Selection Sort Explanation

explain with example step by step then give code for selection and count sort

continue

Selection Sort Explanation

Selection Sort works by repeatedly selecting the smallest (or largest, depending on sorting order) element from the unsorted portion and swapping it with the first unsorted element. Here's a step-by-step explanation:

1. **Start at the beginning:** Set the first element as the minimum.
2. **Find the minimum:** Traverse the unsorted portion to find the smallest element.
3. **Swap:** Swap this minimum element with the first unsorted element.
4. **Repeat:** Move to the next unsorted element and repeat the process until the array is sorted.

Example:

Initial array:
`[29, 10, 14, 37, 13]`
Step-by-step process:

1. **First Pass:**
   - Find the minimum element in `[29, 10, 14, 37, 13]` → `10`
   - Swap `10` with `29`
   - Array after first pass: `[10, 29, 14, 37, 13]`
2. **Second Pass:**
   - Find the minimum in `[29, 14, 37, 13]` → `13`
   - Swap `13` with `29`
   - Array after second pass: `[10, 13, 14, 37, 29]`
3. **Third Pass:**
   - Find the minimum in `[14, 37, 29]` → `14`
   - No swap needed, `14` is already in the correct position.
   - Array after third pass: `[10, 13, 14, 37, 29]`
4. **Fourth Pass:**
   - Find the minimum in `[37, 29]` → `29`
   - Swap `29` with `37`

- Array after fourth pass: `[10, 13, 14, 29, 37]`

The array is now sorted.

Selection Sort Java Code:

```java
public class SelectionSort {
    public static void selectionSort(int[] arr) {
        int n = arr.length;

        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;

            // Find the minimum element in unsorted part
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }

            // Swap the found minimum element with the first unsorted element
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
    }

    public static void main(String[] args) {
        int[] arr = {29, 10, 14, 37, 13};
        selectionSort(arr);

        System.out.print("Sorted array: ");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}
```

Counting Sort Explanation

Counting Sort works by counting the number of occurrences of each unique element in the array. It then uses this count to place the elements in the correct position.

1. **Find the range of elements:** Determine the minimum and maximum values in the array.
2. **Create a count array:** Create an auxiliary array to store the count of each element.
3. **Fill the count array:** Count the occurrences of each element.
4. **Modify the count array:** Modify it so that each element at each index contains the sum of previous counts. This will tell us the position of each element in the sorted array.
5. **Place elements into the sorted array:** Traverse the original array backward, placing elements in their correct positions based on the count array.

Example:

Initial array:
`[4, 2, 2, 8, 3, 3, 1]`
Step-by-step process:

1. **Find the range:**
   Minimum = `1`, Maximum = `8`
2. **Create a count array:**
   Create a count array of size `8` (since the max element is 8):
   `[0, 0, 0, 0, 0, 0, 0, 0]`
3. **Fill the count array:**
   Count occurrences of each element in the array:
   `[0, 1, 2, 2, 1, 0, 0, 0, 1]`
   (Count of `1` is `1`, `2` is `2`, etc.)
4. **Modify the count array:**
   Modify it to reflect positions:
   `[0, 1, 3, 5, 6, 6, 6, 6, 7]`
5. **Place elements into the sorted array:**
   Traverse the original array from the last element, place them based on the count array:
   Sorted array: `[1, 2, 2, 3, 3, 4, 8]`

Counting Sort Java Code:

java

```java
import java.util.Arrays;

public class CountingSort {
    public static void countingSort(int[] arr) {
        int n = arr.length;

        // Find the maximum value to define the range of the count array
        int max = Arrays.stream(arr).max().getAsInt();

        // Create a count array to store the count of each unique object
        int[] count = new int[max + 1];

        // Store count of each element
        for (int i = 0; i < n; i++) {
            count[arr[i]]++;
        }

        // Modify the count array such that each element at each index stores the sum of
previous counts
        for (int i = 1; i <= max; i++) {
            count[i] += count[i - 1];
        }

        // Output array to store the sorted order
        int[] output = new int[n];

        // Build the output array by placing elements at the correct position
        for (int i = n - 1; i >= 0; i--) {
            output[count[arr[i]] - 1] = arr[i];
            count[arr[i]]--;
        }

        // Copy the sorted elements into the original array
        System.arraycopy(output, 0, arr, 0, n);
    }

    public static void main(String[] args) {
        int[] arr = {4, 2, 2, 8, 3, 3, 1};
        countingSort(arr);

        System.out.print("Sorted array: ");
        for (int num : arr) {
```

```
                System.out.print(num + " ");
            }
        }
    }
```

Both **Selection Sort** and **Counting Sort** are simple but useful algorithms with different use cases. Selection Sort is easier to understand but slower, while Counting Sort is much faster for small ranges of integers but requires additional space.