Counting sort algorithm

***Source:* https://github.com/rusevrosen/codehub.github.io/tree/main/Algorithms/Sorting_Algorithms**

1. What is the Counting Sort algorithm?

Counting sort is an efficient, stable sorting algorithm that can sort a collection of items, where each item is a discrete, non-negative integer within a specific range. The algorithm works by counting the number of occurrences of each item in the input collection and then using that information to place each item in its correct position in the output collection.

Counting sort makes assumptions about the data, for example, it assumes that values are going to be in the range of 0 to 10 or 10 – 99, etc. Some other assumption counting sort makes is input data will be all real numbers.

Like other algorithms, this sorting algorithm is not comparison-based that hashes the value in a temporary count array and uses them for sorting.

2. Code example.

```cpp
#include <iostream>

void counting_sort(int a[], int n, int range) {
    int* b = new int[n];      // output array
    int* c = new int[range];  // counting array

    for (int i = 0; i < range; i++) {
        c[i] = 0;
    }
    for (int i = 0; i < n; i++) {
        c[a[i]]++;
    }
    for (int i = 1; i < range; i++) {
        c[i] += c[i - 1];
    }
    for (int i = n - 1; i >= 0; i--) {
        b[--c[a[i]]] = a[i];
    }
    for (int i = 0; i < n; i++) {
        a[i] = b[i];
    }
    delete[] b;
    delete[] c;
}

int main() {
    int a[] = {3, 2, 1, 4, 5, 3, 5, 2, 1, 4};
    int n = sizeof(a) / sizeof(a[0]);
    int range = 6; // since the numbers in the array are between 1 and 5 inclusive
    counting_sort(a, n, range);
    for (int i = 0; i < n; i++) {
        std::cout << a[i] << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

This program will output the sorted array:

```
1 1 2 2 3 3 4 4 5 5
```

The program sorts the array "**a**" containing the integers **`{3, 2, 1, 4, 5, 3, 5, 2, 1, 4}`** and since the range is set to **6** since the numbers are between **1** and **5** inclusive. The function **`counting_sort`** is called and it sorts the array by counting the occurrences of each integer in the array and then uses that information to place each integer in its correct position in the output array. The output array, sorted, is printed out at the end of the main function so you can see the result of the sorting.

3. Code explanation.

The example program is a demonstration of how to use the counting sort algorithm in C++ to sort an array of integers. The program first defines a function called **`counting_sort`** that takes in three parameters: an array of integers **a**, the number of elements in the array **n**, and the range of integers that the array can contain a range.

The counting sort algorithm works by first initializing a counting array c with a size of the range, where the range is the maximum value of the integers in the input array plus one, in this case, is **6** as the numbers in the array are between **1** and **5** inclusive. Then the algorithm iterates through the input array counting the number of occurrences of each integer using the counting array **c**, so **`c[i]`** will contain the number of occurrences of the integer **i** in the input array.

The next step is to cumulative the counting array, so **`c[i]`** will contain the number of integers less than or equal to i. Then it iterates the input array in reverse order, and for each element **`a[i]`** it decreases the value in the counting array **`c[a[i]]`** and sets the output array **b** in the position **`c[a[i]]`** with the value **`a[i].`**

Finally, it copies the output array **b** over the input array **a**. And it deletes the memory of output and counting arrays

In the main function, the program declares an array containing the integers **`{3, 2, 1, 4, 5, 3, 5, 2, 1, 4}.`** Then it calls the **`counting_sort`** function, passing in the array a, its size n and its range of integers, and **6**, the program then prints out the sorted array which is **`1 1 2 2 3 3 4 4 5 5`** in this case.

4. Time complexity.

The time complexity of the counting sort algorithm in the example I provided is **`O(n + k),`** where **n** is the number of elements in the input array, and **k** is the range of integers in the array (i.e. the number of possible values that each element can take on).

The algorithm has three nested loops. The first one iterate range times, initializing the counting array c with zeroes, which has a complexity of **`O(k).`** The second one iterates n times, counting the number of occurrences of each integer, which has a complexity of **`O(n).`** The third one iterates **n** times, placing each integer in its correct position in the output array, which also has a complexity of **`O(n).`**

So, the overall time complexity of counting sort is `O(n + k)`.

It's important to note that counting sort is efficient only when the range of integers is not too high compared to the length of the array, otherwise, it will consume a lot of memory and will be less efficient than other sorting algorithms like quicksort or **`mergesort`** that have a complexity of **`O(nlogn)`**.

..

I wish you happy learning,

Your Rosen Rusev