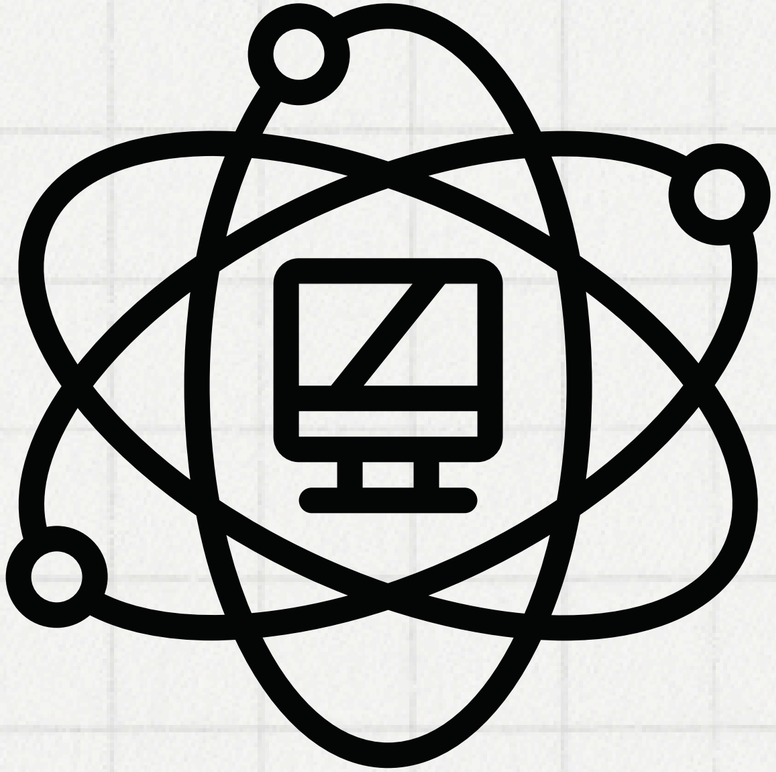
# SORTING ALGORITHM VISUALIZER IN PYTHON

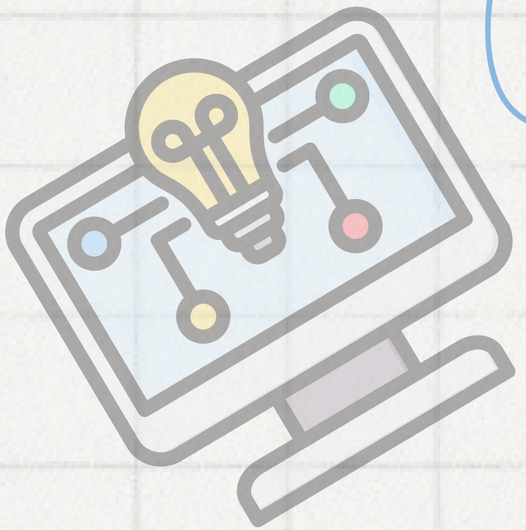

Presented by

- SATYABRATA MOHANTY
- MANSI SHARMA
- ASHUTOSH KUMAR RAI

# WHY SORTING ALGORITHM MATTERS ?

Sorting algorithms are essential in computer science as they organize data for efficient processing, making it easier to search, retrieve, and analyze. They are fundamental to optimizing performance in applications like databases, search engines, and data analysis, helping reduce time complexity and improving overall system efficiency.





# CHALLENGES OF UNDERSTANDING THE SORTING CONCEPT.

01. Abstract Nature Sorting algorithms involve complex steps like comparisons and swaps, which can be hard to visualize without practical examples.

02. Time Complexities: Different algorithims have varying time complexities, making it difficult to choose the most efficient one for specific case.

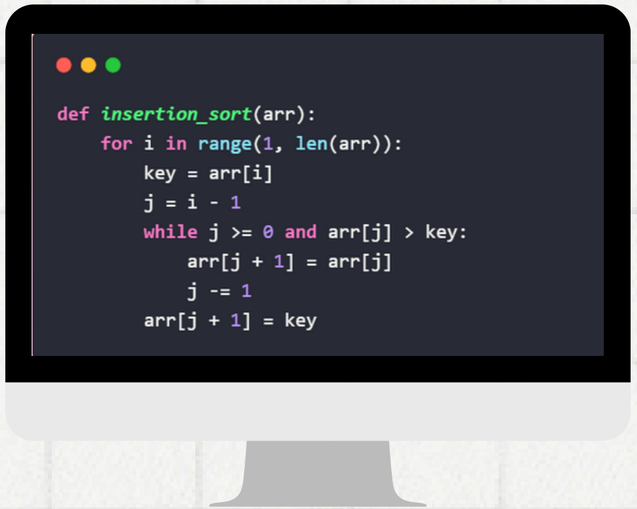
03. Visualization Benefits: Visualizing sorting process clarifies how algorithims work, aiding understanding o efficiency, stablity, and practical applications.

# Sorting Algorithm implemented.

- Bubble Sort: Compares adjacent elements, swapping them if they’re in the wrong order. Repeats until the entire list is sorted, making it simple but inefficient for large datasets.

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1]
              = arr[j + 1], arr[j]
```

- Insertion Sort: Iteratively takes one element, compares it to previous elements, and inserts it in the correct position. It's efficient for small or partially sorted datasets.



```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```

- Selection Sort: Finds the smallest element in the unsorted part of the list and swaps it with the first unsorted element, repeating the process until the list is sorted.

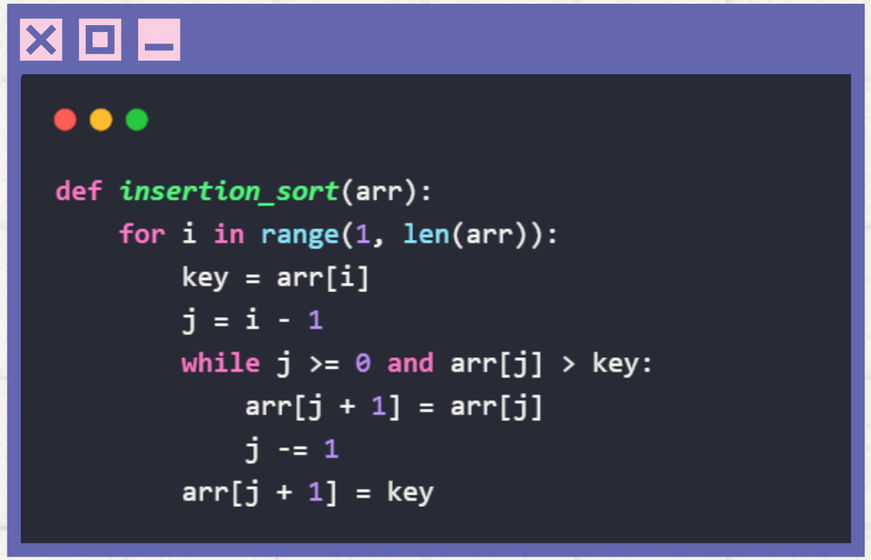
```
ef selection_sort(arr):
    for i in range(len(arr)):
        min_idx = i
        for j in range(i + 1, len(arr)):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[
```

# Visualizing Sorting with Python

Visualizing sorting algorithms with Python's `matplotlib.animation` helps demystify complex processes. By animating each step, you can see how elements are compared, swapped, or inserted in real-time, enhancing understanding of algorithm behavior and efficiency, making it easier to grasp and analyze sorting techniques effectively.

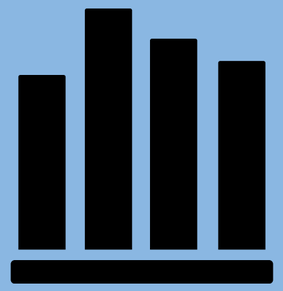
# Code Explained

The visualizer combines sorting algorithms with matplotlib.animation to animate the sorting process. Each algorithm, like Insertion Sort, uses yield to produce the array's state step-by-step. This allows users to observe sorting in real-time, enhancing comprehension.



```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```

# How the Visualization works.

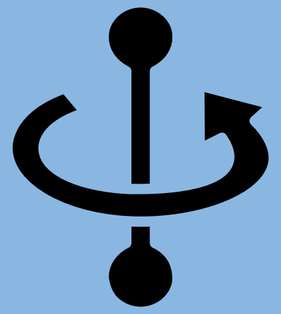


Bar Heights: Bars represent array values. As sorting progresses, their heights change to reflect updates, showing the algorithm's effect visually.

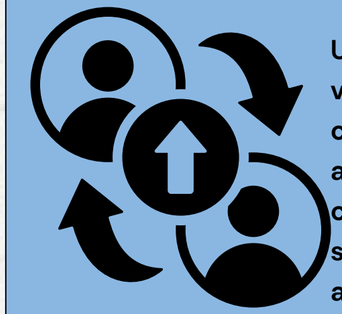
Algorithm Yielding: Each sorting algorithm yields the array's state at each step, allowing matplotlib to capture and animate the changes dynamically.

Real-Time Update: FuncAnimation updates the bar heights in real-time, synchronizing with the sorting algorithm's progress for clear step-by-step visualization.

Iteration Count: An on-screen counter displays the number of operations performed, helping viewers understand the algorithm's efficiency and complexity.



Dynamic Axis Limits: Axis limits adjust automatically based on the data range, ensuring all bars are visible and properly scaled throughout the animation.



User Interaction: The visualization allows users to choose between different algorithms, providing a comparative view of how each sorting method processes the array.

# APPLICATIONS AND USE CASES.

Sorting visualizations are crucial for educational purposes, helping learners grasp complex algorithms. They also aid in debugging and optimizing code, and offer insights into algorithm efficiency, which is valuable for data analysis, database management, and improving system performance across various applications.

# Conclusion and Future Enhancements.

Sorting visualizations provide a clear, interactive way to understand and compare algorithms. Future improvements could include integrating more advanced algorithms like Quick Sort and Merge Sort, handling larger datasets, and incorporating user-driven parameters for enhanced analysis. Sharing this tool with the community could gather feedback and inspire further development, advancing educational and practical applications.

# Thank you very much!

codeforcoders.tech