Counting Sort Project

(a) Required Algorithms for Counting Sort

The **Counting Sort algorithm** works for non-negative integers and sorts them in O(n+k) O(n + k) O(n+k), where nnn is the number of elements in the input array and kkk is the range of the input values.

Counting Sort Algorithm

1. **Find the range of input data**: Determine the largest element in the array (denoted as kkk) to create a count array.
2. **Count occurrences**: Create a count array of size k+1k+1k+1, and count the occurrences of each value in the input array.
3. **Compute prefix sums**: Modify the count array to store the cumulative sum of counts to determine the position of each element in the sorted output.
4. **Sort the elements**: Iterate through the input array and place each element in its correct position in the output array using the count array.

CODE

```
COUNTING-SORT(A, B, k):

   1. Initialize count array C of size (k + 1) with all elements as 0

   2. For each element in A:

        Increment the value in C at index equal to the element in A

   3. Compute prefix sums in C:

        For i from 1 to k:

           C[i] = C[i] + C[i - 1]

   4. Build output array B:

        For each element in A (traverse from right to left for stability):

           Place element A[i] at index (C[A[i]] - 1) in B

           Decrement C[A[i]]

5.    5. Copy the sorted elements from B back to A (optional if sorting in-place is
   needed)
```

- **Input**: Array AAA of integers, an empty array BBB for the result, and kkk, the maximum value in AAA.

- **Output**: Sorted array BBB.

(b) Analysis of the Counting Sort Algorithm

1. Time Complexity:

- **Counting elements**: O(n)O(n)O(n), where nnn is the size of the input array.
- **Prefix sums**: O(k)O(k)O(k), where kkk is the range of numbers.
- **Sorting**: O(n)O(n)O(n), as each element is placed in its sorted position.
- **Total Complexity**: O(n+k)O(n + k)O(n+k).

2. Space Complexity:

- The count array requires O(k)O(k)O(k) space.
- The output array requires O(n)O(n)O(n) space.
- Total space complexity: O(n+k)O(n + k)O(n+k).

3. Stability: Counting Sort is a stable sorting algorithm because elements with the same value retain their relative order from the input array.

4. Constraints:

- Works only for non-negative integers.
- Performance depends on kkk. If kkk is very large compared to nnn, the algorithm may not be efficient

```
#include <iostream>

#include <vector>

#include <algorithm> // for max_element



void countingSort(std::vector<int>& A) {



    int k = *std::max_element(A.begin(), A.end());
```

```
    std::vector<int> C(k + 1, 0);
    std::vector<int> B(A.size(), 0);



    for (int num : A) {
        C[num]++;
    }

    for (size_t i = 1; i < C.size(); ++i) {
        C[i] += C[i - 1];
    }

        for (int i = A.size() - 1; i >= 0; --i) {
        B[C[A[i]] - 1] = A[i];
        C[A[i]]--;
    }



    A = B;
}

int main() {
```

```
    // Example usage
    std::vector<int> array = {4, 2, 2, 8, 3, 3, 1};

    std::cout << "Original array: ";
    for (int num : array) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    countingSort(array);

    std::cout << "Sorted array: ";
    for (int num : array) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

الشرح بالعربي

خطوات الخوارزمية:

1. **حساب نطاق البيانات:**
   - تحديد القيمة العظمى (kkk) في المصفوفة.
2. **إنشاء مصفوفة العد:**
   - مصفوفة العد (CCC) تخزن عدد مرات ظهور كل رقم.

3. **حساب المجاميع التراكمية:**
   - تعديل مصفوفة العد لتحتوي على المواضع النهائية لكل عنصر في المصفوفة.
4. **فرز العناصر:**
   - استخدام مصفوفة العد لتحديد المواضع الصحيحة لكل عنصر، مع ضمان ثبات الترتيب (Stability).

تحليل التعقيد:

1. **التعقيد الزمني:**
   - O(n+k)O(n + k)O(n+k): حيث nnn عدد العناصر، و kkk هو القيمة العظمى.
2. **التعقيد المكاني:**
   - O(n+k)O(n + k)O(n+k): يتطلب مساحة إضافية لمصفوفة العد ومصفوفة الإخراج.

مثال عملي:

لنفترض أن لدينا مصفوفة [4,2,2,8,3,3,1][4, 2, 2, 8, 3, 3, 1][4,2,2,8,3,3,1]:

1. يتم إنشاء مصفوفة عدّ CCC: [0,1,2,2,1,0,0,0,1][0, 1, 2, 2, 1, 0, 0, 0, 1][0,1,2,2,1,0,0,0,1].
2. تُحسب المجاميع التراكمية: [0,1,3,5,6,6,6,6,7][0, 1, 3, 5, 6, 6, 6, 6, 7][0,1,3,5,6,6,6,6,7].
3. يتم إنشاء المصفوفة المُرتبة [1,2,2,3,3,4,8][1, 2, 2, 3, 3, 4, 8][1,2,2,3,3,4,8].