**Exercise 3: Sorting Customer Orders**

* + **Explain Big O notation and how it helps in analyzing algorithms.**
  + **Describe the best, average, and worst-case scenarios for search operations.**

**1. Bubble Sort**

**Description**: Bubble Sort is a simple comparison-based sorting algorithm. It repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted.

**Algorithm**:

1. Compare each pair of adjacent elements.
2. Swap them if they are in the wrong order.
3. Repeat the process for all elements until no swaps are needed.

**Time Complexity**:

* Best Case: O(n) (when the array is already sorted)
* Average Case: O(n^2)
* Worst Case: O(n^2)

**2. Insertion Sort**

**Description**: Insertion Sort builds the final sorted array one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.

**Algorithm**:

1. Start with the second element (the first element is considered sorted).
2. Compare the current element with the sorted elements and insert it into the correct position.
3. Repeat for all elements.

**Time Complexity**:

* Best Case: O(n) (when the array is already sorted)
* Average Case: O(n^2)
* Worst Case: O(n^2)

**3. Quick Sort**

**Description**: Quick Sort is an efficient, comparison-based, divide-and-conquer sorting algorithm. It works by selecting a ‘pivot’ element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot.

**Algorithm**:

1. Pick a pivot element.
2. Partition the array into two sub-arrays: elements less than the pivot and elements greater than the pivot.
3. Recursively apply the above steps to the sub-arrays.

**Time Complexity**:

* Best Case: O(n log n)
* Average Case: O(n log n)
* Worst Case: O(n^2) (when the smallest or largest element is always chosen as the pivot)

**4. Merge Sort**

**Description**: Merge Sort is an efficient, stable, comparison-based, divide-and-conquer sorting algorithm. It works by dividing the unsorted list into n sublists, each containing one element, and then repeatedly merging sublists to produce new sorted sublists until there is only one sublist remaining.

**Algorithm**:

1. Divide the array into two halves.
2. Recursively sort each half.
3. Merge the two halves to produce a sorted array.

**Time Complexity**:

* Best Case: O(n log n)
* Average Case: O(n log n)
* Worst Case: O(n log n)
  + Compare the performance (time complexity) of Bubble Sort and Quick Sort.
  + Discuss why Quick Sort is generally preferred over Bubble Sort.

**Bubble Sort**

* **Time Complexity**:
  + **Best Case**: O(n) (when the array is already sorted)
  + **Average Case**: O(n^2)
  + **Worst Case**: O(n^2)
* **Space Complexity**: O(1) (in-place sorting)

**Quick Sort**

* **Time Complexity**:
  + **Best Case**: O(n log n)
  + **Average Case**: O(n log n)
  + **Worst Case**: O(n^2) (when the smallest or largest element is always chosen as the pivot)
* **Space Complexity**: O(log n) (due to recursive calls)

Quick sort is generally preffered over Bubble sort because of the following reasons:-

1. **Efficiency**:
   * **Quick Sort**: On average, Quick Sort is much faster than Bubble Sort. The average time complexity of Quick Sort is O(n log n), which is significantly better than Bubble Sort’s O(n^2).
   * **Bubble Sort**: Bubble Sort is inefficient for large datasets due to its quadratic time complexity.
2. **Scalability**:
   * **Quick Sort**: Quick Sort scales well with larger datasets. Its divide-and-conquer approach allows it to handle large arrays efficiently.
   * **Bubble Sort**: Bubble Sort does not scale well with larger datasets. As the size of the array increases, the number of comparisons and swaps grows quadratically, making it impractical for large datasets.
3. **Practical Performance**:
   * **Quick Sort**: In practice, Quick Sort is often faster than other O(n log n) algorithms like Merge Sort and Heap Sort due to its efficient in-place partitioning and cache performance.
   * **Bubble Sort**: Bubble Sort is rarely used in practice for sorting large datasets due to its poor performance.
4. **Adaptability**:
   * **Quick Sort**: Quick Sort can be optimized with different pivot selection strategies (e.g., median-of-three, random pivot) to improve its performance and reduce the likelihood of hitting the worst-case scenario.
   * **Bubble Sort**: Bubble Sort has limited scope for optimization and remains inefficient even with minor improvements.