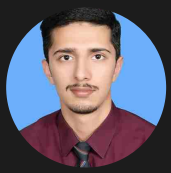
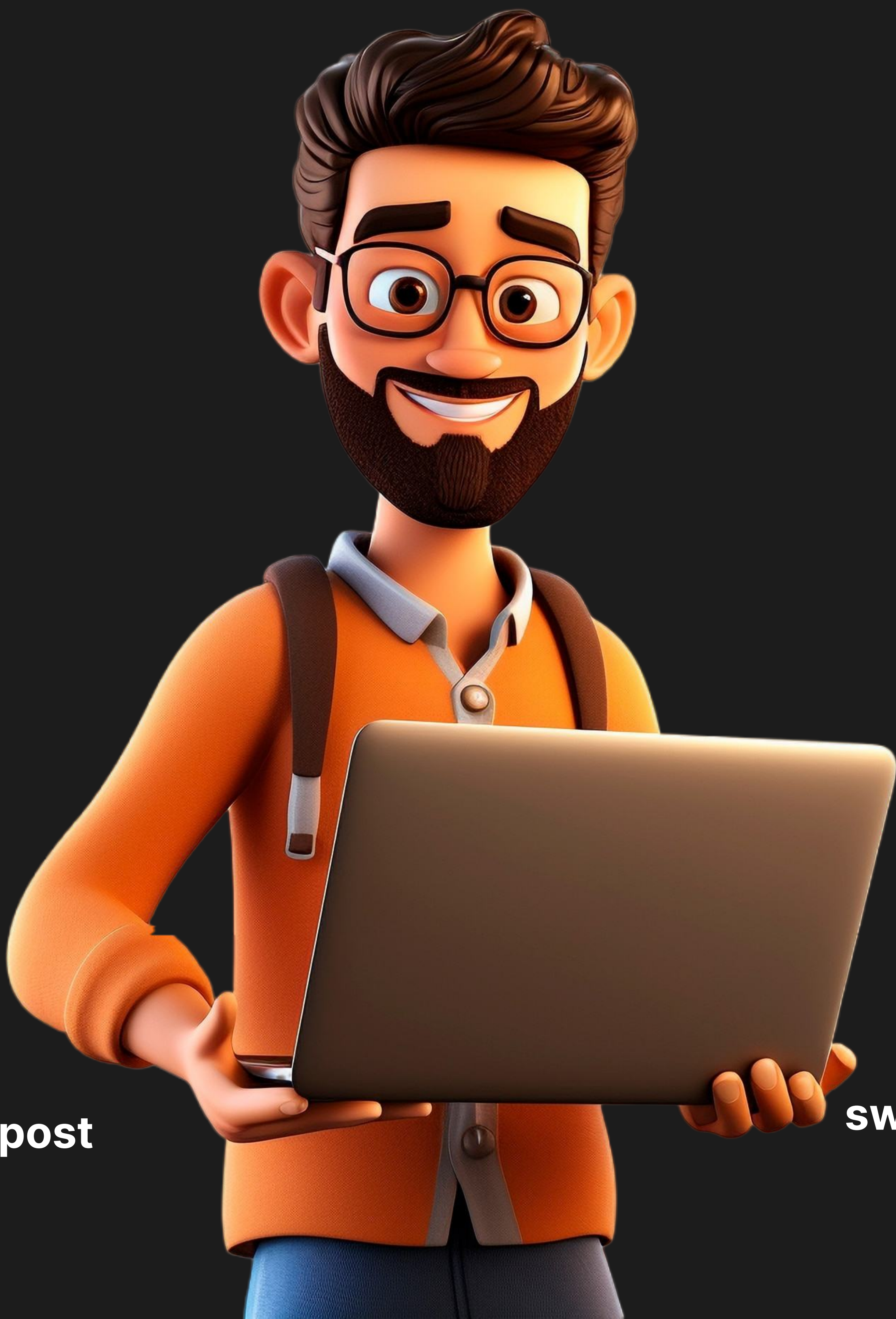


Coding Expert

data structure / algorithm

Algorithms Every Developer Should Know



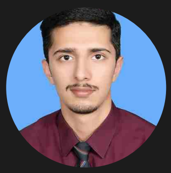
save post

swipe to know

Sorting Algorithm

Sorting algorithms are used to arrange elements in a particular order (either ascending or descending) within a data structure.

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Merge Sort
5. Quick Sort
6. Heap Sort



Coding Expert

data structure / algorithm

Bubble Sort

Bubble Sort is a simple comparison-based algorithm where each pair of adjacent elements is compared and swapped if they are in the wrong order.

Time Complexity:

- Worst-case: O(n^2)
- Best-case: O(n) (when the list is already sorted)

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

```

save post

swipe to know

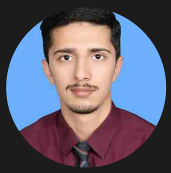
Selection Sort

Selection Sort is a comparison-based algorithm that divides the input list into two parts: the sorted part at the left end and the unsorted part at the right end.

Time Complexity:

- **Worst-case: O(n^2)**
- Best-case: O(n^2)

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

```



Coding Expert

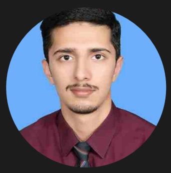
Insertion Sort

Insertion Sort is a algorithm that builds the sorted list one element at a time by repeatedly taking the next element and inserting it into the correct position.

Time Complexity:

- Worst-case: O(n^2)
- Best-case: O(n) (when the list is already sorted)

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



Coding Expert

data structure / algorithm

Merge Sort

Merge Sort is a divide-and-conquer algorithm that divides the input array into two halves, recursively sorts each half, and then merges the two sorted halves to produce the final sorted array.

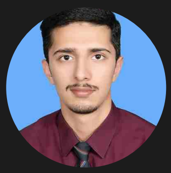
Time Complexity:

- Worst-case: O(n log n)
- Best-case: O(n log n)

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        merge_sort(left_half)
        merge_sort(right_half)

        i = j = k = 0

        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1
```

save post

swipe to know



Coding Expert

data structure / algorithm

Quick Sort

Quick Sort is a divide-and-conquer algorithm that selects a 'pivot' element from the array and partitions the other elements into two sub-arrays. The sub-arrays are then sorted recursively.

Time Complexity:

- Worst-case: O(n^2) (though rare with good pivot selection)
- Best-case: O(n log n)

```python
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[len(arr) // 2]
        left = [x for x in arr if x < pivot]
        middle = [x for x in arr if x == pivot]
        right = [x for x in arr if x > pivot]
        return quick_sort(left) + middle + quick_sort(right)

```

save post

swipe to know

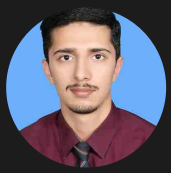
Heap Sort

Heap Sort that uses a binary heap data structure. It builds a max heap from the input data, then repeatedly extracts the max element from the heap and rebuild the heap until the array is sorted.

Time Complexity:

- Worst-case: O(n log n)
- Best-case: O(n log n)

```python
def heapify(arr, n, i):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2

    if left < n and arr[i] < arr[left]:
        largest = left

    if right < n and arr[largest] < arr[right]:
        largest = right

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

def heap_sort(arr):
    n = len(arr)

    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)

```



Keep Learning

PS: Remembers these are just introduction of advanced journey with Data structure.
There’s always more to learn and explore so keep coding and keep growing !