Sorting Algorithms

- Algorithms designed to arrange data in a specific order (ascending or descending)
- Key in tasks like searching, data organization, and optimization
- **Types of Sorting Algorithms:**
  1. Comparison-Based Sorting:
     a) Compare elements to determine their order.
     b) **Example**: Bubble Sort, Quick Sort, Merge Sort.
  2. Non-Comparison-Based Sorting:
     1. Use counting or distribution properties instead of direct comparisons.
     2. **Example**: Counting Sort, Radix Sort, Bucket Sort.

Bubble Sort

- Bubble Sort is a simple comparison-based sorting algorithm. It repeatedly steps through the list, compares adjacent items, and swaps them if they are in the wrong order.

- Works by repeatedly "bubbling up" the largest element to the end.

- Easy to understand and implement.
- Best for Small or nearly sorted datasets.

Bubble Sort algorithm

- **Input**:
  - A list of prices.
- **Output**:
  - The sorted list in ascending order.
- **Steps:**
  1. Start with the given list of prices.
  2. Outer loop: Repeat n−1 passes through the list (where n is the number of prices).
     a) Assume the list is sorted at the start of each pass
  3. Inner loop: Compare each pair of adjacent prices in the unsorted portion of the list
     a) If the left price is greater than the right price, swap them
     b) If any swap occurs, mark the list as not sorted.
  4. If no swaps occur during a pass, the list is already sorted, so stop early
  5. Repeat until the entire list is sorted.
  6. Output the sorted list

Why Two Loops?

- **Multiple Passes**:
  - Bubble sort requires multiple passes over the list to sort it completely. The outer loop controls the number of passes.
- **Comparison and Swapping:**
  - The inner loop performs the actual comparison and swapping of adjacent elements within each pass.
- The outer loop ensures that each element gets its chance to be compared and potentially swapped.
- The inner loop performs the pairwise comparisons and swaps within each pass.

Example pass 1

- Price list = [99.99, 49.95, 299.49, 19.95]
- n = 4;
- Pass 1 (i=0):

1. Comparison 1:
   - Compare 99.99 and 49.95 (j=0):
   - 99.99 > 49.95, so swap.
   - List after swap: [49.95, 99.99, 299.49, 19.95]
2. Comparison 2:
   - Compare 99.99 and 299.49 (j=1)
   - 99.99 < 299.49, no swap.
   - List after swap: [49.95, 99.99, 299.49, 19.95]
3. Comparison 3:
   - Compare 299.49 and 19.95 (j=2):
   - 299.49 > 19.95, so swap.
   - List after swap: [49.95, 99.99, 19.95, 299.49]

Example pass 2

- List after pass1 = [49.95, 99.99, 19.95, 299.49]
- n = 4;
- Pass 2 (i=1):
  1. Comparison 1:
     - Compare 49.95 and 99.99 (j=0):
     - 49.95 < 99.99, no swap.
     - List after swap: [49.95, 99.99, 19.95, 299.49]
  2. Comparison 2:
     - Compare 99.99 and 19.95 (j=1):
     - 99.99 > 19.95, so swap.
     - List after swap: [49.95, 19.95, 99.99, 299.49]

Example pass 3

- List after pass 2 = [49.95, 19.95, 99.99, 299.49]
- n = 4;
- Pass 3 (i=2):
  1. Comparison 1:
     - Compare 49.95 and 19.95 (j=0):
     - 49.95 > 19.95, so swap.
     - List after swap: [19.95, 49.95, 99.99, 299.49]

- Final sorted price list = [19.95, 49.95, 99.99, 299.49]

Time Complexity of Bubble Sort

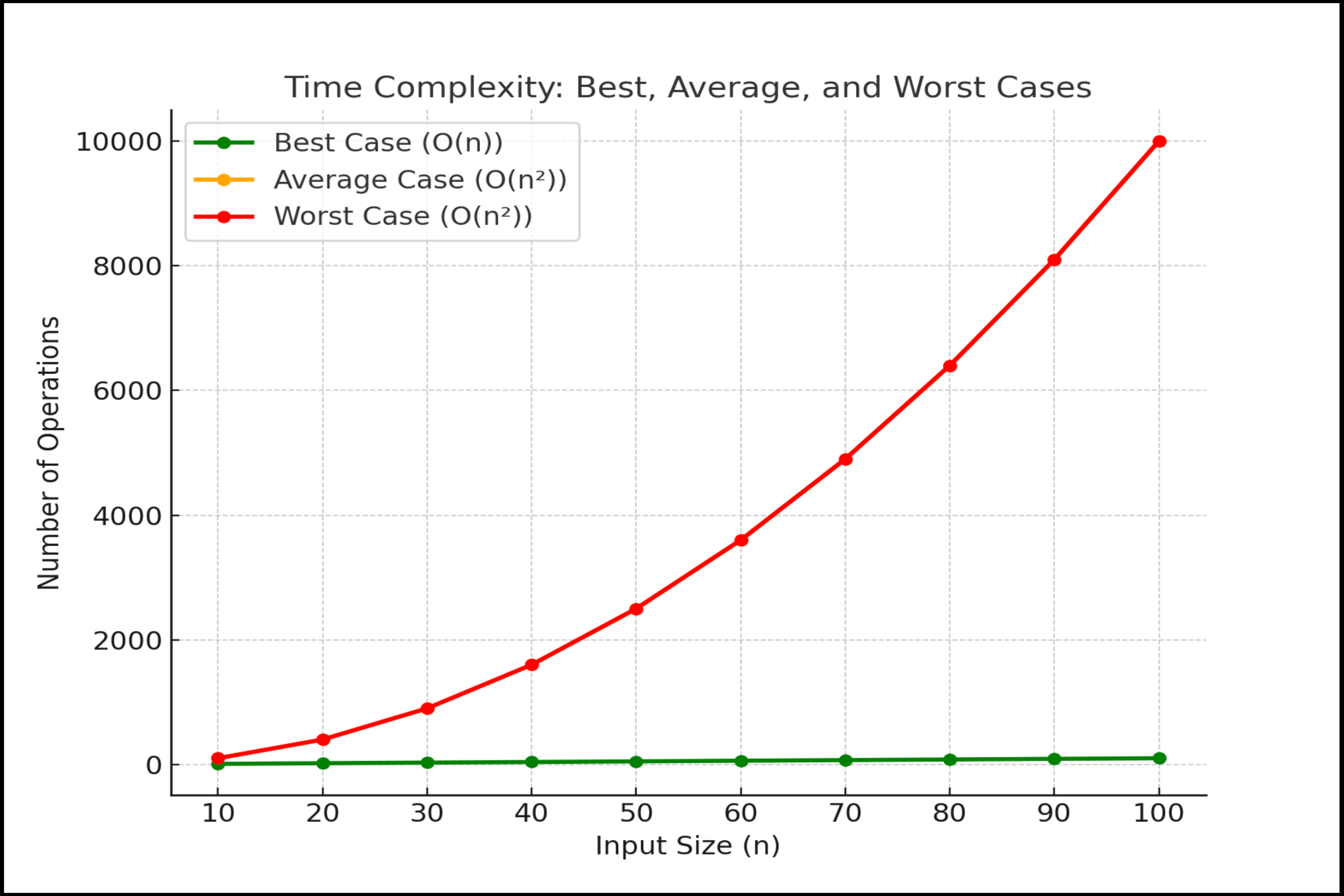
- **Best Case** O(n):
  - The list is already sorted
  - In this case, the outer loop only needs to run once, and the inner loop will terminate early without any swaps.
- **Average Case** $O(n^2)$:
  - The list is in a random order.
  - In most cases, the algorithm will require multiple passes through the array to sort it completely
- **Worst Case** $O(n^2)$:
  - The list is sorted in reverse order.
  - In this case, the maximum number of comparisons and swaps will be required.

- Due to its quadratic time complexity, bubble sort is generally not considered efficient for large datasets. It's better suited for small datasets or educational purposes.

Why O(n²)?

- Outer Loop: Runs n−1 times.
  - For example: n=4, it runs 3 times.
- Inner Loop: Runs n−i−1 times for each outer loop iteration.
  - i=0: Inner loop runs 3 times, means (n–1) times
  - i=1: Inner loop runs 2 times, means (n–2) times
  - i=2: Inner loop runs 1 time, means (n–3) times
- Total Comparisons:
  - (n–1)+(n–2)+….+1
- Using the Arithmetic Series Formula:
- $\frac{(\text{first term}+\text{last term})\times\ \text{number of terms}}{2}$
- $\frac{(n-1)+1\times\ (n-1)}{2}$
- $\frac{n\,(n-1)}{2} = \frac{n^2-n}{2}$ = so the dominant term is n^2 = $O(n^2)$.

Quadratic time complexity



Space Complexity of Bubble Sort

- Space Complexity O(1).
- The space complexity of bubble sort is O(1) that means the amount of extra space required by the algorithm is constant, regardless of the input size.

- Bubble sort is an in-place sorting algorithm, which means it sorts the elements within the original array without requiring any additional data structures that grow with the input size.