KTH Computer Science
and Communication

Automatic Grading System in Microsoft .NET Framework

Evaluating the performance of different languages on the Microsoft .NET platform

WILLIAM LUNDIN FORSSÉN

Master's Thesis at NADA
Supervisor: Alexander Baltatzis
Examiner: Olle Bälter

TRITA xxx yyyy-nn

Abstract

A common challenge for software consulting companies is recruiting the right people. In the software industry, the recruitment process usually involves several steps before a contract is signed; a single job interview is rarely enough. Thus, the interview tends to involve some test to make sure that the person seeking the position is qualified. The testing procedure is usually part of the interview or conduct at the same occasion. Interviewing applicants who are not qualified for the position might be a waste of time.

This waste can be minimized by only interviewing qualified applicants. In the software industry, qualifications are commonly asserted by letting an applicant solve programming problems. This process can be automated using an Automatic Grader. Such systems already exist on some universities today and are used extensively in various programming courses and also in programming contests.

This thesis evaluates such a system with regard to execution speed and memory consumption. It also explains how such a system can be built and attempts to demonstrate the performance differences between the supported programming languages. The system is unique in the aspect that it is built using only Microsoft .NET Framework while still supporting multiple programming languages. The supported languages are C#, Java and Python. This support is enabled through the use of a Java byte code to CIL compiler called IKVM. Python is supported through the use of IronPython.

The results showed that C# and Java performed almost equally in terms of execution speed and memory usage, with Java being slightly behind. Python had greater performance and memory issues than the other two.

Future work involves implementing additional language support and improving the system with usability in mind.

Referat

Automaträttning i Microsoft .NET Framework

En gemensam utmaning för konsultföretag inom mjukvaruutveckling är att rekrytera rätt personer. Rekryteringsprocessen inom mjukvaruindustrin innefattar vanligtvis flera steg innan ett anställningsavtal undertecknas. Enbart en anställningsintervju räcker sällan för att avgöra en sökandes lämplighet. Således tenderar intervjun att involvera någon form av test för att se till att sökanden är kvalificerad. Testproceduren är oftast en del av intervjun eller genomförs vid samma tillfälle. Att intervjua sökande som inte är kvalificerade för positionen i fråga kan vara ett slöseri med tid.

Detta slöseri kan minimeras genom att endast intervjua kvalificerade sökanden. Inom mjukvaruindustrin säkerställs kvalifikationer vanligtvis genom att låta sökanden lösa programmeringsproblem. Denna process kan automatiseras med hjälp av ett s.k.Automaträttningssystem. Sådana system finns redan på flera universitet i dag och används i stor utsträckning i flera programmeringskurser och förekommer även i samband med programmeringstävlingar.

Denna uppsats utvärderar ett sådant system med hänsyn till exekveringshastighet och minneskonsumtion. Den förklarar också hur ett sådant system kan byggas och hur de olika språken presterade. Systemet är unikt i och med att det är byggdes med enbart Microsoft .NET Framework och samtidigt klarar av att stödja flera olika programmeringsspråk. De språk som stöds är C#, Java och Python. Detta stöd möjliggjordes genom användning av en Java byte-kod till CIL kompilator som kallas IKVM och Python stöds genom användning av IronPython.

Resultaten visar att C# och Java presterar nästan lika med hänsyn till exekveringshastighet, med Java på andraplats. Python verkade ha större prestandaproblem än de andra två. Java konsumerade mest minne av de tre, med Python som tvåa.

Framtida arbete omfattar stöd för fler programmeringsspråk och förbättra systemet med användbarhet i åtanke.

Foreword

This thesis is my Master Thesis project which was carried out at Valtech in Stockholm, Sweden. It is part of my last course in the Masters Programme in Computer Science at The Royal Institute of Technology (KTH). The most difficult part of this thesis was finding out what subject to research. Then one day some words from my supervisor Alexander Baltatzis echoed in my head from one of his lectures: "Write code in any language and run it in .NET". That sentence sparked my interest in finding out whether it was possible or not since most people who write programs on the .NET platform do so using C# or sometimes Visual Basic; one rarely hear about someone using any other back-end language (even though many others are supported by the .NET platform).

Glossary

Term	Description
AGS	Automatic Grading System, a system for evaluating or grading code.
CIL	Common Intermediate Language, the lowest level human-readable programming language in .NET Framework.
CLI	Common Language Infrastructure, a specification that describes the runtime environment of Microsoft .NET Framework.
CLR	Common Language Runtime, is the virtual machine of Microsoft .NET Framework.
CLS	Common Language Specification, a set of rules and features that a .NET language must implement and understand.
GC	Garbage Collection, reclaim memory from objects that are not in use.
GUI	Graphical User Interface, an interface that is image oriented rather than text oriented.
IKVM	An implementation of the Java for Mono and Microsoft .NET Framework.
IL	Intermediate Language, a language of an abstract machine.
IronPython	An implementation of the Python programming language targeting the Microsoft .NET Framework and Mono.
JVM	Java Virtual Machine, a program that executes byte-code.
Mono	An open source project created to enable .NET Framework cross platform ccompatibility.
MSIL	Microsoft Intermediate Language, a synonym for CIL.

Contents

Chapter 1

Introduction

This chapter introduces the problem at hand, the goal with this thesis and some brief information about the company where all this work took place.

1.1 Problem Statement

A common challenge for Valtech and other consulting companies is to recruit the right people. The recruitment process in the IT business often involves several steps and can be time consuming. This is because knowledge and skill can differ significantly from programmer to programmer and a need to interview many people arises (just to find one that is deemed suitable for the job).

The recruitment process usually involves some test to assert the skill of the programmer. A test which usually contains one or several programming problems. The test is sometimes performed on paper (questionnaire) or a white board. Both of these environments are not suitable for programming, hence most programmers tend to do their work on computers with the assistance of a web browser and a suitable IDE.

These two issues present a challenge. The optimal solution would result in less time spent recruiting and at the same time asserting the skills of the programmers before an interview is considered.

1.2 Goal

The main goal was to evaluate a system which could satisfy the demands stated above. Thus, since no such system was present at Valtech, there was a need to build the system and then evaluate it through test cases with performance and memory consumption in mind.

The system would allow users to submit solutions to common programming problems and then have their code evaluated. Their code would have to have a limit on execution time since a "good" programmer should be able to solve problems efficiently. The results of the evaluation would be used to estimate how much time

the different programming languages would need in order to solve problems as all of these languages would be run in the Microsoft .NET Platform and some are expected to perform better than others.

1.3 About Valtech

Valtech is an independent, global IT-consulting company with offices in Europe, United States and Asia. The company was founded 1993 in France and has over 1600 employees worldwide. The Swedish division is primarily oriented towards web solutions. Some of Valtech's most well-known award winning projects includes the websites 1177.se, Antagning.se and Riksbank.se.

All work concerning this thesis was carried out at Valtech in Stockholm, Sweden.

Chapter 2

Background

This chapter gives a short presentation of what automatic grading is, some history and how the systems that are in use today are built.

2.1 What is Automatic Grading?

An Automatic Grading System is a computer system that can judge code. The process starts with the user being given a programming task or problem to solve. The user then attempts to solve this problem by writing some code which he or she then sends to the automatic grading system. The system compiles the code (if needed) and then runs it. The output generated is then compared with the correct output for that problem. If the output matches, the system returns a status message indicating a successful submission (e.g. "Accepted") or if the output does not match a different message is returned, indicating that something went wrong (e.g. "Wrong answer").

There are some variations to the process above, sometimes more verbose feedback than "Wrong answer" is used, often indicating exactly which test went wrong and why [1]. This feedback helps college students to debug their code and by doing so enhances their learning experience.

Having an automatic grading system results in several benefits [2]. There is no longer a need for humans correcting code in detail (a time consuming process) since the system acts as a "judge". Using this judge also gives greater consistency while evaluating code since it is impartial and all submissions are judged equally. The system can provide instant feedback. This is useful for universities where students no longer have to wait for a teacher or an assistant to grade their homework. Submissions to this system are saved using a database, providing whoever administrates the system a way to trace all interactions with it.

2.2 History

The concept of automatic grading is not new. The earliest known system was built in 1960 by Hollingsworth [3]. This system used punch cards to write programs. Using this student-system approach rather than the traditional student-teacher resulted in several benefits. It cut costs considerably for the staff since the time they needed to grade the students work was severely reduced. The students themselves also spent less time on each task, since they were able to have their work graded immediately instead of waiting for a teacher to do it. This system also made it possible to increase the number of students taking the course. It did, however, have some shortcomings. For instance, a student's program could modify the grader program, making cheating possible.

An article from 2005 [4] describes three generations of automatic graders. The first generation systems were those regarded as being built and/or used in the 1960's and 1970's. Unsurprisingly, they used code that were close to pure machine code. In order to make them work, it was sometimes necessary to modify both the compiler and the operating system.

The second generation systems (1980-2000) introduced script-based tools. These involved various verification schemes and also asserted that the code was written in a certain way/style (decided by the teacher). Typically these graders involved command-line GUIs. Languages like C and Java were used extensively.

The third generation (2000-) differs from the second generation systems primarily in two ways. They mostly use web based GUI:s and they often include a plagiarism detection system since students sometimes shared code amongst each other. There were some minor issues among these detection systems [1] [4]. If the programming tasks were too easy or if a lecturer had been excessively thorough when describing the homework, the submissions tended to be similar and thus picked up by the plagiarism detection system. Sometimes this made it difficult to distinguish between real plagiarism and the false positives.

2.3 Today's Systems

Today's modern systems, such as those in [1] [2] [4] [5] [6] (considered to be third generation systems) commonly contain a web based front-end together with a general back-end. The back-end's main purpose is to save submissions to the database and to send the submitted code to the other back-ends (one for each supported language) while preserving system security through sandboxing. The database can be used to inspect submissions. The modern systems mainly differ in their support of different programming languages, and not every one of them contain a plagiarism detection system.

The modern systems also differ from the first generation in that most have adopted a test-driven education paradigm. Students are no longer penalized for attempting to submit more than one solution, in fact, it is encouraged. An article

from 1969 [7] describes how student were discouraged from ever making a mistake by giving them a lower score for each subsequent submission on the same problem. Students also received an even lower score if the program had not run to completion or if some results were incorrect. The system described (and built) in this thesis adapts the modern test-driven paradigm in the sense that it does not penalize users for multiple submissions or any submission errors.

Chapter 3

Method

This chapter explains how the system was implemented. It contains details about the work-flow, the GUI, how multiple languages are supported, how the security works and some of the difficulties and limitations encountered.

3.1 System Description

The system was named CELINE, an abbreviation that comes from the descriptive phrase "**C**ode **E**va**L**uation **I**n .**NE**t" (the uppercase bold letters forming the abbreviation).

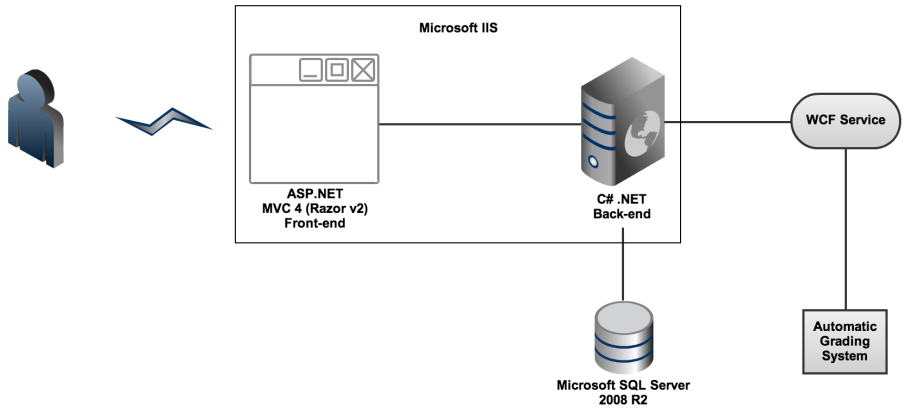

Figure 3.1. Overview of CELINE.

Figure 3.1 describes an overview of the system. The system uses a web based GUI for listing problems and submitting solutions to them. Going through this figure from left to right we can see that a user connects to the website through a web based GUI, which is built using a combination of ASP.NET MVC4 (Razor v2) technology and JavaScript. The system uses Microsoft IIS to enable this web

support, which the standard web server software used in .NET. When the user submits code to solve a problem, that submission is saved to a database which is a Microsoft SQL Server 2008 R2 database. The submission is then forwarded to a WCF Service which in turn creates a new instance of the automatic grading system (built using C#) using the submission as input. The automatic grader compiles the source if needed and runs the code. It then returns a status code generated from the submission, this is described in section 3.1.3.

3.1.1 The Web GUI

The GUI is built using the ASP.NET MVC4 template. This saved both time and effort while still giving the website an easy to understand simplistic style.

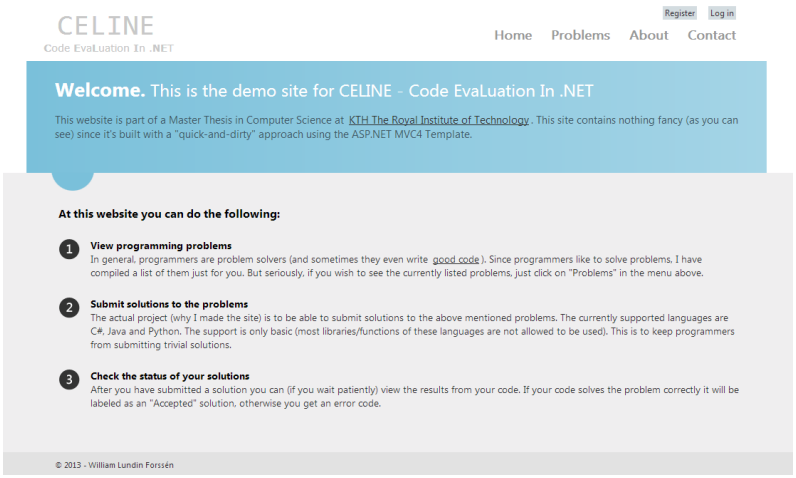

Figure 3.2. The start page of CELINE.

In Figure 3.2, we see the start page. The middle area contains some informative text; the top right contains the site menu, the register and login buttons.

In Figure 3.3, we see the submission form on the left side, this page contains some useful information on the right side concerning language-specific signatures. These signatures specify the point of entry for the application (where the execution of the code starts). Users are free to name this method to anything they see fit, the only limitation is that it takes a string as input and returns a string as output. This is discussed in further detail in section 3.1.3.

3.1.2 The Automatic Grading System

When a user has submitted his or her code, it eventually reaches the AGS. Figure 3.4 describes the flow of the most common paths for a submission through the system.

Depending on the programming language, a compiler is chosen. The code is then compiled into an assembly file and loaded into a separate and secure Application

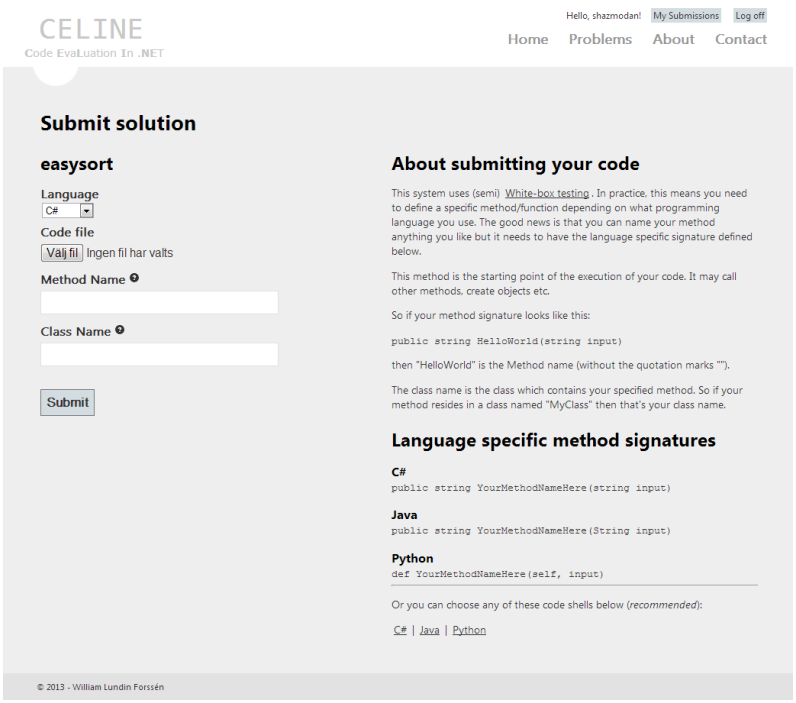

Figure 3.3. Code submission page.

Domain (AppDomain, see section 3.3.1). The AGS then invokes the user specified method with a string containing the test data. The AGS waits for the code to complete or for the problem to timeout. It then compares the string returned by this method with another string containing what should be the correct output. If the strings match, the submission is regarded as being a success; otherwise it is regarded as a failure. The AGS then returns the appropriate status code.

3.1.3 System-User Feedback

The system gives feedback to the user in the form of a status code, one for each submission. The status code is a simplistic message that indicates whether the submission was successful or not. This system attempts to be more verbose on errors than other modern systems (described in section 2.3). It uses grey-box testing, which is more prone to submission related errors, than black-box testing (commonly used in today's AGS:s). The concepts of white- and black-box testing are described more thoroughly in section 3.3.3. The following list contains the possible status codes:

- Accepted - The code has compiled, run and gave the correct answer.
- Wrong Answer - The code has compiled and run but gave the wrong answer.

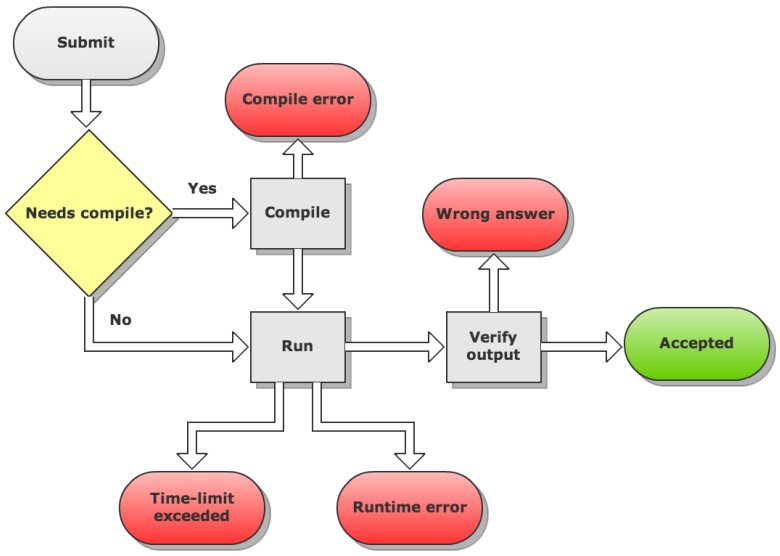

Figure 3.4. Common paths of problem flow in the AGS.

- Server Assembly Error - The AGS failed to build an assembly from the code file, thus making the code unable to run.
- Submission Error - The code tries to reference a code library that is not available/allowed.
- Illegal Operation - Occurs if the AGS detects a forbidden system call (e.g. accessing files, using the network etc...).
- Class or Function Error - Occurs if the class or method name does not correspond to the name provided by the user, thus resulting in the AGS being unable to invoke it.
- Time Limit Exceeded - The code ran longer than the time-limit for this problem allowed. This could be an indication that the code needs to be improved (performance wise).
- Rejected - This is a general error which indicates that the AGS has been unable to determine why the submission failed.
- Server Error - This indicates that the AGS instance has crashed.

The status messages “Server Assembly Error”, “Submission Error”, “Illegal operation” and “Class or Function Error” are all related to incorrect formatting of the code by the user. These might be less verbose than one would like (e.g. each

submission could have returned exactly where in the code the error occurred). Any more verbose feedback would have required an expansion of the scope of this thesis.

3.2 Supported Languages

3.2.1 C#

The support for C# was implemented with the use of the Microsoft CSharp library [8] and the System CodeDom Compiler library [9]. Since C# can be considered a native .NET language, no 3rd party libraries were required to achieve this support.

3.2.2 Java

The Java support comes from using javac [10], the standard Java compiler that comes with the Java Development Kit (JDK) [11]. Javac compiles Java code to bytecode. This bytecode is then compiled to CIL using ikvmc [12], a tool that is part of IKVM.NET (an implementation of Java for Mono and the Microsoft .NET Framework) [13].

3.2.3 Python

Python is supported through the use of IronPython [14]. IronPython provides a library (a dll) which contains a ScriptEngine from which python code can run without the need for compiling. Some issues encountered with the IronPython implementation are described in section 3.3.2.

3.3 Difficulties and Limitations

3.3.1 Security

Executing unknown code can be dangerous for several reasons. The code might have unforeseen side effects or even attempt to gain root access (compromise of the entire system). Thus, it was necessary to avert this danger by running the code in a safe and secure sandboxed environment.

In .NET this can be handled through the use of AppDomains which act as a layer of isolation between applications [15]. AppDomains make it possible to protect the core application from malicious code and exceptions that might occur during the execution of unknown code. This protection comes at the cost of performance since AppDomains are able to load assemblies but cannot unload them. To get rid of a loaded assembly, the entire AppDomain must be unloaded, which is the case with each new submission. CELINE creates a new AppDomain for each submission and then uses proxy objects (remotable objects) called “Marshals” [16] to enable communication between the core-AppDomain and the untrusted-AppDomain, in order to get the output that the unknown code produced.

When a new submission enters the AGS, the code is copied to a temporary location on disk and then gets executed with minimal privileges using this AppDomain. An alternate way to handle the security is mentioned in [2], where user impersonation is used by the core application, essentially letting the operating system handle the security through the use of a minimal privilege user.

There is an exception to the rule about AppDomains being safe from each other. That is when the code that is executing in the untrusted-AppDomain generates a stack overflow exception. This exception will travel back into its parents AppDomain (in this case the core-AppDomain) since it is no longer possible to catch this exception as of .NET version 2.0 and later [17]. In order to avoid crashes caused by stack overflow exceptions, the AGS is instantiated with each new submission by the WCF Service. Thus, the website never experiences any problems, only a timeout of the WCF Service.

3.3.2 IronPython Limitations

IronPython does not have the same performance as the other languages for a number of reasons. The first reason is that it is 61.6% slower on average than CPython (version 2.7) [18]. The startup time is also significantly slower than the other languages since this language is scripted using a ScriptEngine. This adds another layer of interpretation (see Chapter 5 for performance results). However it is necessary to mention that IronPython does have the capability to compile the code into an assembly, but with a serious drawback, the assembly's methods and main entry point cannot be invoked like other assemblies since the MSIL is not CLS-compliant [19] and, therefore, cannot be directly accessed from other .NET languages [20].

3.3.3 White- and Black-box Testing

White-box testing is a method of testing a software's internal structures as opposed to black-box testing, which tests application functionality/behavior on a higher level (usually done with users in mind). These concepts are usually discussed in relation to Test-Driven Development (TDD) but are also relevant in this context since the code is being tested and there are different ways of doing it.

As mentioned in section 3.1.3, CELINE is a mixture of both, a grey-box system since it requires the user to define a method that has a pre-defined signature (used as the invocation point), but it does not care about any other implementation details. The choice for this implementation was made because it saved time compared with the approach of using standard system in and system out (black-box testing) to provide input and output channels.

Chapter 4

Testing Methodology

This chapter discusses which aspects of the code that were tested and explain which algorithms were used.

4.1 Aspects to Test

There are several ways to evaluate code. Code can be evaluated on its structure, understandability, execution speed, memory usage and others. Some programming exercises are of the nature that one would be interested in looking at execution speed and memory usage. Such is the case with many programming exercises used in KTH Kattis [5] and since CELINE is primarily used for securing programming knowledge of job applicants, one can expect this system to be used primarily for the recruitment of college graduates (those who have worked in the industry for a while tend to know how to program). Execution speed and memory usage are also of particular interest since they are free of any human opinion.

4.1.1 Execution Speed

The primary objective of the code should be to execute quickly. In general, the faster any code executes the more efficient the implementation is. Apart from saving time this priority enables users to compete amongst each other (using leaderboards on the website), for the shortest execution time by writing the most efficient code. The time is measured using the .NET Stopwatch class [21] which starts from the point of executing the thread which invokes the submissions entry method and ends when the thread ends, either by returning an answer or timing out.

4.1.2 Memory Consumption

The secondary objective is memory consumption. Every problem has the option to restrict memory consumption. This will allow administrators of the system to create problems in which the goal could be to force users to make a trade-off between speed and memory, illustrating real life situations such as optimizing code in order

to avoid the unnecessary cost of purchasing more memory. The tool used to measure the memory consumption (and more) is the Windows Performance Monitor. This tool allows for tracking of Windows processes, and more specifically, tracking of individual resources contained in the .NET CLR.

4.2 The Chosen Tests

The tests are a mixture of general functionality and two algorithms. This section assumes that the reader is familiar with the Big O notation [22].

The goal of these tests is to obtain a scaling factor between the different programming languages. This is needed in order to keep the judgement of user solutions fair. For example, it is very likely that C# will execute faster than Java or Python in the context of the .NET Framework, therefore one needs to offset this difference if users are to be able to compete for optimal solutions regardless of the programming language they use.

4.2.1 Language Overhead

Each programming language has an overhead startup cost. This is measured by submitting code which does the minimum given this context. In Figure 4.1, a minimum code block for the C# programming language is demonstrated.

```
public class OverheadTest{
        public string SimpleReturn(string input){
                return "";
        }
        public static void Main(){}
}
```

Figure 4.1. C# code for language overhead testing.

This code will receive an empty string as input, and return an empty string. Measuring the overhead makes it possible to compensate for a language which has a slower start up time. For example, it would be unfair to restrict Python code to the same time-limit as C# code since Python has a considerably longer startup time than the native .NET language C# (see Chapter 5).

4.2.2 Common Operators

The common operators are addition (+), subtraction (-), multiplication (*), division (/) and modulo (%). This test will measure how the language performs doing the simplest operations, giving an indication of its general performance. Figure 4.2 shows the code used for the addition test in Python.

```
class AdditionTest:
    def AddNumbers(self, input):
        input = input.split(" ")
        array = list(map(int, input))
        total = sum(array)
        return str(total)
```

Figure 4.2. Python code for testing the addition operation.

This code takes a string containing numbers as input, divides it into an array of strings, converts the strings to integers, sums these integers and returns this summation.

4.2.3 Insertion Sort

Insertion Sort is a comparison based sorting algorithm useful for sorting small inputs. The algorithm can be expressed using code, but this does not illustrate it well enough for people not already familiar with it. So instead of code, playing cards (using a common 52 card deck) will be used to explain it.

- Start with an empty left hand and imagine seeing a bunch of cards on a table.
- Pick a card from the table one at a time from left to right.
- Insert this card into the correct position in your left hand.
- The correct position is determined by comparing this card to each other card in your hand from right to left.
- The cards in your left hand are always sorted before each new card is picked.

The algorithm has a worst-case running time of $O(n^2)$ but can still outperform more advanced algorithms such as Merge Sort (section 4.2.4). However, this is only true when the input is small (the definition of "small" varies from system to system) [23] because Merge Sort has an extra overhead from its recursive function calls and also uses more memory.

Insertion Sort is a suitable algorithm for testing performance between different languages since its implementation is easy, straightforward and similar in these languages.

4.2.4 Merge Sort

Merge Sort is a divide and conquer, comparison based sorting algorithm. It is stable in the sense that it preserves the input order of equal elements in the output. The algorithm goes as follows (once again using the common 52 card deck analogy):

- Imagine yourself seeing a full deck of cards spread out in a line on a table.
- Divide all the cards into two equally sized piles. Now there should be 26 cards on your left side and 26 cards on your right side.
- Keep dividing these piles into halves until there are only piles with one card in them.
- Now merge these one card piles together so that one ends up with a pile that contains 2 cards that are sorted from left to right (in ascending order).
- Keep merging all piles together until one ends up with one pile containing all of the cards (52) in sorted order.

Figure 4.3 illustrates this process using eight cards.

There are several sorting algorithms that have a worst-case time complexity of $O(n * log(n))$, Merge Sort was chosen because it has an auxiliary worst space complexity of $O(n)$, thus using more memory than other similar algorithms (e.g. Heap Sort) [24]. This helps in illustrating how different programming languages differ in memory consumption.

4.3 About Just-in-time compilation

All languages used in this thesis use Virtual Machines (VM:s) (and/or interpreters) to execute their code (though Microsoft prefers to call the CLR an *Execution Engine* [25]). This means that they can benefit from Just-in-time compilation (JIT). JIT is a method to improve runtime performance. It works by using *heuristics* in order to determine whether or not to compile a method from byte-code into machine-code or to simply execute the code through an interpreter. JIT also uses runtime statistics to aid in these decisions [26].

This means that some programs might experience slow start-up times because the VM has yet to determine the best compiling action for all the methods the program uses since it has not yet been able to collect any runtime statistics. In order to make sure programs run in their optimized state, a warm-up time is commonly used.

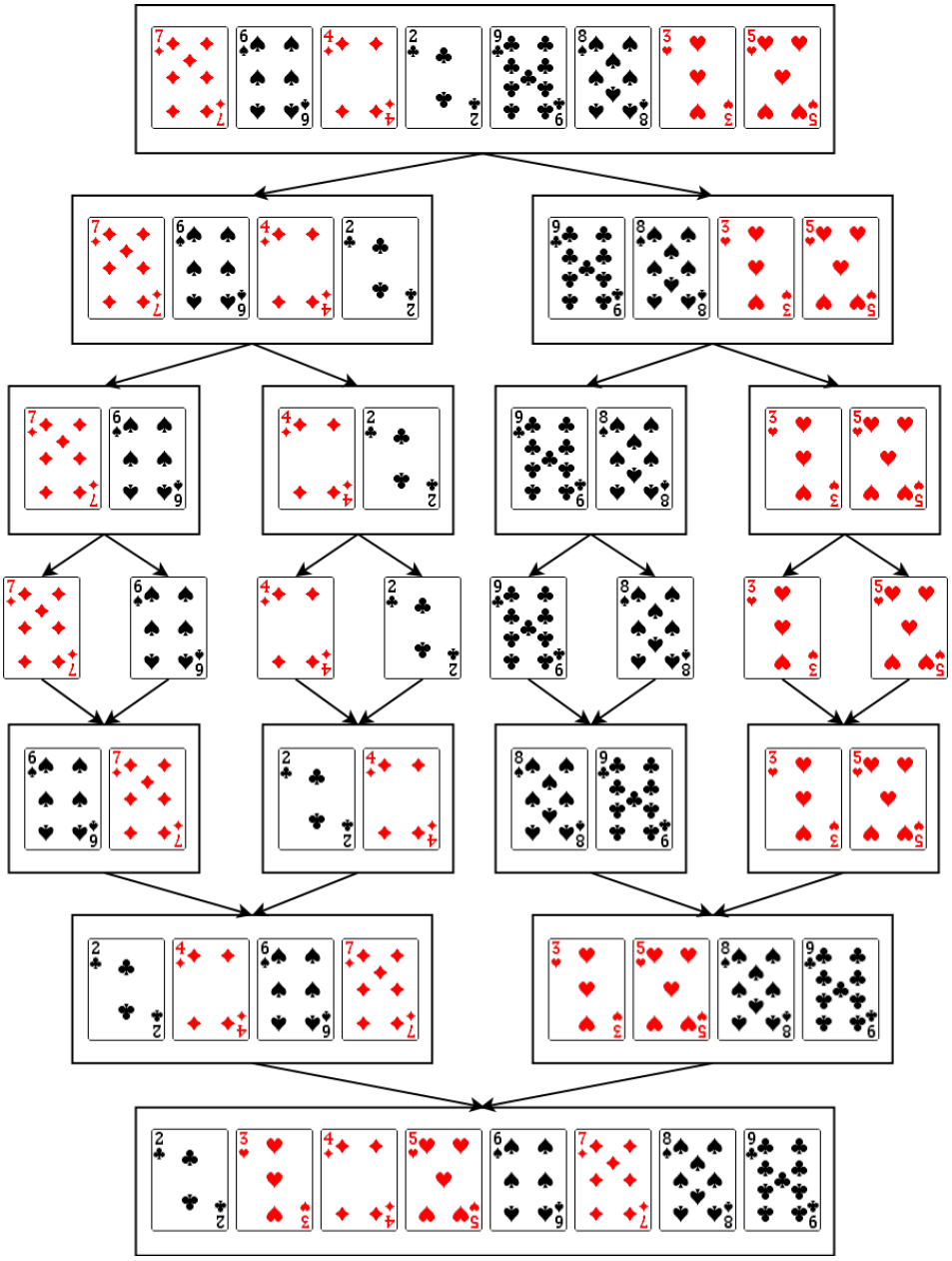

Figure 4.3. Illustration of the Merge Sort algorithm using eight cards. The cards are subsequently divided up and then merged back together in sorted order.

Chapter 5

Testing Results

This chapter describes the system specifications of the two computers that were used during testing; how to tests were carried out and present the results of the tests.

5.1 Testing Systems Specifications

The tests were conducted on two computers. The system specifications can be seen in Table 5.1. The MacBook Pro runs Windows 7 with the help of Parallels Desktop 8. Additional tests were also made using Bootcamp with Windows 7 to exclude any severe overhead penalties added by Parallels.

	MacBook Pro	**Windows 8 Desktop**
Operating System	OSX 10.8.2	Windows 8 Pro 64-bit v6.2
Processor	2.6 GHz Intel Core i7	3.4GHz Intel Core i7
Memory	16 GB 1600 MHz	16GB 1333 MHz

Table 5.1. System specifications for the computers used.

5.2 Testing Procedure

All tests presented in this chapter were run on both computers. The ratios on the Windows 8 computer were close to the OSX computer running Parallels and Bootcamp. The small differences in the results between the two machines are likely due to pure variance. The test results should be viewed in relative terms and not in absolute terms.

While the best time is the only time presented in this chapter, the mean-time was also recorded to make sure that the best time was not just a one-time fluke (which could have depended on unknown factors).

Every test was run 100 times on each computer in order to reduce variance. All results were adjusted according to the overhead for each language (see section

5.3.3). The input for each test varies from 1000 elements up to 10 million elements. The elements are randomly generated positive integers.

All tests in the Java and .NET environments were run with disabled debug flags to optimize performance.

5.3 Results

In this section, the results are presented using tables and charts. Note that on some charts the vertical axis is base 10 logarithmic (for viewing purposes). The implementation of all the algorithms for all languages can be found in Appendix A.

5.3.1 Native environments

This section display the results for the tests made in each language native environment. As is illustrated in each subsection, these tests tend to do better performance wise than the corresponding tests in the .NET environment.

Addition Operator

The test takes 10 million positive randomly generated integers and calculates the sum of them. See Figure 5.1 for results. Other operators such as subtraction, multiplication, division and modulo were also tested but generated the same results.

Insertion Sort

Figure 5.2 illustrates the time it takes for each language to sort 10'000 randomly generated positive integers in ascending order. Note the severe time differences between Python and the other languages.

Merge Sort

Figure 5.3 illustrates the time it takes for each language to sort one million randomly generated positive integers in ascending order.

5.3.2 .NET environment

This section display the results for the tests made in each language when run in the .NET environment. The tests tend to perform poorer performance wise than the corresponding tests in the native environment.

5.3.3 Language Overhead

As can be seen in Table 5.2 Python has a greater overhead cost than the other two.

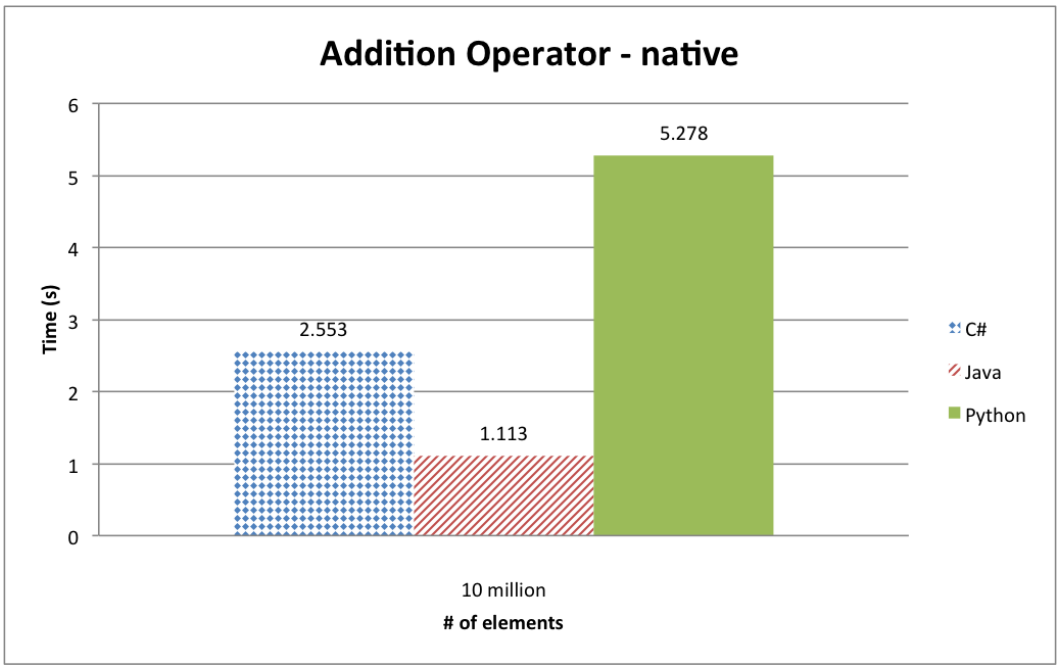

Figure 5.1. This tests a language ability to sum 10 million elements. Lower is better. The results show that Java running in its native environment is the fastest of the three languages with 1.113 seconds. Python is the slowest of the three with a runtime of 5.278 seconds, even when using its native library functions to sum the elements, see A.2.1.

Addition Operator

The test takes 10 million positive randomly generated integers and calculates the sum of them. See Figure 5.4 for results. Other operators such as subtraction, multiplication, division and modulo were also tested but generated the same results.

Insertion Sort

Figure 5.5 illustrates the time it takes for each language, run in the .NET environment, to sort 10'000 randomly generated positive integers in ascending order.

Merge Sort

Figure 5.6 illustrates the time it takes for each language, run in the .NET environment, to sort one million randomly generated positive integers in ascending order.

Figure 5.7 illustrates the amount of MB of memory used when running the merge sort algorithm in the .NET environment by the different languages. C# and Java are almost equal but Python uses more memory.

Figure 5.8 illustrates the percentage of the processor used when running the merge sort algorithm in the .NET environment by the different languages.

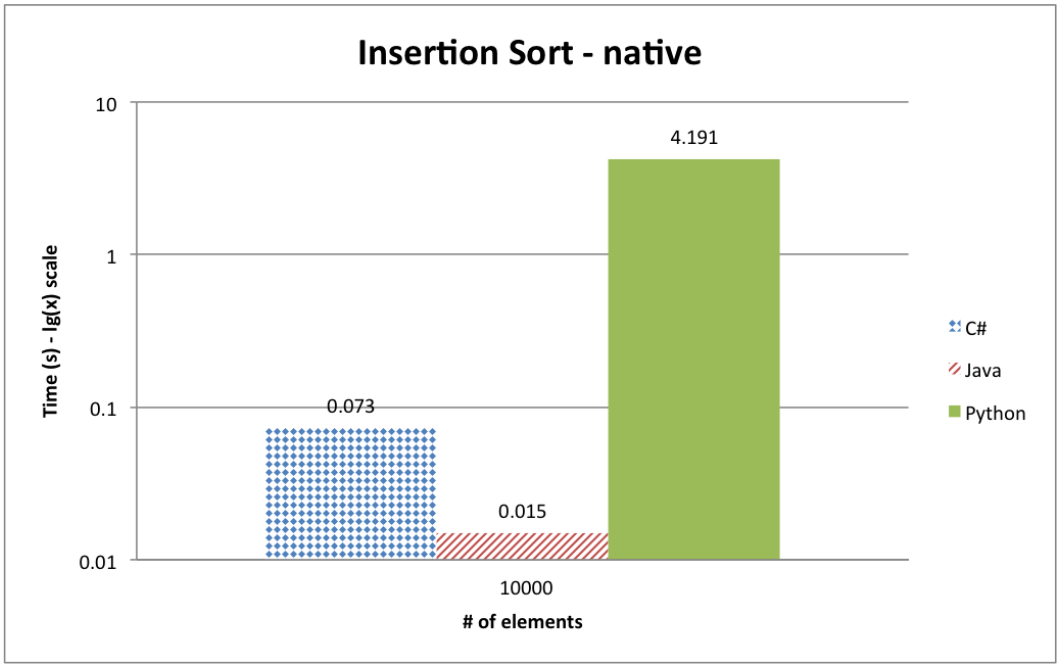

Figure 5.2. This test uses the Insertion sort algorithm to sort 10'000 elements in ascending order. Lower is better. Note that the vertical axis is base 10 logarithmic. Java is the fastest with 0.015 seconds while Python is the slowest with 4.191 seconds.

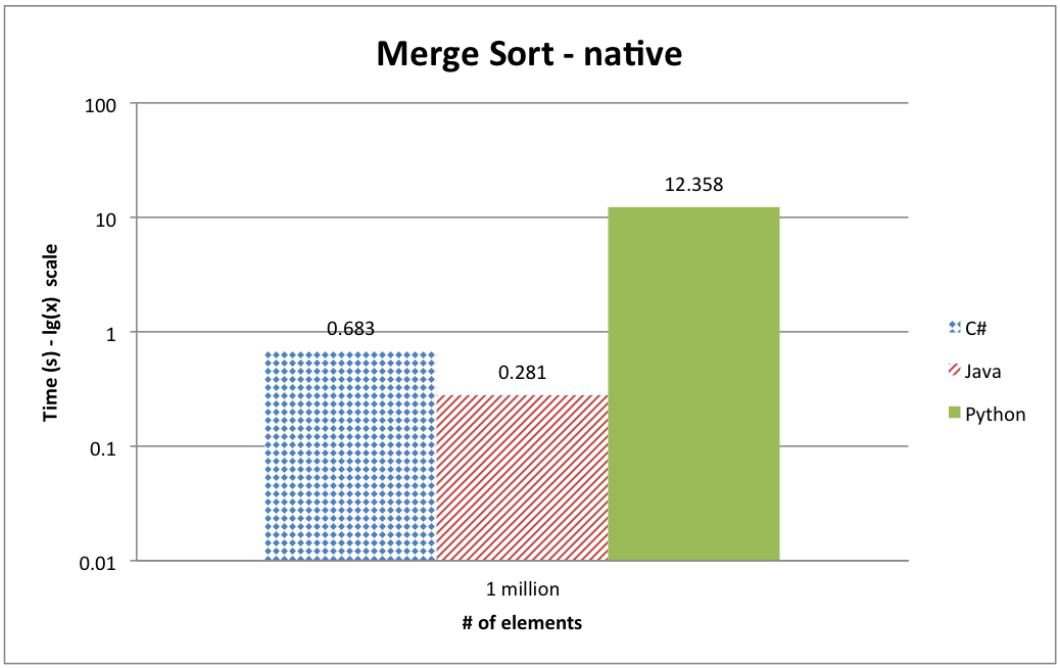

Figure 5.3. This test uses the Merge sort algorithm to sort one million elements in ascending order. Lower is better. Note that the vertical axis is base 10 logarithmic. Java is the fastest with 0.281 seconds while Python is the slowest with 12.358 seconds.

Language	Time
C#	0.014s
Java	0.038s
Python	1.006s

Table 5.2. The overhead startup cost for each language when run in the .NET environment. Since C# is the native language it has the lowest startup time with 0.014 seconds while Python has the longest with 1.006 seconds.

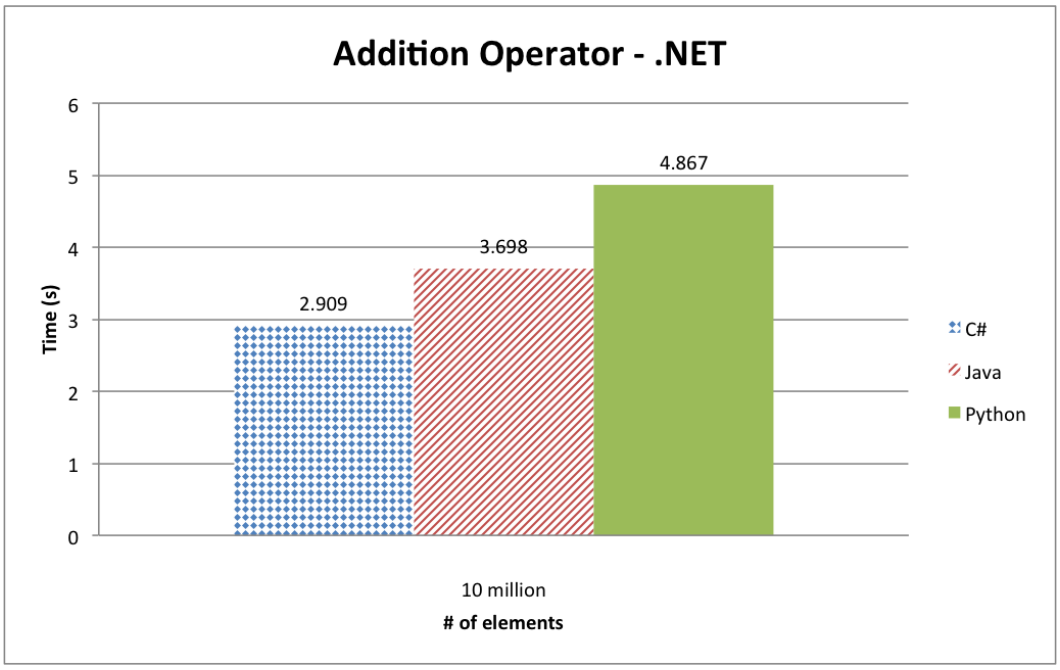

Figure 5.4. This tests a language ability to sum 10 million elements. Lower is better. The results show that C# running in its native environment (.NET) is the fastest of the three languages with 2.909 seconds. Python is the slowest of the three with a runtime of 4.867 seconds.

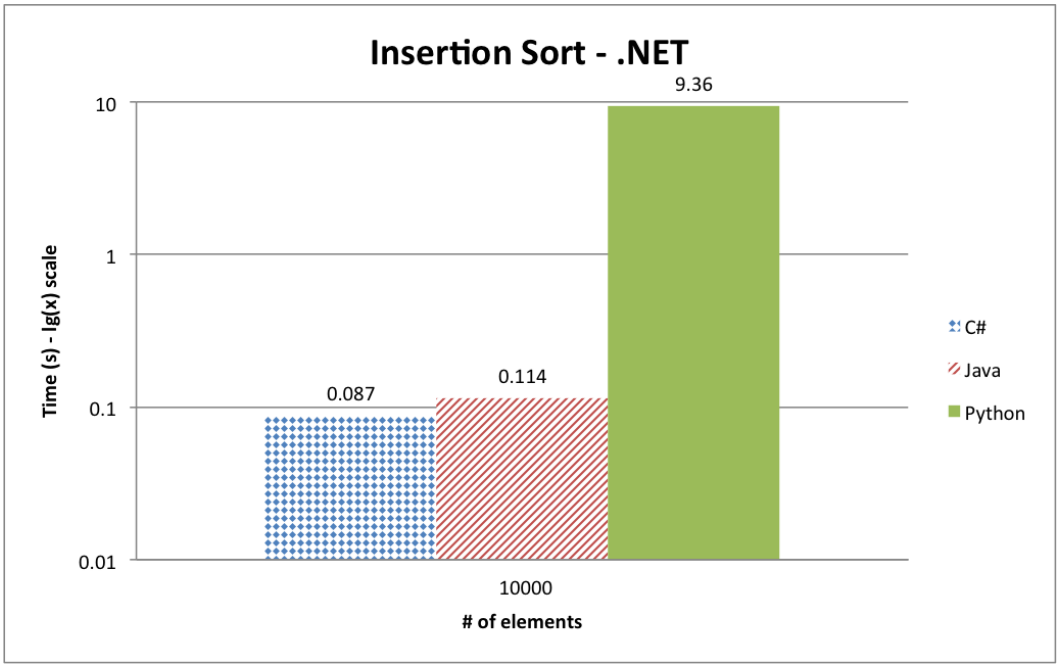

Figure 5.5. This test is run in the .NET environment and uses the Insertion sort algorithm to sort 10'000 elements in ascending order. Lower is better. Note that the vertical axis is base 10 logarithmic. C# is the fastest with 0.087 seconds. Java is a close second with 0.114 seconds. Python is the worst with 9.36 seconds.

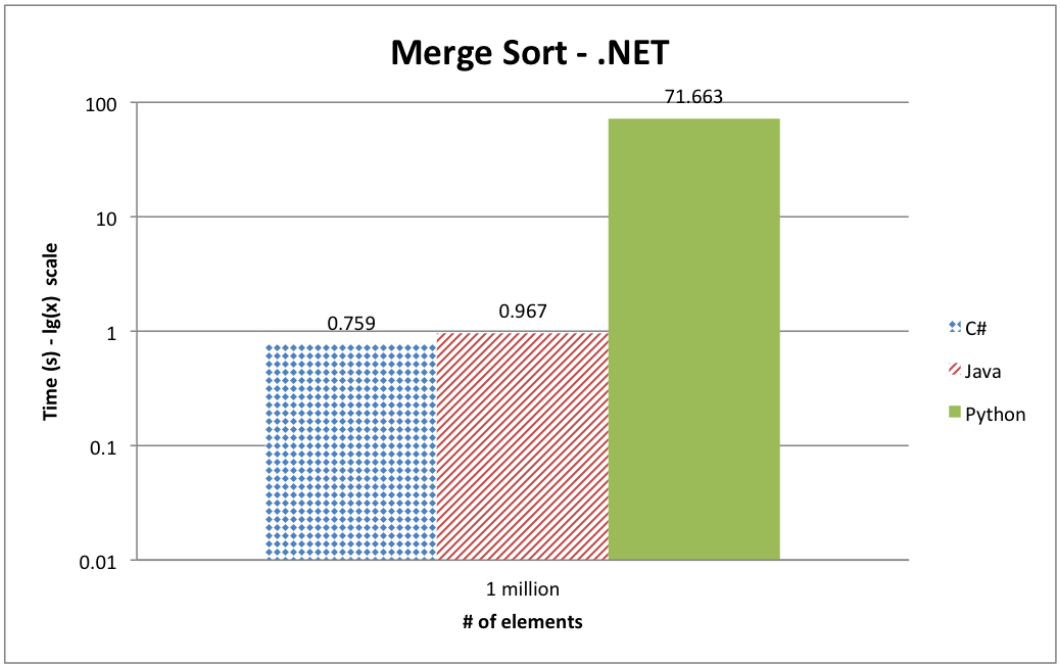

Figure 5.6. This test is run in the .NET environment and uses the Merge sort algorithm to sort one million elements in ascending order. A lower value is better. Note that the vertical axis is base 10 logarithmic. C# is the fastest with 0.759 seconds. Java comes in second with 0.967 seconds. Python is the slowest with 71.663 seconds.

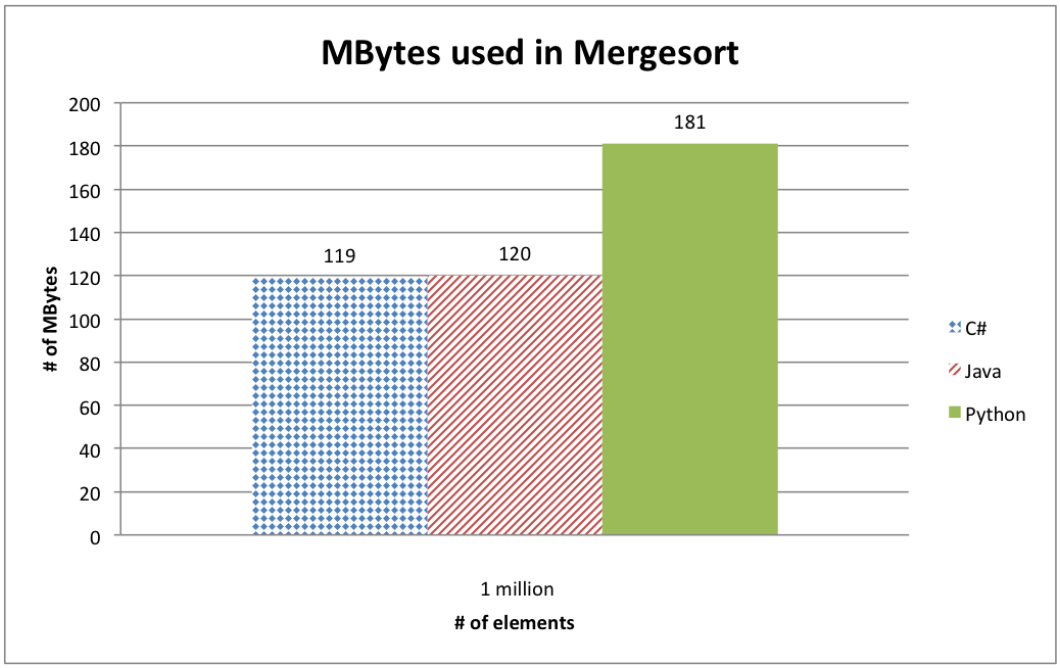

Figure 5.7. This test is run in the .NET environment and uses the Merge sort algorithm to sort one million elements in ascending order. A lower value is better. C# and Java perform equally with 119MB and 120MB of memory used. Python performs the worst with 181MB of memory used.

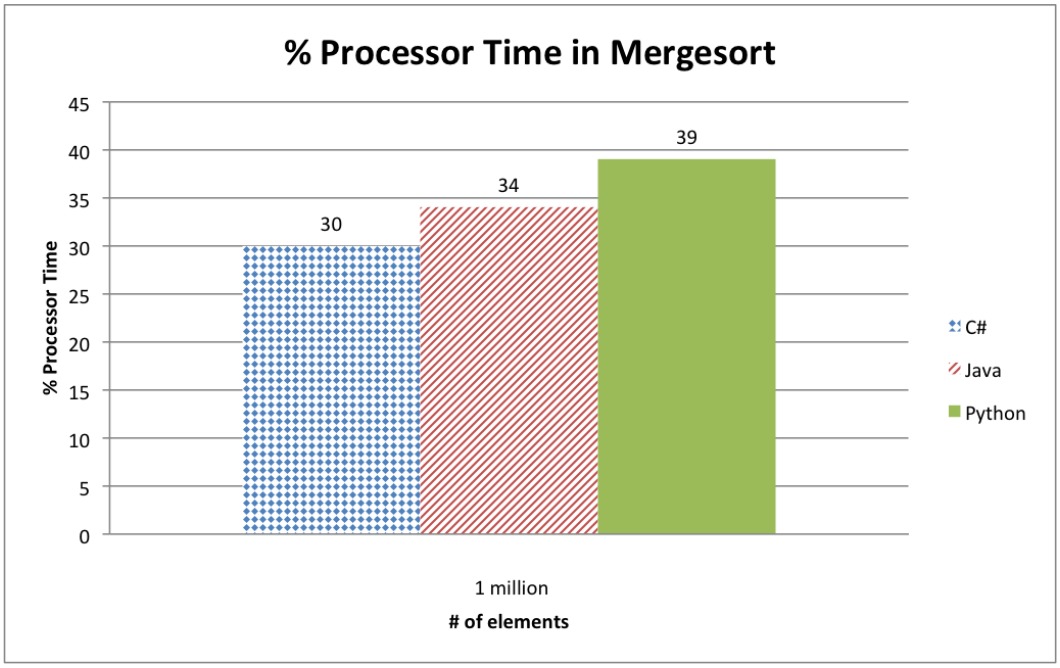

Figure 5.8. This test is run in the .NET environment and uses the Merge sort algorithm to sort one million elements in ascending order. A lower value is better. C# performs the best with 30% processor usage, Java is a close second with 34% and Python performs the worst with 39%.

Chapter 6

Discussion

This chapter contains reflections on the test results as well as the thesis conclusion and possible future work.

6.1 Result Analysis

The common operator tests do not contain any surprises other than that Python addition is faster than its other operator tests. The reason is because that code uses a built-in *sum* function [27] (see Figure 4.2) while the others use the *reduce* function [28].

Many of the slow execution results from Python derives from a couple of things. Firstly, Python is an interpreted language which means it suffers from a rather significant performance penalty from the start, this can be seen in Figure **??** where Python gets heavily outperformed by the other two languages, both of which are compiled languages. Also, in the .NET environment the Python code is run by a ScriptEngine, adding another layer of interpretation. This is also reflected in the language overhead test in Section 5.3.3. Good Python performance seem dependent on using the built-in functions of the language (since these are written in C), for instance a 3-4 time speed increase can be seen using the built in sort function *Tim Sort* [29] (a hybrid algorithm using a mix of insertion sort and merge sort), see Figure **??**.

Java outperforms C# on every test when run in the native environment. An even further speed increase could be achieved by adding the *-server* tag to the JVM making Java the fastest language by far in the native comparison tests.

The memory consumption of Merge Sort with 10'000 elements is about the same as with Insertion Sort. It would seem that this amount of elements is too small to have an impact on the amount of memory consumed. One would expect Merge Sort to consume significantly more memory, but the data obtained seem to be a reflection of a static cost.

Java is the least conservative language in relation to memory consumption. This can be seen in Figure **??**. It is, in fact, so memory hungry that the GC gets invoked

every other run. It also happens to be the fastest native language regardless of the number of elements that need to be sorted; both of these tendencies are also confirmed by [30] [31], however, keep in mind that these articles concerns native Java and not CIL compiled byte code in .NET. The memory consumption could either be a result of *javac* producing memory inefficient byte code or IKVM compiling the byte-code to CIL inefficiently.

6.2 Conclusion

The goal of this thesis was to evaluate the performance of different languages run in the .NET environment. The results indicate that Python suffers from significant performance problems. One could argue that the problem is not Python but rather the tests. Sorting for instance is best handled using the built-in functions. However, if one was to apply for a job and the task was to implement sorting algorithms in order to demonstrate their programming proficiency, the recruiter would not have them use the built-in functions as this would not reveal whether the applicant understands how to implement the algorithm or not.

This thesis concludes that while implementing multiple language compatibility in .NET is possible it is not feasible for all languages with concerns to performance. See, for example, the time it takes for Python to sort 1 million elements in .NET using Merge Sort (Figure **??**). The compensation times of the Python code varies considerably depending on how many built-in functions one can use or is allowed to use for solving a problem. This makes compensating for Python performance troublesome.

Instead of having .NET be the only back-end it could be used for handling input/output, security and then use separate processes for each language where this process simply executes the submitted code in its native language environment (e.g. Java code using *java.exe* or Python code using *python.exe*). This is the way many modern systems are built (see Section 2.3), and for good reason.

6.3 Future Work

Evaluating if other languages are suited for the .NET environment. Among these are, JavaScript using JavaScript.NET [32] or Jint [33], PHP using Phalanger [34], Ruby using IronRuby [35]. There are many more languages that have a CLI implementation but basing on the current popularity of programming languages [36] those are primarily the ones of interest.

CELINE can also be improved in many ways since the focus of this thesis was language compatibility and testing, rather than usability. An administrative web interface could be built along with better feedback on submissions. Energy can also be spent on usability testing in order to improve the user experience.

Appendix A

Code

A.1 Language Overhead

```
public class CSharpOverheadTest
{
        public string TestMethod(string input)
        {
                return "";
        }
        public static void Main(string[] args){}
}
```

```
public class JavaOverheadTest {

        public String TestMethod(String input){
                return "";
        }

        public static void main(String [] args){}
}
```

```
class PythonOverheadTest:
        def TestMethod(self, input):
                return "";
```

A.2 Common Operators

A.2.1 Addition

```
using System;

public class AdditionTest
{
    //int.parse() is faster than convert.ToInt32() since it performs an additional null check.
    public string AddNumbers(string input)
```

```
    {
        int sum = 0;

        string[] splitted = input.Split(' ');
        for (var i = 0; i < splitted.Length; i++)
        {
            sum += Int32.Parse(splitted[i]);
        }
        return sum.ToString();
    }

    public static void Main(){}
}
```

```
public class AdditionTest {

        public String AddNumbers(String input){
                int sum = 0;
                String [] splitted = input.split(" ");

                for(int i = 0; i < splitted.length; i++){
                        sum += Integer.parseInt(splitted[i]);
                }

                return "" + sum;
        }

        public static void main(String[] args){}
}
```

```
import sys
class AdditionTest:
    def AddNumbers(self, input):
        input = input.split(" ")
        array = list(map(int, input))
        total = sum(array)
        return str(total)
```

A.2.2 Subtraction

```
using System;
public class SubtractionTest
{
    //int.parse() is faster than convert.ToInt32() since it performs an additional null check.
    public string SubtractNumbers(string input)
    {
        int sum = 0;

        string[] splitted = input.Split(' ');
        for (var i = 0; i < splitted.Length; i++)
        {
            sum -= Int32.Parse(splitted[i]);
        }
        return sum.ToString();
    }
```

```
        public static void Main(){}
}
```

```
public class SubtractionTest {

        public String SubtractNumbers(String input){
                int sum = 0;
                String [] splitted = input.split(" ");

                for(int i = 0; i < splitted.length; i++){
                        sum -= Integer.parseInt(splitted[i]);
                }

                return "" + sum;
        }

        public static void main(String[] args){}
}
```

```
import operator
class SubtractionTest:
    def SubtractNumbers(self, input):
        input = input.split(" ")
        array = list(map(int, input))
        total = reduce(operator.sub, array, 1)
        return str(total)
```

A.2.3 Multiplication

```
using System;
public class MultiplicationTest
{
    //int.parse() is faster than convert.ToInt32() since it performs an additional null check.

    public string MultiplyNumbers(string input)
    {
        int sum = 1; //multiplication starts at 1, not 0.

        string[] splitted = input.Split(' ');
        for (var i = 0; i < splitted.Length; i++)
        {
            sum *= Int32.Parse(splitted[i]);
        }
        return sum.ToString();
    }

        public static void Main(String[] args){}

}
```

```
public class MultiplicationTest {

        public String MultiplyNumbers(String input){
                int sum = 1;
                String [] splitted = input.split(" ");
```

```
                for(int i = 0; i < splitted.length; i++){
                        sum *= Integer.parseInt(splitted[i]);
                }

                return "" + sum;
        }

        public static void main(String[] args){}
}
```

```
import operator
class MultiplicationTest:
    def MultiplyNumbers(self, input):
        input = input.split(" ")
        array = list(map(int, input))
        total = reduce(operator.mul, array, 1)
        return str(total)
```

A.2.4 Division

```
using System;

public class DivisionTest
{
    //int.parse() is faster than convert.ToInt32() since it performs an additional null check.

    public string DivideNumbers(string input)
    {
        float sum = 1; //divide by zero is a bad idea

        string[] splitted = input.Split(' ');
        for (var i = 0; i < splitted.Length; i++)
        {
            sum /= Int32.Parse(splitted[i]);
        }
        return sum.ToString();
    }

        public static void Main(String[] args){}

}
```

```
public class DivisionTest {

        public String DivideNumbers(String input){
                float sum = 1;
                String [] splitted = input.split(" ");

                for(int i = 0; i < splitted.length; i++){
                        sum /= Integer.parseInt(splitted[i]);
                }

                return "" + sum;
        }

        public static void main(String[] args){}
```

```
}
```

```
import operator
class DivisionTest:
    def DivideNumbers(self, input):
        input = input.split(" ")
        array = list(map(int, input))
        total = reduce(operator.div, array, 1)
        return str(total)
```

A.2.5 Modulo

```
using System;

public class ModuloTest
{
    public string ModuloNumbers(string input)
    {
        double sum = double.MaxValue;

        string[] splitted = input.Split(' ');
        for (var i = 0; i < splitted.Length; i++)
        {
            sum %= Double.Parse(splitted[i]);
        }
        return sum.ToString();
    }

        public static void Main(String[] args){}
}
```

```
public class ModuloTest {

        public String ModuloNumbers(String input){
                double sum = Double.MAX_VALUE;
                String [] splitted = input.split(" ");

                for(int i = 0; i < splitted.length; i++){
                        sum %= Double.parseDouble(splitted[i]);
                }

                return "" + sum;
        }

        public static void main(String[] args){}
}
```

```
import operator
class ModuloTest:
    def ModuloNumbers(self, input):
        input = input.split(" ")
        array = list(map(int, input))
        total = reduce(operator.mod, array, 1)
        return str(total)
```

A.3 Insertion Sort

```
using System;
using System.Text;

public class CSharpInsertionSort
{
        public String Sort(String input)
        {
                // Organize input.
                String[] splitted = input.Split(' ');
                int[] array = new int[splitted.Length];
                for (int i = 0; i < splitted.Length; i++)
                {
                        array[i] = int.Parse(splitted[i]);
                }

                // Start insertion sort.
                for (int i = 1; i < array.Length; i++)
                {
                        int a = array[i];
                        int j;
                        for (j = i - 1; j >= 0 && array[j] > a; j--)
                        {
                                array[j + 1] = array[j];
                        }
                        array[j + 1] = a;
                }

                // Prepare output.
                StringBuilder builder = new StringBuilder();
                for (int i = 0; i < array.Length; i++)
                {
                        builder.Append(array[i] + " ");
                }
                return builder.ToString().TrimEnd();
        }

        public static void Main(String [] args){}
}
```

```
public class JavaInsertionSort {
        public String Sort (String input) {

                // Organize input.
                String [] splitted = input.split(" ");
                int [] A = new int[splitted.length];
                for(int i = 0; i < splitted.length; i++){
                        A[i] = Integer.parseInt(splitted[i]);
                }

                // Start insertion sort.
            for (int i = 1; i < A.length; i++) {
              int a = A[i];
              int j;
              for (j = i - 1; j >=0 && A[j] > a; j--){
                A[j + 1] = A[j];
```

```
            }
            A[j + 1] = a;
        }

        // Prepare output.
        StringBuffer buffy = new StringBuffer();
        for(int i = 0; i < A.length; i++){
            buffy.append(A[i] + " ");
        }
        return buffy.toString().trim();
    }

    public static void main(String[] args){
    }
}
```

```
import sys
class PythonInsertionSort:
    def Sort(self, s):

        # Organise input
        s = s.split(" ")
        s = list(map(int, s))

        # Start insertion sort
        for i in range(1, len(s)):
            val = s[i]
            j = i - 1
            while (j >= 0) and (s[j] > val):
                s[j+1] = s[j]
                j = j - 1
            s[j+1] = val

        s = map(str, s)
        s = " ".join(s)
        return s
```

A.4 Merge Sort

```
using System;
using System.Text;

public class CSharpMergeSort
{
    public static int[] Sort(int [] array)
    {
        if (array.Length > 1)
        {
            int elementsInA1 = array.Length / 2;
            int elementsInA2 = array.Length - elementsInA1;
            int[] arr1 = new int[elementsInA1];
            int[] arr2 = new int[elementsInA2];

            for (int y = 0; y < elementsInA1; y++)
            {
                arr1[y] = array[y];
            }
```

```
                for (int x = elementsInA1; x < elementsInA1 + elementsInA2; x++)
                {
                        arr2[x - elementsInA1] = array[x];
                }


                arr1 = Sort(arr1);
                arr2 = Sort(arr2);

                int i = 0, j = 0, k = 0;

                while (arr1.Length != j && arr2.Length != k)
                {
                        if (arr1[j] <= arr2[k])
                        {
                                array[i] = arr1[j];
                                i++;
                                j++;
                        }
                        else
                        {
                                array[i] = arr2[k];
                                i++;
                                k++;
                        }
                }

                while (arr1.Length != j)
                {
                        array[i] = arr1[j];
                        i++;
                        j++;
                }
                while (arr2.Length != k)
                {
                        array[i] = arr2[k];
                        i++;
                        k++;
                }
        }
        return array;
}

public static String HandleInputOutput(String input)
{
        // Organize input.
        String[] splitted = input.Split(' ');
        int[] array = new int[splitted.Length];
        for (int i = 0; i < splitted.Length; i++)
        {
                array[i] = int.Parse(splitted[i]);
        }


        // Sort the array (using mergesort)
        array = Sort(array);


        // Prepare output.
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < array.Length; i++)
```

```
                {
                        builder.Append(array[i] + " ");
                }
                return builder.ToString().TrimEnd();
        }

        public static void Main(String [] args){}
}
```

```
public class JavaMergeSort{

        public int[] Sort(int array[]) {
                if(array.length > 1)  {
                        int elementsInA1 = array.length/2;
                        int elementsInA2 = array.length - elementsInA1;
                        int arr1[] = new int[elementsInA1];
                        int arr2[] = new int[elementsInA2];

                        for(int i = 0; i < elementsInA1; i++)
                                arr1[i] = array[i];

                        for(int i = elementsInA1; i < elementsInA1 + elementsInA2; i++)
                                arr2[i - elementsInA1] = array[i];

                        arr1 = Sort(arr1);
                        arr2 = Sort(arr2);

                        int i = 0, j = 0, k = 0;

                        while(arr1.length != j && arr2.length != k) {
                                if(arr1[j] <= arr2[k]) {
                                        array[i] = arr1[j];
                                        i++;
                                        j++;
                                } else {
                                        array[i] = arr2[k];
                                        i++;
                                        k++;
                                }
                        }

                        while(arr1.length != j) {
                                array[i] = arr1[j];
                                i++;
                                j++;
                        }
                        while(arr2.length != k) {
                                array[i] = arr2[k];
                                i++;
                                k++;
                        }
                }
                return array;
        }

        public String HandleInputOutput(String input){

                // Organize input.
                String [] splitted = input.split(" ");
```

```
                int [] A = new int[splitted.length];
                for(int i = 0; i < splitted.length; i++){
                        A[i] = Integer.parseInt(splitted[i]);
                }

                // Start merge sort.
                A = Sort(A);

                // Prepare output.
                StringBuffer buffy = new StringBuffer();
                for(int i = 0; i < A.length; i++){
                        buffy.append(A[i] + " ");
                }
                return buffy.toString().trim();
        }

        public static void main(String[] args){}
}
```

```
import sys
import time
import array

class PythonMergeSort:
        def Sort(self, arr):
                if len(arr) == 1:
                        return arr

                m = len(arr) // 2
                l = self.Sort(arr[:m])
                r = self.Sort(arr[m:])

                if not len(l) or not len(r):
                        return l or r

                result = []
                i = j = 0
                while (len(result) < len(r)+len(l)):
                        if l[i] < r[j]:
                                result.append(l[i])
                                i += 1
                        else:
                                result.append(r[j])
                                j += 1
                        if i == len(l) or j == len(r):
                                result.extend(l[i:] or r[j:])
                                break

                return result

        def HandleInputOutput(self, input):
                # Organise input.
                input = input.split(" ")
                array = list(map(int, input))

                # Start merge sort.
                derp = MergeSort()
                array = derp.Sort(array)

                # Prepare output.
```

```
        array = map(str, array)
        returnString = " ".join(array)

        return returnString
```

Bibliography

[1] Colton D, Fife L, and Thompson A. A Web-based Automatic Program Grader. *Information Systems Education Journal*, 2006.

[2] H, Suleman. Automatic marking with Sakai. *SAICSIT '08 Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology Pages 229-236*, 2008.

[3] Hollingsworth J. Automatic graders for programming classes. *Communications of the ACM*, 1960.

[4] Douce C, Livingstone D, and Orwell J. Automatic Test-Based Assesment of Programming: A Review. *Journal on Educational Resources in Computing*, 2005.

[5] E, Enström and G, Kreitz and F, Niemelä and P, Söderman and V, Kann. Five Years with - Kattis Using an Automated Assessment System in Teaching. *41st ASEE/IEEE Frontiers in Education Conference*, 2011.

[6] M, Amelung and P, Forbrig and D, Rösner. Towards Generic and Flexible Web Services for E-Assessment. *TiCSE '08 Proceedings of the 13th annual conference on Innovation and technology in computer science education*, 2008.

[7] Hext J, B and Winings J, W. An automatic grading scheme for simple programming exercises. *Communications of the ACM, Volume 12 / Number 5 / May*, 1969.

[8] MSDN. CSharpCodeProvider.
`http://msdn.microsoft.com/en-us/library/microsoft.csharp.csharpcodeprovider.aspx`.
Accessed 2013-03-04.

[9] MSDN. System CodeDom Compiler.
`http://msdn.microsoft.com/en-us/library/system.codedom.compiler.icodecompiler.aspx`.
Accessed 2013-03-04.

[10] Oracle. The Java Programming Language Compiler - javac.
http://docs.oracle.com/javase/6/docs/technotes/guides/javac/.
Accessed 2013-03-04.

[11] Oracle. Java Development Kit SE.
http://www.oracle.com/technetwork/java/javase/overview/index.html.
Accessed 2013-03-04.

[12] Frijters J. Ikvmc.
http://sourceforge.net/apps/mediawiki/ikvm/index.php?title=Ikvmc.
Accessed 2013-03-04.

[13] Frijters J. IKVM.NET.
http://www.ikvm.net/.
Accessed 2013-03-04.

[14] Viehland D. IronPython.
http://ironpython.net/.
Accessed 2013-03-04.

[15] MSDN. Application Domains.
http://msdn.microsoft.com/en-us/library/cxk374d9.aspx.
Accessed 2013-03-04.

[16] MSDN. Remotable Objects.
http://msdn.microsoft.com/en-us/library/aa720494(v=vs.71).aspx.
Accessed 2013-03-04.

[17] MSDN. StackOverflowException Class.
http://msdn.microsoft.com/en-us/library/system.stackoverflowexception.aspx.
Accessed 2013-03-22.

[18] IronPython. Performance Report.
http://ironpython.codeplex.com/wikipage?title=IP27A1VsCPy27Perf.
Accessed 2013-03-04.

[19] MSDN. CLSCompliantAttribute.
http://msdn.microsoft.com/en-us/library/system.clscompliantattribute.aspx.
Accessed 2013-03-04.

[20] IronPython documentation. Accessing Python code from other .NET code.
https://github.com/IronLanguages/main/blob/master/Languages/IronPython/Public/Doc/dotnet-integration.rst#accessing-python-code-from-other-net-code.
Accessed 2013-03-04.

[21] MSDN. Stopwatch class.
http://msdn.microsoft.com/en-us/library/system.diagnostics.stopwatch.aspx.
Accessed 2013-03-04.

[22] Goodrich M T and Tamassia R. *Algorithm Design: Foundations, Analysis and Internet Examples*, pages 13–16. John Wiley & Sons, Inc., 2002.

[23] D, R Martin. Sorting Algorithms: Insertion Sort.
http://www.sorting-algorithms.com/insertion-sort.
Accessed 2013-03-04.

[24] D, R Martin. Sorting Algorithms: Merge Sort.
http://www.sorting-algorithms.com/merge-sort.
Accessed 2013-03-04.

[25] B, Abrams. Is the CLR a Virtual Machine?
http://blogs.msdn.com/b/brada/archive/2005/01/12/351958.aspx.
Accessed 2013-10-20.

[26] J L, Schilling. The Simplest Heuristics May Be the Best in Java JIT Compilers.
http://www.sco.com/developers/java/news/jit-heur.pdf.
2003.

[27] Python 2.7.3 Documentation: Sum Function.
http://docs.python.org/2/library/functions.html#sum.
Accessed 2013-03-22.

[28] Python 2.7.3 Documentation: Reduce Function.
http://docs.python.org/2/library/functions.html#reduce.
Accessed 2013-03-22.

[29] Python 2.7.3 Documentation: List Functions.
http://docs.python.org/2/tutorial/datastructures.html.
Accessed 2013-03-22.

[30] Brent Fulgham. Computer Language Benchmark Game.
http://benchmarksgame.alioth.debian.org/u64/csharp.php.
Accessed 2013-03-04.

[31] Decebal Mihailescu. Benchmark start-up and system performance for .Net, Mono, Java, C++ and their respective UI.
http://www.codeproject.com/Articles/92812/Benchmark-start-up-and-system-performance-for-Net.
Accessed 2013-03-04.

[32] JavaScript.NET.
http://javascriptdotnet.codeplex.com/.
Accessed 2013-03-04.

[33] Jint.
http://jint.codeplex.com/.
Accessed 2013-03-04.

[34] Phalanger.
http://phalanger.codeplex.com/.
Accessed 2013-03-04.

[35] IronRuby.
http://ironruby.codeplex.com/.
Accessed 2013-03-04.

[36] TIOBE Programming Community Index.
http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.
Accessed 2013-03-20.