Ouriginal

Document Information

Analyzed document	submission.odt (D136606144)
Submitted	2022-05-15T16:41:00.0000000
Submitted by	
Submitter email	G00398318@gmit.ie
Similarity	19%
Analysis address	dominic.carr.gmit@analysis.urkund.com

Sources included in the report

SA	**Galway-Mayo Institute of Technology / submission.pdf** Document submission.pdf (D104582482) Submitted by: G00387832@gmit.ie Receiver: dominic.carr.gmit@analysis.urkund.com	1
W	URL: https://www.geeksforgeeks.org/in-place-algorithm/ Fetched: 2022-05-15T16:41:00.0000000	1
J	URL: 59cd1872-5345-4692-bdd7-bd9c6383dda5 Fetched: 2019-05-03T13:50:18.3470000	2
SA	**Galway-Mayo Institute of Technology / submission.docx** Document submission.docx (D136218652) Submitted by: G00398581@gmit.ie Receiver: dominic.carr.gmit@analysis.urkund.com	1
SA	**Galway-Mayo Institute of Technology / submission.docx** Document submission.docx (D136572806) Submitted by: G00048625@gmit.ie Receiver: dominic.carr.gmit@analysis.urkund.com	2
W	URL: https://iq.opengenus.org/time-complexity-of-selection-sort/ Fetched: 2022-05-15T16:41:00.0000000	3
SA	**Galway-Mayo Institute of Technology / submission.pdf** Document submission.pdf (D135762562) Submitted by: G00398279@gmit.ie Receiver: dominic.carr.gmit@analysis.urkund.com	4
SA	**Galway-Mayo Institute of Technology / submission.pdf** Document submission.pdf (D104980609) Submitted by: G00387935@gmit.ie Receiver: dominic.carr.gmit@analysis.urkund.com	1
W	URL: https://www.studytonight.com/data-structures/merge-sort Fetched: 2022-05-15T16:41:00.0000000	1
W	URL: https://iq.opengenus.org/time-and-space-complexity-of-counting-sort/ Fetched: 2022-05-15T16:41:00.0000000	1

Curiginal

W	URL: https://iq.opengenus.org/insertion-sort-analysis/ Fetched: 2022-05-15T16:41:00.0000000	2
SA	**Galway-Mayo Institute of Technology / submission.pdf** Document submission.pdf (D104742236) Submitted by: G00287906@gmit.ie Receiver: dominic.carr.gmit@analysis.urkund.com	2
SA	**Galway-Mayo Institute of Technology / submission.pdf** Document submission.pdf (D136586566) Submitted by: G00398244@gmit.ie Receiver: dominic.carr.gmit@analysis.urkund.com	6
SA	**Galway-Mayo Institute of Technology / submission.docx** Document submission.docx (D136590819) Submitted by: G00270683@gmit.ie Receiver: dominic.carr.gmit@analysis.urkund.com	1

Entire Document

CTA Project

Benchmarking Sorting Algorithms

Introduction

This section of the report is to introduce different concepts of sorting and the use of sorting algorithms. This introduction will hopefully demonstrate an understanding of concepts such as performance, complexity, comparator functions, comparison/non-comparison based sorts, and in-place sorting.

This project outcome is to demonstrate a well-designed algorithm, as an algorithm is to get an expected output from input with a set of rules. The request is to investigate and demonstrate this using a sorting algorithm, producing the correct output by demonstrating the efficiency of the sorting algorithm over space/time and varying sizes of the input (n=100 to n=10000).

What is algorithms' time complexity? The time complexity of an algorithm is estimated by counting the number of steps performed till its final execution. By the number of steps which approach would you consider is the best solution? A python for loop or using a mathematical operator of *

To demonstrate let us try two methods to output the square of 16. Example 1 uses a for loop to cycle from 1*1 eventually reaching 16*16 and printing out each step.

Example 1 For loop [3] Example 2 uses a Mathematical operator of *

Example 2 Mathematical operator of *

The time complexity of steps performed till their final execution using the mathematical operator of * returning the result in one line [4] is a better solution.

What is algorithms' space complexity? Space complexity is considered the amount of memory a computer requires to solve concerning the input size, another aspect that can also affect the outcome can include the type of machine, programming language or how the program is compiled. [1]

So if we re-run example 3 of the for loop with a bigger number (123456) this will take up more space in memory and resources, making the PC unusable until the program finishes executing.

Example 3 for loop Heavy on CPU and memory consumption.

Now let us get the square of 123456 using the mathematical operator of *. Example 4

Example 4 negligible CPU and memory consumption

The second squared example of a mathematical operator * requires little to no memory to achieve the same result and even with larger numbers did not reach the same requirements in example 3.

What methods measure complexity? With an algorithm, there are two methods of calculating this Fig 1 gives a further breakdown. .• A Priori analysis looks at from a theoretical, device agnostic and measure of complexity. .• A Posteriori analysis measures the performance on the same platform

Fig 1 Difference between Posteriori and Priori [1]

These methods indicated performance versus complexity and if we refer to the Examples 1 and 3 python for loops, performance never affects complexity but as the complexity increased input size n it affected the performance. This is shown in Fig 2 when the increased input size n growth functions Fig 2 Comparing growth functions [5]

What is the Big O notation? (Worst case) From a theoretical perspective, the

88%	MATCHING BLOCK 1/28	SA	submission.pdf (D104582482)

Big O notation is the most common metric used for calculating time complexity.

Referring to examples Example 1 and 3 pythons for loops increasing the size of the input n, increased the number of the operation taken. Big O notation measures the growth rate of a function increase or decrease in a worst-case scenario representing the lower bounds

If Example 2 has a less complex Big 0 notation than Example 1 it can be inferred that Example 2 is more efficient in terms of space/time requirements. [5]

What is Ω (omega) notation? (Best Case) This is the best-case scenario with a linear growth rate in execution time as n is increased, representing the upper bounds [5]

What is Θ (theta) notation? (Average Case) Θ (theta) notation is the upper and lower bounds of the runtime and is used for analysing the metric used for calculating the average time of complexity.[5]

What is a sorting algorithm? A sorting algorithm is to take an input (an array or list) to arrange the output in a particular order and has desirable properties of stability, efficiency and in-place sorting.[1] There are several sorting algorithms each with its efficiency in complexity by the number of operations or time it takes to complete known as Time and the amount of memory required to run also known as Space.

Why do we need sorting algorithms? It is said that 25 per cent of the running time of computers in the 1960s was spent sorting with some tasks responsible for more than half of the computing time. [2]

While we humans deal with sorting information on a day to day for example your shopping list and finding food in a supermarket. We generally use technology to simplify tasks for us, this is usually through sorting, how we search the web and the results and the order of the return results, how to find a car's make, model, by year, mileage, etc.

Sorting memory usage using in-place Different sorting techniques require different allocations of memory, to keep memory usage to a minimum the most desirable is in-place. In-place sorting does not require extra memory as the input data is usually overwritten by the output, only swapping elements within the input size n. This avoids creating an empty array for copying data to double the memory requirements.

In-place

98%	MATCHING BLOCK 2/28	W	https://www.geeksforgeeks.org/in-place-algorithm/

Definition: In-place means that the algorithm does not use extra space for manipulating the input but may require a small though non-constant extra space for its operation. Usually, this space is O(log n), though sometimes anything in O(n) (Smaller than linear) is allowed. [6]

Comparison-based and non-comparison-based sorts Comparison based means that elements within input size n are compared with each other. This is called a Comparator function which compares one element against another element and returns a value. Comparing elements can never have a worst-case time better than O(N log N) Fig 2.

A Non-Comparison is a way to design an algorithm not to compare elements, for instance making an assumption about data, taking a range of values from the data and the distribution of

69%	MATCHING BLOCK 3/28	J	59cd1872-5345-4692-bdd7-bd9c6383dda5

the data. Non-comparison sorts can achieve linear running time in the best case but are less flexible.

Fig 3 Overview of sorting algorithms [7]

Sorting Algorithms This section of the report is to introduce the five chosen sorting algorithms referencing

88%	MATCHING BLOCK 4/28	SA	submission.docx (D136218652)

their space and time complexity, how each algorithm works using bespoke diagrams and different example input instances. [8]

The five chosen sorting algorithms are as follows: 1. Bubble Sort is a simple

29%	MATCHING BLOCK 5/28	SA	submission.docx (D136572806)

comparison-based sort. 2. Merge Sort as an efficient comparison-based sort 3. Counting Sort as a non-comparison sort 4. Selection Sort is a simple comparison-based sort. 5. Insertion Sort is a simple comparison-based sort. Bubble Sort is a simple comparison-based sort. Bubble Sort is a comparison based sort which compares two elements, swapping them and continuing until they are in

the correct order. The name comes from how the highest number bubbles up to the top as the largest element is repeatedly compared to the neighbour on the right and swapped. This is repeated until the largest element is in the furthest position to the right, then the next highest element and so on. See the Diagram 1 to show the example [5]

Diagram 1

It may better visualise if the input the bubbling up affects if the array is transposed, this diagram shows 20 bubbling up, then 4, then 2. Diagram 2

Space complexity [9] .• O(1) The algorithm only requires using a temporary variable while swapping the largest element

Time complexities [9] .•

58%	MATCHING BLOCK 7/28	W	https://iq.opengenus.org/time-complexity-of-se ...

Worst Case Time Complexity of Bubble Sort is O(N^2) .◦ This is the case when the array is

reversely sort .•

73%	MATCHING BLOCK 6/28	SA	submission.docx (D136572806)

Best Case Time Complexity of Bubble Sort is O(N) .◦ This case occurs when the

given array is already sorted. .• Average Case Time Complexity of Bubble Sort is

90%	MATCHING BLOCK 8/28	SA	submission.pdf (D135762562)

O(N^2) .◦ The number of comparisons is constant in Bubble Sort

so in the average case, there are O(N^2) comparisons.

Python code and execution of the above diagrams

Example 5 cta-sorts.ipynb code and execution of the Bubble sort of above diagrams

Merge Sort

75%	MATCHING BLOCK 9/28	SA	submission.pdf (D104980609)

is an efficient comparison-based sort Merge Sort is a recursive divide-and-conquer

approach by diving the index into halves and subdividing until the elements can be easily solved and then merging the results [10]. To visualise the diagram shows the index split into sub-indexes [2,4] [20] [1,19]

Diagram 3

Space complexity [11] .• O(N) as N is the space required for several elements in the index, this is divided and stored for the total space and merged back to a single array.

Time complexities [12] .•

95%	MATCHING BLOCK 11/28	W	https://www.studytonight.com/data-structures/m ...

Merge Sort is O(n*Log n) in all the 3 cases (worst, average and best) as merge sort always divides the array into two halves and takes linear time to merge two halves.

Example 6 cta-sorts.ipynb code output of a merge sort of the above diagrams

Counting Sort as a non-comparison sort A Counting Sort is a non-comparison based algorithm based on keys in a range. It counts the number of elements with distinct values. The distinct value is held in a temporary array and used that to sort the input array [13] Please see the diagram below

Diagram 4

Space complexity [14] .• O(n+k) as an auxiliary array is created to the size of k, k in the example is 20 there for the auxiliary array is the size of 20 with only 5 elements (Example 7) on the input array or create an auxiliary array of 3 if the range is between input is between 1-3 with 5 elements (Example 8). This indicates that space complexity is not great for a wide range of integers as the auxiliary array of that size is required to be created.

Time complexities [9] .• O(n+k)

100%	MATCHING BLOCK 14/28	W	https://iq.opengenus.org/time-and-space-comple ...

range k of the elements is significantly larger than the other elements

deepening the range of elements, Comparing example 7 with 40 steps and example 8 with 23 steps Example 7 cta-sorts.ipynb input index of 5 elements with range 1-20 with the auxiliary of 20 Example 8 cta-sorts.ipynb input index of 5 elements with range 1-3 with the auxiliary of 3

Selection Sort is

57%	MATCHING BLOCK 10/28	SA	submission.pdf (D135762562)

a simple comparison-based sort. A Selection Sort essentially finds the minimum element in an array and inserts it

into its position with the element in its place. [15] Diagram 5

Space complexity [16] .• O(1) Similar to bubble sort the algorithm only requires using a temporary variable for the to hold an element in the array while swapping the elements

Time complexities [16] .•

44%	MATCHING BLOCK 12/28	W	https://iq.opengenus.org/time-complexity-of-se ...

Worst Case Time Complexity of Selection Sort is O(N^2) .◦ This is the case if the given array is sorted except for the last element is the

smallest .•

55%	MATCHING BLOCK 13/28	W	https://iq.opengenus.org/time-complexity-of-se ...

Best Case Time Complexity of Selection Sort is O(N^2) .◦ This case occurs when the given array is already sorted. .•

Average Case Time Complexity of Selection Sort is O(N^2) .◦ The number of comparisons is constant in Selection Sort so in an average case, there are O(N^2) comparisons.

Example 9 cta-sorts.ipynb code output of a Selection sort of the above diagrams Insertion Sort is a simple comparison-based sort.

An Insertion Sort compares each pair in if the minimum element in an array is found it inserts that element into the correct position. [17] Diagram 6

Space complexity [18] .• O(1) Similar to Bubble sort and Selection sort the algorithm only requires

80%	MATCHING BLOCK 15/28	SA	submission.pdf (D135762562)

space by shifting the element to right and inserting the element at the appropriate position.

Time complexities [18] .•

100%	MATCHING BLOCK 17/28	W	https://iq.opengenus.org/insertion-sort-analysis/

Worst Case Time Complexity of Insertion Sort is O(N^2) .◦

This is the case if

55%	MATCHING BLOCK 16/28	SA	submission.pdf (D135762562)

the array is reversely sorted .• Best Case Time Complexity of Insertion Sort is O(N) .◦ This case occurs when the

given array is already sorted. .•

100%	MATCHING BLOCK 18/28	W	https://iq.opengenus.org/insertion-sort-analysis/

Average Case Time Complexity of Insertion Sort is O(N^2) .◦ The

number of comparisons is constant in Insertion Sort so in an average case, there are O(N^2) comparisons.

Example 10 cta-sorts.ipynb code output of an Insertion sort of the above diagrams

Implementation & Benchmarking This section of the report is to introduce the implementation using a Jupyter notebook to time and test all the python sorting functions and output the required table and graphs. The results of each of the benchmarks are discussed and broken into 3 parts. Information on extra notes and scripts used in the project and a conclusion.

Jupyter notebook ctaproject-V05.ipynb The project was implemented in a Jupyter notebook called ctaproject-V05.ipynb, with the five sorting algorithms. Each of the sorting functions is labelled and commented on appropriately and included some information from this project.

.• Bubble Sort is a

44%	MATCHING BLOCK 20/28	SA	submission.pdf (D104742236)

simple comparison-based sort [20] .• Merge Sort is an efficient comparison-based sort [21] .• Counting Sort as a non-comparison sort [22] .• Selection Sort is a simple comparison-based sort. [23] .• Insertion Sort is a simple comparison-based sort. [24]

Settings Example 11 Settings (not including imports) have the following

.• A list of sorting algorithms names (algos) .• To call the functions .• A list with varying integers sizes (Nsizes) .• To create the sizes of N .• A dictionary (a_dict) .• To hold the results of the timer.

Example 11 Setting Code Snippet

For Loop Example 12 A for loop [19] iterates over list Nsizes for every element in algos calling the function timer and adds them to a a_dict Example 12 For Loop Code Snippet

Benchmarking Example 13 For benchmarking a timer function was adapted from 1.2.2 Python of the CTA Project Specification pdf [8]. This accepted two inputs x (Nsizes) and y (algos). A random array is created using Numpy with a range of numbers 1 – 100 and size (Nsizes). A section is commented out on a second random array to check how a larger range of 1 - 100000 would affect the timing on the Count Sort. A While Loop is used to count 10 runs of sorting algorithms, the time.time module records the times in seconds before and after each run of an algorithm and array size. The recorded times are subtracted and multiplied by 1000 to get the milliseconds as required by the Project Specifications. All 10 runs record are added and then divided by 10 to get the average run. The average run is returned and stored in the dictionary a_dict.

Example 13 Benchmarking Code Snippet

Pandas Example 14 – 17 A pandas dataframe is created with the index for each of the sorting algorithms and a_dict is inserted into the Pandas Dataframe.

Example 14 Pandas Code Snippet

All data is then rounded to 3 decimal places and a table of the results is presented to the user. Example 15 Pandas Code Snippet

This produces a graph overlay of each of the chosen sorting algorithms.

Example 16 Pandas Code Snippet

The second graph is produced for each sorting algorithm

Example 17 Pandas Code Snippet

Results of Benchmarking Benchmarking was tested in 3 different ways 1. n size 100 – 10000 with a random range of 1 – 100 2. n size 100 – 10000 with a random range of 1 – 100000 3. n size 100 – 100000 with a random range of 1 – 100

Part 1 n=10000 Range 1-100 Example 18 – 20 The results of the benchmarking are presented show simple comparison sorts, Bubble Sort taking over 23 seconds to complete n=10000 and followed close behind with Selection and Insertion sort over 11 seconds. All three of the simple comparison-based sort displayed the growth rate of O(N^2).

Merge sort in 84 milliseconds and Counting Sort in 15 milliseconds

Example 18 Results of Benchmarking Part 1

Example 19 Results of Benchmarking Part 1

Example 20 Results of Benchmarking Part 1

Part 2 n=10000 Range 1-100000 Example 21 – 24

Part 2 was to test if the range of numbers was increased 1000 times from a range of 100 to a range of 100000 and what effect this would have on the benchmark results. The results showed no significant changes in 4 of the 5 sorting algorithms. The time and space were affected in the Counting Sort indicating a nearly ~ 2.4 increase in the time to sort the range with an n=10000

Example 21 Comparing Solve times of ranges 100 versus 100000 Example 22 Results of Benchmarking Part 2

Example 23 Results of Benchmarking Part 2 Example 24 Results of Benchmarking Part 2

Part 3 n=100000 Range 1-100 Example 25 – 31 This was the initial testing of using n=100000 was not efficient due to time constraints and an error in the layout of the script. Both of these issues produced some interesting results and contributed to my understanding of sorting algorithms' time and space. These are the discoveries:

• This was only run once as the total runtime was 544 minutes, as all the sorting algorithms required a higher time and space complexity

• When compared to a run of n=10000 table it was noted the average time was ~100 times slower. This means an estimation could be made for all n sizes, for example below estimating an n=1000000 Example 25 Estimating n=10000000

• An error in the coding when the run on the same random array 10 times (getarray) Insertion time did not follow the curve and had significantly lower times than expected of ~10 times faster, Bubble Sort was ~2 times as fast. Selection, Merge and Count times were not significant enough to note. See Example 28 for comparison Example 26 getarray outside while loop Example 27 getarray inside while loop

Example 28 Compare results of error in code

Example 29 Results of Benchmarking Part 3 Example 30 Results of Benchmarking Part 3

Example 31 Results of Benchmarking Part 3

Conclusion Example 32 This project has allowed me a greater understanding of sorting algorithms and space and time complexities. The times in the table and graphs comparisons are the expected results given in the course. As demonstrated various factors came into play including the machine's performance. Testing the Jupyter notebook on another machine will not exactly match output tables and graphs but the expected behaviour of each of the chosen sorting algorithms. The test was run over ten times for each Nsize on a random NumPy array with different ranges. Changing the ranges did affect the outcome of the Counting Sort increasing the times to solve by ~2.4 times. Counting Sort was still overall the fast algorithm of the five chosen. The changes made also proved that Comparisons based (In-place) sorting does not require extra memory as the input data is usually overwritten by the output, only swapping elements within the input size n. In the larger range of numbers 1 – 100000, a

75%	MATCHING BLOCK 19/28	J	59cd1872-5345-4692-bdd7-bd9c6383dda5
Non-comparison sort can achieve linear running time in the best case but are less flexible,			

as demonstrated with the Counting Sort creating a larger second key array significantly increasing the memory requirements. It was also noted that the error made on the script using the same random array saw a significant decrease in times for Insertion (10 times) and Bubble Sort (2 times), and would require further investigation.

In Summary, the results of the benchmarking are in line with the introduction of each of the chosen sorting algorithms.

Example 32 Conclusion

Extra Scripting and notes

A second Jupyter notebook cta-sorts.ipynb is included to demonstrate the bespoke diagrams run in code and the outputs at the different stages. The code cta-sorts.ipynb is a reference in the examples 5 – 10

A third script is included square.py used in Examples 1 – 3

Requirement.txt contains the required imports for the Jupyter notebook.

References [1] Mannion, P., 2022. Sorting Algorithms Part 1

[2] John D. Cook | Applied Mathematics Consulting. 2022. Sorting. [online] Available at: >https://www.johndcook.com/blog/2011/07/04/sorting/< [3] square, P. and Parikh, J., 2022. Python for loops program with square. [online] Stack Overflow. Available at: >https://stackoverflow.com/questions/41876325/python-for-loops-program-with-square< [4] Studytonight.com. 2022. Time Complexity of Algorithms |

100%	MATCHING BLOCK 21/28	SA	submission.pdf (D136586566)
Studytonight. [online] Available at: >https://www.studytonight.com/data-structures/			

time-complexity-of-algorithms< [5] Mannion, P., 2022. Sorting Algorithms Part 2 [6] GeeksforGeeks. 2022.

In-Place Algorithm - GeeksforGeeks. [online] Available at: >

100%	**MATCHING BLOCK 27/28**	SA	submission.docx (D136590819)

https://www.geeksforgeeks.org/in-place-algorithm/< [7] Mannion, P., 2022. Sorting Algorithms Part 3 [8]

Mannion, P., 2022. CTA_Project. [9] Simplilearn.com. 2022. Time and Space complexity of

44%	**MATCHING BLOCK 22/28**	SA	submission.pdf (D136586566)

Bubble Sort [online] Available at: >https://www.simplilearn.com/tutorials/data-structure-tutorial/bubble-sort-algorithm< [10]GeeksforGeeks. 2022. Merge Sort - GeeksforGeeks. [online] Available at: >https://www.

geeksforgeeks.org/merge-sort/< [11]Opengenus.org. 2022 . Time & Space Complexity of Merge Sort. [online] Available at: >https://iq.opengenus.org/time-complexity-of-

62%	**MATCHING BLOCK 23/28**	SA	submission.pdf (D136586566)

merge-sort/< [12]Studytonight.com. 2022. Merge Sort Algorithm | Studytonight. [online] Available at: >https://www.studytonight.com/data-structures/merge-sort< [13]GeeksforGeeks. 2022. Counting Sort - GeeksforGeeks. [online] Available at: >https://www.

geeksforgeeks.org/counting-sort/< [14]Opengenus.org. 2022 . Time & Space Complexity of Counting Sort. [online] Available at: >https://iq.opengenus.org/time-and-space-complexity-of-counting-sort/< [15]GeeksforGeeks. 2022.

80%	**MATCHING BLOCK 24/28**	SA	submission.pdf (D136586566)

Selection Sort - GeeksforGeeks. [online] Available at: >https://www.geeksforgeeks.org/stable-selection-sort/?

ref=gcse< [16]Opengenus.org. 2022 . Time & Space Complexity of Selection Sort. [online] Available at: >https://iq.opengenus.org/time-complexity-of-selection-sort/< [17]

76%	**MATCHING BLOCK 25/28**	SA	submission.pdf (D136586566)

GeeksforGeeks. 2022. Insertion Sort - GeeksforGeeks. [online] Available at: >https://www.geeksforgeeks.org/insertion-sort/< [18]

Opengenus.org. 2022 . Time & Space Complexity of Insertion Sort. [online] Available at: >https://iq.opengenus.org/insertion-sort-analysis/< [19] O'Reilly Online Learning. 2022. Python Cookbook. [online] Available at: >https://www.oreilly.com/library/view/python-cookbook/0596001673/ch01s15.html< [20] GeeksforGeeks. 2022. Python Program for Bubble

43%	**MATCHING BLOCK 26/28**	SA	submission.pdf (D136586566)

Sort - GeeksforGeeks. [online] Available at: >https://www.geeksforgeeks.org/python-program-for-bubble-sort/< [21] GeeksforGeeks. 2022. Iterative Merge Sort - GeeksforGeeks. [online] Available at: >https://www.geeksforgeeks.org/iterative-merge-sort/< [22] GeeksforGeeks. 2022. Counting Sort - GeeksforGeeks. [online] Available at: >https://www.

geeksforgeeks.org/counting-sort/< [23]

Runestone.academy. 2022. 6.8.

48% **MATCHING BLOCK 28/28** SA submission.pdf (D104742236)

The Selection Sort — Problem Solving with Algorithms and Data Structures 3rd edition. [online] Available at: >https://runestone.academy/ns/books/published/pythonds3/SortSearch/TheSelectionSort.html< [24] Runestone.academy. 2022. 6.9. The Insertion Sort — Problem Solving with Algorithms and Data Structures 3rd edition. [online] Available at: >https://runestone.academy/ns/books/published/pythonds3/SortSearch/

TheInsertionSort.html< [25]

Pandas.pydata.org. 2022. pandas.DataFrame.plot — pandas 0.23.1 documentation. [online] Available at: >https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.DataFrame.plot.html< [26]Pandas.pydata.org. 2022. pandas.DataFrame.plot.line — pandas 1.4.2 documentation. [online] Available at: >https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.line.html<

Hit and source - focused comparison, Side by Side

Submitted text	As student entered the text in the submitted document.
Matching text	As the text appears in the source.

1/28	SUBMITTED TEXT 14 WORDS	88% MATCHING TEXT 14 WORDS
	Big O notation is the most common metric used for calculating time complexity.	Big O notation is the most common metric for calculating time complexity.

SA submission.pdf (D104582482)

2/28	SUBMITTED TEXT 44 WORDS	98% MATCHING TEXT 44 WORDS
	Definition: In-place means that the algorithm does not use extra space for manipulating the input but may require a small though non-constant extra space for its operation. Usually, this space is O(log n), though sometimes anything in O(n) (Smaller than linear) is allowed. [6]	definition is, In-place means that the algorithm does not use extra space for manipulating the input but may require a small though non-constant extra space for its operation. Usually, this space is O(log n), though sometimes anything in O(n) (Smaller than linear) is allowed.

W https://www.geeksforgeeks.org/in-place-algorithm/

3/28	SUBMITTED TEXT 40 WORDS	69% MATCHING TEXT 40 WORDS
	the data. Non-comparison sorts can achieve linear running time in the best case but are less flexible.	the data to be sorted. Non comparison sorting algorithms can achieve better running time in the best case but they are less flexible.

J 59cd1872-5345-4692-bdd7-bd9c6383dda5

4/28	SUBMITTED TEXT 18 WORDS	88% MATCHING TEXT 18 WORDS
	their space and time complexity, how each algorithm works using bespoke diagrams and different example input instances. [8]	their space and time complexity, and explain how each algorithm works using your own bespoke diagrams and different example input instances.

SA submission.docx (D136218652)

5/28	SUBMITTED TEXT	59 WORDS	29% MATCHING TEXT	59 WORDS
comparison-based sort. 2. Merge Sort as an efficient comparison-based sort 3. Counting Sort as a non-comparison sort 4. Selection Sort is a simple comparison-based sort. 5. Insertion Sort is a simple comparison-based sort. Bubble Sort is a simple comparison-based sort. Bubble Sort is a comparison based sort which compares two elements, swapping them and continuing until they are in			Comparison based sort 2) Merge Sort Algorithm-Efficient comparison based sort 3) Bucket Sort Algorithm-Non comparison sort 4) Insertion Sort Algorithm-Comparison based sort 5) Count Sort Algorithm-Non comparison based sort Bubble-Sort Algorithm: The bubble sort algorithm is an example a comparison based algorithm in each pair of adjacent elements are compared and if they are not in	

SA submission.docx (D136572806)

7/28	SUBMITTED TEXT	25 WORDS	58% MATCHING TEXT	25 WORDS
Worst Case Time Complexity of Bubble Sort is O(N^2) .◦ This is the case when the array is			Worst Case Time Complexity of Selection Sort The worst case is the case when the array is	

W https://iq.opengenus.org/time-complexity-of-selection-sort/

6/28	SUBMITTED TEXT	21 WORDS	73% MATCHING TEXT	21 WORDS
Best Case Time Complexity of Bubble Sort is O(N) .◦ This case occurs when the			best-case time complexity of bucket sort is O(n + k). • case complexity - occurs when the	

SA submission.docx (D136572806)

8/28	SUBMITTED TEXT	18 WORDS	90% MATCHING TEXT	18 WORDS
O(N^2) .◦ The number of comparisons is constant in Bubble Sort			O(n 2). Average Case: The number of comparisons is constant in Bubble Sort,	

SA submission.pdf (D135762562)

9/28	SUBMITTED TEXT	12 WORDS	75% MATCHING TEXT	12 WORDS
is an efficient comparison-based sort Merge Sort is a recursive divide-and-conquer			is an efficient comparison-based sort. It is similar to Merge Sort in that it is a recursive divide and conquer	

SA submission.pdf (D104980609)

Curiginal

11/28	SUBMITTED TEXT 52 WORDS	95% MATCHING TEXT 52 WORDS
	Merge Sort is O(n*Log n) in all the 3 cases (worst, average and best) as merge sort always divides the array into two halves and takes linear time to merge two halves.	Merge Sort is O(n*Log n) in all the 3 cases (worst, average and best) as merge sort always divides the array in two halves and takes linear time to merge two halves. •
W	https://www.studytonight.com/data-structures/merge-sort	

14/28	SUBMITTED TEXT 13 WORDS	100% MATCHING TEXT 13 WORDS
	range k of the elements is significantly larger than the other elements	range k of the elements is significantly larger than the other elements.
W	https://iq.opengenus.org/time-and-space-complexity-of-counting-sort/	

10/28	SUBMITTED TEXT 18 WORDS	57% MATCHING TEXT 18 WORDS
	a simple comparison-based sort. A Selection Sort essentially finds the minimum element in an array and inserts it	A Simple Comparison Based Sort) Selection Sort takes the smallest element in an unsorted array and brings it
SA	submission.pdf (D135762562)	

12/28	SUBMITTED TEXT 34 WORDS	44% MATCHING TEXT 34 WORDS
	Worst Case Time Complexity of Selection Sort is O(N^2) .◦ This is the case if the given array is sorted except for the last element is the	Worst Case Time Complexity of Selection Sort The worst case is the case when the array is already sorted (with one swap) but the smallest element is the
W	https://iq.opengenus.org/time-complexity-of-selection-sort/	

13/28	SUBMITTED TEXT 27 WORDS	55% MATCHING TEXT 27 WORDS
	Best Case Time Complexity of Selection Sort is O(N^2) .◦ This case occurs when the given array is already sorted. .•	Best Case Time Complexity of Selection Sort The best case is the case the array is already sorted.
W	https://iq.opengenus.org/time-complexity-of-selection-sort/	

15/28	SUBMITTED TEXT 15 WORDS	80% MATCHING TEXT 15 WORDS
	space by shifting the element to right and inserting the element at the appropriate position.	space by shifting the elements to right and insert the element at the appropriate position. 4.
SA	submission.pdf (D135762562)	

17/28	SUBMITTED TEXT 17 WORDS	100% MATCHING TEXT 17 WORDS
	Worst Case Time Complexity of Insertion Sort is O(N^2) .◦	worst case time complexity of Insertion sort is O(N^2) •

W https://iq.opengenus.org/insertion-sort-analysis/

16/28	SUBMITTED TEXT 27 WORDS	55% MATCHING TEXT 27 WORDS
	the array is reversely sorted .• Best Case Time Complexity of Insertion Sort is O(N) .◦ This case occurs when the	the array is already sorted. best-case time complexity of counting sort is Ω(n + k). • Worse Case: This is when the

SA submission.pdf (D135762562)

18/28	SUBMITTED TEXT 18 WORDS	100% MATCHING TEXT 18 WORDS
	Average Case Time Complexity of Insertion Sort is O(N^2) .◦ The	average case time complexity of Insertion sort is O(N^2) • The

W https://iq.opengenus.org/insertion-sort-analysis/

20/28	SUBMITTED TEXT 39 WORDS	44% MATCHING TEXT 39 WORDS
	simple comparison-based sort [20] .• Merge Sort is an efficient comparison-based sort [21] .• Counting Sort as a non-comparison sort [22] .• Selection Sort is a simple comparison-based sort. [23] .• Insertion Sort is a simple comparison-based sort. [24]	Simple comparison-based sort (Bubble Sort) .... 7 2.2 Efficient comparison-based sort (Quicksort).. 8 2.3 Non-comparison sort (Counting Sort) .... 5 2.4 Simple comparison-based sort (Selection Sort) .... 10 2.5 Simple comparison-based sort (

SA submission.pdf (D104742236)

19/28	SUBMITTED TEXT 19 WORDS	75% MATCHING TEXT 19 WORDS
	Non-comparison sort can achieve linear running time in the best case but are less flexible,	Non comparison sorting algorithms can achieve better running time in the best case but they are less flexible.

J 59cd1872-5345-4692-bdd7-bd9c6383dda5

21/28	SUBMITTED TEXT	5 WORDS	100% MATCHING TEXT	5 WORDS
Studytonight. [online] Available at: >https://www.studytonight.com/data-structures/			Studytonight. [online] Available at: >https://www.studytonight.com/data-structures/	

SA submission.pdf (D136586566)

27/28	SUBMITTED TEXT	10 WORDS	100% MATCHING TEXT	10 WORDS
https://www.geeksforgeeks.org/in-place-algorithm/< [7] Mannion, P., 2022. Sorting Algorithms Part 3 [8]			https://www.geeksforgeeks.org/in-place-algorithm/ Mannion, P (2022). Sorting Algorithms Part 2.	

SA submission.docx (D136590819)

22/28	SUBMITTED TEXT	16 WORDS	44% MATCHING TEXT	16 WORDS
Bubble Sort [online] Available at: >https://www.simplilearn.com/tutorials/data-structure-tutorial/bubble-sort-algorithm< [10]GeeksforGeeks. 2022. Merge Sort - GeeksforGeeks. [online] Available at: >https://www.			Bubble Sort Algorithm | Studytonight. [online] Available at: >https://www.studytonight.com/data-structures/bubble-sort< w3resource. 2022. Python: Insertion sort - w3resource. [online] Available at: >https://www.	

SA submission.pdf (D136586566)

23/28	SUBMITTED TEXT	21 WORDS	62% MATCHING TEXT	21 WORDS
merge-sort/< [12]Studytonight.com. 2022. Merge Sort Algorithm | Studytonight. [online] Available at: >https://www.studytonight.com/data-structures/merge-sort< [13]GeeksforGeeks. 2022. Counting Sort - GeeksforGeeks. [online] Available at: >https://www.			merge%20sort%20on%20halves.< Studytonight.com. 2022. Sort Algorithm | online] Available at: >https://www.studytonight.com/data-structures/bubble-sort< w3Python: Insertion sort - resource. [online] Available at: >https://www.	

SA submission.pdf (D136586566)

24/28	SUBMITTED TEXT	8 WORDS	80% MATCHING TEXT	8 WORDS
Selection Sort - GeeksforGeeks. [online] Available at: >https://www.geeksforgeeks.org/stable-selection-sort/?			Selection Sort - GeeksforGeeks. [online] Available at: >https://www.geeksforgeeks.org/python-program-for-selection-sort/<	

SA submission.pdf (D136586566)

25/28	SUBMITTED TEXT 11 WORDS	76% MATCHING TEXT 11 WORDS
	GeeksforGeeks. 2022. Insertion Sort - GeeksforGeeks. [online] Available at: >https://www.geeksforgeeks.org/insertion-sort/< [18]	GeeksforGeeks. 2022. Counting Sort - GeeksforGeeks. [online] Available at: >https://www.geeksforgeeks.org/counting-sort/<
	SA submission.pdf (D136586566)	

26/28	SUBMITTED TEXT 30 WORDS	43% MATCHING TEXT 30 WORDS
	Sort - GeeksforGeeks. [online] Available at: >https://www.geeksforgeeks.org/python-program-for-bubble-sort/< [21] GeeksforGeeks. 2022. Iterative Merge Sort - GeeksforGeeks. [online] Available at: >https://www.geeksforgeeks.org/iterative-merge-sort/< [22] GeeksforGeeks. 2022. Counting Sort - GeeksforGeeks. [online] Available at: >https://www.	Sort - GeeksforGeeks. [online] Available at: >https://www.geeksforgeeks.org/python-program-for-selection-sort/< Bubble Sort Algorithm with Python using List Example. [online] Available at: >https://www.guru99.com/bubble-sort.HackerEarth. 2022. Merge Sort visualize | Sorting | Algorithms | HackerEarth. [online] Available at: https://www.
	SA submission.pdf (D136586566)	

28/28	SUBMITTED TEXT 38 WORDS	48% MATCHING TEXT 38 WORDS
	The Selection Sort — Problem Solving with Algorithms and Data Structures 3rd edition. [online] Available at: >https://runestone.academy/ns/books/published/pythonds3/SortSearch/TheSelectionSort.html< [24] Runestone.academy. 2022. 6.9. The Insertion Sort — Problem Solving with Algorithms and Data Structures 3rd edition. [online] Available at: >https://runestone.academy/ns/books/published/pythonds3/SortSearch/	The Bubble Sort — Problem Solving with Algorithms and Data Structures (date).Available at: https://runestone.academy/runestone/books/published/pythonds/SortSearch/TheBubbleSort.html (Accessed: 9 The Selection Sort — Problem Solving with Algorithms and Data Structures (date). Available at: https://runestone.academy/runestone/books/published/pythonds/SortSearch/
	SA submission.pdf (D104742236)	