Insertion Sort

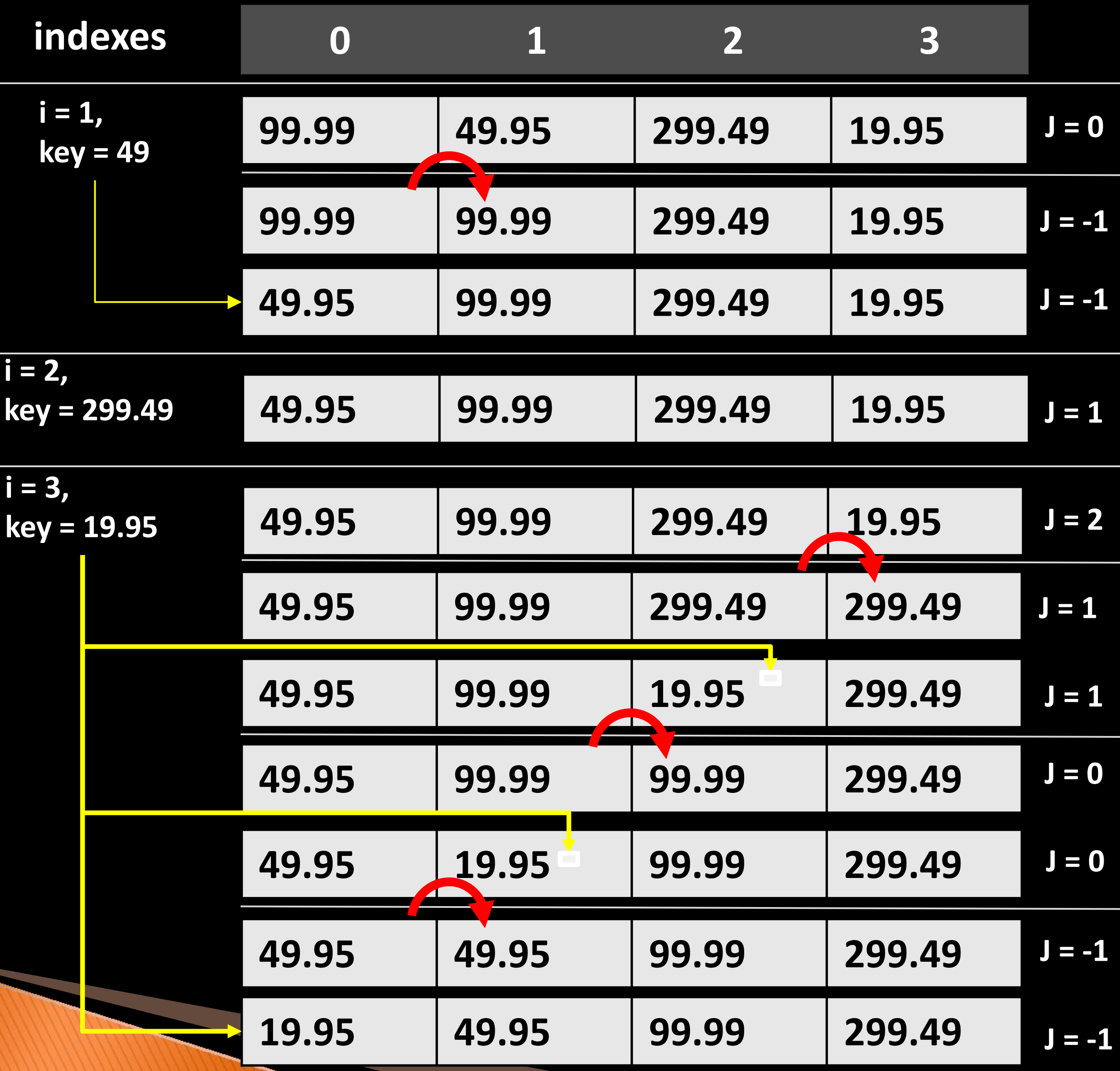
- A comparison-based sorting algorithm that builds the final sorted array one element at a time.

- Elements from the unsorted part are picked and inserted into the correct position in the sorted part.
- Efficient for small data sets or nearly sorted arrays.

Insertion Sort algorithm

- **Input**:
  - A list of elements to be sorted.
- **Output**:
  - The sorted list.
- **Steps:**
  1. Start with the second element (index 1), considering the first element as sorted.
  2. Compare the current element with elements in the sorted part.
  3. Shift elements of the sorted part to the right to make room for the current element if necessary.
  4. Insert the current element into its correct position.
  5. Repeat until the entire array is sorted.

How insertion Sort works

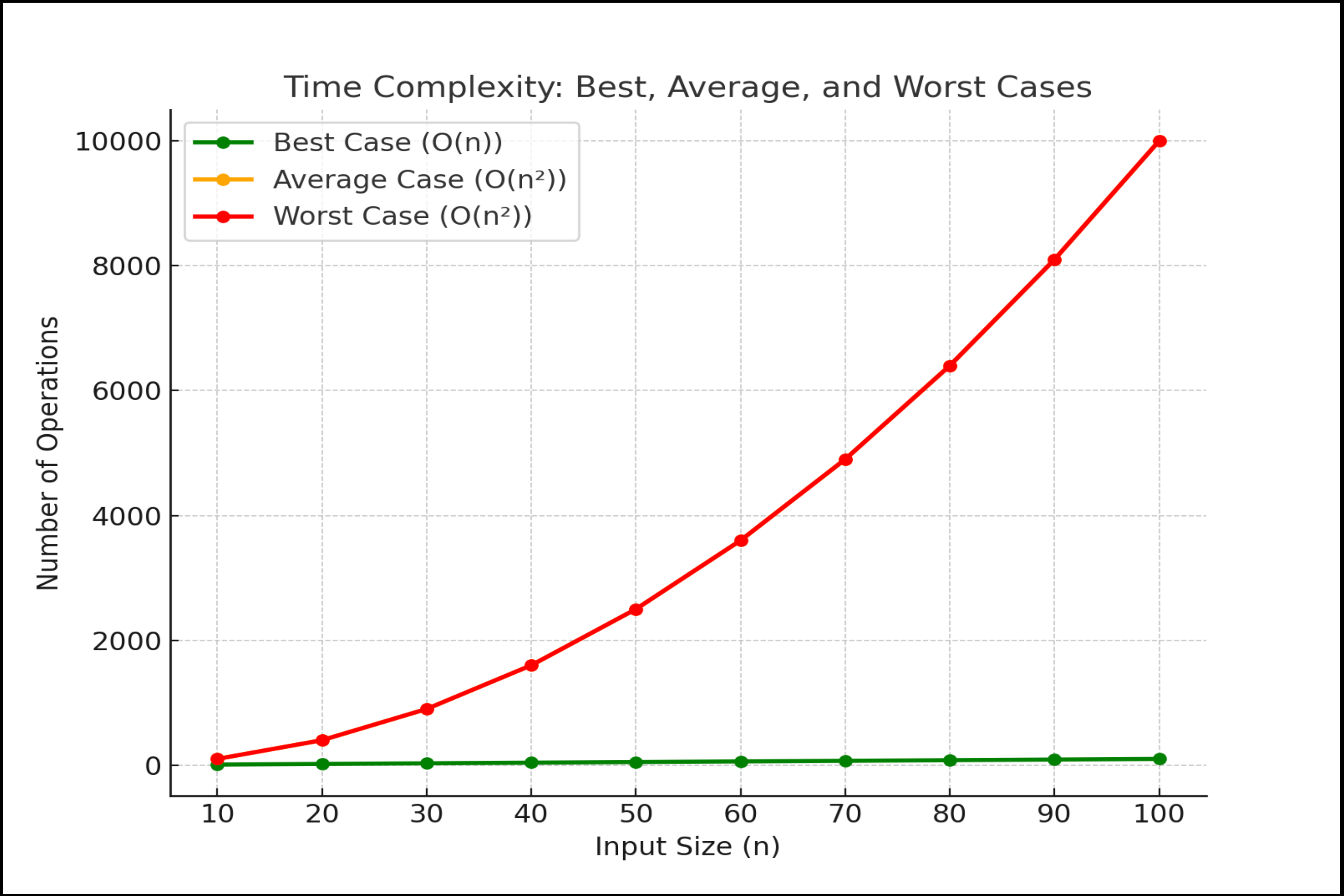


indexes	0	1	2	3	
i = 1, key = 49	99.99	49.95	299.49	19.95	J = 0
	99.99	99.99	299.49	19.95	J = -1
	49.95	99.99	299.49	19.95	J = -1
i = 2, key = 299.49	49.95	99.99	299.49	19.95	J = 1
i = 3, key = 19.95	49.95	99.99	299.49	19.95	J = 2
	49.95	99.99	299.49	299.49	J = 1
	49.95	99.99	19.95	299.49	J = 1
	49.95	99.99	99.99	299.49	J = 0
	49.95	19.95	99.99	299.49	J = 0
	49.95	49.95	99.99	299.49	J = -1
	19.95	49.95	99.99	299.49	J = -1

Time Complexity of insertion Sort

- **Best Case** O(n):
  - The list is already sorted
  - In this case, the inner loop does not need to make any shifts, it only compares each element once.
- **Average Case** O(n^2):
  - The list is in a random order.
  - each element must be compared with half of the already sorted elements and potentially shifted. Over all insertions, this leads to a quadratic number of comparisons and shifts.
- **Worst Case** O(n^2):
  - The list is sorted in reverse order.
  - Each element has to be compared with all previously sorted elements, leading to the maximum number of comparisons and shifts.

Quadratic time complexity



Space Complexity of insertion Sort

- Space Complexity O(1).
- Insertion Sort is an in-place sorting algorithm.

- It only requires a constant amount of extra space (for example, a few variables) regardless of the input size.

- Insertion Sort is efficient for small datasets or nearly sorted arrays due to its simple implementation and low overhead. However, it becomes less efficient with larger, unsorted datasets.