Bubble Sort Questions:

1. Explain the basic idea behind the bubble sort algorithm. How does it work, and why is it called "bubble sort"?

- Basic idea: Bubble sort is a straightforward sorting algorithm that compares nearby elements in an array repeatedly and swaps them if they are not in the correct order.

- How it works: Bubble sort operates by repeatedly going over the array, comparing each element to its neighbours, and exchanging those that are in incorrect order. After then, the system keeps doing this until no more swaps are required.

- Why is it called bubble sort: The reason bubble sort is termed a "bubble sort" is because it constantly bubbles the biggest elements to the end of the array as it operates. This is comparable to how bubbles in a glass of water rise to the top.

2. Write the pseudocode for the bubble sort algorithm. Be sure to include the necessary steps and loop structures.

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

3. What is the time complexity of the bubble sort algorithm in the worst, average, and best cases? Explain your answers.

- Worst Case - O(n^2): happens when the elements in the input array need to bubble up to their proper positions since the array's input is in reverse order.
- Average Case - O(n^2): when items are sorted randomly, this is the anticipated time complexity.
- Best Case - O(n): when the input array is already sorted, bubble sort must do comparisons to ensure no swaps are required.

4. Provide an example of a small unsorted array, and walk through the steps of the bubble sort algorithm to demonstrate how it sorts the array.

   Unsorted Array: [4, 1, 8, 6]

   Example:

   - Step 1: compare 4 and 1.
     Then, swap. => [1,4,8,6]
   - Step 2: compare 4 and 8.
     Then, no swap. => [1,4,8,6]
   - Step 3: compare 8 and 6
     Then, swap. => [1,4,6,8]

   Hence, the array is sorted.

5. Discuss the advantages and disadvantages of using bubble sort for sorting large datasets.

   Advantages:

   - Simple and easy to understand
   - Sorts the array while it is in use and doesn't need further memory (less space complexity)
   - Stable sorting algorithm

   Disadvantages:

   - Very slow algorithm, especially for large datasets (worst-case time complexity)
   - Lack of flexibility, still requires O(n^2) even most data is already in order

Bubble Sort Questions:

1. Describe the insertion sort algorithm. How does it work, and why is it called "insertion sort"?

- Basic idea: insertion sort is a simple sorting algorithm that builds a sorted array one element at a time. The method iterates over the unsorted array, putting every element into the sorted array in the proper location before starting with an empty sorted array.

- How it works: Insertion Sort moves bigger elements to the right by repeatedly picking each element in the array's unsorted section and putting it in the appropriate spot inside the sorted portion. Repeat this process until all of the elements are ordered in ascending order.

- Why is it called insertion sort: because it works by picking an element from the array's unsorted part and inserting each element into the sorted array in its correct position.

2. Write the pseudocode for the insertion sort algorithm. Be sure to include the necessary steps and loop structures.

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

3. What is the time complexity of the insertion sort algorithm in the worst, average, and best cases? Explain your answers.

   - Worst Case - O(n^2): happens when each element needs to be relocated to the beginning of the array because the input array is in reverse order.
   - Average Case - O(n^2): The predicted time complexity when components are sorted at random.
   - Best Case - O(n): occurs when there is no need for swaps since the input array is already sorted.

4. Provide an example of a small unsorted array, and walk through the steps of the insertion sort algorithm to demonstrate how it sorts the array.

Unsorted Array: [6, 1, 8, 2]

Example:

- Step 1: the sorted portion starts with [6], and the unsorted portion has [1, 8, 2].
  - Sorted array: [6]
  - Unsorted array: [1,8,2]

- Step 2: start with 1 (second element) and compare with 6. Since 1 is smaller than 6, so shift 6 to the right and 1 insert in correct position.
    - Sorted array: [1, 6]
    - Unsorted array: [8,2]

- Step 3: start with 8 (third element) and compare it with 6. (no shift)
    - Sorted array: [1, 6, 8]
    - Unsorted array: [2]

- Step 4: start with 2(fourth element) and compare it with 8. Since 2 is smaller than 8, so shift 8 to the right. Then, compare 2 and 6 by shifting 6 to the right.
    - Sorted array: [1, 2, 6, 8]
    - Unsorted array: []

Hence, the array is sorted from [6,1,8,2] to [1,2,6,8].

5. Compare and contrast bubble sort and insertion sort. What are the key differences between the two algorithms in terms of their performance and usage?

   Key differences:

   - Compared to Insertion Sort, Bubble Sort is less effective and less tolerant of partly sorted data.
   - While Insertion Sort picks each element and positions it correctly, Bubble Sort continuously compares and swaps nearby items.
   - In practical applications with partially sorted data, Insertion Sort frequently outperforms Bubble Sort.
   - Both techniques, which have O(n2) time complexity in the worst and average circumstances, are typically used for sorting small to medium-sized datasets or when the capacity to handle partially sorted data is critical.
   - While Insertion Sort has greater real-world uses in situations where fresh data is regularly supplied to a sorted dataset, Bubble Sort is mostly utilized for instructional reasons.

6. Can you think of a real-world scenario where insertion sort would be a better choice than bubble sort for sorting data? Explain your reasoning.

   Scenario: storing a contact list in mobile phone

   For sorting a contact list on a mobile device, insertion sort is an appropriate choice since it is effective for tiny arrays and arrays that are already partially sorted. On mobile devices, contact lists are generally tiny arrays that are only partially sorted, with the most frequently

contacted contacts at the top. It maintains the relative order of like entries in the list, making it more stable than bubble sort. This is crucial for contact lists because, even if they share a name, we might want to retain the people who get in touch with us the most at the top.

However, for bubble sort, even if the contact's name is in alphabetical order, it still compare with each other one by one until the array is sorted. It is inefficient for time complexity.