School of Computer Science

Semester I, Academic Session 2022/2023

CPT 316

Programming Language Implementations & Paradigms

Assignment 1

Lecturer: Dr. Nibras Abdullah Ahmed Faqer

Group members:

Name	Matric number
LIM ZI QIANG	153116
KHOO LAY SIN	153534
LOH WAN TENG	149104

Submission Date: 13 Nov 2022

Table of Contents

1. Abstract

There are many programming languages available to tackle computerized issues. This paper emphasizes on comparing Java and Python using the algorithm selected from the three reviewed algorithms from their respective research papers. The paper also provides a brief introduction to the background of Java and Python. With the algorithm implemented in Java and Python, performances were evaluated, and the results were discussed further in this paper.

2. Introduction on Java and Python

Java and Python are two of the most popular programming languages. Each is well-established, platform-independent, and supported by a large community.

2.1. Introduction on Java

Java is an object-oriented, and network-centric high-level programming language that can be used in multiple platforms for coding web and mobile applications. It was first released in 1995, by Sun Microsystems Inc and designed by a team of researchers led by James Gosling. (*Introduction to Java*, 2022)

Java has been a popular choice among developers nowadays as it has multiple inbuilt functions and libraries and there are many learning resources available. Moreover, it speeds up the application development process efficiently by providing tools to support debugging, testing, and deployment.

Java is the first language to combine compilers and interpreters using a Java Virtual Machine (JVM). Different from other programming languages, Java code converts code into Bytecode instead of the instructions for a specific type of computer. Upon the conversion, Java Runtime Environment (JRE) or the Java virtual machine interprets Bytecode and translates it for the host computer. (*Introduction to Java*, 2022)

2.2. Introduction on Python

Python is a widely used general-purpose, high-level programming language. It was first released in 1991, designed by Guido van Rossum, emphasis on code readability. (*History of Python - GeeksforGeeks*, n.d.) With the rise of Data Science and the large standard library supporting Python, it has become the fastest-growing language. (Khoirom et al., 2020)

Python emphasizes on syntax simplicity, and hence, is easier and faster to learn and perform testing compared to Java. (*Python vs. Java: Which Should I Learn? | Coursera*, n.d.) While being an interpreted language where the interpreters execute commands line-by-line, it is significantly slower in execution when compared with other languages and is not preferred for memory comprehensive task. (Khoirom et al., 2020)

3. Review of three string sorting algorithms

Three string sorting algorithms from three research papers: Bubble Sort, Insertion Sort, and Quick Sort are selected for review.

3.1. Bubble Sort

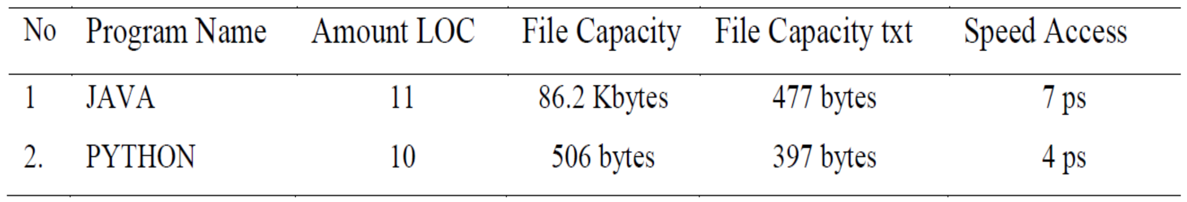
Bubble sort is a comparison-based sorting algorithm in which the sorting process involves comparing data elements next to each other and allocating them in the correct position. (Gill et al., n.d.) Bubble sort has the following properties:

1. Its time complexity is $O(n^2)$ as it has nested loops where the first loop and second loop are overlapping each other. are overlapping each other.
2. It has constant space complexity, $O(1)$ since it only needs a fixed amount of extra space for the flag and size variables.
3. It is an in-place sorting algorithm such that the input will be modified throughout the sorting process. (*What Is Bubble Sort Algorithm? Time Complexity & Pseudocode | Simplilearn*, n.d.)
4. It is a simple algorithm that everyone can understand the algorithm quickly and implement it easily.

5. It is efficient in sorting a small amount of data but not in a big amount of data as more loops are needed to execute. (Gill et al., n.d.)

The research team has implemented bubble sort algorithms using Java and Python on the array with 9 integer elements. The result is shown in the table below:'

Table 1. Comparison source source java vs python bubble sort method.



No	Program Name	Amount LOC	File Capacity	File Capacity txt	Speed Access
1	JAVA	11	86.2 Kbytes	477 bytes	7 ps
2.	PYTHON	10	506 bytes	397 bytes	4 ps

Based on the table above, it clearly stated that Python is better than Java in implementing the bubble sort algorithm. The lines of code (LOC) of Python are one line lesser than Java. The file capacity of Python is much lesser than Java, regardless of the source files or txt files generated. (Li et al., 2019)

3.2. Insertion Sort

Insertion sort is an in-place, straightforward, and efficient sorting algorithm suitable for tiny and almost sorted lists. At each iteration, insertion sort takes one element from input data and inserts it into the correct position. It needs only a fixed amount of additional memory space for inserting the element at the appropriate position in the sorted sub-list. (Siddhant Grover, 2019)

The data used in this research is numerical. Java uses large-size sorted and unsorted arrays having$1 - n$￼distinct elements, where n is ranging from 50K to 10000 K. As for Python, the large size sorted and unsorted array having ￼distinct elements, where n is ranging from 5 K to 1000 K. $1 - n$ Grover, 2019)distinct elements, where n is ranging from 5 K to 1000 K. (Siddhant Grover, 2019)

In the research, both Java and Python are used to implement insertion sort on sorted and unsorted arrays separately. The results in the table below, Python programs run comparatively slower than Java while running the insertion sort algorithm,

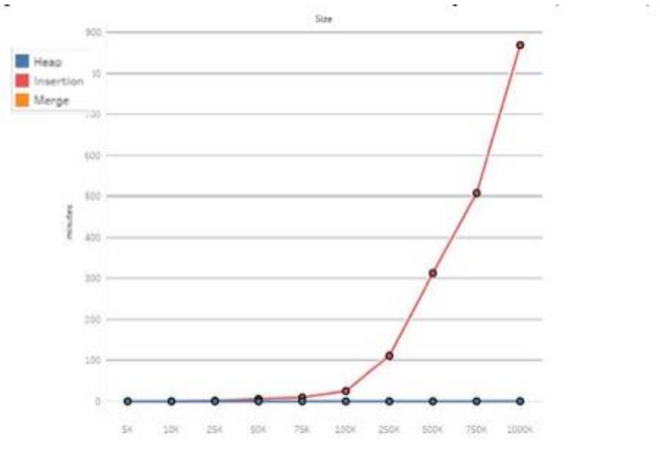
Table 1: Time taken(milliseconds) to sort a sorted array by Insertion sort in Java.

Size	Time (ms)
50K	7.8
100K	9.2
250K	13
500K	17.8
750K	18.3
1000K	**19**
2500K	**21**
5000K	**25**
7500K	**27**
10000K	32

Table 2: Time taken(seconds) to sort a sorted array by Insertion sort in Python.

Size	Time (s)
5K	0.00199866
10K	0.00199819
25K	0.00703454
50K	0.01094365
75K	0.02797771
100K	0.02100396
250K	0.06605411
500K	0.14139819
750K	0.19499683
1000K	0.2761302

While comparing the execution speed of Java and Python on the unsorted array, based on the graph, the graph shows that there is a wide time gap between 250 K elements in Java and 25 K elements in Python. This is due to the reason that Python being an interpreted language takes longer time to run as compared to Java compiled language. (Siddhant Grover, 2019)



Graph 4: Time taken to sort an unsorted array in Python(minutes)

In conclusion, the point at which the algorithms noticeably differ should be taken into consideration when choosing a programming language.

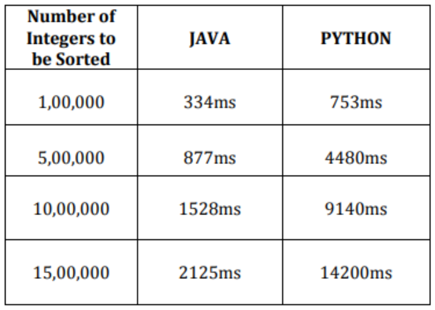
3.3. Quick Sort

Quick Sort is a divide-and-conquer algorithm. It performs in-place sorting with no additional storage space needed. It picks an element as a pivot and partitions the given array around the picked pivot. The worst time complexity and the worst space complexity of quick sort are $O(n^2)$ and $O(1)$ respectively. (*QuickSort - GeeksforGeeks*, n.d.)

The research team has performed the experiment for both Python and Java using Quick Sort Algorithm. The team created four text files containing one hundred thousand, five hundred thousand, one million, and five million randomly generated numbers respectively to compare the performance of the sorting algorithm. (Khoirom et al., 2020)

'*timeit*' is for Python while 'Instant.now()' is used for Java to calculate the approximate execution time. The execution time comparison is shown below:

Table 1: Quick Sort Execution Time Comparison



Number of Integers to be Sorted	JAVA	PYTHON
1,00,000	334ms	753ms
5,00,000	877ms	4480ms
10,00,000	1528ms	9140ms
15,00,000	2125ms	14200ms

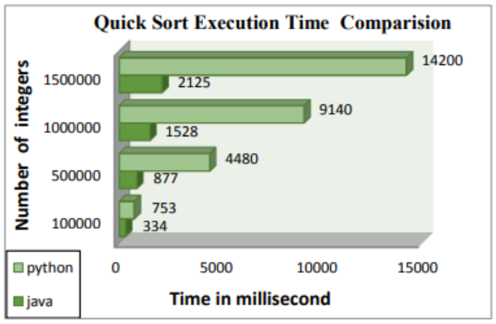


Fig-4: Quick Sort Execution Time Comparison Graph

From the result shown, Java executes faster than Python for every text file in the experiment. However, comparing Java and Python when the input gets larger, the time needed for Python to complete sorting the numbers increases exponentially compared to Java which only increases around two to three times from the previous input size.

Comparing the file size and lines of code for Python and Java implementing Quick Sort Algorithm, Python has relatively fewer lines of code and smaller file size compared to Java.

Table 2: File Size and Lines of Code Comparison in Quick Sort

Programming Language	Lines of Code (LOC)	File Size (in bytes)
Python	22	1372
Java	57	1533

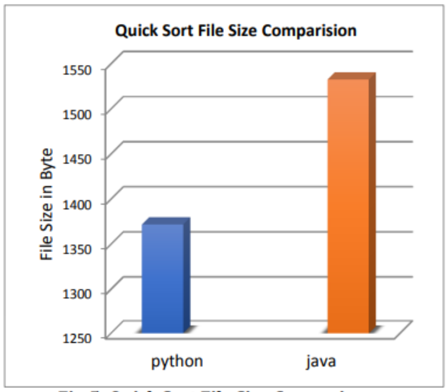


Fig-5: Quick Sort File Size Comparison

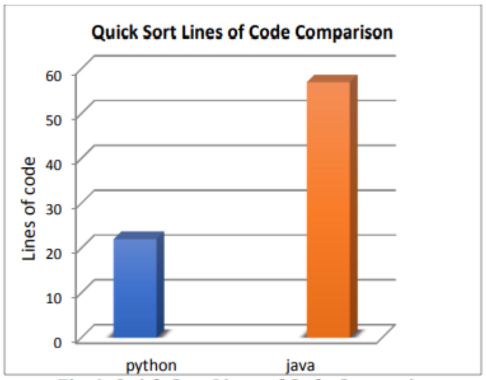


Fig-6: Quick Sort Lines of Code Comparison

The result shown by the team states that Java has better execution speed, while Python has fewer lines of code and smaller file size compared with Java. It significantly shows the advantage of Java as a compiled language in taking less time to execute the code, while Python as an interpreted language takes longer time to interpret and perform each line of code and update the machine state. However, in terms of code readability, lines of code, and file size, Python shows a great advantage.

4. Reason/Justification of selecting one algorithm to be implemented

Quick sort algorithm is chosen to be implemented as it is the fastest algorithm among the three algorithms above based on the time complexity analysis. It is worth to testify the algorithm with the large amount of data given to observe the memory usage and the speed of execution. It can be used to understand how fast Python or Java interprets or compile the program.

Another reason to use quick sort is it needs not any extra memory allocation for the sorting process. It is also a cache-friendly algorithm when it is used with arrays as it will access the whole array very frequently during the process which leads to good locality of reference. (*What Is Bubble Sort Algorithm? Time Complexity & Pseudocode | Simplilearn*, n.d.) This will reduce the memory usage of both Java and Python and focus on the time taken for Python and Java to execute the program.

5. String sorting implementation in Java

5.1. Pseudocode for Java Implementation

```
                    $partition(array, left, right)$
Begin
    If index of first element ($low$) is smaller than last element ($right$)
    Begin if
        Find a pivot by partitioning the array and assign it
                $pi = partition(array, left, right)$
        Sort elements on the left of the pivot (calling
        $quck\_sort(array, left, pi - 1)$)
        Sort elements on the right of the pivot
        (calling$quck\_sort(array, pi + 1, right)$
    End if
End
```

```
                    $partition(array, left, right)$
                    $partition(array, low, high)$
    Set the rightmost element as pivot
    Set $i = left - 1$ where $i$ is the position where partition is done

    For $j$ from low to high:
    Begin for loop
        If element at index $j$ is less than or equal to pivot:
            Increase $i$ by 1
            Swap element at index $i$ with element at index $j$
    End for loop

    Swap element at index $i + 1$ with element at $right$ index

    Return $i + 1$
End
```

```
Initialise an array called $wordlist$
Open the text file that contains the strings that need to be sorted and store in the
$wordlist$

Get current time and assign to ￼
Call function $quick\_sort(wordlist, 0, len(wordlist) - 1)$
Subtract current time and $startTime$ , then assign to $endTime$
```

5.2. Code explanation for Java

The Quick Sort Algorithm is implemented using Java programming language. The algorithm is created using three functions namely $quickSort$, $partition$, $swap$. The program also contains a $main$ function that takes the dataset as input and runs the sorting algorithm via the function all.all.

In the $main$ function, it first creates an empty array list, $wordList$ for storing strings. The words from the text file are then read and added line by line to the $wordList$. After reading until the end of the file, the ￼is converted into an array - ￼before $wordList$ ￼function.$quickSort$ function.

In the $quickSort$ function, if the left index is lower than the right index, it will recursively update the partitioning index by calling ￼ and sort the left and right partitions using ￼ and $quickSort(arr, pi + 1, right)$respectively.

In the $partition$ function, the pivot is first initialized as the rightmost element of the array. Variable ￼is initialized as the smaller index next to the current smallest index, $low\ i$ After that, a loop is implemented to compare the current element to the $pivot$. Ilow the current element j is smaller than the ￼, it swaps the smaller and larger elements to the left and right of the pivot respectively. The loop terminates once the $pivot$ is incremented to the same value as the right index. Then, another $swap$ is called to swap the element at index $i + 1$ with the element at index $right$ to ensure that the pivot is the value in between the left and right values.f the current element j is smaller than the $pivot$, it swaps the smaller and larger elements to the left and right of the pivot respectively. The loop terminates once the j is incremented to same value as right index. Then, another $swap$ is called to swap the element at index $i + 1$ with the element at index $right$ to ensure that the pivot is the value in between the left and right values.

6. String sorting implementation in Python

6.1. Pseudocode for Python Implementation

```
$quick\_sort(array, low, high)$
Begin
    If index of first element ($low$) is smaller than last element ($high$)
    Begin if
        Find a pivot by partitioning the array and assign it $pi$ ($pi = partition(array, low, high)$
        Sort elements on the left of the pivot (calling $quick\_sort(array, low, pi - 1)$)
        Sort elements on the right of the pivot (calling $quick\_sort(array, pi + 1, hight)$
    End if
End
```

```
$partition(array, low, high)$
Begin
    Set the rightmost element as pivot
    Set $i = low - 1$ where $i$ is the position where partition is done

    For $j$ from low to high:
    Begin for loop
        If element at index $j$ is less than or equal to pivot:
            Increase $i$ by 1
            Swap element at index $i$ with element at index $j$
    End for loop

    Swap element at index $i + 1$ with element at $high$ index

    Return $i + 1$
End
```

```
Initialise an array called $wordlist$
Open the text file that contains the strings that need to be sorted and store in the $wordlist$

Get current time and assign to $startTime$
Call function $quick\_sort(wordlist, 0, len(wordlist) - 1)$
Subtract current time and $startTime$, then assign to $endTime$
```

6.2. Code explanation for Python

The Quick Sort Algorithm is deployed in Python programming language. The whole program is designed based on two functions and a few executive lines of code outside. Firstly, the program initializes an empty list named $wordlist$. The list is created to store all the strings that need to be sorted. The strings are then read from the file. Then, the $quick\_sort(wordlist, 0, len(wordlist) - 1)$ function is called where 0 is the index of the first element in ￼ and $len(wordlist) - 1$ is the index of the last element in $wordlist$. is the index of the first element in $wordlist$ and $len(wordlist) - 1$ is the index of the last element in $wordlist$.

After passing the arguments to $quick\_sort(array, low, high)$, the lower index is compared to the higher index, if low is smaller than the $high$, then, the $partition(array, low, high)$ is called. In the $partition$ function, a variable $pivot$ is assigned with the rightmost element in the array passed to $partition$. Then, $partition$riable i is assigned to be the position where partition is done. i is initially set one index smaller than the current smallest index, low. $i$for loop is used to get index from low to $high - 1$.$i$the for loop, if the value of the current element with index j is smaller than the $pivot$, then the position where partition is done, i is incremented by 1. The element at index $jlow$ then swapped with the element at index i. This swapping process in the for loop ensures that elements smaller than the pivot are moved to the left, and the larger elements are moved to the right of the pivot. After the for-loop ends, another swapping with element at index ￼ is done with the elownt at index $high$. This is to move the pivot element which located at the index $high$ to be moved to the position $i + 1$ where the elements on the left-hand side of position $i + 1$ are all the elements smaller t$high - 1$the pivot and the elements on the right-hand side of position $i + 1$ are all the elements larger than the pivot. The position $i + 1$ is then returned from the $partition$ function and assigned to the variable pi in the $quick\_sort$ function. Then, another $quick\_sort$ is called to sort the elements on the left of the pivot, followed by another call for the elements on the right of the pivot.at index j is then swapped with the element at index i. This swapping process in the for loop ensure that elements smaller than the pivot are move to the left, and the larger elements are move to the right of the pivot. After the for-loop ends, another swapping with element at index $i + 1$ is done with the element at index $high$. This is to move the pivot element which located at the index $high$ to be moved to the position $i + 1$ where the elements on the left-hand side of position $i + 1$ is all the elements smaller than the pivot and the elements on the right-hand side of position $i + 1$ is all the elements larger than the pivot. The position $i + 1$ is then returned from the $partition$ function and assigned to the variable pi in the $quick\_sort$ function. Then, another $quick\_sort$ is called to sort the elements on the left of the pivot, followed by another call for the elements on the right of the pivot.

Once, the $partition$ function is done, the $wordlist$ will be shown in a sorted sequence. A time stamp is added before and after calling the $quick\_sort(wordlist, 0, len(wordlist) - 1)$ to record the time taken for the sorting algorithm to run. The memory required to run the program can get from the file description.

7. Performance evaluation and comparison of results

We will evaluate the performance evaluation from the following aspects: code readability, file capacity, and execution speed.

In terms of code readability, Python is a more human-readable programming language compared to Java. The programmers can grasp the syntax and semantics of Python with lesser effort than Java.

The second part is file capacity. The file capacity refers to the LOC in the source file. Python only uses 2 kilobytes of space in the workstation while Java uses 3 kilobytes of space. Based on our observation, we can see that the Python source file has only 48 lines of code while Java has 84 lines of code which is almost double if compared with Python. From there, we can conclude that Python is better in the aspect of file capacity.

The third part will be execution speed. Based on the table below, we can observe that the execution speed of Python is way faster than Java. Hence, Python is better than Java in execution speed.

Times of execution (ms) / Languages	1st	2nd	3rd	4th	5th
Python	14.857	13.171	14.441	12.768	13.082
Java	104	91	84	81	73

8. **Lesson learned and conclusion**

In this assignment, we experienced implementing the Quick sort in Java and Python language. This has further bridged the gap between our theoretical and practical understanding of the Quick sort algorithm.

Next, we get to know the distinct characteristics of Java and Python. We learn that Python code is easier to understand than Java in regards to its code readability. This is due to the reason that Java has a more complex syntax than Python. Although Java is not as readable as Python, however, it has better memory management, and it is more secure. (Khoirom et al., 2020)

Furthermore, Java and Python both are different types of languages. Java is a compiled, static typed language while Python is an interpreted, dynamic typed language. (Khoirom et al., 2020) There is a difference between interpreted and compiled languages in the program running process. The interpreter will directly show the result of the program while the compiler outputs an assembly program and transform it into binary code. Since assembly language is machine and architecture-dependent, the compiled program is not able to run on computers with different architectures. On the other hand, Java is known for its “Write Once, Run Anywhere” features as it uses a runtime environment that will compile the source code to byte code. The byte code will be read by the runtime environment and translated into computer-specific commands. (*What Is the Difference between a Compiled and an Interpreted Program?*, n.d.) The difference between static typed and dynamic typed language is that when the programmer uses the variable, static typed language requires the programmer to declare the variable and its type first before using the variable while dynamic typed language need not do so. (Khoirom et al., 2020)

In conclusion, Python is better in execution and readability and Java is better in managing memory and is more secure.

9. References

1) Khoirom, S., Sonia, M., Laikhuram, B., Laishram, J., & Singh, T. D. (2020). Comparative Analysis of Python and Java for Beginners. *International Research Journal of Engineering and Technology (IRJET)*, *7*(8), 4384–4407. Retrieved from: https://www.irjet.net/archives/V7/i8/IRJET-V7I8755.pdf
2) Coursera. (2022, Jul 29) Python vs. Java: Which Should I Learn? Retrieved from: https://www.coursera.org/articles/python-vs-java
3) GeeksforGeeks (2022, May 27). History of Python. Retrieved from: https://www.geeksforgeeks.org/history-of-python/#:~:text=Python%20is%20a%20widely%2Dused,in%20fewer%20lines%20of%20code
4) GeeksforGeeks (2022, Sep 27). QuickSort. Retrieved from: https://www.geeksforgeeks.org/quick-sort/
5) Gill, S. K., Singh, V. P., Sharma, P., & Kumar, D. (n.d.). *A Comparative Study of Various Sorting Algorithms*. Retrieved October 26, 2022, from https://ssrn.com/abstract=3329410
6) *History of Python - GeeksforGeeks*. (n.d.). Retrieved November 5, 2022, from https://www.geeksforgeeks.org/history-of-python/
7) Khoirom, M. S., Sonia, M., Laikhuram, B., Laishram, J., & Singh, D. (2020). Comparative Analysis of Python and Java for Beginners. *International Research Journal of Engineering and Technology*. www.irjet.net
8) Li, B., Xia, D., Ma, J., Katayama, Y., Tachikawa, N., & Insanudin, E. (2019). Implementation of python source code comparison results with Java using bubble sort method. *Journal of Physics: Conference Series, 1280*, 32027. https://doi.org/10.1088/1742-6596/1280/3/032027
9) *Python vs. Java: Which Should I Learn? | Coursera*. (n.d.). Retrieved November 5, 2022, from https://www.coursera.org/articles/python-vs-java
10) *QuickSort - GeeksforGeeks*. (n.d.). Retrieved November 5, 2022, from https://www.geeksforgeeks.org/quick-sort/
11) *What is Bubble Sort Algorithm? Time Complexity & Pseudocode | Simplilearn*. (n.d.). Retrieved October 31, 2022, from https://www.simplilearn.com/tutorials/data-structure-tutorial/bubble-sort-algorithm
12) (2022). Introduction to Java. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-to-java/
13) Grover, S. (2019). Performance Analysis of Heap, Merge, and Insertion Sort. cs.winona.edu. https://cs.winona.edu/cs-website/current_students/Projects/CSConference/2020conference.pdf#page=20
14) What is the difference between a compiled and an interpreted program? (n.d.). Retrieved November 8, 2022, from https://kb.iu.edu/d/agsz

Division of Task

Names	**Division of Task**
Lim Zi Qiang	Introduction on Java, Review of Insertion Sort research paper, String sorting implementation in Java, Implement Quick Sort algorithm, GitHub code pull request review, Report proofreading
Khoo Lay Sin	Review of Bubble Sort, Reason/Justification of selecting one algorithm to be implemented, Performance evaluation and comparison of results, Implementation of time complexity analysis in Java, Lesson learned and conclusion, GitHub code pull request review
Loh Wan Teng	Abstract, Introduction on Python, Quick Sort, Reason/Justification of selecting one algorithm to be implemented, String sorting implementation in Python, Implement Quick Sort and timeit, GitHub code pull request review