     - C++ (NetBeans 3.7.1, compilado)
     - Python 3.3.0 (compilador PyPy.exe 1.9.0)
   - Lenguaje interpretado
     - Smalltalk
     - Python 3.3.0
   - Byte Code Java
     - Java JDK 1.7, máquina virtual default
     - Java JDK 1.6, con máquina virtual jrockit

   Si comparamos lenguaje de máquina Vs Byte Code Java

   - Java (máquina virtual default), tiempo total: 312 milisegundos.
   - Java (máquina virtual jrockit), tiempo total: 219 milisegundos.
   - C++ (Microsoft Visual Studio, compilado): 655 milisegundos.
   - C++ (NetBeans 3.7.1, compilado): 335 milisegundos.
   - Python 3.3.0 (compilador PyPy.exe) 1497,4249 miliseg.

   Las relaciones son:
   - Java 655/219 = 2,99 veces más rápido que C++ M.V.Studio.
   - Java 335/219 = 1,53 veces más rápido que C++ NetBeans
   - Java 1497,4249/219 = 6,84 veces más rápido que Python compilado con PyPy

   Si comparamos lenguaje interpretado Vs Byte Code Java
   - Java (máquina virtual default), tiempo total: 312 milisegundos.
   - Java (máquina virtual jrockit), tiempo total: 219 milisegundos.
   - Smalltalk interpretad 16584 milisegundos.
   - Python interpretado

     7283,15 milis.

   Las relaciones son:
   - Java 16584/219 = 75,73 veces más rápido que Smalltalk.
   - Java 7283,15 /219 = 33,26 veces más rápido que Python

2) - Algunos lenguajes no tienen recursos para resolver este problema:
Por ejemplo, Borland C++, se pueden definir dinámicamente el tamaño de arreglos mediante el operador new. Pero ocurre que el parámetro esperado por new es de tipo int, implementado en 2 bytes, y que puede valer como máximo 32.767 o sea que es imposible definir un arreglo de 100000 casilleros.

Conclusión
Son muy dispares los tiempos utilizados por los diferentes lenguajes. Esto hace que para compararlos sea poco adecuado referirse a porcentajes, es más adecuado usar órdenes de magnitud.
Efectivamente, si comparamos Java (Byte Code) con Smalltalk (interpretado):

SmallTalk, tiempo total: 16584 milis.
Java, tiempo total: 219 milis.

Podemos decir que Java demora **1,32%** de lo que demora Smalltalk. Pero es más claro decir que Java es 75,73 veces más rápido que Smalltalk, como ya lo expuse en la discusión.

Esta disparidad torna necesario categorizar los lenguajes:
Eficientes o eficaces. Las definiciones de estos conceptos son:

Eficiencia: podemos definirla como la relación entre los recursos utilizados en un proyecto y los logros conseguidos con el mismo. Se entiende que la eficiencia se da cuando se utilizan menos recursos para lograr un mismo objetivo. O al contrario, cuando se logran más objetivos con los mismos o menos recursos.

Eficacia: podemos definirla como el nivel de consecución de metas y objetivos. La eficacia referencia a nuestra capacidad para lograr lo que nos proponemos.

Las categorizaciones de lenguajes que podemos hacer:

Eficientes:

Java (con VM default o jrockit) está en primer lugar

Luego siguen los **C++** (compilados) en cualquiera de los entornos utilizados.

Un poco más atrás Python con PyPy.

Eficaces: A los dos restantes, (Phyton interpretado, Smalltalk) debemos reconocerles eficacia, dado que son capaces de resolver el problema, pero obviamente no consideran el tiempo un factor importante.

Es asimismo muy dispar el nivel de ayuda que proporcionan los distintos entornos de programación de los lenguajes utilizados en este trabajo.
Esta conclusión no correspondería al análisis del presente trabajo. Pero a la hora de efectivamente programar algoritmos que si bien son simples usan recursos específicos del lenguaje, se puede tropezar con dificultades. Algunos entornos brindan toda la ayuda eventualmente requerida en línea, inclusive abundan en ejemplos codificados, mientras que en otros apenas existen algunos comentarios documentativos para los informáticos que desarrollan estos lenguajes. Este es un tema muy importante, y debería agregarse a los 14 puntos que actualmente considera el material de la cátedra PPR relacionados a los criterios de evaluación de un lenguaje de programación.

Agradecimientos
Ing. Valerio Fritelli. Aporta el tema algorítmico Prob2_SUM, usado en este trabajo. Aporta el soft Python 3.3.3 (Interpretado)
Ing. Julio Castillo. Aporta información sobre el soft. PyPy 1.9.0, (Python compilado). Aporta información sobre "A Gentle Introduction to Haskell"
Ing. Pablo Frías Aporta Información sobre soft. Jrockit (Java compilado)
Cont. Jeremías Tymoschuk. Graficación
Ing. Soledad Romero. Revisión del material/Sugerencias/Correcciones

Referencias
[1] – Material de estudio de la cátedra Paradigmas de Programación. UTN FRC.
[2] – A Gentle Introduction to Haskell. Reuben Thomas, Paul Hudak
[3] – Tutoriales y ayudas en línea de los diferentes lenguajes
[4] – Programación en C++. Luis Joyanes Aguilar, editorial Mc Graw Hill
[5] – Programación Orientada a Objetos con C++. Francisco Javier Ceballos Sierra, editorial Ra-Ma
[6] – Estructuras de datos y algoritmos en Java. Goodrich/Tamasia. Editorial Continental.

Datos de contacto
Jorge Pablo Tymoschuk
Universidad Tecnológica Nacional Facultad Regional Córdoba
Albert Sabin 6012 Dpto 2, Villa Belgrano, Córdoba, Pcia Córdoba, Argentina.
jptymos@gmail.com

Tablas – Todos los tiempos en milisegundos. Cada línea de la tabla 1 es graficada mediante diagramas de barras. En el caso de C++ NetBeans sólo se dispone del tiempo total acumulado.

Tabla 1 – Tiempos (milisegundos) por etapa / lenguaje de programación

A	B	C	D	E	F	G	H
	Python 3-3-3	PyPy 1-9	Smalltalk	Java Default	Java Jrockit	C++ MVS	C++ NetBeans
Lectura/carga arregloV	162,46	281,95	2051	46	47	187	
Carga tabla hash	43,26	23,44	163	32	47	250	
Búsqueda tabla hash	163,76	41,02	2482	62	31	172	
Orden QuickSort	961,44	869,55	264	16	16	15	
Búsqueda binaria	5952,23	281,46	11588	156	78	31	
Total acumulado	7283,15	1497,42	16548	312	219	655	335

Tabla 2 – Tiempo acumulado total (milisegundos)

Python 3-3-3	PyPy 1-9	Smalltalk	Java Default	Java Jrockit	C++ MVS	C++ NetBeans
7283,15	1497,42	16548	312	219	655	335

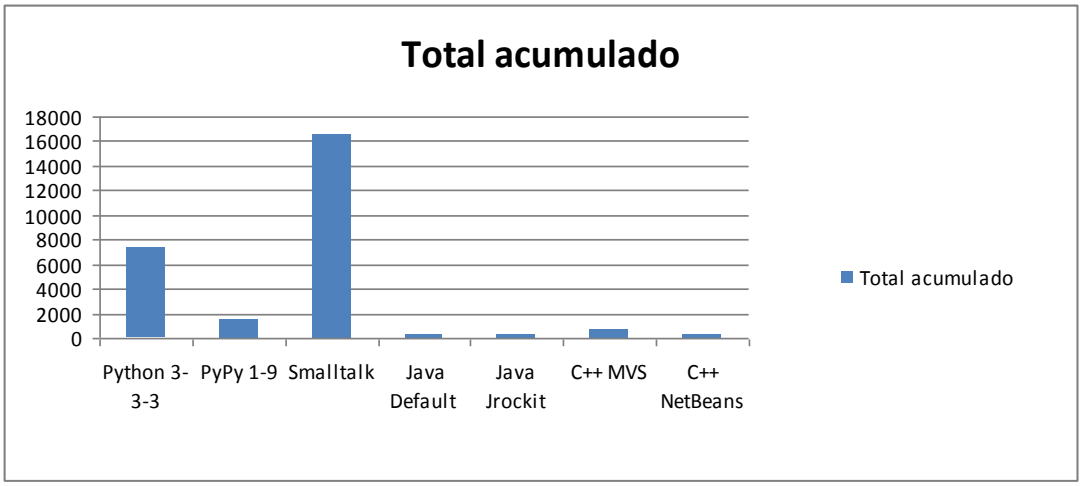


Tabla 3 – Tiempo de lectura/carga (milisegundos) en arreglo

Python 3-3-3	PyPy 1-9	Smalltalk	Java Default	Java Jrockit	C++ MVS	C++ NetBeans
162,46	281,95	2051	46	47	187	0

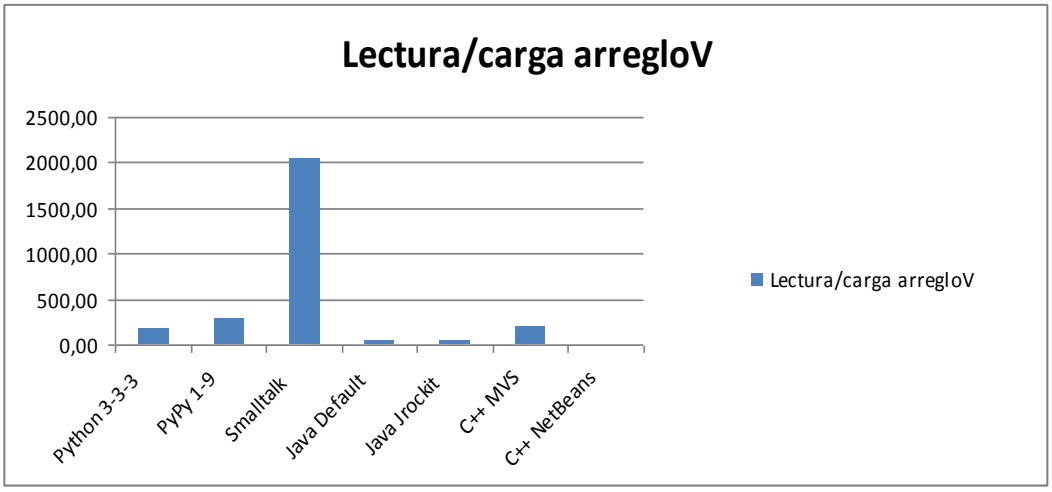


Tabla 4 -Tiempo de carga arreglo en tabla hash (milisegundos)

Python 3-3-3	PyPy 1-9	Smalltalk	Java Default	Java Jrockit	C++ MVS	C++ NetBeans
43,26	23,44	163	32	47	250	0

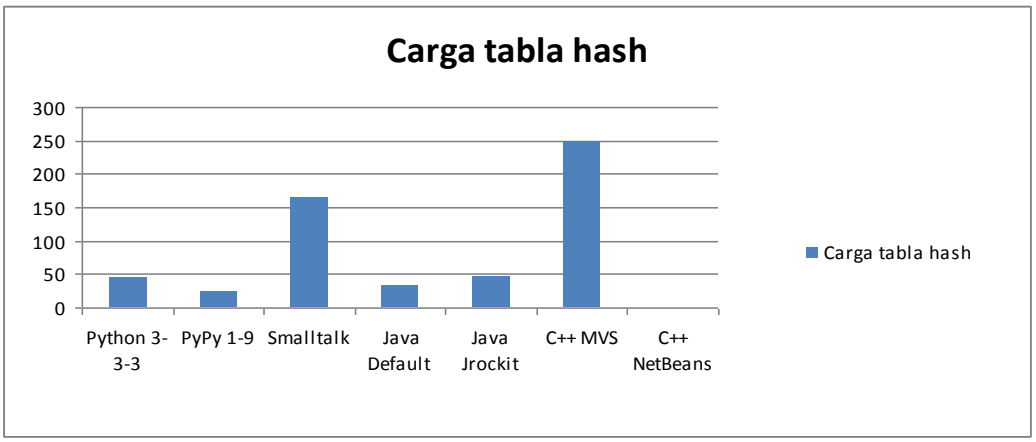


Tabla 5 – Tiempo de búsqueda (milisegundos) en tabla hash

Python 3-3-3	PyPy 1-9	Smalltalk	Java Default	Java Jrockit	C++ MVS	C++ NetBeans
163,76	41,02	2482	62	31	172	0

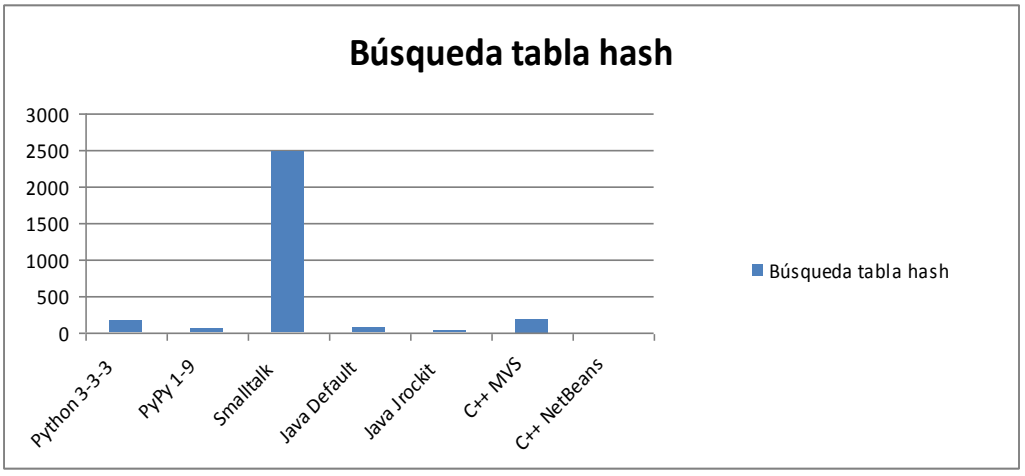


Tabla 6 – Tiempo de ordenamiento (milisegundos) usando QuickSort

Python 3-3-3	PyPy 1-9	Smalltalk	Java Default	Java Jrockit	C++ MVS	C++ NetBeans
961,44	869,55	264	16	16	15	0

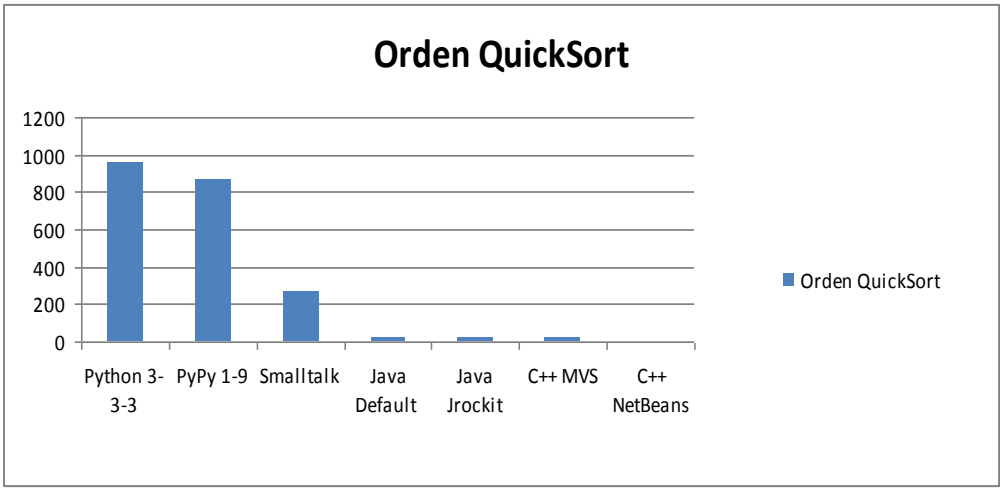


Tabla 7 – Tiempo de búsqueda binaria (milisegundos)

Python 3-3-3	PyPy 1-9	Smalltalk	Java Default	Java Jrockit	C++ MVS	C++ NetBeans
5952,23	281,46	11588	156	78	31	0