Colin Fitzpatrick

Independent Study

May 5, 2021

Comparison of Programming Language Efficiency

Java vs. Python

There are over 250 different coding languages that are notable and used today. Each one of those languages specialize in certain areas of expertise, but each one of these languages have some way to output the same information. So, the question that is being asked in this paper is, which programming languages handles different data structures and sorting algorithms with a better performance? Better performance in this study is defined by a lower average time to do the computation on a simulation of 1000 times and then doing that 50 times to come up with an average. The computations that were chosen for this study were 3 sorting algorithms of selection, bubble, and bucket sort. Another 2 data structures that were also used is the function of inserting and deleting from a linked list and a binary tree. All these sorting algorithms take in an array of data and sort them in numerical order for lowest to highest, but they are done using different formulations and strategies. The selection sort algorithm is a simple algorithm in which the algorithm sorts an array by repeatedly finding the minimum value in the array and putting the value at the beginning of the array and it does this until the array is sorted. The unique process of this algorithm is that selection sort maintains two subarrays in each array. The selection sort algorithm has a time complexity of O(n2), meaning that the more values that an array contains it takes n2 longer for each value in the array. The bubble sort algorithm is the simplest sorting algorithm that repeatedly swaps the two adjacent values in an array if they are in the wrong order. The time complexity of the bubble sort algorithm is also O(n2). The bucket sorting algorithm is by far the most complicated sorting algorithm that was tested in this study. Bucket sort is most useful when input is uniformly distributed over a range, so for example let take the range from 0 to 1. First thing that bucket sort does is that it creates ten empty buckets. Then if the first value that is being sorted is .19 it gets put into bucket 1 because the value starts with a .1 and this is done for each value in the given array. Once all the values are but into their respective buckets then its sorts each bucket using an insertion sort. All insertion sort does is that it iterates over all values within the bucket and compares the current values to is predecessor and if the current values is smaller than its predecessor than they swap places otherwise it does nothing. Once the insertion sort is done within each individual bucket, the buckets are then put back together for the final masterpiece.

Next, the data structures that were used in this experiment were linked lists and binary trees. With these data structures it is important to talk about their insert and delete methods. These methods allow for the adding and removing of nodes in the data structure. Now, there are many ways to add and delete node from data structures. The linked list insert inserts all nodes at the head of the list. This is true for both the java and python code. The remove method is also the same for each program, as the method cycles through each of the list until a certain key value is found then removes from the space. Now for the binary trees the insert method is recursive and follows and ordered tree approach. For example, if the root of the tree contains 50 for its data value, then any number greater than 50 would be inserted to the right and any number less than 50 is inserted to the left. Both the java and python implementations have this same insert method. The deletion method is also recursive in which the method takes in a key value and then searches for that key in the tree. It uses the same logic of the insert method, so it saves time searching for the key within the binary tree.

Now, let us talk about how the average time for a set of variable n instructions was calculated for a certain method, whether it be for the data structures or the sorting algorithms. The method that was written to calculate the average times for both the data structures and for the sorting algorithms all have the same foundation underneath them, so explaining only one of them will suffice. To build the average time calculation for these methods it takes three inputs the number of instructions that the user wants to loop through, how many times does the first initially set of instructions wants to run, and for the sorting algorithms it has an additional input of the length of the array that the user wants to sort. The average time is calculated by using java and pythons internal clock system. This allows for the subtraction from the start to the end of the instructions. Then average calculator then runs that set of n instructions m number of times to after one set of n instructions the system then find the time it took to run those n instructions. After that, then we take that time and store it in an array that will be used for later. Then once the m times is done and there are m values stored in the array. Then the average was calculated out by using the numbers that were stored the array divided by the length of the array. The problem that came up was that java wall clock time is in nanoseconds and python wall clock time is in seconds. In the java programs there is a division by 1,000,000,000 to convert the nanosecond back to seconds. Then average output is displayed along with the array of values that was used to calculate the average.

The results of this test are now listed below. All calculations were performed in the statistical program of SPSS. To check for differences in each area for both the sorting algorithms and the data structures, a difference of means was done to see if java or python has the faster speed when executing these instructions.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Paired Samples Test** | | | | | | | | | |
|  | | Paired Differences | | | | | t | df | Sig. (2-tailed) |
| Mean | Std. Deviation | Std. Error Mean | 95% Confidence Interval of the Difference | |
| Lower | Upper |
| Pair 1 | javaSel - pySel | -.098928974 | .018661844 | .002639183 | -.104232611 | -.093625336 | -37.485 | 49 | .000 |
| Pair 2 | javaBub - pyBub | -.006129213 | .007841022 | .001108888 | -.008357607 | -.003900819 | -5.527 | 49 | .000 |
| Pair 3 | javaBuck - pyBuck | -.045954680 | .008201737 | .001159901 | -.048285587 | -.043623772 | -39.619 | 49 | .000 |
| Pair 4 | javaListInsert - pyListInsert | .164235844 | .099017352 | .014003168 | .136095424 | .192376264 | 11.728 | 49 | .000 |
| Pair 5 | javaListRemove - pyListRemove | -.204665002 | .022980788 | .003249974 | -.211196070 | -.198133934 | -62.974 | 49 | .000 |
| Pair 6 | javaTreeInsert - pyTreeInsert | -.001511499 | .004736373 | .000669824 | -.002857562 | -.000165437 | -2.257 | 49 | .029 |
| Pair 7 | javaTreeRemove - pyTreeRemove | -.000613887 | .003092211 | .000437305 | -.001492684 | .000264910 | -1.404 | 49 | .167 |

After running the test listed above, we can conclude that Python is better when using wall clock time in all categories expect the binary tree removal at the alpha equals .05 level. A reason for this finding is that python wall clock sometimes reports the times at 0 if they are low enough. This is something that could have skewed the results enough to have these results.

In conclusion Python in this study prevailed over Java in terms of time needed to run the code. Also, the time it took to code the everything in Python was much faster than the time in Java. This is because arrays in Python are much easier to work with due to Python’s way of handling arrays. Overall, this independent study shows that even though Python might be faster than Java, but both of these programming are similar enough where it does not matter.