Bucket Sort – Practice Set

Q1. Dry run Bucket Sort on the array:
`[0.42, 0.32, 0.23, 0.52, 0.25, 0.47, 0.51]`

Q2. For the array `[0.78, 0.17, 0.39, 0.26, 0.72, 0.94, 0.21]` (values in the range [0,1)):

- Show how the elements will be distributed into **10 buckets.**
- Sort each bucket and give the final output.

Q3. Dry run Bucket Sort on the array:
`[42, 32, 23, 52, 25, 47, 51]` using **5 buckets.**

Q4. Perform Bucket Sort on the array:
`[0.36, 0.12, 0.45, 0.72, 0.54, 0.89]` using **6 buckets.**

Q5. If all elements of the array fall into **a single bucket**, what will be the **time complexity** of Bucket Sort? Explain with an example.

Q6. Show how Bucket Sort sorts the array:
`[0.11, 0.21, 0.31, 0.41, 0.51, 0.61, 0.71, 0.81]`

Q7. Dry run Bucket Sort on the array:
`[0.14, 0.98, 0.37, 0.25, 0.75, 0.45, 0.63, 0.82, 0.19, 0.59]`

Q8. Apply Bucket Sort to the array:
`[0.23, 0.45, 0.23, 0.12, 0.45, 0.89, 0.12]`

Q9. Consider the array:
`[5, 5, 5, 5, 5]`

- Show how Bucket Sort processes this input.
- What does this example tell us about the **distribution of elements**?

Q10. How can Bucket Sort be modified to handle an array containing negative numbers like:
`[-2, -5, -3, -8, -6]`

Q11. Why is Bucket Sort more efficient when the input data is **uniformly distributed**? Give an example.

Q12. Give a real-world example where **Bucket Sort** would be more efficient than **Quick Sort** or **Merge Sort.**

Q13. If you have an array of size `n = 1,000,000` with values in the range `[0,1)`:

- How many buckets would you ideally use?
- Why?

Q14. How is **Bucket Sort** different from **Counting Sort**? Which one would you use for sorting integers in a limited range?

Q15. Perform Bucket Sort on the array:

```
[0.91, 0.02, 0.72, 0.48, 0.33, 0.11, 0.67, 0.83, 0.54]
```