Bucket Sort

Bucket Sort is a **non-comparison-based sorting algorithm** that distributes elements into several groups called **buckets.**
Each bucket is then sorted individually (using another algorithm like Insertion Sort, Quick Sort, or even recursively Bucket Sort), and finally, the buckets are combined to form the sorted array.

It is mainly useful when input is **uniformly distributed** over a range.

🔹 How Bucket Sort Works (Step by Step)

1. Create n empty buckets (arrays or lists).
2. Scatter the input elements into different buckets based on a **hashing function or value range.**
3. Sort each bucket individually (using Insertion Sort or any other algorithm).
4. Concatenate all the buckets to get the final sorted array.

🔹 Example

Sort the array **[0.42, 0.32, 0.23, 0.52, 0.25, 0.47, 0.51]** using Bucket Sort.

- Step 1: Create 10 buckets for the range `[0, 1)`.
- Step 2: Scatter elements into buckets:
  - 0.42 → Bucket 4
  - 0.32, 0.23, 0.25 → Bucket 2
  - 0.52, 0.51, 0.47 → Bucket 5

- Buckets now look like:
    - Bucket 2 → [0.32, 0.23, 0.25]
    - Bucket 4 → [0.42]
    - Bucket 5 → [0.52, 0.47, 0.51]
- Step 3: Sort each bucket:
    - Bucket 2 → [0.23, 0.25, 0.32]
    - Bucket 4 → [0.42]
    - Bucket 5 → [0.47, 0.51, 0.52]
- Step 4: Concatenate buckets → **[0.23, 0.25, 0.32, 0.42, 0.47, 0.51, 0.52]**

Final sorted array obtained.

◆ Time Complexity

- **Best Case:** O(n + k) → when elements are evenly distributed, and bucket sorting is efficient.
- **Average Case:** O(n + k) → depends on distribution of elements.
- **Worst Case:** O(n²) → when all elements go into one bucket and a quadratic algorithm (like Insertion Sort) is used inside.

Here,

- n = number of elements
- k = number of buckets

🔹 Space Complexity

- **O(n + k)** (extra buckets are used).

🔹 Characteristics

Works very well for uniformly distributed input.
Efficient when using buckets + efficient internal sorting (like Insertion Sort for small buckets).
Performance heavily depends on **distribution of data**.
Not suitable when the input data is **not uniformly distributed**.
Requires **extra space** for buckets.

Real-life Analogy:

Think of **arranging exam papers by marks**:

- Create buckets for each range (0–10, 11–20, …, 91–100).
- Put each student's paper in the correct bucket.
- Sort within each bucket (if needed).
- Finally, stack all buckets in order.