Structure of computer systems PROJECT Microbenchmark (C#, Java, Python)

Faculty: Automation and Computer Science

Department: Computer Science

Student: Cofaru Vlad-George , **Group** 30433/1

Contents

1. Introduction

- ## Context

The purpose of this project is to establish a set of Microbenches in order to study the performance of various programming languages . In this paper we will crate a set of tests to see how different programming languages perform in a various set of scenarios , in the end providing a comparison between them . To this purpose, the languages that we will be studied are Java , C# and Python.

Java and C# are were chosen because they very popular choices when it comes to working with the Object Oriented programming paradigm (OOP), and we want to observe their strong points.

Python was selected because it has recently became the most popular programming language, according to the PYPL index. The strength of the language is in the fact that is a very versatile programming language, that can be applied to anything.

In the end we also want to establish a comparison between two types of typing , static typing (represented by the Java , C#) and dynamic typing (Python)

- ## Specifications

The project will use a series of benchmarks to study the response time of different programing languages in following scenarios:

  - Memory allocation
  - Memory access (static versus dynamic)
  - Predefined data structures (creation and use)
  - Object Oriented programming
  - Threads manipulation(creation, synchronization , use)

The secondary aim of this project is to obtain a comparison between a static type of language , where the types of all the variables are specified in their definition, and dynamically typed , where the types of the variables is determined only an compile time

- ## Objectives

Design and use a series of tests in order to analyze the performance of three programming languages in different scenarios. The results will be used to draw a comparison between the three to see which is more efficient in which type of scenario .Also it is followed to see the comparison between two popular OOP languages (Java , C#) and two types of structural typing (static and dynamic)

2. Bibliographic study

• Memory management

The Garbage collectors[1] serves as an automatic memory manager. The garbage collector manages the allocation and release of memory for an application. Among the responsibilities of a garbage collector are : Allocates objects on the managed heap efficiently , Reclaims objects that are no longer being used, clears their memory, and keeps the memory available for future allocations. Provides memory safety by making sure that an object cannot use the content of another object.[2]

When it comes to memory management all the three languages chosen for this project have some characteristics , the presence of garbage collectors , the function calls being allocated on the stack and the objects on the heap .

Their particularities though can have an impact on the performance , so they will be analyzed bellow:

❖ Java

The Java follows basic memory management rules , as many other static typed languages . The function calls , along with any local variables and primitive types(int , char float) are stored in the stack , and all the Objects (predefined or user created) are stored in a heap , and referenced from the stack .

Java does memory management automatically ,the management being divided into two major parts: JVM(Java Memory Structure) and the Garbage Collector. The JVM is responsible for creates various run time data areas in a heap, which are then used during the execution. The data is destroyed when the execution thread ends , while the areas are destroyed when the JVM exits[3] during the execution.

❖ Python

Memory management in python involves a private heap containing all Python objects and data structures. When an allocation is required a raw memory allocator ensures that there is enough room in the private heap for storing all Python-related data by interacting with the memory manager of the operating system.[4]

In python the stack is used only for storing functions and local variables inside those functions . Any other variable declaration , since python is a dynamic typed language , is treated as an object , therefore the heap will store that variable.[5]

Since Dynamic Memory allocation underlies the memory management, when an object is no longer needed, the memory manager will automatically reclaim memory from them. Memory is not necessarily directly released into the operating system, but returned to the python interpreter

Another specific memory management technique is that fact that , when two variables have the same value, the virtual manager , instead of creating two objects on the heap , it makes both variables to point towards the same region .

❖ C#\

Objects are allocated on the heap in a continuous manner , the small objects being put in a separate small objects heap. Memory allocation is a very fast process as it is just the adding of a value to a pointer.

For a better performance the heap is divided in three sections , named generations. When objects are just created, they are placed to the Generation 0 (Gen 0), When this section is full , the Gc(Garbage collector) performs a garbage collection , any unreachable objects , all the usable objects being promoted to Gen 1 . The process is similar in the case Gen 1 reaches its maximum storage , every usable object being promoted to Gen 2 , while Gen 0 and Gen 1 go through garbage collection .

This way of dividing the heap ensures a better memory management , all the unused objects being eliminated each time a generations its full

- Static Typing versus Dynamic Typing

The languages that will be studied in this paper can be divided in two

categories static typed languages(Java ,C#) and dynamic typed(Python) .

In Static Typed , when a variable is declared this sets aside an area of memory for holding values allowed by the data type of the variable. The memory allocated will be interpreted as the data type suggests. If it’s an integer variable the memory allocated will be read as an integer and so on. When we assign or initialize it with some value, that value will get stored at that memory location. At compile time, initial value or assigned value will be checked. So we cannot mix types. The variable types will checked constantly before run-time[6]

The advantage of this type of languages is the fact that the code is constantly compiled , therefore the types are checked before running (static) and the type error are immediately caught before the actual program can begin .The fact that the types are known allows for machine code optimization.

It is expected from statically typed languages to have a better performance because the types are not check dynamically rather are calculated before running the program

In Dynamically Typed language , the type of the variable is unknown until the code is run. What it does is, It stores that value at some memory location and then binds that variable name to that memory container making it accessible through that variable name. The type of a variable is determined during the execution [7]

The advantages of this type of language is the fact that you can reuse the variables on the fly changing their type .While this approach offers a more comfortable way of writing code , it lacks in speed , due to the fact that the code can not be precompiled beforehand. This means that the errors will be caught only on run time , and the actual execution time will be longer because the compiler can not perform any code optimization , and it had to figure out at run time the type of the variables it is working with [8]

3. Benchmarks

- Java -- **JMH(java-microbenchmark-harness)**

JMH is an API which allows us to write microbenchmarks for Java. This API takes care of warming up the JVM and code-optimization patch in order to obtain results as close to reality of possible .

To use JMH we simply have to add the dependencies in our java project , after that in order to mark the methods we want to benchmark we will use annotations @ , making the test fully customizable . In terms of annotation we can specify :

@ Warm-up iterations which before starting the measurement procedure will run the function a certain amount of times .

@BenchmarkMode : this in turn can be set as **Throughput** (Measures the number of operations per second) , **AverageTime**(Measures the average time it takes for the benchmark method to execute) , **SampleTime(**measures how long time it takes for the benchmark method to execute, including max, min time) , **ALL**(measures all of the above)

@OutputTimeUnit, the time unit of the measurements

@Scope , this allows to specify certain variables and function that are needed in the benchmark function , but their execution time is not a part of the benchmark measurements

@Measurements : specify the number of iterations of the measured function

- C# -- BenchmarkDotNet API

BenchmarkDotNet is a special API specifically designed to benchmark specific functions of your code . In is easy to use , once you install the package you only have to identify functions by writing [Benchmark] . This api takes take or memory warmups in order to obtain results as close as possible to reality . The numbers of runs and parameters again being fully customizable . The results can then be easily reported in a requested format being html , csv , json etc

This benchmark above all is very reliable being used in over 4500 projects and 650,000 project files .

- Python

Pytest was used in order to measure the performance of various script written in Python. This api runs script that measures the time of execution of the marked function , generating a report and saving it in a .json file .

4. Tests

- Memory allocation , here the purpose is to test the stack versus heap , the time to create and store a variable on the stack , and the time to access this variable , vs the same process on the Heap . Static Memory Allocation is done before program execution and there is no space for re-usability whereas in dynamic addressing the memory can be freed when not required . In static memory allocation, once the memory is allocated, the memory size can not change.
- Predefined data structures (creation and use) : Here the goal is to use a set of already build in containers (arrays , lists , map , dictionaries etc) and to test the efficiency of working with them (retrieve an element , search for a certain one , add , sort the container) , and to compare the time between programs .

- Object Oriented programming : The goal is to create a certain Custom class , and to test the efficiency of creating an object , updating one , using in certain functions(data retrieval time for custom objects)
- Threads manipulation(creation, synchronization , use): The goal is to test the performance of the programming languages in the following scenarios : Thread creation , thread synchronization (maybe different mechanisms and their efficiency) , multithreading programs .

5. Tests Design

General: In order to avoid the dead code elimination of the compilers , especially the loop optimization (the process when if the compiler identifies a loop or a process as having no effect , it will simply not execute it) we applied the following structure :

Instead of :
```
public int ArrayListBinaryFind()
  {
                for (int j = 0; j < 10; j++)
ArrList.BinarySearch(RandomNumbers[j]);
  }
```
The following model was used :

```
public int ArrayListBinaryFind()
    {
       int cnt = 0;
       for (int j = 0; j < 10; j++)

       { cnt = ArrList.BinarySearch(RandomNumbers[j]);
          cnt++;
       }
       return cnt;
    }
```

Thus the loop becomes functional having a result , so no more loop optimization is applied ;

- Memory allocation , here the purpose is to test the stack versus heap , the time to create

and use a variable , being located in the stack , or on the Heap. In order to keep the design simple to limit the number of extra operations that are measured the following test was designed : a simple increment function was created , which saves on the stack an int , increments it and returns the new value , thus testing allocation , writing and reading from the stack . In order to test the heap the same principle was applied , but working with an object string , rather than an int .

In this suite of tests I also wanted to verify the efficiency of recursion functions , knowing that they are very stack expensive , to make this test the factorial function was defined , in two manners , one being the classical recursive way , and the other one being the factorial defined in a functional programming stile of writing (using lambda expressions) .

- Predefined data structures , the purpose of these tests was to compare different

predefined data structures , them being : array (int [] arr) , ArrayLists , Lists , LinkedLists(the data type stored must be of type node) , HashSet , Stack , Queue.

These generic containers were tested in the following scenarios :

- Creating and populating the containers with a number of elements (here is to be observed also the difference between specifying the range of an array , and leaving it empty , where the internal resize mechanism is always applied)
- The Find Function : given a random element that is in the containers , the average time it takes to return the element .
- The Not Find Function : given a random elements that certainly is not in the array , what is the time of the functions to return a result
- Adding at End : having a populated container , what is the time to add an element to the end .
- Removing element: what is the time to eliminate an element from a container , with the condition that the container must also be resized .

- Object Oriented programming : here the test was to observe the time it would take to

work with a simple custom class . During these tests we also observed how it changed the execution time of a method was changed in regard with the parent child inheritance relationship . So were tested the creation of an object , parent vs child , the call of a custom method from the parent , from the child , and also a method belonging to the parent , called from the child class . And finally , we observed the time it takes for the equality method to operate in these two cases , parent and child . And also the efficiency of the setter and getters methods .

- Threads manipulation : Here the goal was to observe the time it will take for a basic

program to work with threads . So it was measured the time it takes to create a thread , to run it , to get the results . We also wanted to measure how context switch affects the program , so in order to achieve that , a synchronization mechanism was applied , a lock . The create and test this multithreading scenario , a basic class was created , having a member , that will generate the race condition , and the need of synchronization . Two methods were created , both using the race condition variable , but one using a lock to limit the access to it . So using this class we were able to test , the time it takes to create a thread , to attribute a method to it , to run it , and the time difference between having a lock to mitigate the access to the race condition elements , and letting the threads run freely .

6. Planning

- Performing the suite of tests on Java using the JMH (the Java Microbenchmark

Harness)[9] to measure the results in the following scenarios Memory allocation ,Memory access (static versus dynamic) ,Predefined data structures (creation and use) .Object Oriented programming ,Threads manipulation(creation, synchronization , use)

- Performing the suite of tests on Python measuring the results

in the following scenarios Memory allocation ,Memory access (static versus dynamic) ,Predefined data structures (creation and use) .Object Oriented programming ,Threads manipulation(creation, synchronization , use)

- Performing the suite of tests on C# using BenchmarkDotNet to measure

the results in the following scenarios Memory allocation ,Memory access (static versus dynamic) ,Predefined data structures (creation and use) .Object Oriented programming ,Threads manipulation(creation, synchronization , use)

- OOP comparison of Java and C#
- Static vs Dynamic typing comparison between Python and

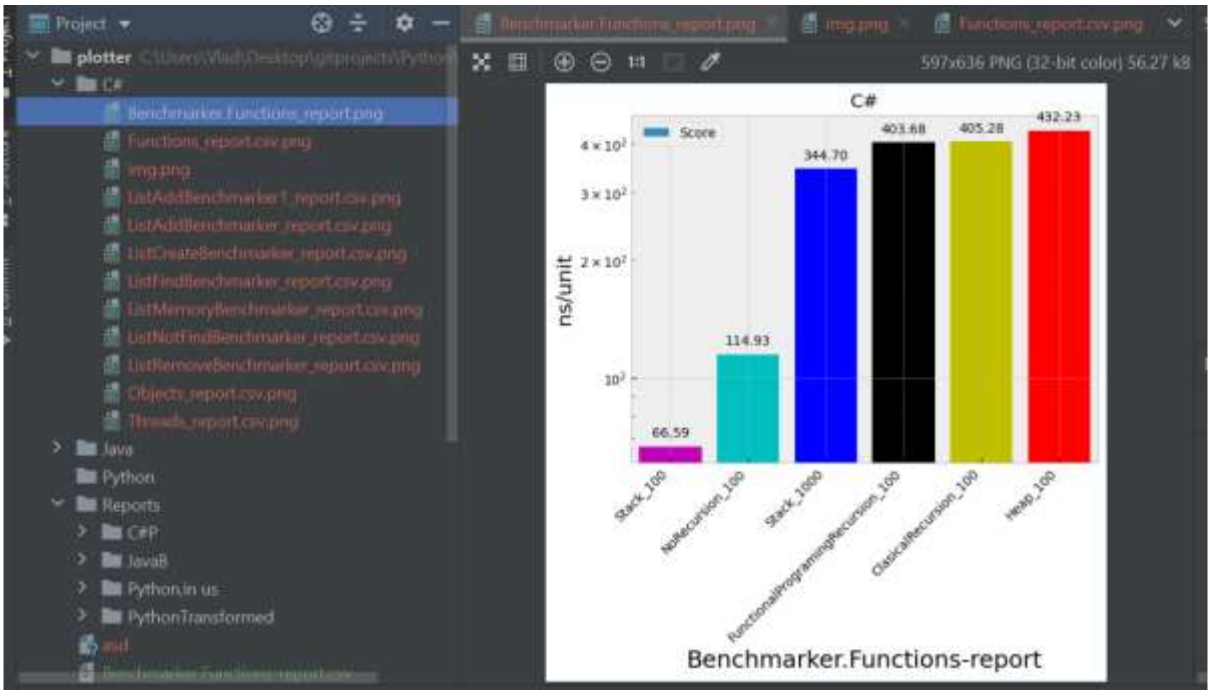
Java/C#

7. Results

• Plotter

To be able to draws a comparison between the programming languages mentioned above , a

python script was written witch is able to read the data from the csv files ,in which the results of the measurements were saved, and generate a bar graph for each measurement. The script itself takes care of parsing the input files, normalizing the results(bringing them in the same unit of measurement and renames the columns of the csv all to have the same name , in order to be able to group them latter in the comparison part of the program) and generates a folder where the graphs are saved , for each programming language.



Project
plotter C:\Users\Vlad\Desktop\gitprojects\Python
C#
Benchmarker.Functions_report.png
Functions_report.csv.png
img.png
ListAddBenchmarker1_report.csv.png
ListAddBenchmarker_report.csv.png
ListCreateBenchmarker_report.csv.png
ListFindBenchmarker_report.csv.png
ListMemoryBenchmarker_report.csv.png
ListNotFindBenchmarker_report.csv.png
ListRemoveBenchmarker_report.csv.png
Objects_report.csv.png
Threads_report.csv.png
Java
Python
Reports
C#P
JavaB
Python.in us
PythonTransformed
md
Benchmarker.Functions_report.csv
Benchmarker.Functions_report.png
img.png
Functions_report.csv.png
597x636 PNG (32-bit color) 56.27 kB
C#
Score
ns/unit
4 × 10²
3 × 10²
2 × 10²
10²
66.59
114.93
344.70
403.68
405.28
432.23
Stack_100
NoRecursion_100
Stack_1000
FunctionalProgramingRecursion_100
ClasicalRecursion_100
Heap_100
Benchmarker.Functions-report

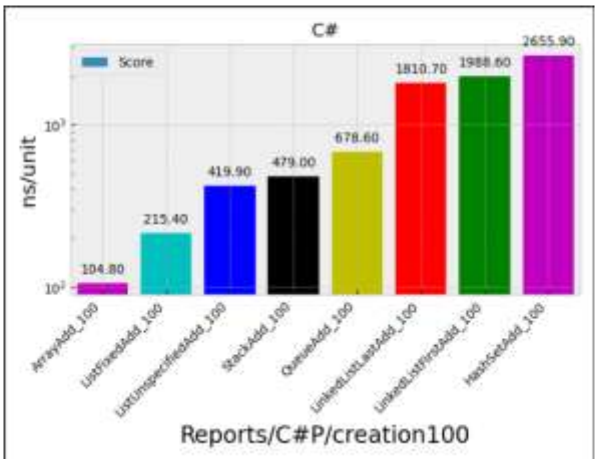
This is how the program is structured and how the results might look .

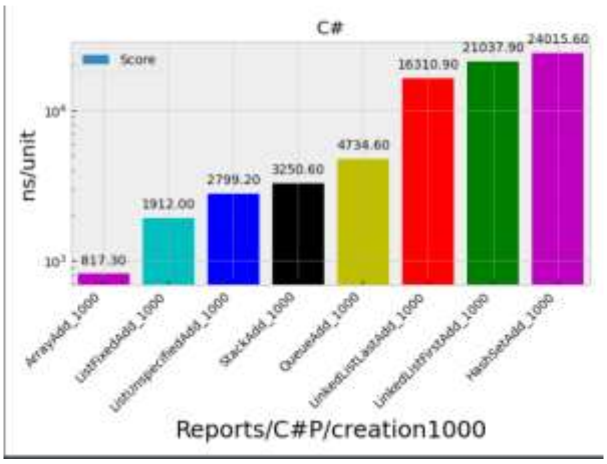
- C#

❖ Containers

Adding an element

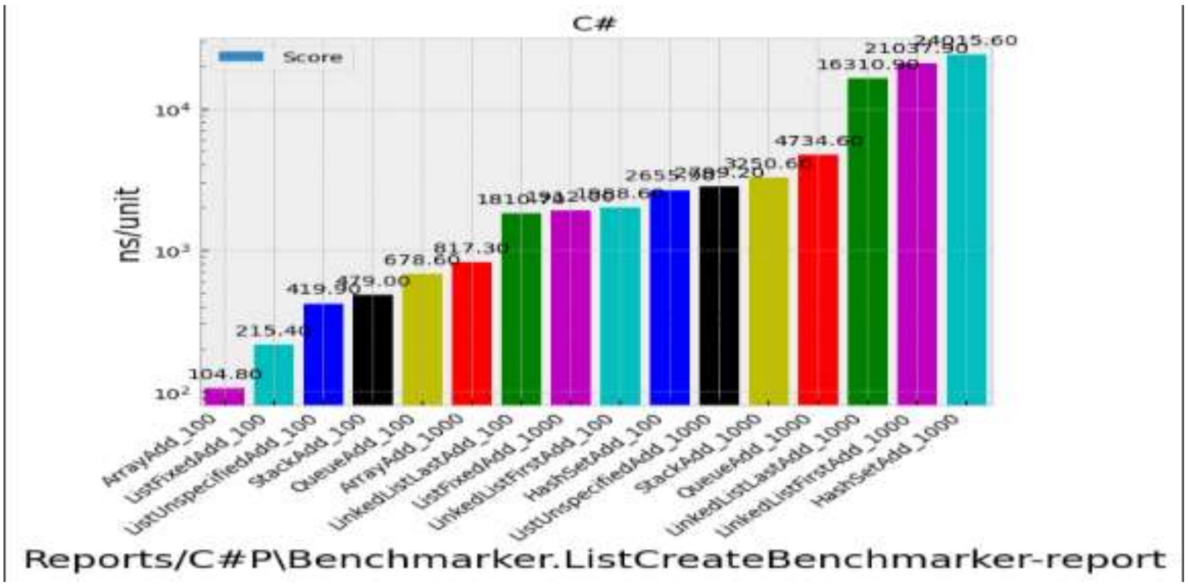
Here , besides the classical containers that were tested, it was followed to see the difference between a list with a fixed length(*list[100]*) , and a list with unspecified length(*list[])* .





As we can observe by far the most efficient one , was the array , where simply a value is atributed to a poiter in the memory. Folowed by the list , where as aspected the fixed size list is faster than the unspecified one , that is because of the mechanism that reserves memory for the unspecified one, it start with a fixed threshold(eg 4 elements) and each time this threashlod is reached the value is doubled, thus beeig more expensive in term of time because of that contignuous realocation of memorry . The most expensive one beeing the hashSet , because it has to hash each elemente before adressing it to a certain position in tha memory .

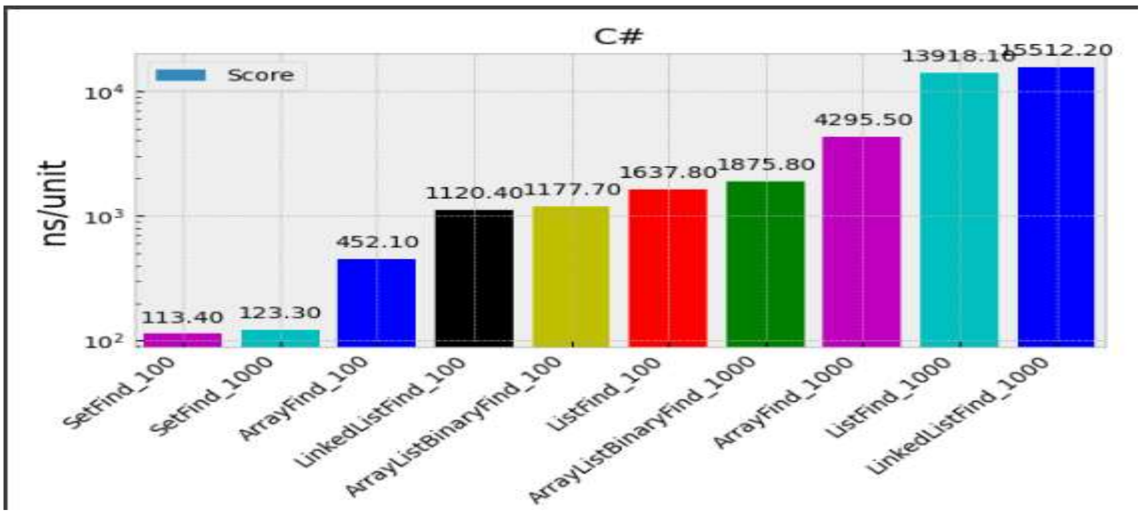
Bellow It is represented a comparison between the run with 100 elements and the run with 1000 elements.



Again we can see that the normal array is the fastest in term of speed , even the 1000 array is faster that the regular linked list with 100 elements.

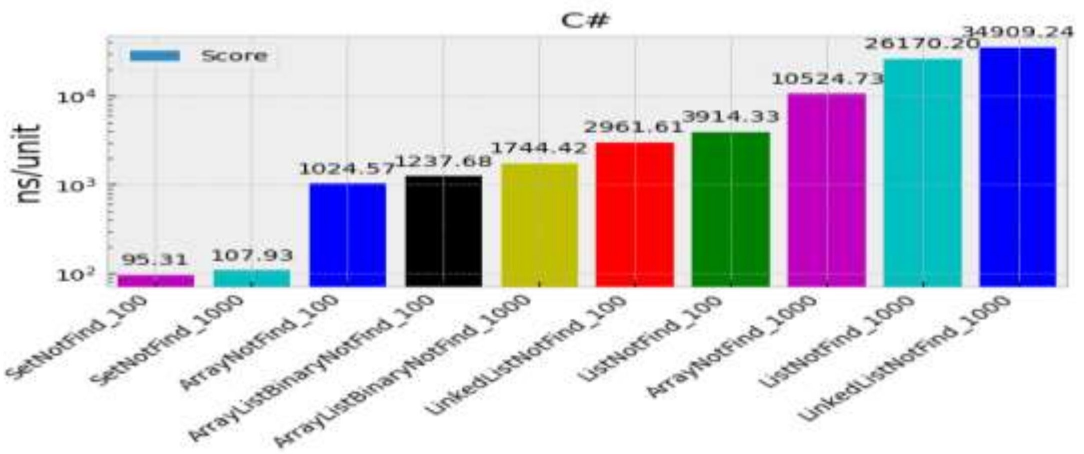
Find test

For this test 10 elements were generated at random , and searched for in all of the predefined containers , with size varying from 100 elements container to 1000.



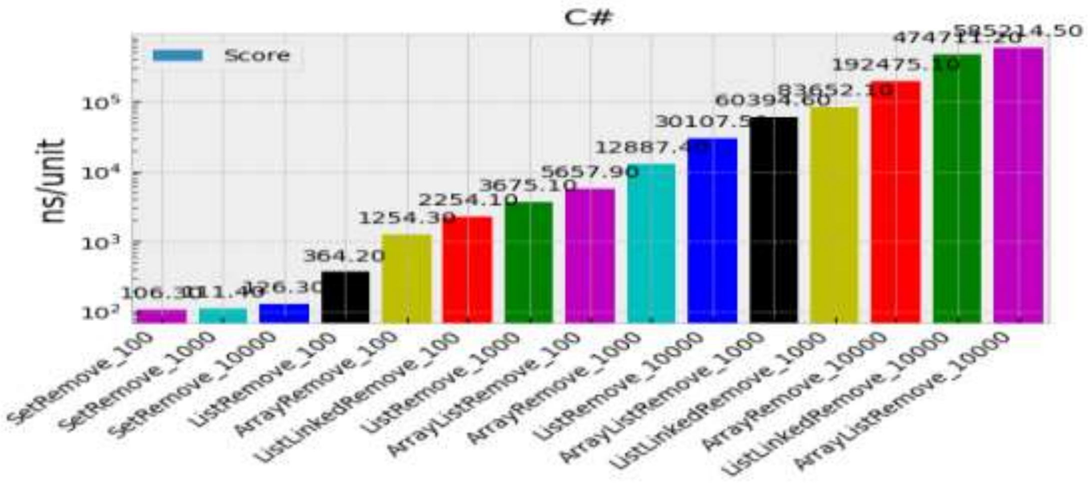
The most efficient one , by far , in terms of finding 10 elements , is the Hashset , we can see that an element in a 1000 element Hashset , is found **3.5 times** faster than an element in an array of only 100 elements. The advantage of the Arraylist(a type in c#) is that we can use a binarySearch function that is predefined in the container , but even with it it came a lot slower that the Hash. Even though in a small number of elements the binary search came below the array of the linked list , as the number increases , We can clearly see that time is much smaller in comparison with the other containters.

Not Find test



Again the performance of the hash set is clear from the graph , even more so , the time between the find search and the not find search is almost the same in the case of the hashSet , being at 1000 elements **, 10 times** faster that the closes to it in terms of , array.

Removal of an element test



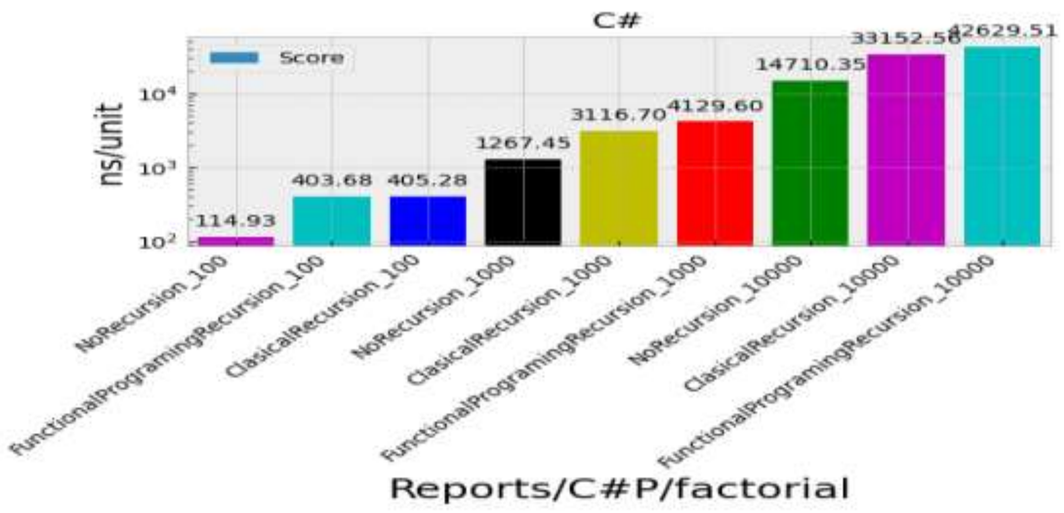
Again in terms of efficiency , the hash set is by far the fastest , and to emphasize this once more , this test was also performed with 10.000 size containers , and even so the has at 10.000 was **3 times** faster than the closest to it.

Conclusion , in terms of efficiency , the hashSet was the most expensive one to created, but the retrieval time of an element that is constant provides a huge boost in performance , sometimes even 10 times faster than the closest one to it in terms of speed .

- Function

Recursive Testing

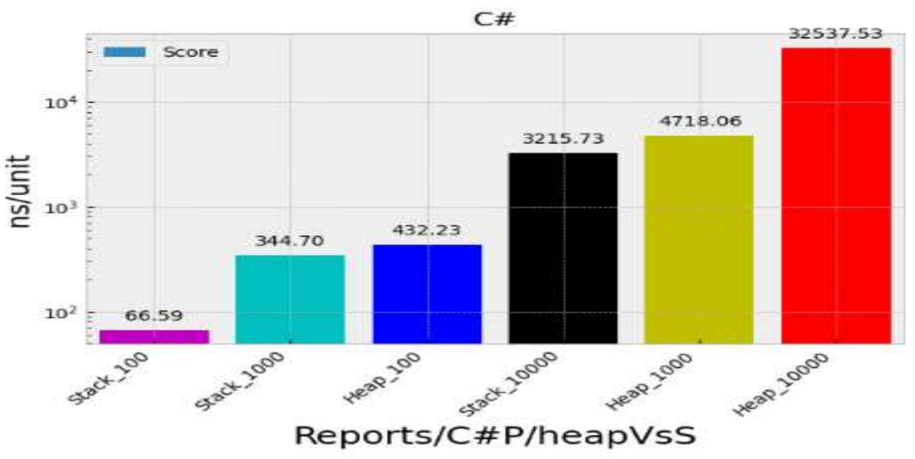
Here as a test , I took the factorial function and implemented it in 3 different methods, with no recursion , with recursion , and with lambda function for the numbers of 100 1000 and 10000



As it can be observed the most efficient is the normal one because it does not have the added cost of the recursive call , that the other two methods have . And in terms of optimizations the functional programming style with lamba is faster only if the number of recursive calls is not so big .

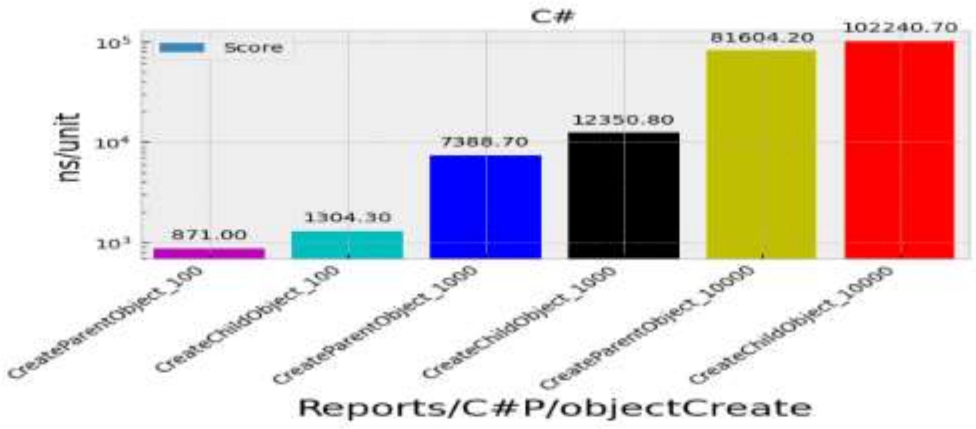
Sack Vs Heap

For this test we incremented a global value a number of times(100 , 1000 , 10000) and modified an object which is stored in the heap .



- OOP

Create

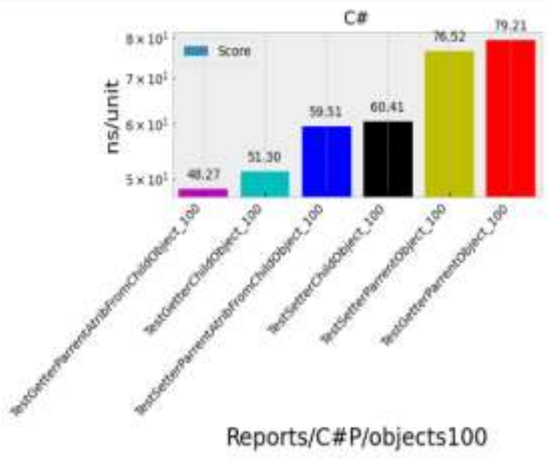


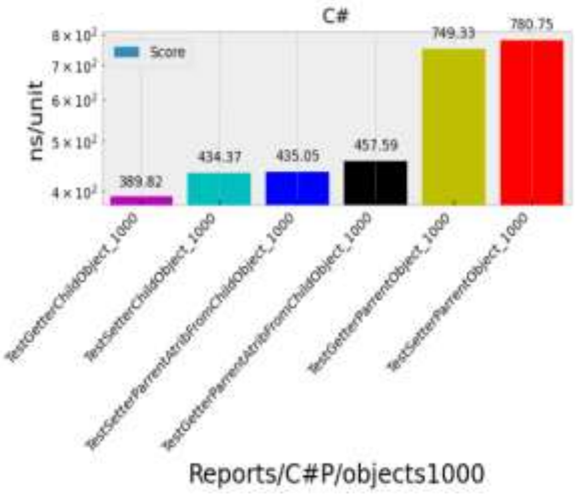
The creation of the parent is faster than that of the child, but the difference is very small.

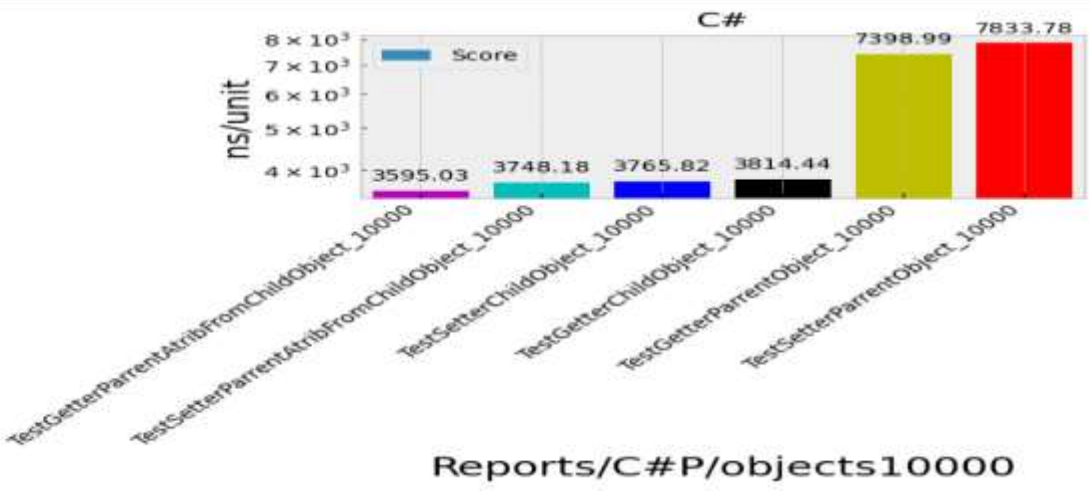
Get and Set

Here a parent class was created (vehicle) and a class Car that inherits it . The aim was to test how well does C# behave in a regular OOP scenario.

So bellow we tested the getter and setter methods, for a repeated call of N(100, 1000 , 10000)



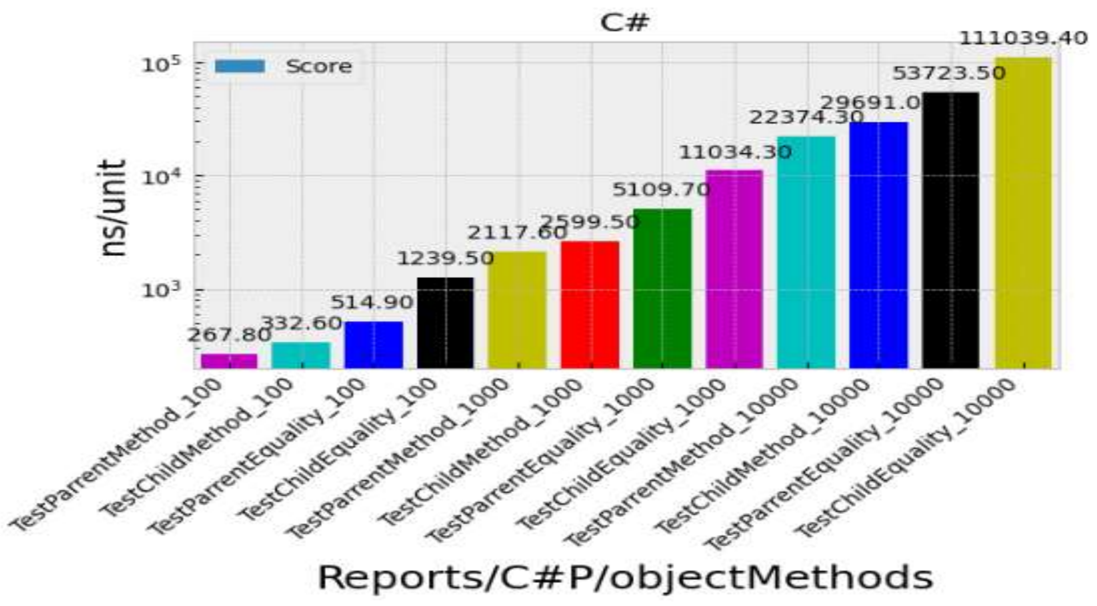




The time increases laniary with the number of calls to the function , keeping the result in the same order , the parent access being the most expensive.

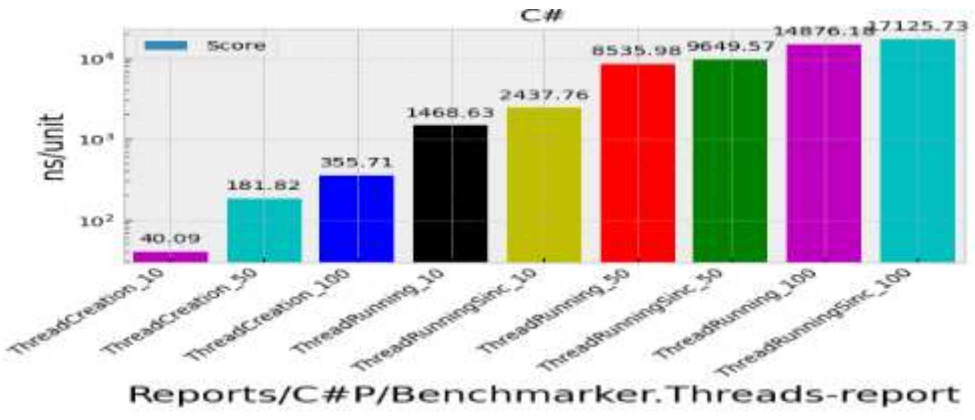
Methods

Here , a simple method was defined both for the children and the parent and it was called in different ways , from parent from child and from one another , and the equality method of the class.



❖ Thread

For testing the threads , a basic class was created called Department , witch has a variable called salary, witch will constitute our race condition and a basic function called withdraw , which decrements this salary. Here to observe the execution time , a number of thread were created(10 50 100) and were given the withdraw method to run , in a synchronous scenario(with a lock in place to facilitate the context switch) and asynchronously.

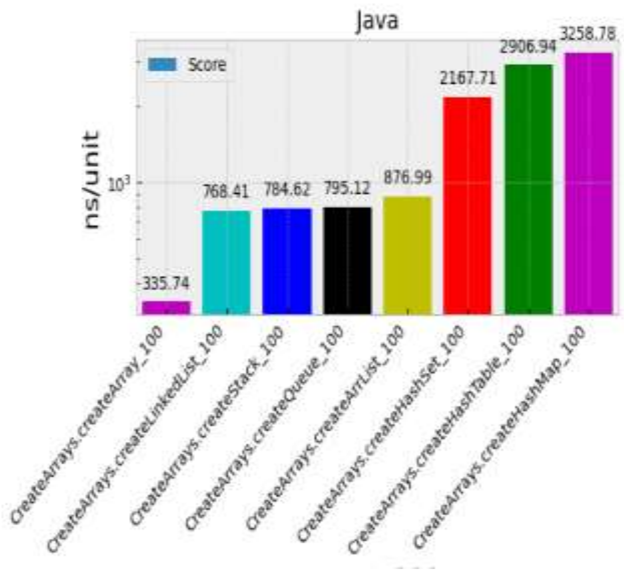


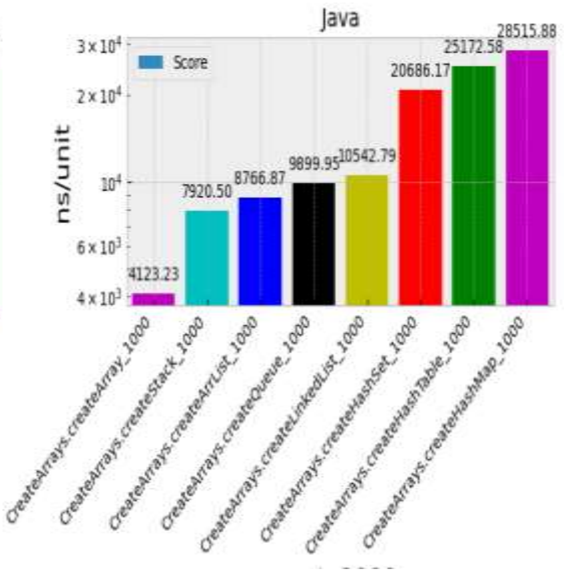
As expected the running synchronous were slower , than the normal ones, but the lock was necessary to avoid the race condition witch will generate a wrong result .

- Java

❖ Containers

Create



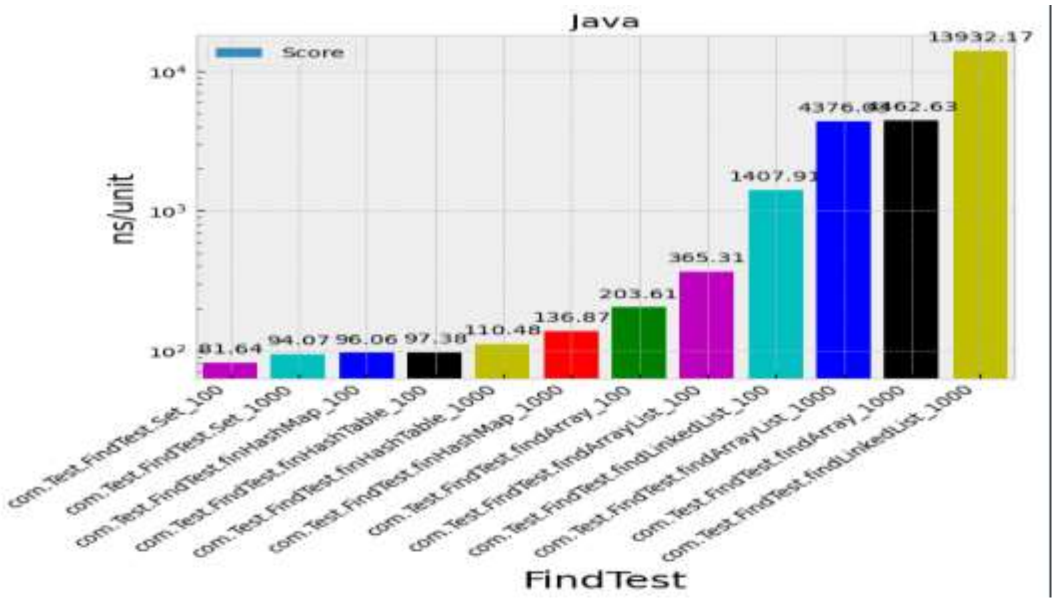


Again here the test is the same , to see how much It will take to create and and 100/1000 elements in a container. Again the simple array is the fastest , and the hashTable the slowest.

Find

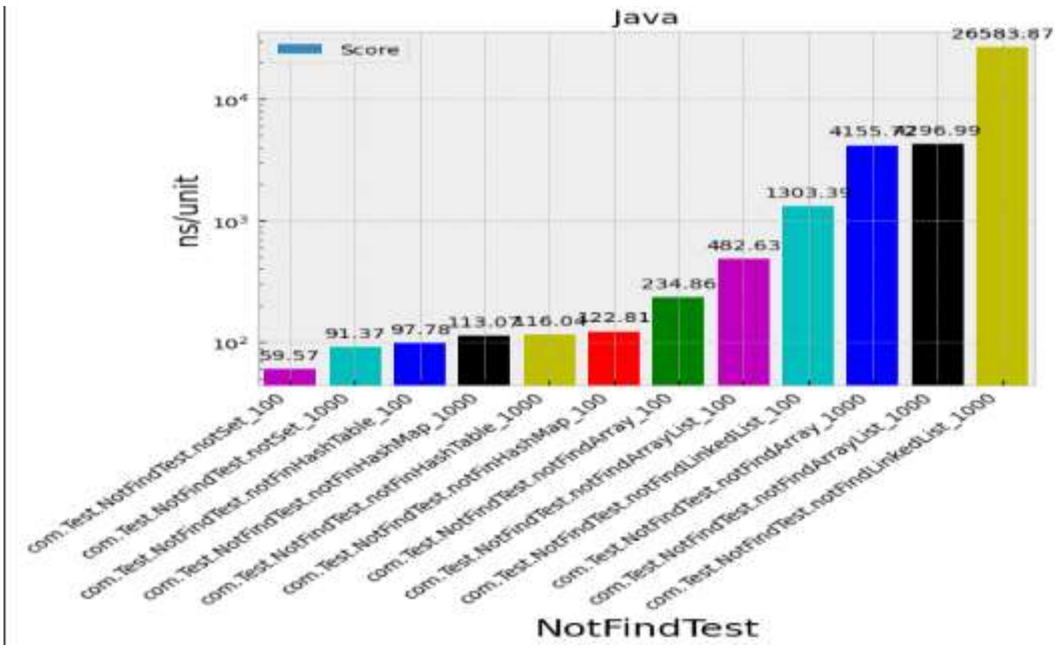
An array of random elements was generated and than , each element was searched in the respective container. The hashSet and HashMap , are constructed on the same data container , the hash table , but the set also ensures unicity, so we wanted to see how they behave when put to comparison.

The find function time , is the smallest by far ,when it comes to the HashTable containers , the slowest it hashTable was **the hashMap of 1000 elements** even so it was **2 times faster** than the finding time the array , and **100 times faster** than the fining in the linked list.



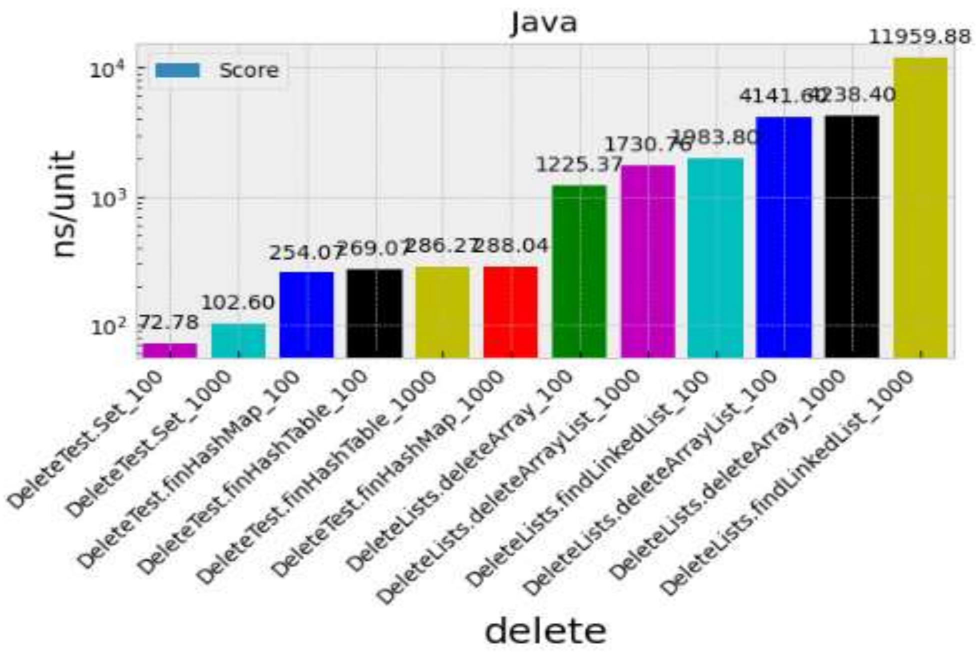
Not Find

Here a set of 10 random numbers that were not in the containers was generated, and then each number was searched in the containers .Again the most efficient one was the set containers, having the search time in the worst case scenarios very similar with the find case time , being constant. The worst again is the linked list , where the time is **almost twice as long as in the find case** and **100 times worse** than the worst hashTable Time.



Delete

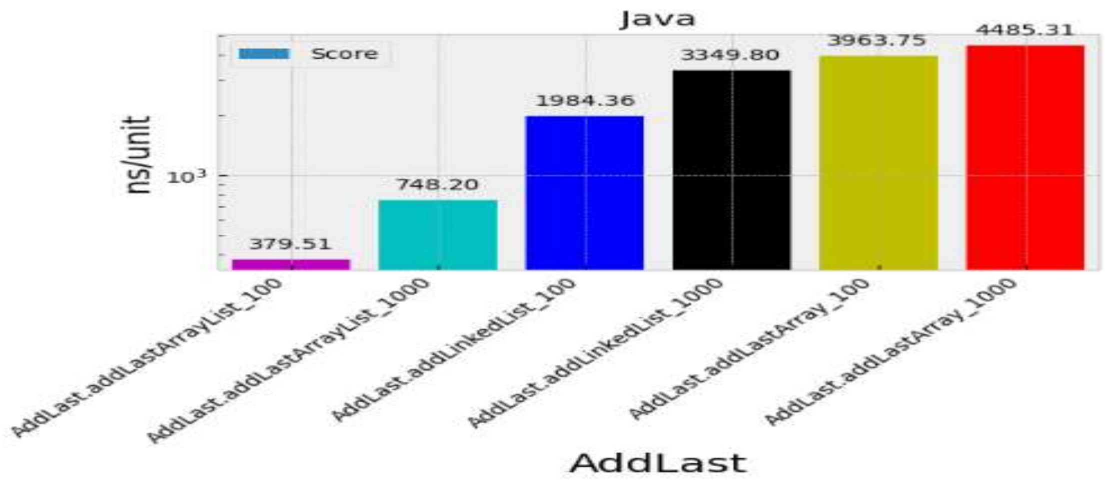
The situation is very similar with the other cases, the hashTable being the fastest by far , while the linkedList is the most expensive one in terms of time.



Add Last

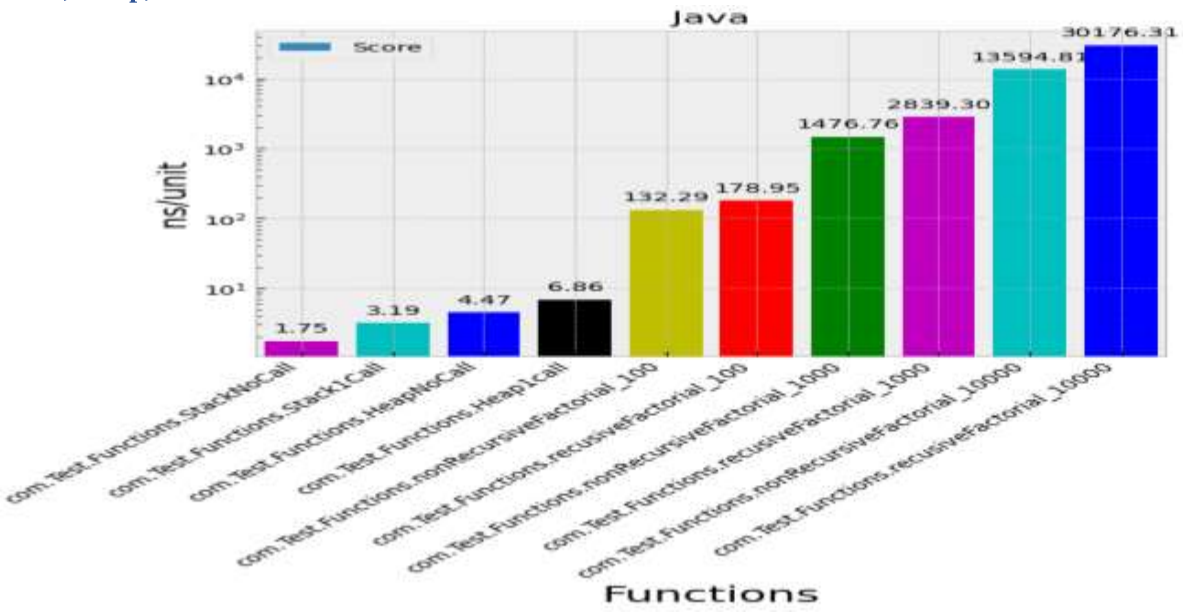
Here were tested the linkedList, the array , and the arrayList to see thee cost of adding N elements to the end of the container.

The best ones are the lists since they have optimizations in place to add at the end of the container an element, whereas , in the array case , the array must be extended with one element , and all the elements copied in the new array before adding the new element.



- Functions

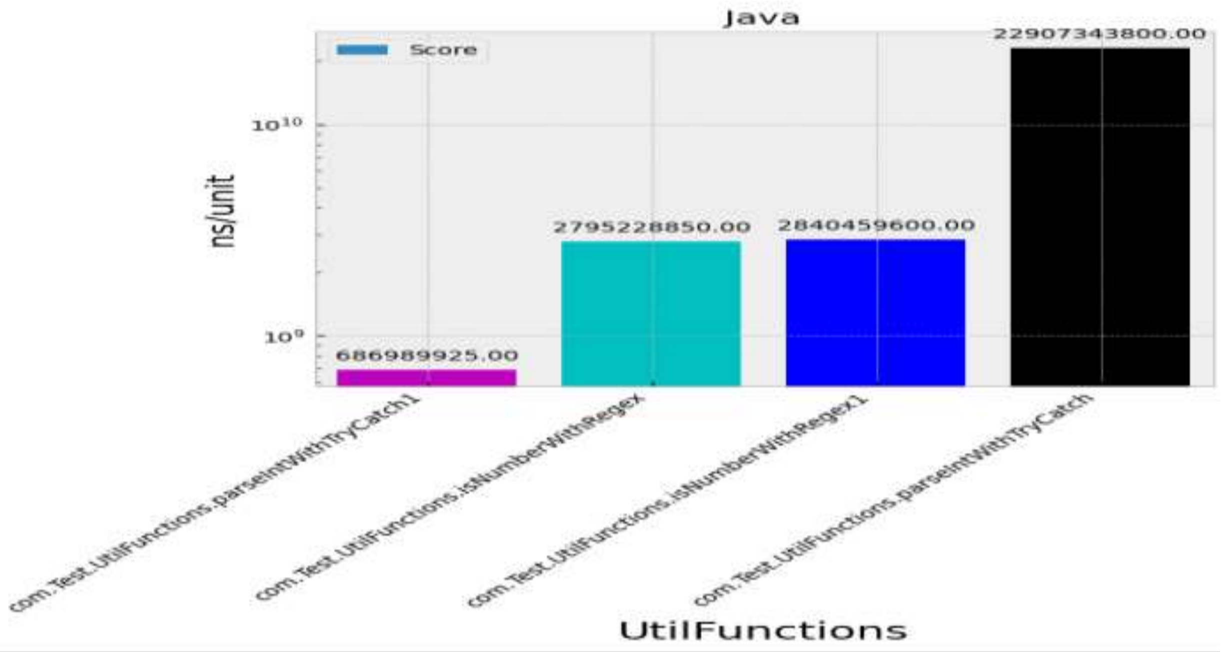
Stack , Heap, Recursion



Here again it was tested the heap Vs stack time , and the factorial implemented in different ways, recursive and non recursive . The heap was tested with no call of a function, simply by incrementing a global value , and with one function call to increment the same value . The heap was tested in the same way with an integer object. Again the results are as expected , the stack is faster than the heap , and the non recursive version is faster in all the cases than the recursive one .

Specific Function test.

Here we followed 2 different parsing methond, one with a try catch to see if the input is a number and the other one is with regex . The input value(1 is a number and empty means that it was not a number) shows us that the tryCatch method is extremely fast if we know that most of the time the value will be indeed a number (look TryCatch1) was **4 times faster** , but for the case where the input was not a number the time **is 10 times larger** that in the regex case . We can see the regex execution time is pretty much constant no matter the input.

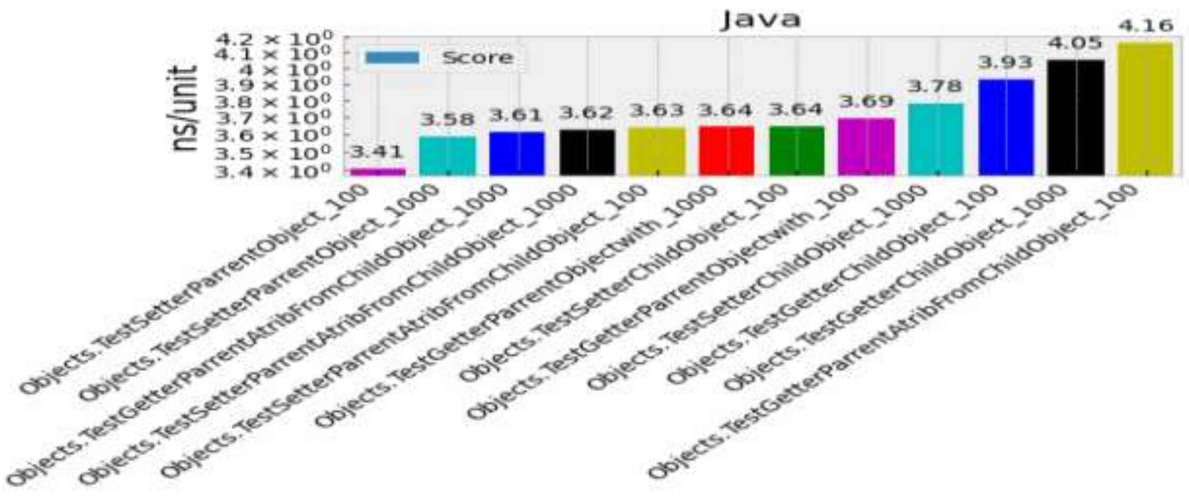


❖ OOP

Here a parent class was created (vehicle) and a class Car that inherits it . The aim was to test how well does C# behave in a regular OOP scenario.

Setter and getter

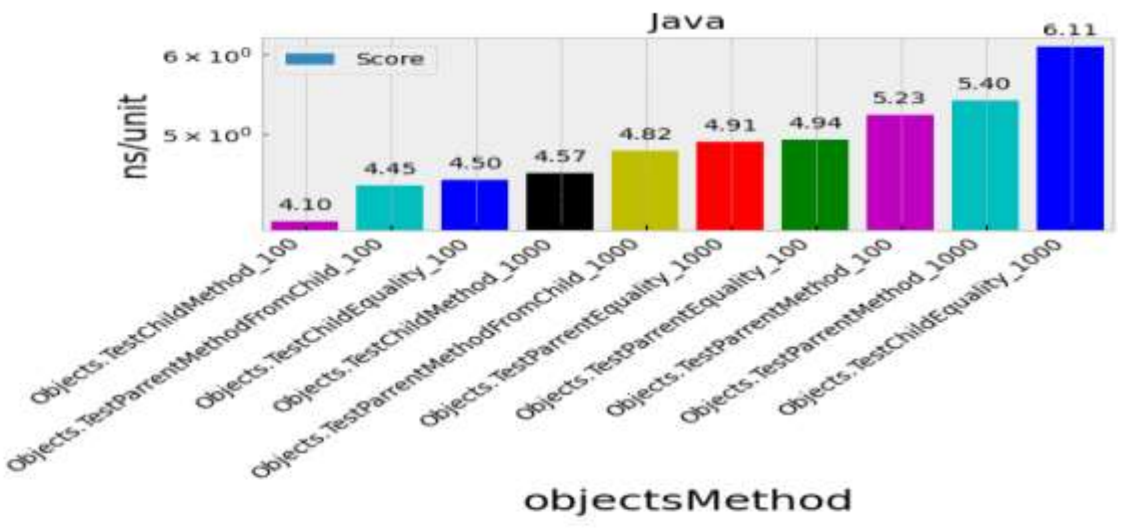
So bellow we teste the getter and setter methods, for a repeated call of N(100, 1000)



We can see that the time is very small, the while graph has a vary small variation from 3.41 to 4.16 ns , so the member access is very efficient .

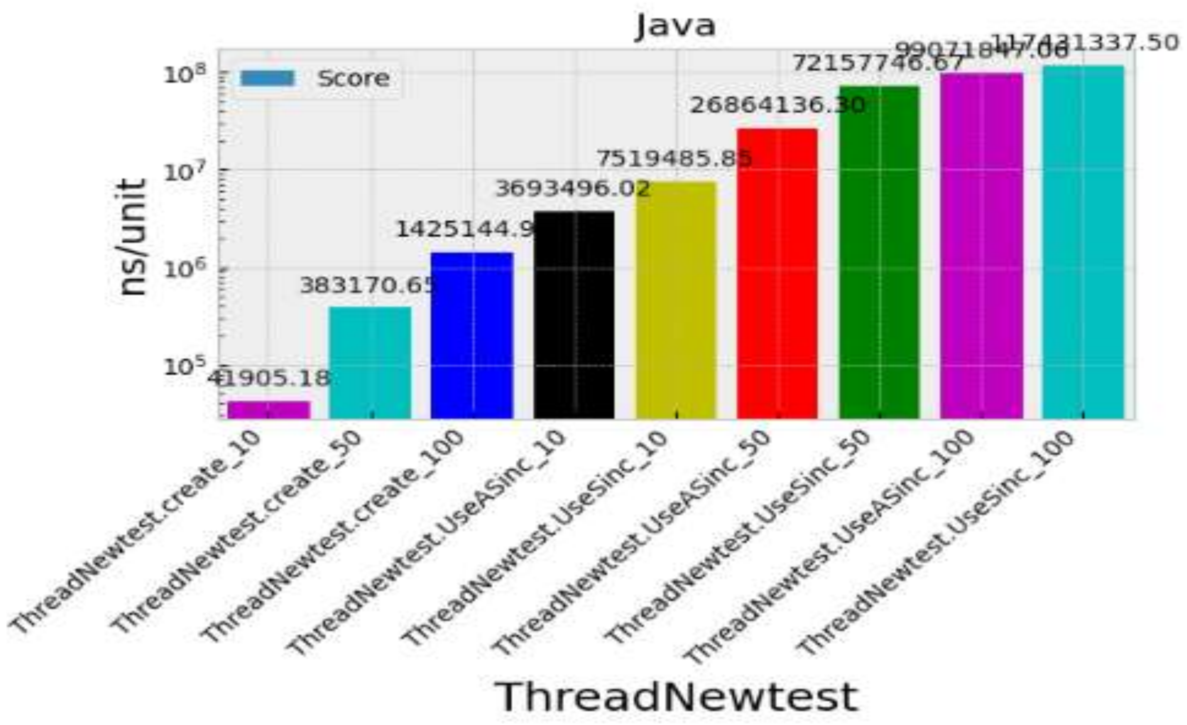
Methods

Here we have the use of some basic methods . including the use of the equality method.



❖ Threads

Here again the same test scenario was designed, with a simple thread creation , and a use case with a racing condition , having synchronization and no synchronization, between the threads .



As it can be seen because of the context switch the , the sync version is always slower.

Python

Python is odd one in this comparison, being first of all a dynamic typed language , meaning that each variable is treated as an object , its type being determined by the compiler only at run time . What it does is, It stores that value at some memory location and then binds that variable name to that memory container. Another important difference between it and the other two languages presented in this project, is the fact that python code is run through an interpreter written in C (Cpython) so , the python code is not run directly . This being said , the python code will prove to be much slower than the other two alternatives , but where python truly shines is its modularity capability , and the speed it provides in writing complex code.

Containers

Python offers only 4 data types : list , tuples, set and dictionary . Anything else is implemented using these 4 units (for example both stacks and queue are implemented with the list container)

Generators

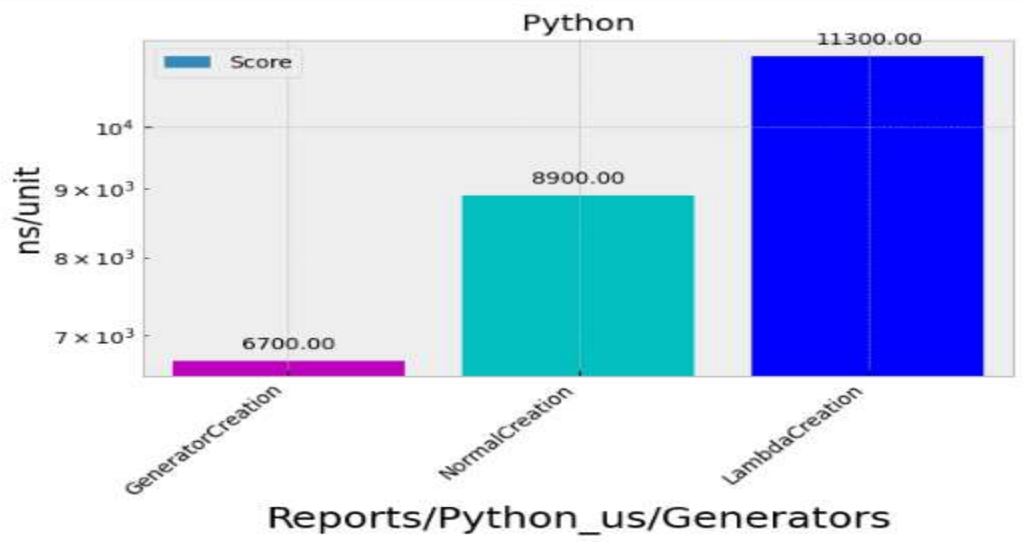
Python offers a great tool for generating a list of elements , called a generator , having the structure

```
list= [ f(n) for n in origArr]
```

.This basically creates an array of length =size , where to each element from *origArr* is applied the function *f*. For example if we want to generate the first 10 perfect squares we cand simply write :

```
list= [ n*n for n in range(10)]
```

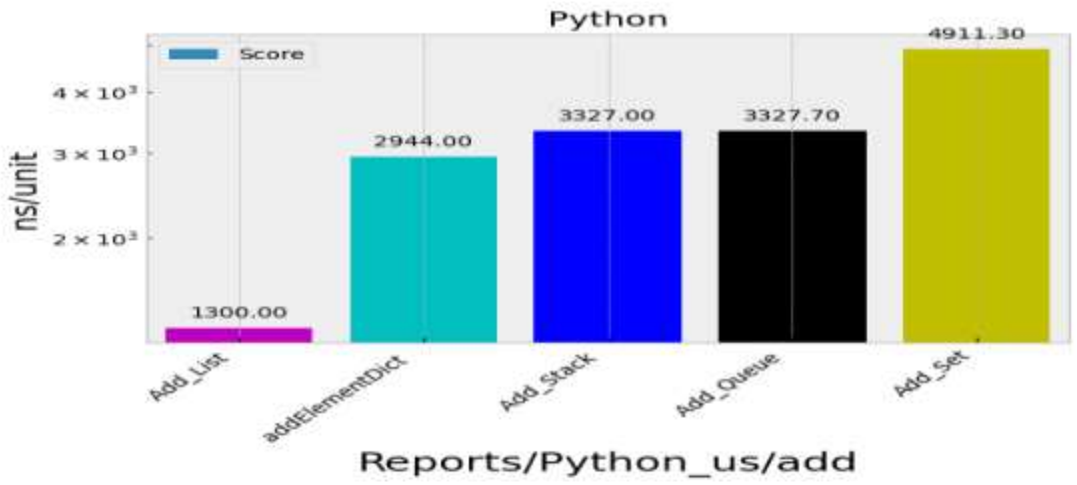
So for this test we wanted to generate a list of 100 numbers , by the rule that *x = x.index*2* and we did so in 3 different ways , the regular way where we go with a *for loop* through the numbers and add to the list i*2 . A generator was used as described above, and a functional programming stile was tested , with the map function and a lambda variable over a list of 100 elemens.



We can see that the method using the generator is the fastest , followed by the normal creation , and lastly the slowest was the lambda function use .

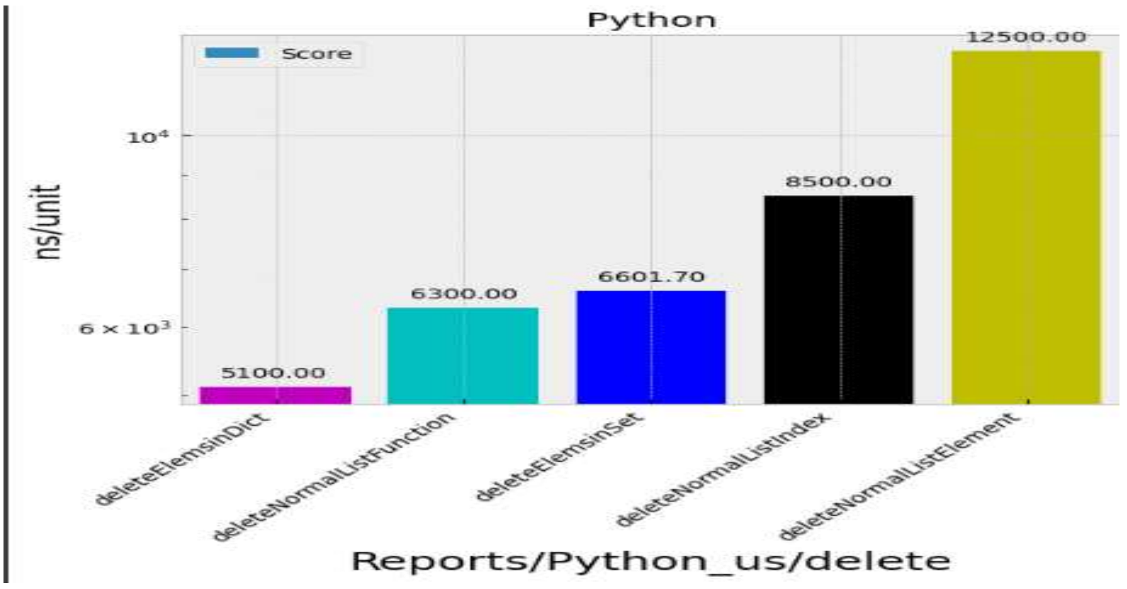
Adding Element

The regular list was the fastest by a large margin , followed by the dictionary. As we can see since the stack and queue are implemented the same way , their time is almost identical . The slowest one being of course the set , because it also has to ensure unicity of each element.



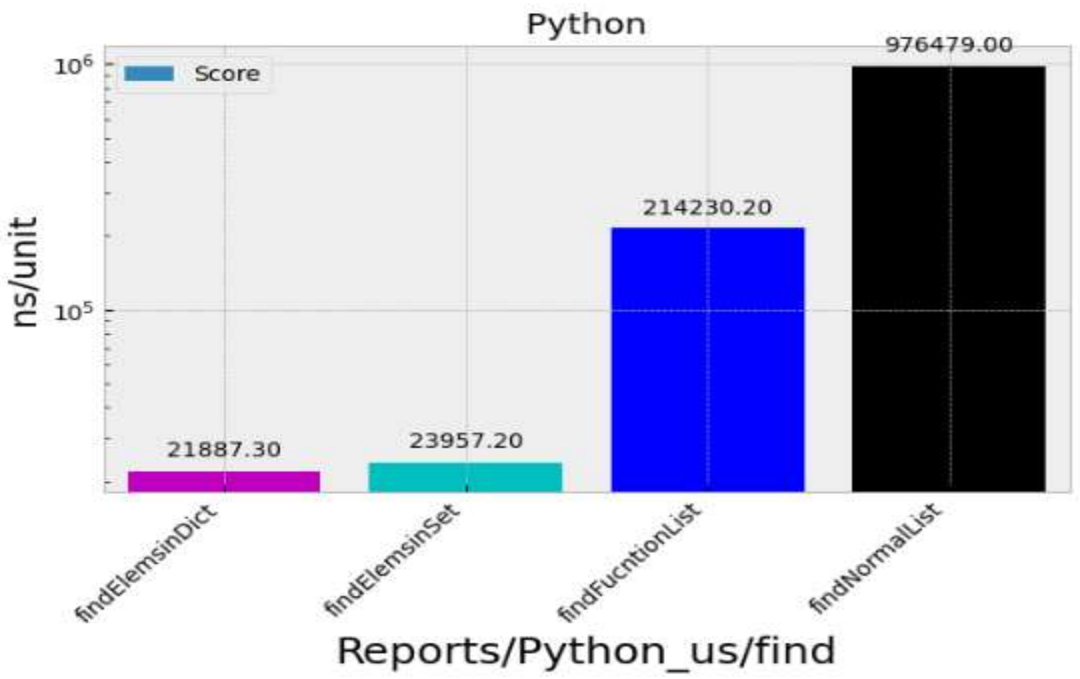
Delete

Here again a set of random numbers was generated and each element was removed in turn from a container. The fastest one was the dictionary , because it acts like a hashTable so finding the element was very fast. Followed by the deleteNormalList function, where the **building delete** method of the list was use. Followed closely by the deletion of the element in a set . The deletion by the index was used **the pop** predefined function, and the slowest was the one where the **removeMethod** of the list was use.



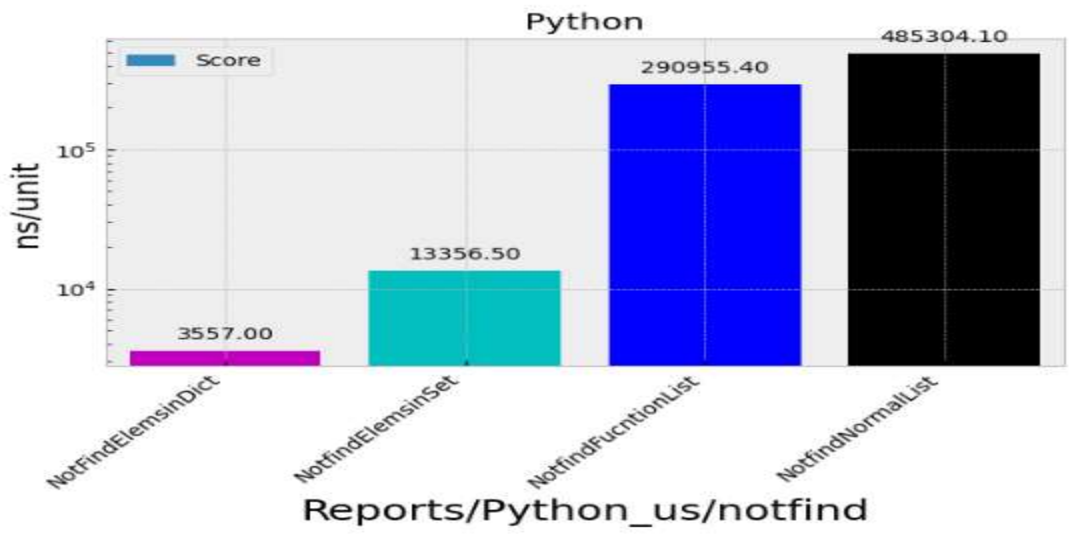
Find

Here the dictionary was the fastest , followed closely by the set , than the list where the build in find function was used , this method being 10 times slower than the set or the dictionary . The slowest by far being the regular search method .



Not Find

The results are very similar , with the added fact that the difference between set and dictionary , is more noticeable now .



❖ Functions .

Here the python introduces a very interesting idea , the namely ***kwargs and args*** . These are special parameters that can be send to a function and modify its behaviour.

The args

parameter tells the function that the number of parameters that are going to be passed is unknows , so for example if we want to create a function that concatenates a series of string writing concat(*args) as it signature allows us to call this function with as many strings as we want .

In order to try to generate a test for this , we created a fucntin that takes as an argument a list , and for each argument in the list we add it to a sum

We can see that the results are much faster using the build in args , that having to use a list .

The kwargs

argument behaves in a similar manner with the args , but instead of accepting positional arguments it accepts keywors arguments , arguments for which you can check later in the code. For example a function create(**kwargd) , might take any number of keywords arguments example :

create(name=”Andrei”,address=”anywhere”) but also create(name=”Andrei”,age=21) is a valid call to the function . This makes it extremely easy to modify the behavior of a function ,depending on its arguments.

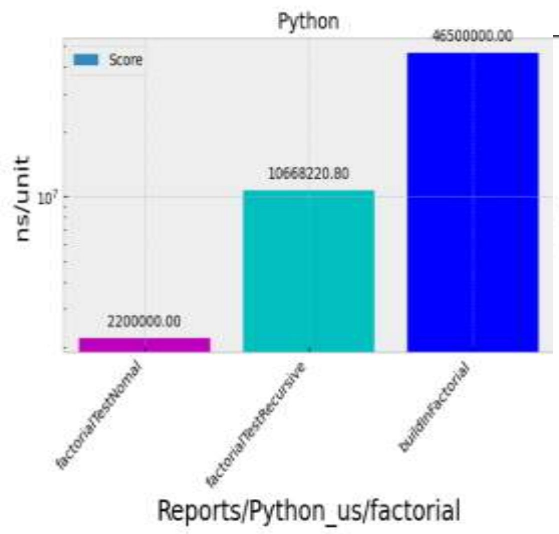
The in operator

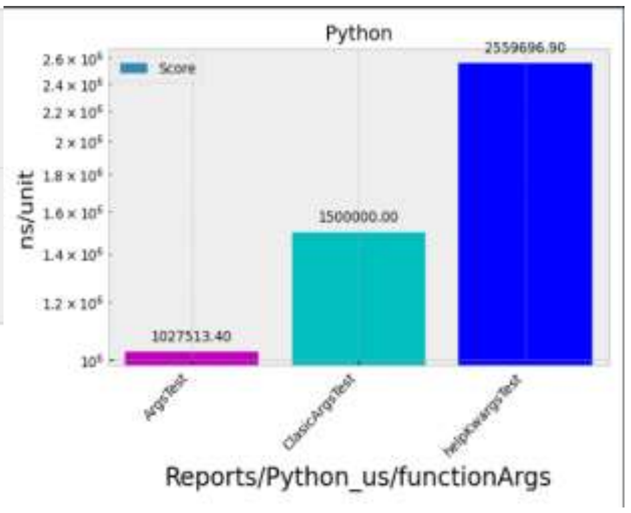
is an extremely powerful tool to sech for any king of element in a list , since the syntax is greatly reduce (x in arr) . Since it works on any list , it can also be used on Strings , making searching for a substring very fast in terms of written code(“arr” in “testing arr func). So we tested the behavior searching for a substring with in and with a regular method(splitting the input in words and looking for the word)

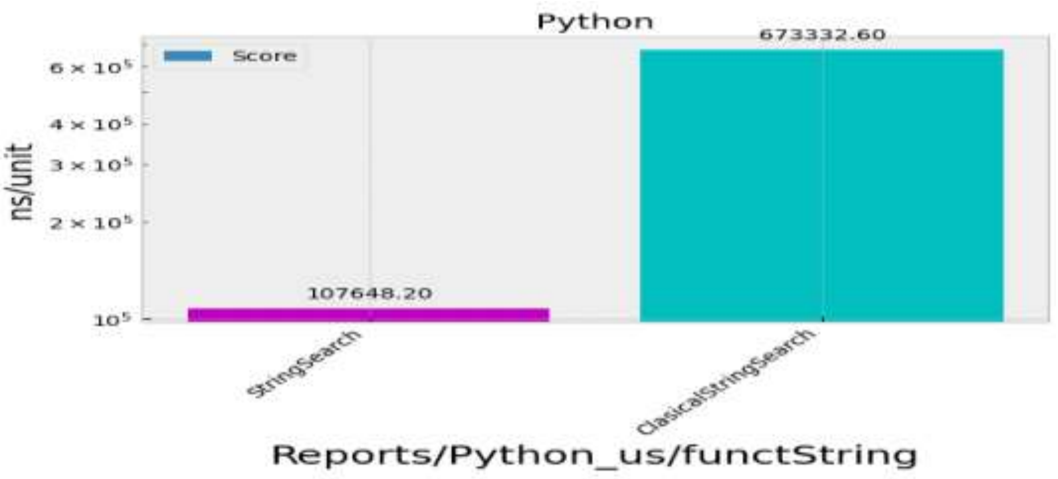
We can see that the in operator is much faster **(6 times)** than the regular sting search.

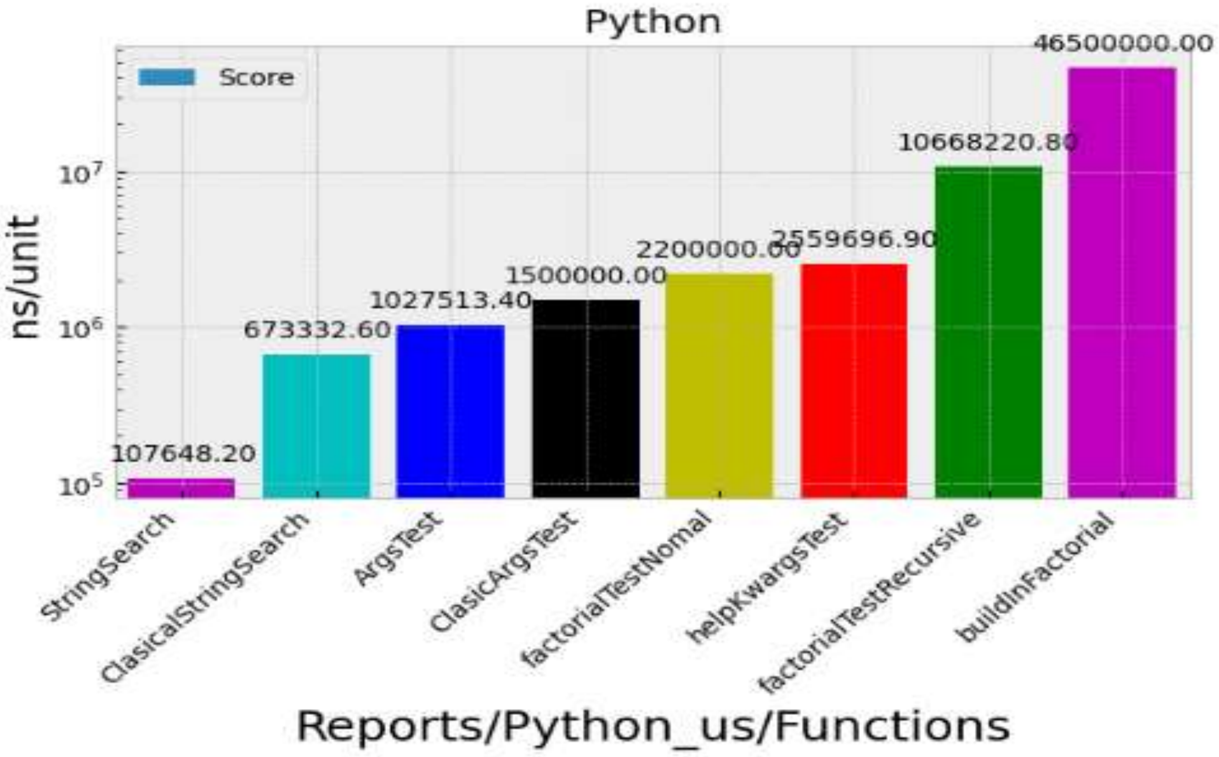
The factorial

Here , the non recursive factorial was put to test against the recursive factorial , and the build in method.



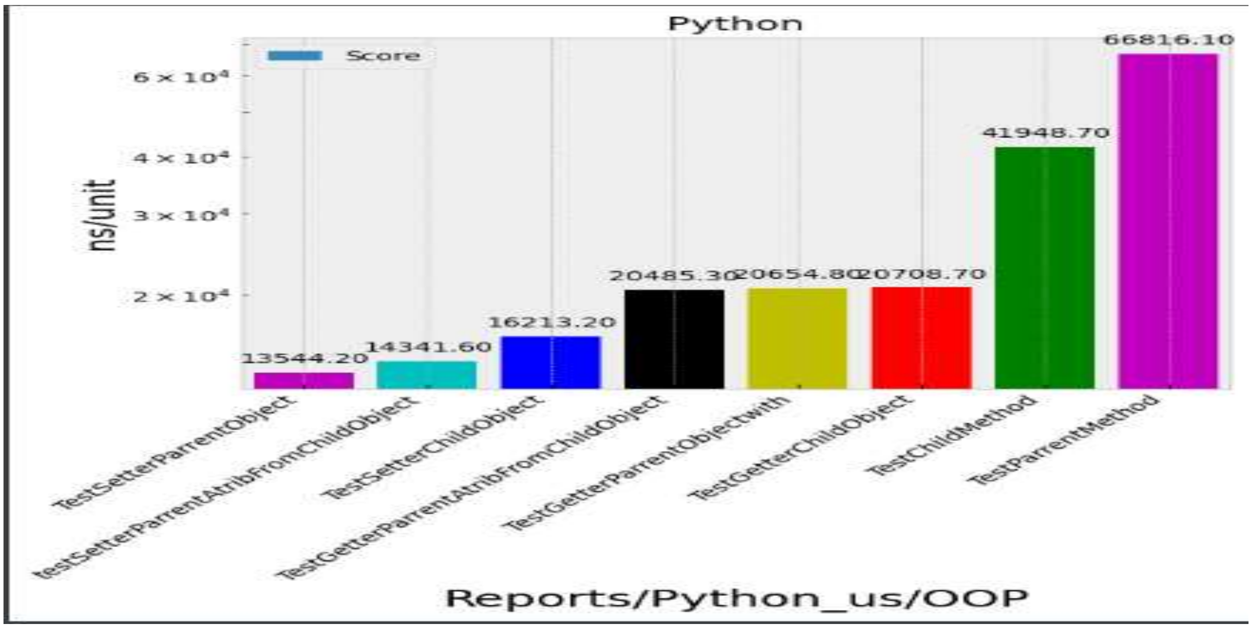



Python
Score
673332.60
107648.20
ns/unit
6 × 10⁵
4 × 10⁵
3 × 10⁵
2 × 10⁵
10⁵
StringSearch
ClasicalStringSearch
Reports/Python_us/functString


Python
Score
ns/unit
10⁷
10⁶
10⁵
107648.20
673332.60
1027513.40
1500000.00
2200000.00
2559696.90
10668220.80
46500000.00
StringSearch
ClasicalStringSearch
ArgsTest
ClasicArgsTest
factorialTestNomal
helpKwargsTest
factorialTestRecursive
buildinFactorial
Reports/Python_us/Functions

❖ OOP

Python has also OOP capabilities. Here for these test I kept the exact structure from the other 2 examples with the car and the vehicle .



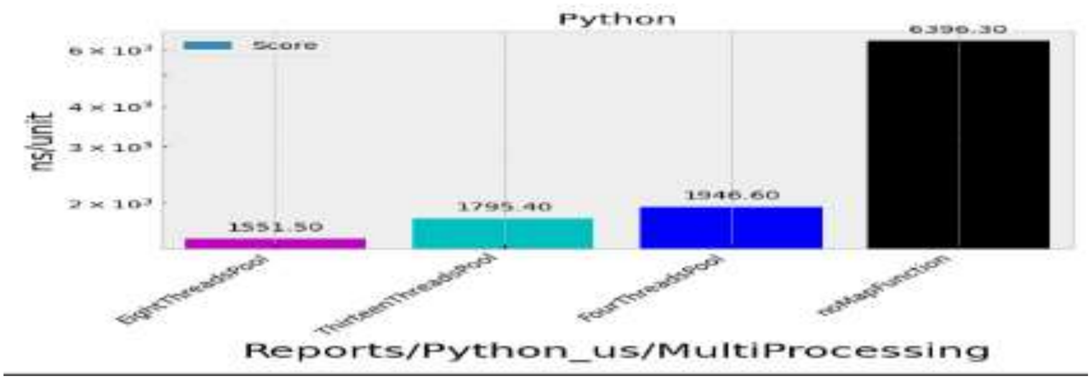
❖ Threading

The main problem with python is that it uses a GIL (Global interpreter lock) which acts as a mutex , not allowing to more than one thread to run inside a process. This lock is necessary mainly because CPython's memory management is not thread-safe.

In order to test some of the threading capabilities, I used a library called *multiprocessing.dummy* which enables us to create a pool of threads each thread being treated as a process that runs on a different core , thus eliminating the GIL problem .

As a test I used the following scenario. We have a list of link and we want to pink each one to see if the website is active or something happened . For that a used a thread pool of 1 (the normal program) , of 4 , 8 and 13 .

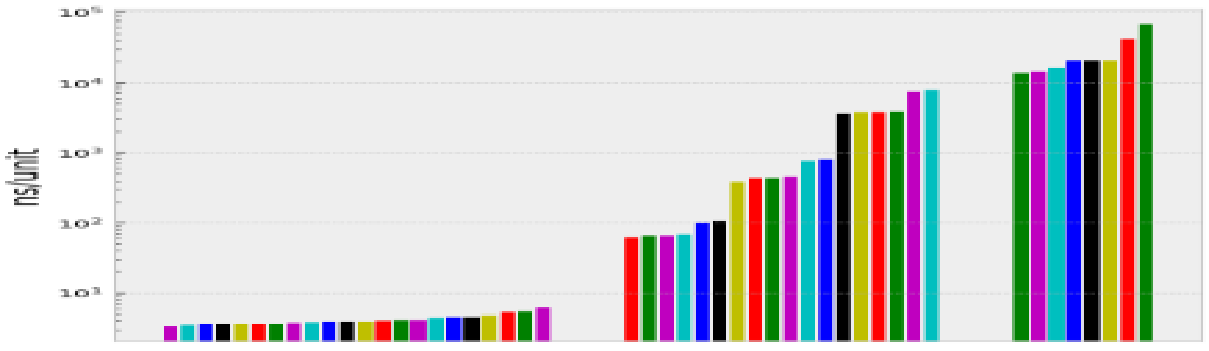
The most efficient one for this scenario will be the 8 pool thread , after that point the time of the execution will increase slightly

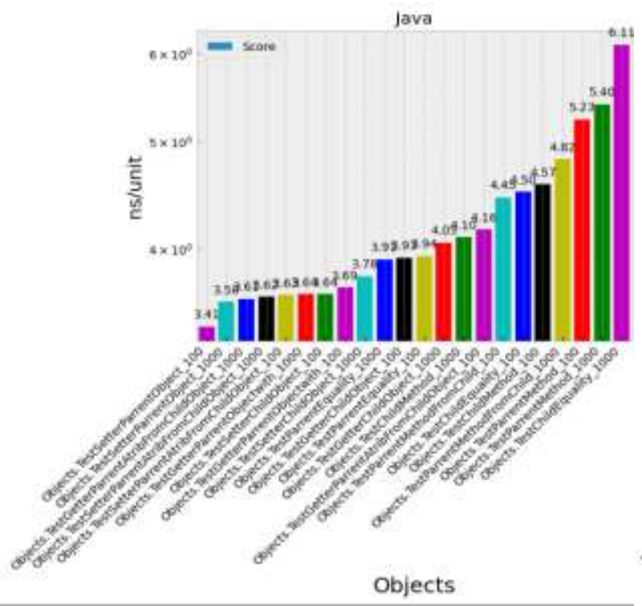


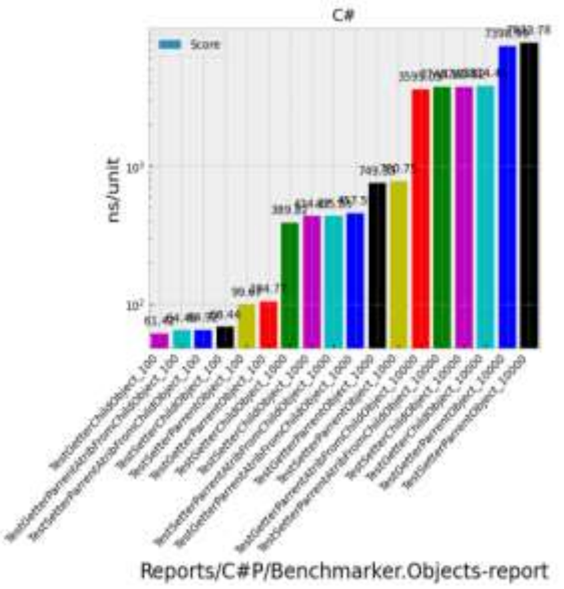
8. Comparison

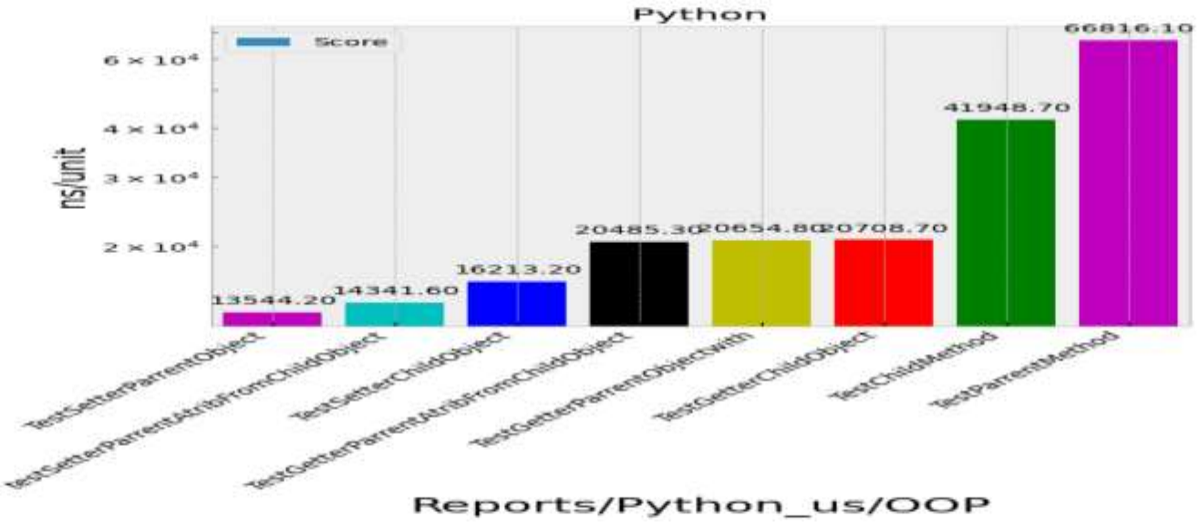
Here the graph will be presented as the Java Graph , followed by the c# and then python

OOP comparison



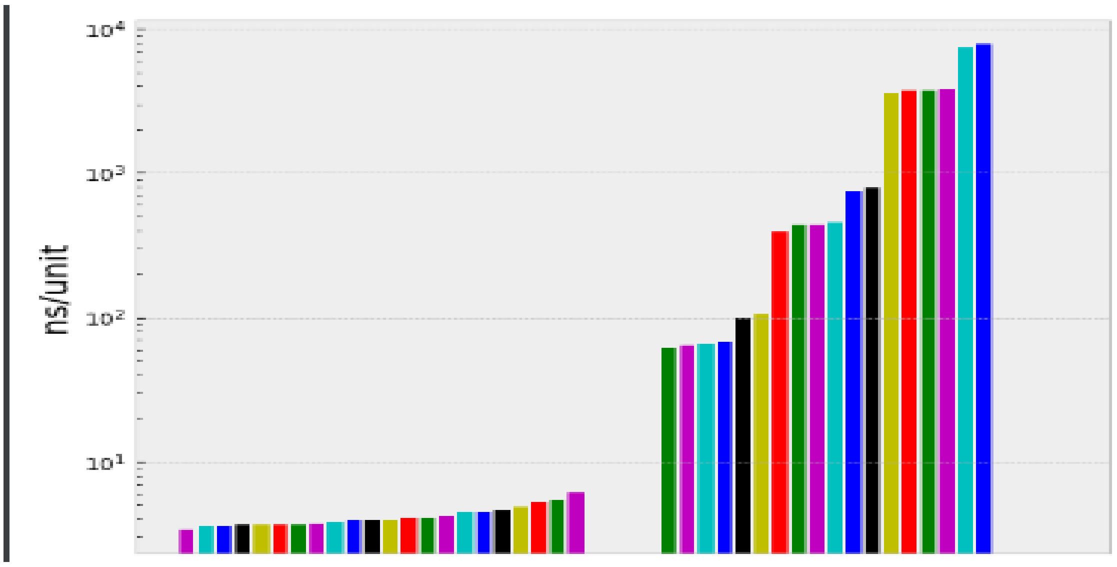






As we can see java is clearly the winner, the most expensive function in the java test suite is 10 times faster than the fastest in C# , Python does not seem much bigger that c# , but it is to keep in mind that the scale is logarithmical so in fact the slowest fucntion in c# is **10 times faster that the fastest in python** .

This graph truly shows the difference between java and c # , Java clearly being the winner .



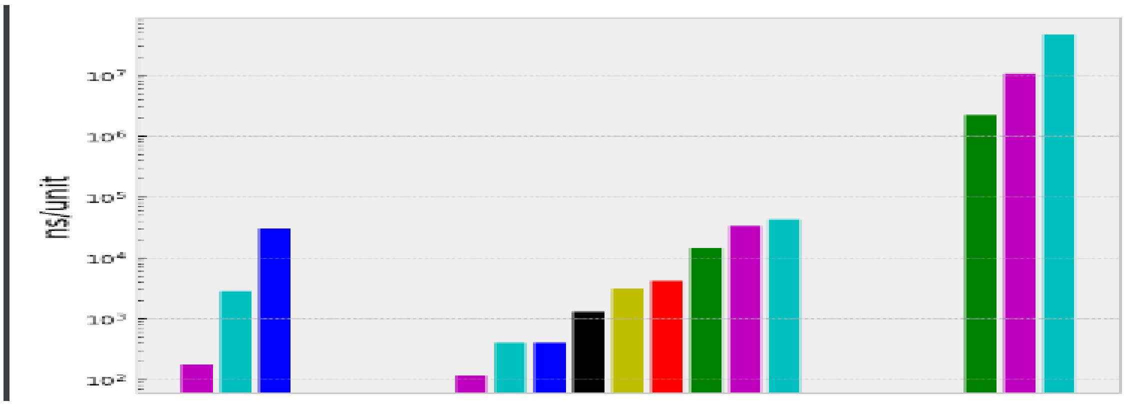
Static vs Dynamic

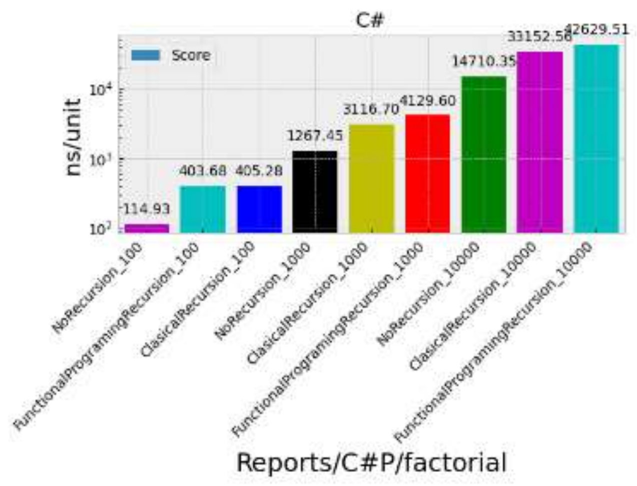
The two static language are Java and c# , Versus the dynamic typed one Python ,

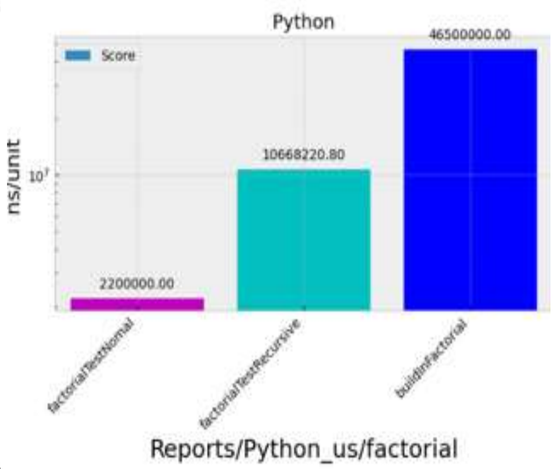
We will look here in the context of containers usage , and the function in general ,

Factorial

As it can be seen java and c # are very close in performance when it comes to computing the factorial , the python remaining the slowest

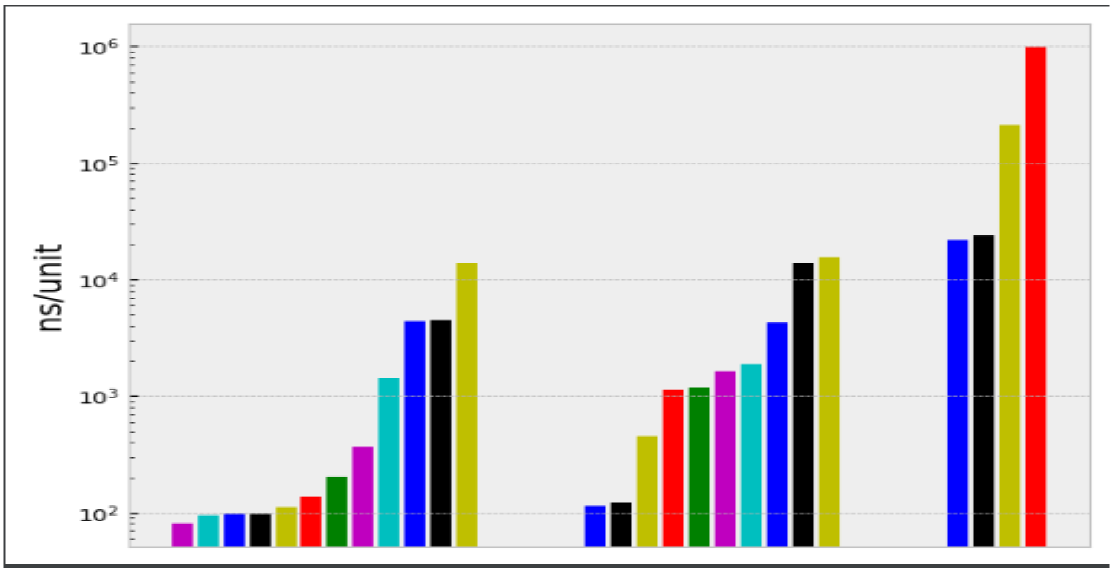






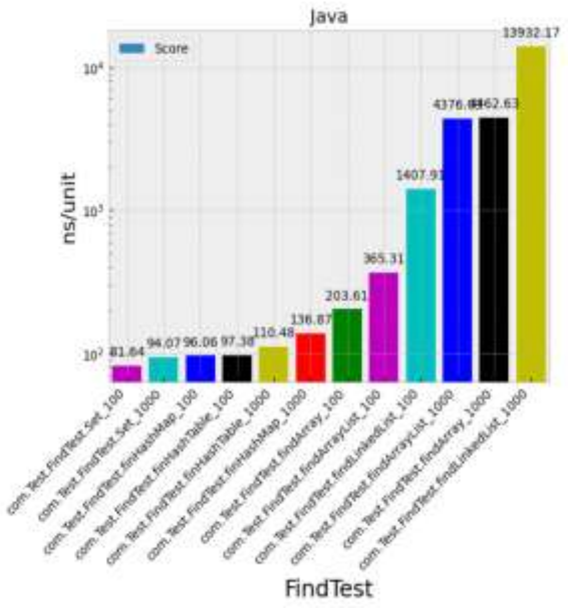
So the difference between the two languages Is from 42.629 to 2.200.000 , so the recursive factorial of 10000 in c# is computed 50 times faster that the factorial of 100 in a non recursive way .

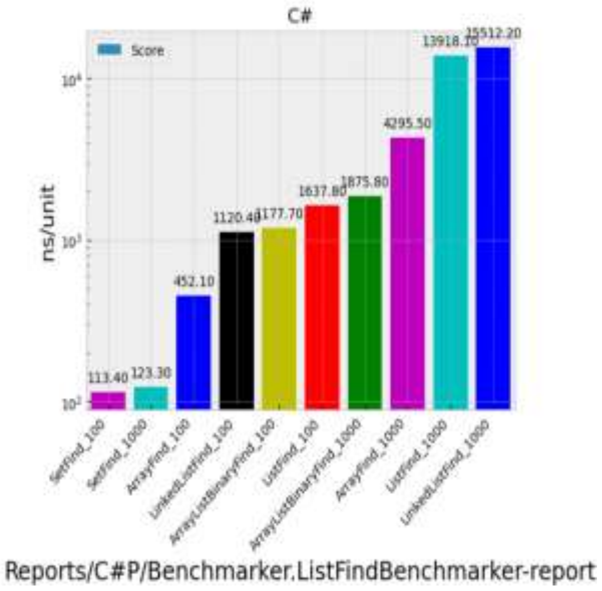
Cotainers Find



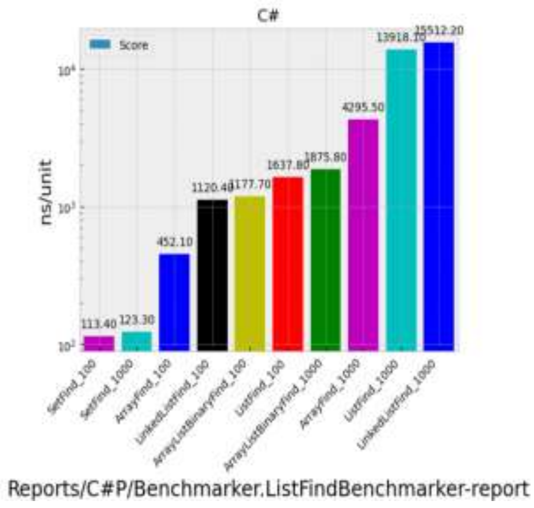
The Java and python are similar in terms of performance, Java though coming on top , and of course python is again the slowest by far

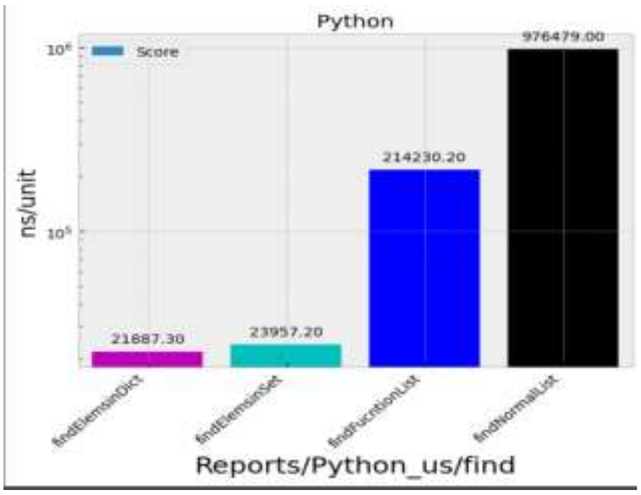
Java vs C#





C# vs Python





Here the difference in efficiency becomes very clear, where the finding time in a hastable in python is 1.3 slower than for looking up an elemenf in a list in c# and 180 times slower than looking up an element in the hashTable of the c# program .

9. Conclusion

Well first of all its clear who the slowest of all is , Python . By a large margin at every test python came up as the slowest . This was to be expected since , python is the only dynamic typed language out of all these 3 , meaning that it has to lose some time during the runtime to determine the type of each variable , everything being threated as an object. Further more since the type is not known the compiler cant make optimization for that data before the run phase. This being said , although python is the slowest of the three , it comes through when it comes to versatility , being use in every filed from web development to network automatization to machine learning. You lose in performance but win a lot in developing speed having many prewritten libraries than make your workflow easier, and has many unique that makes writing the code very easy (eq. list with more than one type [1,"a",[1,2,3]] , the in operator , the functions can return more than one element, the yield that makes function return more elements , list slicing to retrieve elements) .

From the static versus dynamic point of view , as expected the dynamic language(Python) proved to be the slower by far than a regular static typed language.

OOP , from the OOP point of view, Java proved the be the go to option being more efficient than C# at handling classes , and even in general , java proved a bit faster than C#.

So overall Java takes the first place , being the fastest of the 3 , followed by C# and Python being the slowest of them all .

The time measured in this project was before any code optimization to see how the compilers handle the workload, in reality with the optimizations in place Java and C# should have very similar performance , and Python certainly would still be the slowest but not by such a large margin fall

10. Bibliography

1. https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)
2. https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals
3. https://www.javatpoint.com/memory-management-in-java
4. https://stackify.com/python-garbage-collection/
5. https://docs.python.org/3/c-api/memory.html
6. https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html
7. https://stackify.com/python-garbage-collection/
8. https://realpython.com/lessons/dynamic-vs-static/
9. https://www.baeldung.com/java-microbenchmark-harness