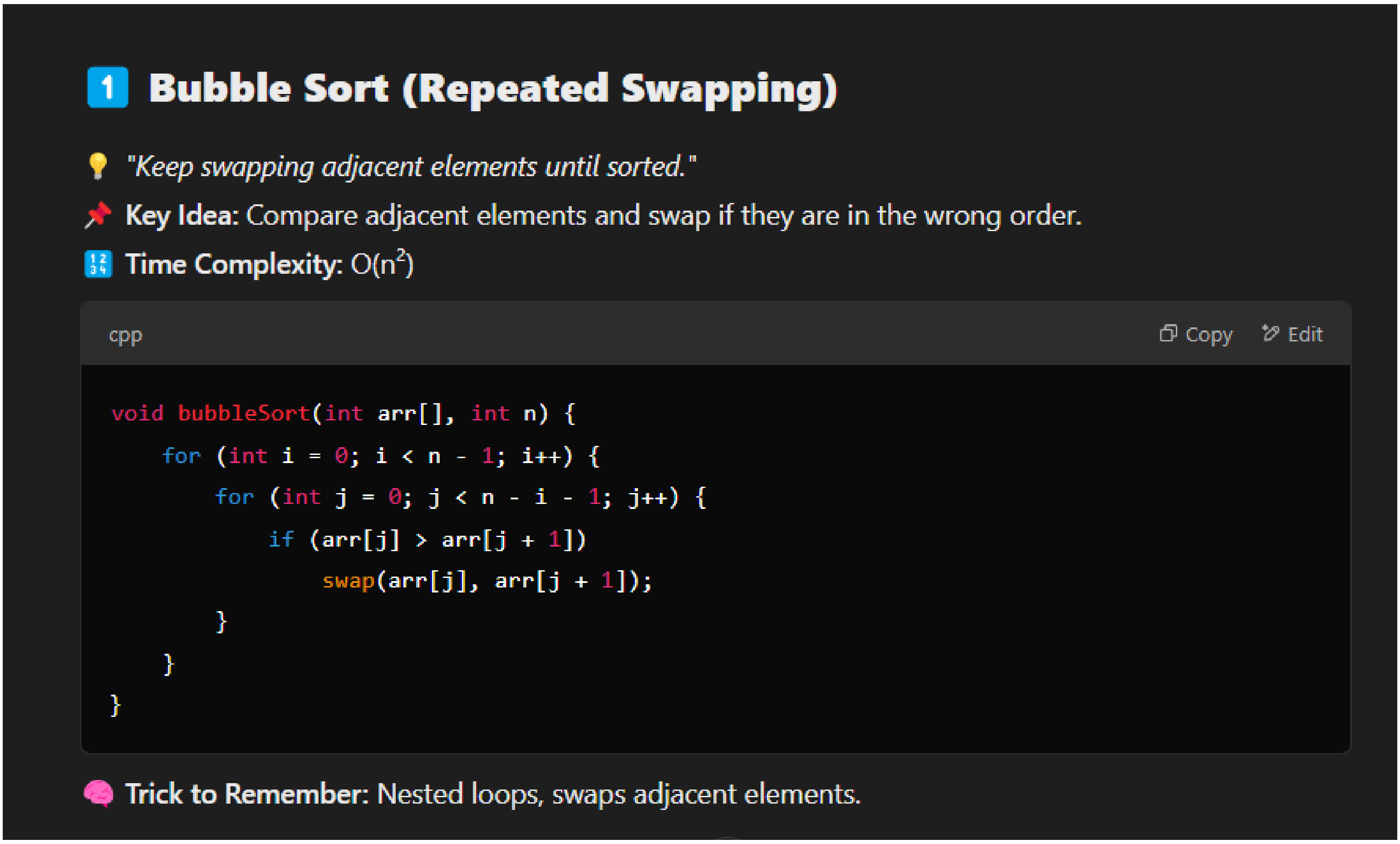


1️⃣ Bubble Sort (Repeated Swapping)

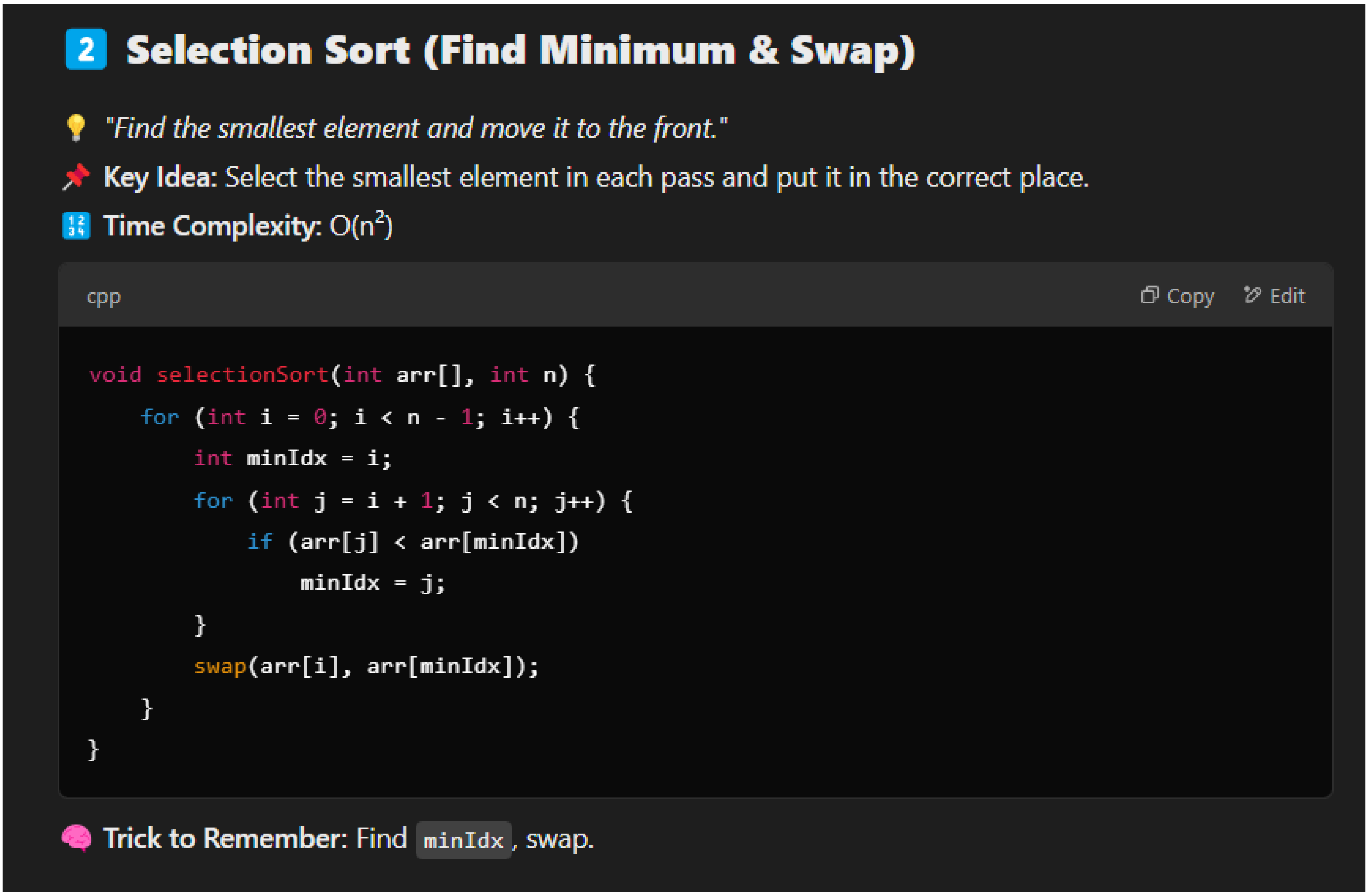
💡 *"Keep swapping adjacent elements until sorted."*

📌 **Key Idea:** Compare adjacent elements and swap if they are in the wrong order.

🔢 **Time Complexity:** O(n^2)

```cpp
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
        }
    }
}
```

🧠 **Trick to Remember:** Nested loops, swaps adjacent elements.



2️⃣ Selection Sort (Find Minimum & Swap)

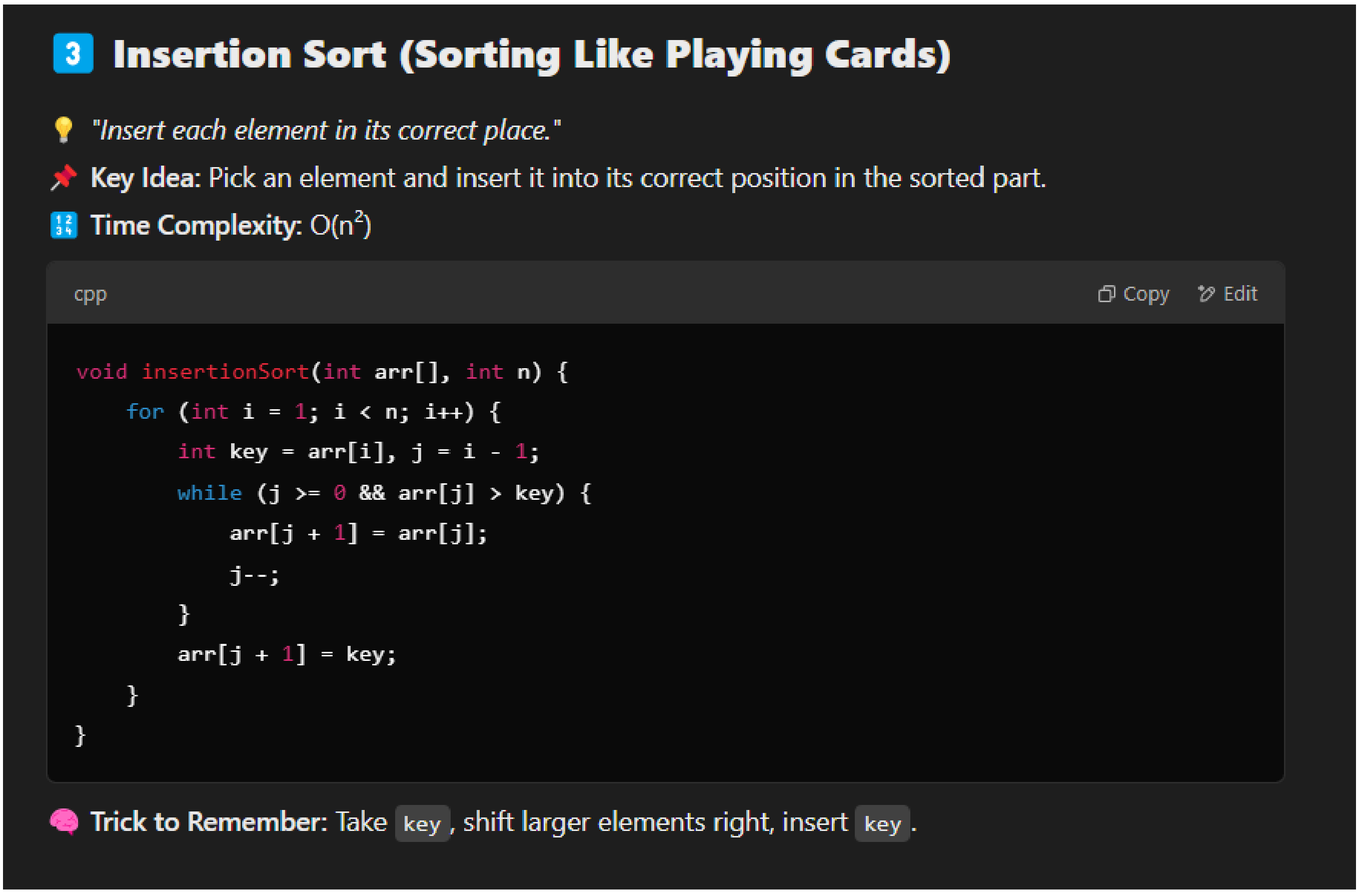
💡 *"Find the smallest element and move it to the front."*

📌 **Key Idea:** Select the smallest element in each pass and put it in the correct place.

🔢 **Time Complexity:** O(n^2)

```cpp
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIdx])
                minIdx = j;
        }
        swap(arr[i], arr[minIdx]);
    }
}
```

🧠 **Trick to Remember:** Find `minIdx`, swap.



3️⃣ Insertion Sort (Sorting Like Playing Cards)

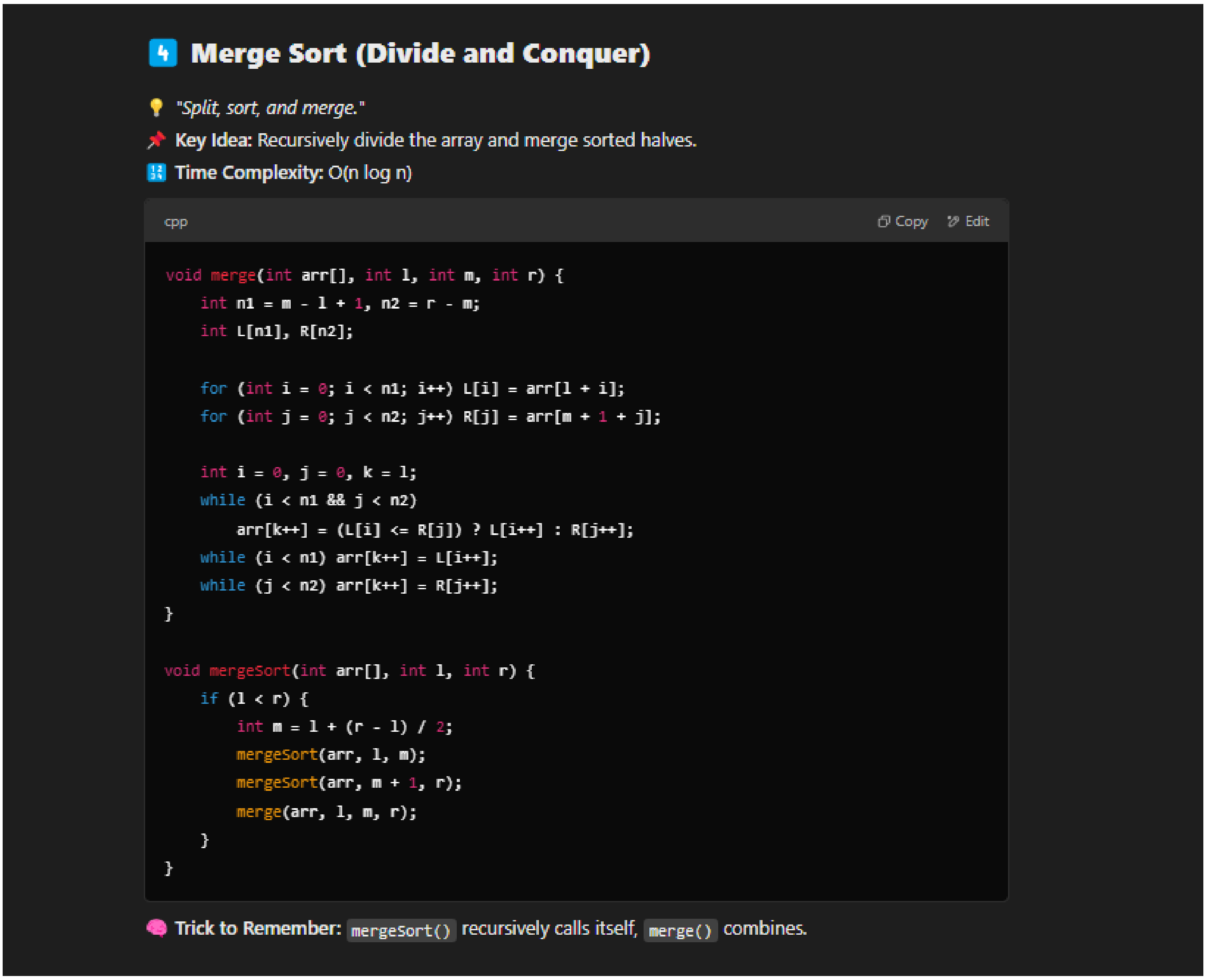
💡 *"Insert each element in its correct place."*

📌 **Key Idea:** Pick an element and insert it into its correct position in the sorted part.

🔢 **Time Complexity:** O(n^2)

```cpp
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i], j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
```

🧠 **Trick to Remember:** Take `key`, shift larger elements right, insert `key`.



4️⃣ Merge Sort (Divide and Conquer)

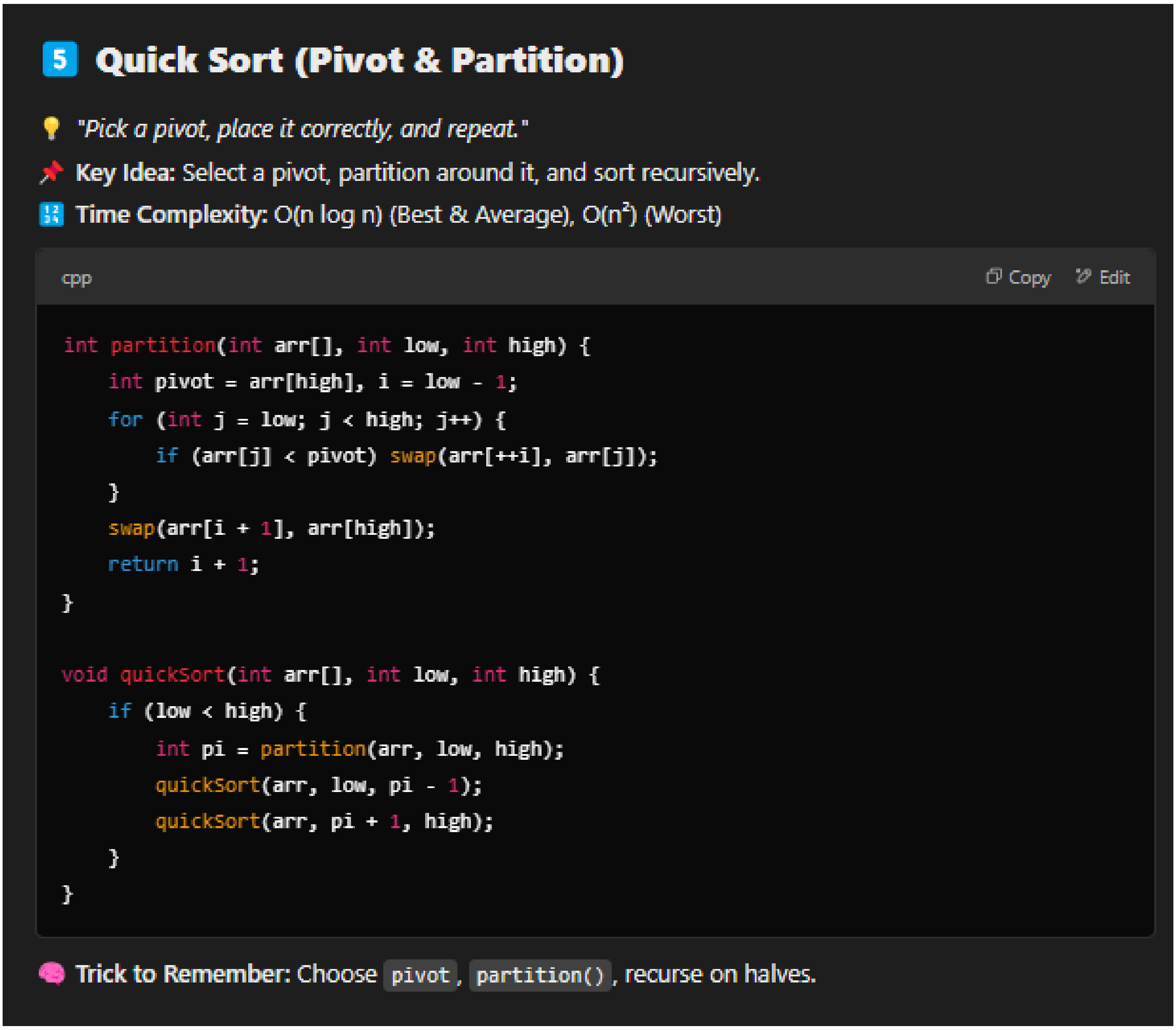
💡 *"Split, sort, and merge."*

📌 **Key Idea:** Recursively divide the array and merge sorted halves.

🔢 **Time Complexity:** O(n log n)

```cpp
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1, n2 = r - m;
    int L[n1], R[n2];

    for (int i = 0; i < n1; i++) L[i] = arr[l + i];
    for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2)
        arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
```

🧠 **Trick to Remember:** `mergeSort()` recursively calls itself, `merge()` combines.



5️⃣ Quick Sort (Pivot & Partition)

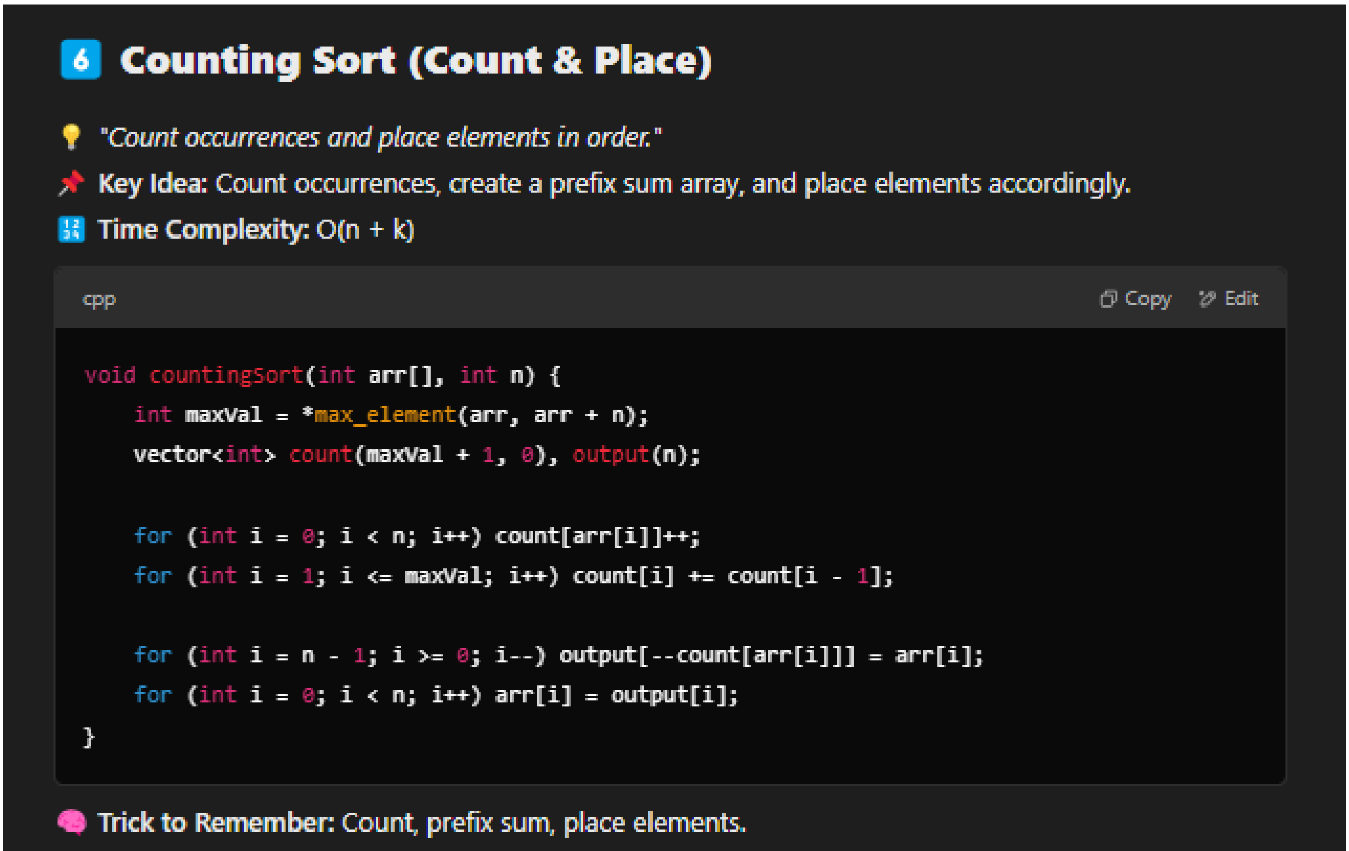
💡 *"Pick a pivot, place it correctly, and repeat."*

📌 **Key Idea:** Select a pivot, partition around it, and sort recursively.

🔢 **Time Complexity:** O(n log n) (Best & Average), $O(n^2)$ (Worst)

```cpp
int partition(int arr[], int low, int high) {
    int pivot = arr[high], i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) swap(arr[++i], arr[j]);
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

🧠 **Trick to Remember:** Choose `pivot`, `partition()`, recurse on halves.



6️⃣ Counting Sort (Count & Place)

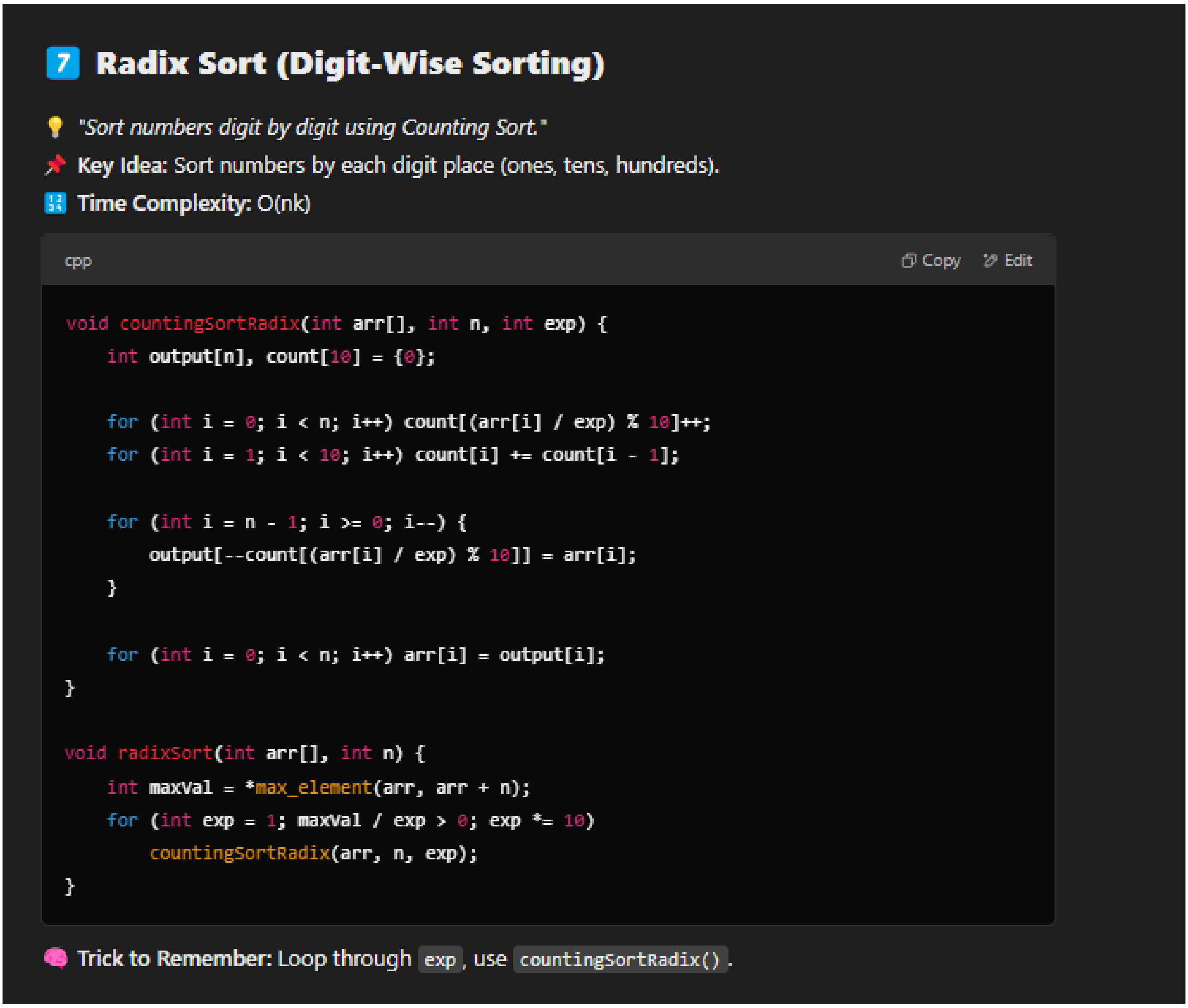
💡 *"Count occurrences and place elements in order."*

📌 **Key Idea:** Count occurrences, create a prefix sum array, and place elements accordingly.

🔢 **Time Complexity:** O(n + k)

```cpp
void countingSort(int arr[], int n) {
    int maxVal = *max_element(arr, arr + n);
    vector<int> count(maxVal + 1, 0), output(n);

    for (int i = 0; i < n; i++) count[arr[i]]++;
    for (int i = 1; i <= maxVal; i++) count[i] += count[i - 1];

    for (int i = n - 1; i >= 0; i--) output[--count[arr[i]]] = arr[i];
    for (int i = 0; i < n; i++) arr[i] = output[i];
}
```

🧠 **Trick to Remember:** Count, prefix sum, place elements.



7️⃣ Radix Sort (Digit-Wise Sorting)

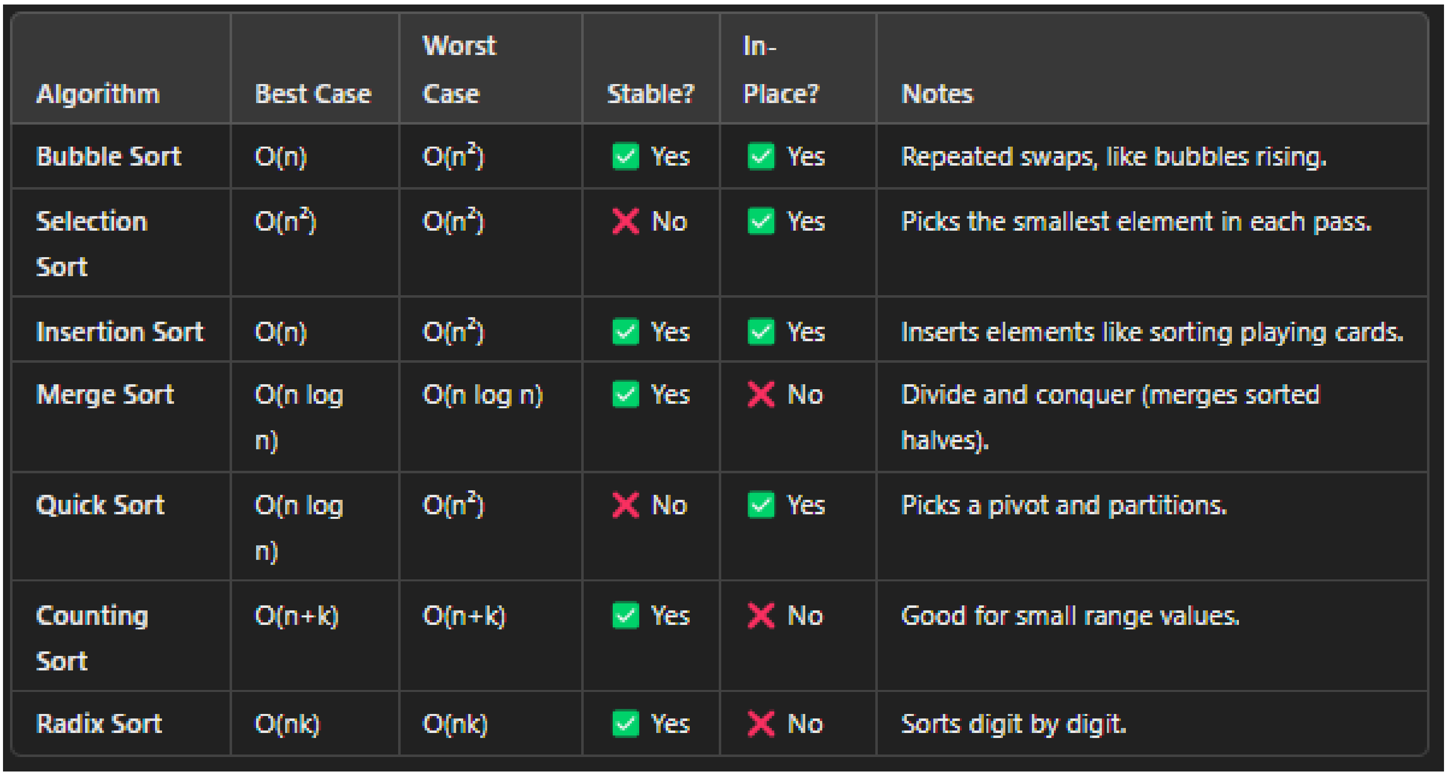
💡 *"Sort numbers digit by digit using Counting Sort."*

📌 **Key Idea:** Sort numbers by each digit place (ones, tens, hundreds).

🔢 **Time Complexity:** O(nk)

```cpp
void countingSortRadix(int arr[], int n, int exp) {
    int output[n], count[10] = {0};

    for (int i = 0; i < n; i++) count[(arr[i] / exp) % 10]++;
    for (int i = 1; i < 10; i++) count[i] += count[i - 1];

    for (int i = n - 1; i >= 0; i--) {
        output[--count[(arr[i] / exp) % 10]] = arr[i];
    }

    for (int i = 0; i < n; i++) arr[i] = output[i];
}

void radixSort(int arr[], int n) {
    int maxVal = *max_element(arr, arr + n);
    for (int exp = 1; maxVal / exp > 0; exp *= 10)
        countingSortRadix(arr, n, exp);
}
```

🧠 **Trick to Remember:** Loop through `exp`, use `countingSortRadix()`.



Algorithm	Best Case	Worst Case	Stable?	In-Place?	Notes
Bubble Sort	O(n)	O(n^2)	✅ Yes	✅ Yes	Repeated swaps, like bubbles rising.
Selection Sort	O(n^2)	O(n^2)	❌ No	✅ Yes	Picks the smallest element in each pass.
Insertion Sort	O(n)	O(n^2)	✅ Yes	✅ Yes	Inserts elements like sorting playing cards.
Merge Sort	O(n log n)	O(n log n)	✅ Yes	❌ No	Divide and conquer (merges sorted halves).
Quick Sort	O(n log n)	O(n^2)	❌ No	✅ Yes	Picks a pivot and partitions.
Counting Sort	O(n+k)	O(n+k)	✅ Yes	❌ No	Good for small range values.
Radix Sort	O(nk)	O(nk)	✅ Yes	❌ No	Sorts digit by digit.