Consider a scenario where an array contains duplicate elements. The problem is to find the frequency of each element in the array. A common approach is to use a hash table to count the occurrences of each element. However, for large datasets, this method can be inefficient due to the overhead of hash table operations.

A more efficient approach is to use bucket sort. The basic idea is to divide the range of possible array elements into equal-sized buckets. Each bucket corresponds to a range of values, and the algorithm distributes the elements into these buckets. Once the elements are in the buckets, the algorithm then processes each bucket separately to count the occurrences of each element.

Bucket sort is particularly useful when the input is uniformly distributed over a range. It is more efficient than traditional sorting algorithms like quicksort or mergesort for certain types of data distributions. However, its performance can degrade if the input is not uniformly distributed or if the number of buckets is not chosen appropriately.

In this lesson, we will explore the implementation of bucket sort and analyze its time complexity and space requirements. We will also compare it with other sorting algorithms to understand its advantages and limitations.
Algorithm 9.1: ENSW Sort on a CROW-PRAM with Additive-Write Conflict Resolution

```plaintext
1. end ENSW.SORT
2. \( |A| = |f/f| \cdot (d/d) \)
3. op each process \( p \) set \( c = f \).
4. \( |E| = |f/f| \cdot (d/d) \)
5. \( \text{if} \ |A| \neq |f/f| \cdot (d/d) \) \( \eta = |f/f| \cdot (d/d) \).
6. \( \text{if} \ |A| > |f/f| \cdot (d/d) \) \( \eta = |f/f| \cdot (d/d) \).
7. \( \text{op each process} \ p \) set \( c = f \).
8. \( \text{proc each process} \ p \) set \( c = f \).
```

Algorithm 9.2: Enumberation Sort

```
1. \( |f/f| = |f/f| \cdot (d/d) \)
2. \( |A| = |f/f| \cdot (d/d) \)
3. op each process \( p \) set \( c = f \).
4. \( |E| = |f/f| \cdot (d/d) \)
5. \( \text{if} \ |A| \neq |f/f| \cdot (d/d) \) \( \eta = |f/f| \cdot (d/d) \).
6. \( \text{if} \ |A| > |f/f| \cdot (d/d) \) \( \eta = |f/f| \cdot (d/d) \).
7. \( \text{op each process} \ p \) set \( c = f \).
8. \( \text{proc each process} \ p \) set \( c = f \).
```

9.6.1 Enumberation Sort

The basic algorithm for the Enumberation Sort is as follows:

1. \( |f/f| = |f/f| \cdot (d/d) \)
2. \( |A| = |f/f| \cdot (d/d) \)
3. op each process \( p \) set \( c = f \).
4. \( |E| = |f/f| \cdot (d/d) \)
5. \( \text{if} \ |A| \neq |f/f| \cdot (d/d) \) \( \eta = |f/f| \cdot (d/d) \).
6. \( \text{if} \ |A| > |f/f| \cdot (d/d) \) \( \eta = |f/f| \cdot (d/d) \).
7. \( \text{op each process} \ p \) set \( c = f \).
8. \( \text{proc each process} \ p \) set \( c = f \).