# ADA Final Project

*"Sorting Algorithm Analysis"*

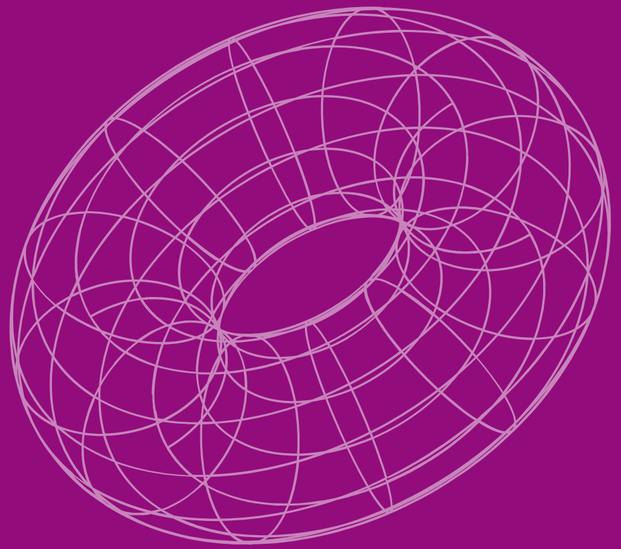
Presented by :

- Alexander Tjiandra
- Kevin Matthew T.
- Sulthan

# BACKGROUND

Sorting Algorithm is used to rearrange a given array or list of elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure.

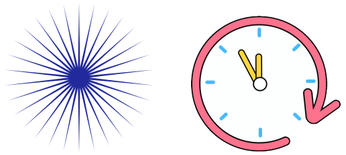
# Problem Statement



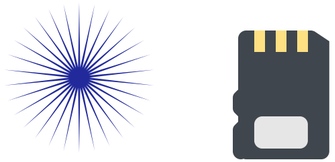
When it comes to sorting, there are several options that are available for us to use, but not all sorting algorithms are made equal. Each have pros and cons and our project seeks to analyze 3 out of the many sorting algorithms and compare them.

# Measured content

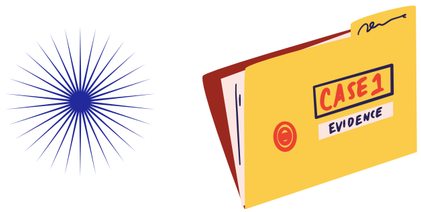
What will be measured/compared?



**Time Complexity**



**Space Complexity**




**Case by case scenario**

# Proposed Solution using Python Library

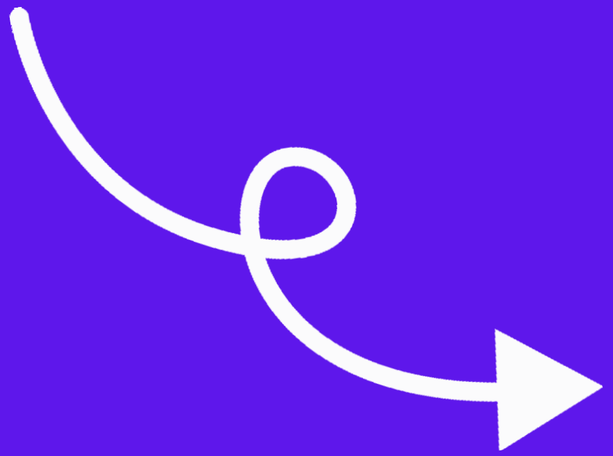
## Tracemalloc

A module is a debug tool to trace memory blocks allocated by Python

## Python Time Library

This module provides various time-related functions.

# Let's jump to the Result



# Algorithm - Sort Algorithm

## Insertion Sort

is a simple sorting algorithm that works similar to the way you sort playing cards in your hands.

## Quick Sort

is a sorting algorithm that use divide and conquer method.

## Bubble Sort

is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order.

**RESULT AND DISCUSSION**

**LIST EXAMPLES USED AND THE RESULTS OF EACH LIST (SMALL CASE)**

1. Unsorted list

- List = [3, 2, 4, 1, 5, 6]

| | Time Complexity | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion Sort | 0.01170000177808106 | (224, 674) |
| Bubble Sort | 0.016700010746717453 | (224, 674) |
| Quick Sort | 0.024200009647756815 | (2048, 2520) |

2. Partially sorted list

- List = [1, 2, 5, 3, 4, 6]

| | Time Complexity | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion Sort | 0.013400014722719789 | (224, 674) |
| Bubble Sort | 0.017099984688684344 | (224, 674) |
| Quick Sort | 0.021299987565726042 | (1592, 2064) |

3. Middle sorted list

- List = [6, 2, 3, 4, 5, 1]

| | Time Complexity | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion Sort | 0.010199990356341004 | (224, 674) |
| Bubble Sort | 0.012500007869675756 | (224, 674) |
| Quick Sort | 0.03640001523308456 | (1592, 2064) |

4. Almost sorted list

- List = [1, 2, 3, 4, 6, 5]

| | Time Complexity | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion Sort | 0.0137000170070678 | (224, 674) |
| Bubble Sort | 0.015900004655122757 | (224, 674) |
| Quick Sort | 0.02060001133941114 | (1592, 2064) |

# SMALLER CASES

# MIDDLE CASES

**LIST EXAMPLES USED AND THE RESULTS OF EACH LIST (MIDDLE CASE)**

1. Unsorted list

- List = [8 5 3 10 4 11 20 9 15 1]

| | Time Complexity | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion Sort | 0.01479999627918005 milliseconds | (224, 706) |
| Bubble Sort | 0.022899999748915434 milliseconds | (224, 706) |
| Quick Sort | 0.023600063286721706 | (2104, 2640) |

| | millisecond | |
|---|---|---|

- List = [5 2 15 10 6 20 37 15 12 7 8 3 14 16 11 30 27 25 23 35]

| | Time Complexity | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion Sort | 0.017799960914999247 milliseconds | (224, 786) |
| Bubble Sort | 0.02770003629848361 milliseconds | (224, 786) |
| Quick Sort | 0.0393999507650733 milliseconds | **(3184, 3880)** |

2. Partially sorted list

- List=[1 5 7 20 11 2 3 12 19 10 ]

| | Time Complexity | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion Sort | 0.011999974958598614 milliseconds | (224, 708) |
| Bubble Sort | 0.03819999983534217 milliseconds | (224, 708) |

| | | |
|---|---|---|
| Quick Sort | 0.04039995837956667 milliseconds | (1136, 1674) |

- List=[1 5 8 10 13 14 17 24 20 56 21 73 45 53 24 15 40 32 30 25 ]

| | Time Complexity | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion Sort | 0.015900004655122757 milliseconds | (224, 792) |
| Bubble Sort | 0.03960001049563289 milliseconds | (224, 792) |
| Quick Sort | 0.03200001083314419 milliseconds | (3296, 3998) |

3. Middle sorted list

- List=[21 18 24 12 14 17 20 3 5 1]

| | Time Complexity | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion Sort | 0.014499993994832039 milliseconds | (224, 712) |
| Bubble Sort | 0.019200029782950878 milliseconds | (224, 712) |

| | | |
|---|---|---|
| Quick Sort | 0.017900019884109497 milliseconds | (2104, 2646) |

- List=[1 38 14 52 43 15 16 20 25 27 30 35 37 8 4 2 40 12 10 7 ]

| | Time Complexity | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion Sort | 0.041599967516958714 milliseconds | (224, 788) |
| Bubble Sort | 0.07040001219138503 milliseconds | (224, 788) |
| Quick Sort | 0.03720005042850971 milliseconds | (6136, 7024) |

4. Almost sorted

- List=[31 16 15 17 18 19 24 30 32 5 ]

| | Time Complexity | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion Sort | 0.011499971151351929 milliseconds | (224, 716) |
| Bubble Sort | 0.017199956346303225 milliseconds | (224, 716) |

| | | |
|---|---|---|
| Quick Sort | 0.03320001997053623 milliseconds | (3752, 4298) |

- List=[51 47 15 17 20 23 25 26 28 29 30 35 37 40 13 10 5 4 57 60]

| | Time Complexity | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion Sort | 0.022299995180219412 milliseconds | (224, 794) |
| Bubble Sort | 0.03880000440403819 milliseconds | (224, 794) |
| Quick Sort | 0.06960006430745125 milliseconds | (9032, 10632) |

# BIGGER CASES

**LIST EXAMPLES USED AND THE RESULTS OF EACH LIST (BIGGER CASE)**

1. Unsorted list

- List = [17 151 95 13 47 15 42 59 137 99 74 57 83 128 29 7 63 131 97 43 148 89 81 34 121 20 24 93 103 18 92 69 10 39 109 67 163 27 55 112 107 3 120 49 52 1 79 37 45 86]

| | Time Complexity (millisecond) | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion sort | 0.13830000534653664 | (224, 1056) |
| Bubble sort | 0.23559998953714967 | (224, 1056) |
| Quick sort | 0.12099999003112316 | (4488, 5832) |

| | | |
|---|---|---|
| | | |

2. Partially sorted list

- List = [13 15 17 42 47 57 59 74 83 95 99 137 151 103 18 92 69 10 39 109 67 163 27 55 112 107 3 120 49 52 1 79 37 45 86 7 20 24 28 29 34 43 63 81 89 93 97 121 131 148]

| | Time (Millisecond) | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion sort | 0.16309999045915902 | (224, 1054) |
| Bubble sort | 0.1773000112734735 | (224, 1054) |
| Quick sort | 0.20070001482963562 | (10056, 13992) |

3. Middle sorted list

- List = [17 151 95 13 47 15 42 59 137 7 10 18 20 24 27 29 34 39 43 57 63 67 69 74 81 83 89 92 93 97 99 103 109 121 128 131 148 163 55 112 107 3 120 49 52 1 79 37 45 86]

| | Time (millisecond ) | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion sort | 0.10830000974237919 | (224, 1056) |
| Bubble sort | 0.14980000560171902 | (224, 1056) |
| Quick sort | 0.10539998766034842 | (6704, 8376) |

4. Almost sorted list

- List = [7 10 13 15 17 18 20 24 27 29 34 39 42 43 47 55 57 59 63 67 69 74 81 83 89 92 93 95 97 99 103 109 121 128 131 137 148 151 163 112 107 3 120 49 52 1 79 37 45 86]

| | Time (millisecond) | Space Complexity/Memory (In Byte and peak) |
|---|---|---|
| Insertion sort | 0.045199994929134846 | (224, 1056) |
| Bubble sort | 0.1343000039923936 | (224, 1056) |
| Quicksort | 0.13010000111535192 | (7784, 9432) |

# Discussion

- Insertion sort performs better in smaller cases than Bubble sort and Quick sort.
- In middle cases, Insertion Sort also performs better. However, Quicksort shows an improvement.
- In bigger cases, both Insertion and Quick performs better and have their respective efficiency in solving problems

# Conclusion

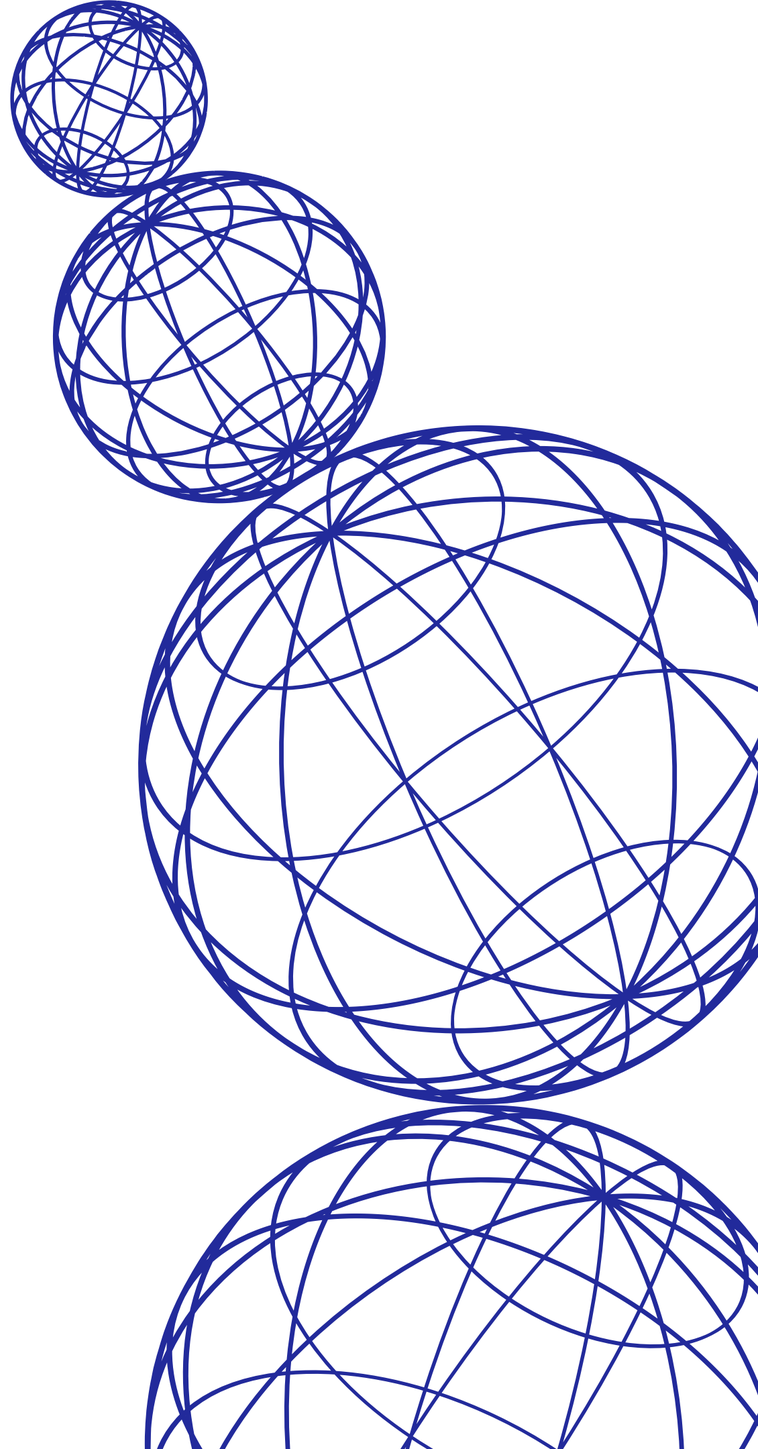

Insertion Sort

Quick Sort

Bubble Sort

# Do you have any questions?

Send it to us! We hope you learned something new.



# THANKYOU FOR LISTENING