ALGORITHMICS - Bachelor of Software Engineering, 2024

LAB GUIDE. SESSION 0

GOALS:

- **Factors that influence execution time of programs**

1. Introduction

Naturally, when a problem is solved using a computer program, it is important to implement it in the best possible way.

The key parameters that are of interest to optimize are these two:

- **Execution time**: time (e.g. in milliseconds) it takes the computer to execute the program designed to solve the problem.
- **Occupied memory**: memory (e.g. bytes) occupied by the program in addition to that occupied by the different data structures used to model the problem.

Between the two previous factors (time and memory), it is usually more interesting to tend to minimize the execution time; at least as long as the memory does not overflow because it cannot accommodate the data structures used by the program being created.

Well, in this initial session we are going to work with different factors that influence the execution time and through a simple example problem we will analyze to what extent they affect the execution time.

Example problem we will work with: "*calculate all prime numbers between 2 and a given integer n*".

2. Factor 1: problem size

The larger the size of the problem, the longer the time to solve it for any algorithm that is designed. Unfortunately, most interesting problems in real life are of considerable size and it is not possible to reduce the magnitude of the problem without losing its essence.

For the example problem proposed above, if we go back to the Computer Fundamentals course of the first year, we would proceed to address its analysis and coding using the Python programming language.

Proceeding incrementally, the following Python modules are attached:

- **PythonA1v0.py**: the function **primoA1** is addressed, which calculates and returns in a certain way (algorithm **A1**) whether the integer passed to it, via parameter, is prime or not.
- **PythonA1v1.py**: another function **listadoPrimos** is addressed, which using the previous function ends up getting all the prime numbers until a given **n**.

- **PythonA1.py**: through this module we intend to validate how the execution time changes and increases as the problem to be solved increases in size (**n** grows).

YOU ARE REQUESTED TO:

Make a table reflecting the execution times of the **PythonA1.py** module for the exposed values of **n** (10000, 20000, 40000, 80000, 160000, 320000, 640000). If for any **n** it takes more than 60 seconds you can indicate OoT ("Out of Time"), both in this section and in subsequent sections.

3. Factor 2: computer performance

It is evident that the same program will take different time to run depending on the performance of the computer on which it is executed. The performance of a computer (work or instructions that it executes per unit of time) depends on its components, the most important being the CPU, as well as -to a lesser extent- the amount of memory, the disc, the power plan of the operating system, etc.

It is obvious that if we have access to several computers, to minimize time, we will be interested in running any program on the most powerful computer.

YOU ARE REQUESTED TO:

Make a table that reflects, at least for two computers to which you have access, the execution times of the **PythonA1.py** module for the exposed values of **n** (10000,20000, …, 640000). Clearly indicate which CPU and RAM memory you are using in each test.

4. Factor 3: implementation environment

We are now going to compare what was done before with the Python interpreter environment with another one. Specifically, the same problem will be solved in Java and thus we will be able to compare the times obtained for both cases / programming languages.

The trivial conclusion that we will reach in the end is: if, to solve a problem we have the possibility of developing it in several environments, we will be interested in selecting the most efficient one to minimize times.

Preliminarily, we will remember how to compile and execute programs in Java.

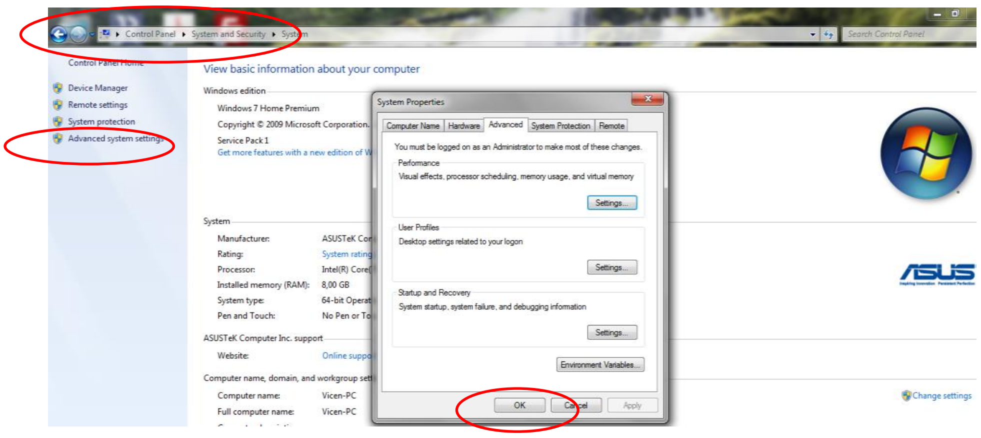
A. Java installation on the computer

As we are going to use the Java programming language for the purposes of this course, the most important thing we need is the JDK (Java Development Kit). It is quite likely that the JDK is already installed on the computer. For example, the JDK for Java 19 in a Windows system could be installed in the path: **C:\Program Files\Java\jdk-19** or similar. Just in case your computer does not have installed the JDK, you should search for it on the Web – The official website can be reached at https://www.oracle.com/es/java/technologies/downloads/.

After that, it is essential that you set the Java environment Path in your system. That way, you can invoke Java commands (e.g., for the compilation process) from different sources such as the command line console or the Eclipse IDE. The best way to know whether you already have it ready is to open a new command line terminal:

- Click **Start**
- Type `cmd` and press `enter`
- Type `javac` and press `enter`
- If you get an error, you need to configure the Java environment Path. If not, you can skip the next step.

Configuring the Java environment path:


Control Panel ▸ System and Security ▸ System
Search Control Panel

Control Panel Home
Device Manager
Remote settings
System protection
Advanced system settings

View basic information about your computer

Windows edition
Windows 7 Home Premium
Copyright © 2009 Microsoft Corporation.
Service Pack 1
Get more features with a new edition of W

System
Manager: ASUSTeK Cor
Rating: System rating
Processor: Intel(R) Core(
Installed memory (RAM): 8,00 GB
System type: 64-bit Operat
Pen and Touch: No Pen or To

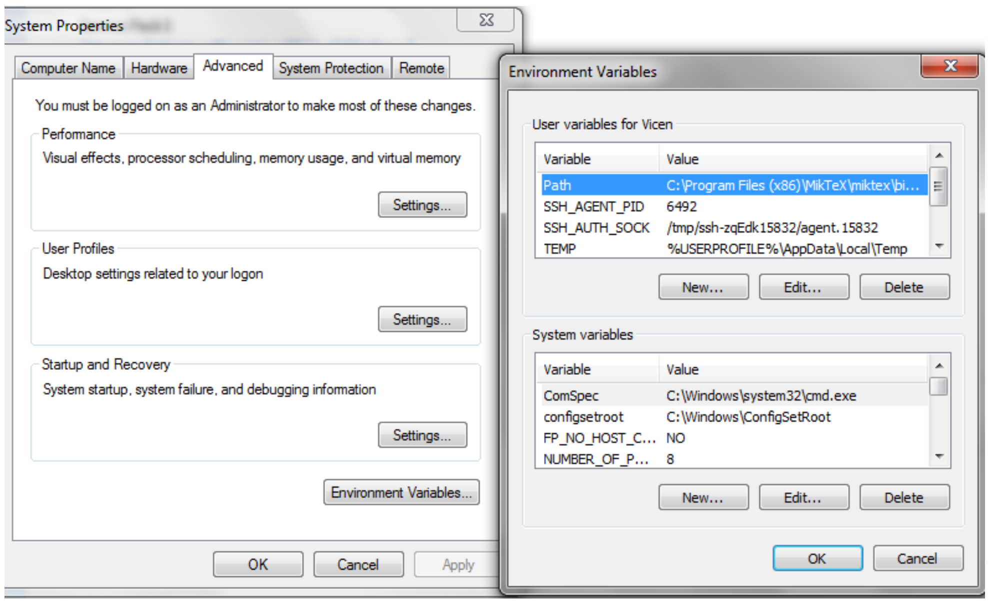
ASUSTeK Computer Inc. support
Website: Online suppo

Computer name, domain, and workgroup sett
Computer name: Vicen-PC
Full computer name: Vicen-PC

Change settings

System Properties
Computer Name | Hardware | Advanced | System Protection | Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your logon
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK Cancel Apply

Figure 1. System properties


System Properties
Computer Name | Hardware | Advanced | System Protection | Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your logon
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK Cancel Apply

Environment Variables

User variables for Vicen

Variable	Value
Path	C:\Program Files (x86)\MikTeX\miktex\bi...
SSH_AGENT_PID	6492
SSH_AUTH_SOCK	/tmp/ssh-zqEdk15832/agent.15832
TEMP	%USERPROFILE%\AppData\Local\Temp

New... Edit... Delete

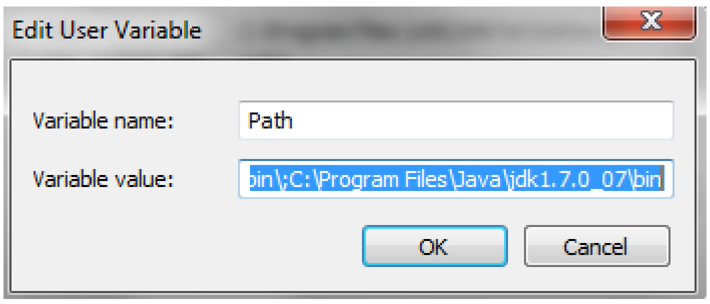
System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
configsetroot	C:\Windows\ConfigSetRoot
FP_NO_HOST_C...	NO
NUMBER_OF_P...	8

New... Edit... Delete
OK Cancel

Figure 2. Environment variables

Finally, add the path in which you have installed the JDK's bin directory:



Variable name: Path

Variable value: bin\;C:\Program Files\Java\jdk1.7.0_07\bin

Figure 3. Edit user variable

B. Integrated development environments (IDEs)

In addition to the basic way (directly with the JDK), you can work with a Java IDE. There are lots of them: Eclipse, NetBeans, BlueJ, DrJava, jGRASP, JBuilder, JCreator, Kawa, etc. I strongly recommend to use the Eclipse IDE as it is free and used in other courses during the Degree and Master programs in our school. In addition, it is broadly used in both academia and industry.

C. Programming with Java in the command line

The basic execution of programs in Java is done from the Windows command line. To use such a window, you should execute the executable from the start menu: `cmd`

When we want to change the hard disk drive, we just must type:

➢ `x: //it will open unit x:`

When we are in a folder and we want to place commands in a child folder we will do:

➢ `cd child //name of the folder we want to move to`

To go from a folder to the parent (to the previous one hierarchically speaking):

➢ `cd ..`

To go to the root folder:

➢ `cd /`

Instead of using the instructions from the previous "Configuring the Java environment path" section, another way to configure the environment is to create a batch file (**.bat**) with the **PATH** of the computer with an editor (e.g., Notepad). That batch file should be executed every time we open a new command line.

- The `SET PATH` command lets you know the currently active PATHs.

In that file, it is also possible to include information about the **CLASSPATH** (an option supported on the command line or using an environment variable that tells the Java Virtual Machine where to look for packages and user-defined classes when running programs).

- The `SET CLASSPATH` command lets you know the currently active CLASSPATH.

The problem with this is that, if the command line windows is closed, the established paths disappear, so when we open another new command line window there is a need to re-establish the paths.

When we create a Java class, it is usually part of a package (set of Java classes with a common purpose). The first valid statement of the class file specifies precisely in which package it is the class. In addition, the folder in which that class is stored must exactly match the package name.

To edit the files, any program editor is valid (even the Windows Notepad, in which case you must save the program with the option "All Files" and the **.java** extension, so that the **.txt** extension is not automatically added). In the labs of the school there are also installed other more powerful editors like Notepad++, Visual Studio Code and Sublime Text.

Java is a programming language appeared in 1996, but it has been adapting and improving over time, which makes it one of the most used and powerful programming languages. To compile programs, Java uses the javac compiler. This tool converts the program into machine code (also called bytecode). In addition, Java has an internal component that has been improved over time: the Java-In-Time Compiler (JIT). This component optimizes the execution of the programs, achieving very important time improvements. For example, if the JIT realizes that we have implemented a loop in our code, but that loop does not change any variables that will be needed later in the same code, it is likely that the JIT "eliminates" that loop at compilation time to optimize the execution of the program. It is very important to take this into account because when we measure times it may happen that measurements are much lower than expected. That would mean that the JIT has found "optimizable" code and changed it to run as fast as possible. Thus, it is highly recommended to use the JIT, but to avoid inconsistencies due to optimizations carried out by the JIT in code fragments that we have not written properly, we can deactivate it to verify that the written code always has the expected theoretical complexity.
To execute the code without using the JIT: java -Djava.compiler=NONE [FILE_TO_EXECUTE]

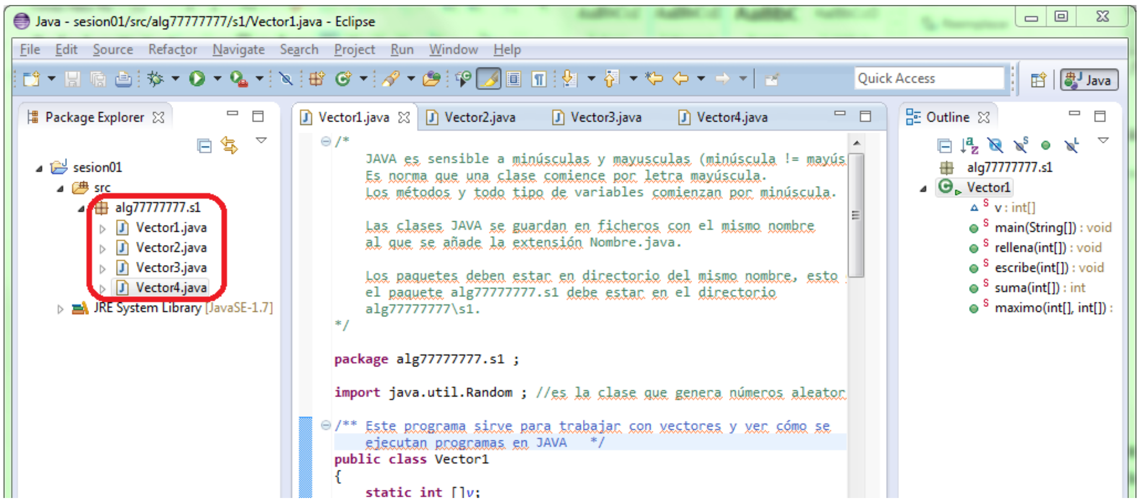
From now on, we will call "WITHOUT OPTIMIZATION" those times obtained without using the JIT and "WITH OPTIMIZATION" those times obtained using the JIT.

D. Programming with Java in the Eclipse environment

One of the most used environments in the Java developer community is Eclipse. This integrated development environment allows you to perform all development-related tasks from a common interface: coding, code analysis, execution and debugging.

In this environment, the minimum development unit is the **Project**, so we must create a new Java Project to start working with the code. Subsequently, you just need to copy & paste the folder corresponding to the package or packages that includes the source code of the Java classes to the `src` folder within the project.

One of the great advantages of the Eclipse environment is that it performs an automatic compilation when changing and saving the code and informs the user of possible errors using marks, both by highlighting the code involved and by establishing an icon in the margin, which provide extra information about the error or warning if we place ourselves on it with the cursor.

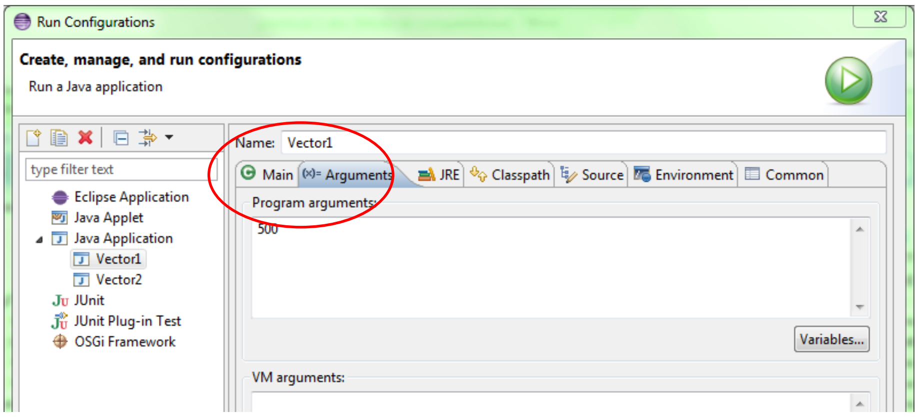


```
/*
    JAVA es sensible a minúsculas y mayusculas (minúscula != mayús
    Es norma que una clase comience por letra mayúscula.
    Los métodos y todo tipo de variables comienzan por minúscula.

    Las clases JAVA se guardan en ficheros con el mismo nombre
    al que se añade la extensión Nombre.java.

    Los paquetes deben estar en directorio del mismo nombre, esto
    el paquete alg77777777.s1 debe estar en el directorio
    alg77777777\s1.
*/

package alg77777777.s1 ;

import java.util.Random ; //es la clase que genera números aleator

/** Este programa sirve para trabajar con vectores y ver cómo se
    ejecutan programas en JAVA   */
public class Vector1
{
    static int []v;
```

Figure 4. Eclipse workspace

To execute and debug the code, we must use the corresponding icons in the toolbar, or the context menu related to the project and use the option "**Run as ...**". To perform this operation, all the changes made to the project files must be saved and the environment will be responsible for performing the necessary operations to execute it. There are several execution modes; the one that interests us is "**Java Application**" that executes Java classes without any container or server.

In addition, Eclipse allows establishing the arguments that accompany the call to the main class to execute it. This can be done through the option "**Run configurations ...**" which can be accessed either from the toolbar execution button or from the context menu (**Right mouse button → Run As → Run Configuration → Arguments → Program Arguments**). We should remember to select in the menu on the left the related class and type the values on the tab on the right called "**Arguments**". Here we can enter the values for the arguments in the "**Program arguments**" field separated by spaces.



Run Configurations

Create, manage, and run configurations

Run a Java application

Name: Vector1

Program arguments:

500

VM arguments:

Figure 5. Execution of a program with arguments

<u>**YOU ARE REQUESTED TO**</u>:

Program a class named **JavaA1.java**, which uses the same **A1** algorithm to find out if a number is a primer number, as in **PythonA1.py**.

Then, a table must be made reflecting the execution times of **JavaA1.java** for the same values of **n** (10000, 20000, ..., 640000).

Finally, compare these times with those obtained in Python (in a previous section) for that same algorithm A1.

5. Factor 4: algorithm that is used

The execution time will critically depend on the algorithm and data structures used to solve the problem. This factor is extremely decisive and is the basic objective of this Algorithmics course.

Frequently, this factor has a greater impact on the search for shorter execution times than the other factors previously seen, and we will see paradigmatic examples of this in this course (for example, in topic 2 dedicated to sorting).

To illustrate the above, we are going to solve our problem in two other ways, which are provided in the following Python modules:

- **PythonA2.py**: the function **primoA2** is introduced, which calculates and returns in another way (algorithm **A2**) if the integer passed to it, via parameter, is prime or not.
- **PythonA3.py**: the function **primoA3** is introduced, which calculates and returns in a third way (algorithm **A3**) whether the integer passed to it, via parameter, is prime or not.

YOU ARE REQUESTED TO:

Make a table reflecting the execution times of the modules **PythonA1.py**, **PythonA2.py** and **PythonA3.py**, for the same values of **n** (10000, 20000, ..., 640000).

Codify those same algorithms **A2** and **A3** in Java, in two classes named respectively **JavaA2.java** and **JavaA3.java**.

Make a table that reflects the execution times of the classes **JavaA1.java**, **JavaA2.java** and **JavaA3.java**, for the values of **n** (10000, 20000, ..., 640000) **WITHOUT OPTIMIZATION** of the Java program.

Make a table that reflects the execution times of the classes **JavaA1.java**, **JavaA2.java** and **JavaA3.java**, for the values of **n** (10000, 20000, ..., 640000) **WITH OPTIMIZATION** of the Java program.

Finally, draw final conclusions by comparing the times previously obtained: with Python, with Java WITHOUT OPTIMIZATION and with Java WITH OPTIMIZATION.

Optionally, an **A4** algorithm can be implemented in Java, which should improve the previous algorithms.

6. Work to be done

- An `algstudent.s0` **package** in your course project. The content of the package should be the Java files created during this session. You can manage the Python files as you wish.

- A `session0.pdf` **document** using the course template (the document should be included in the same package as the code files). The activities of the document should be the following:
    - **Activity 1. Factor 1 (problem size)**
    - **Activity 2. Factor 2 (computer power)**
    - **Activity 3. Factor 3 (implementation environment)**
    - **Activity 4. Factor 4 (algorithm that is used)**

Deadline: The delivery of this lab will be made on the same date as the two following ones. More instructions will be given in coming weeks.