Counting sort

Counting Sort
Idea

- Counting sort operates by counting the number of objects that have each distinct key value, and using arithmetic on those counts to determine the positions of each key value in the output sequence.

- Its running time is linear in the number of items and the difference between the maximum and minimum key values, so it is only suitable for direct use in situations where the variation in keys is not significantly greater than the number of items.

Counting Sort Example

5	12	4	2	4	3	6	4	6	2	9	9	1	9	3

Suppose we have this array, let's calculate count of each element

element	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
count	1	2	2	3	1	2	0	0	3	0	0	1	0	0	0

Counting Sort Example

element	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
count	1	2	2	3	1	2	0	0	3	0	0	1	0	0	0

Now let's place elements from first one back in the array

Counting Sort Example

element	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
count	1	2	2	3	1	2	0	0	3	0	0	1	0	0	0

Now let's place elements from first one back in the array

Counting Sort Example

element	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
count	1	2	2	3	1	2	0	0	3	0	0	1	0	0	0

Now let's place elements from first one back in the array

1														

Counting Sort Example

element	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
count	0	2	2	3	1	2	0	0	3	0	0	1	0	0	0

Now let's place elements from first one back in the array

1	2	2												

Counting Sort Example

element	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
count	0	0	2	3	1	2	0	0	3	0	0	1	0	0	0

Now let's place elements from first one back in the array

1	2	2	3	3										

Counting Sort Example

element	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
count	0	0	0	3	1	2	0	0	3	0	0	1	0	0	0

Now let's place elements from first one back in the array

1	2	2	3	3	4	4	4							

Counting Sort Example

element	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
count	0	0	0	0	1	2	0	0	3	0	0	1	0	0	0

Now let's place elements from first one back in the array

1	2	2	3	3	4	4	4	5						

Counting Sort Example

element	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
count	0	0	0	0	0	2	0	0	3	0	0	1	0	0	0

Now let's place elements from first one back in the array

1	2	2	3	3	4	4	4	5	6	6				

Counting Sort Example

element	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
count	0	0	0	0	0	0	0	0	3	0	0	1	0	0	0

Now let's place elements from first one back in the array

1	2	2	3	3	4	4	4	5	6	6	9	9	9	

Counting Sort Example

element	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
count	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Now let's place elements from first one back in the array

1	2	2	3	3	4	4	4	5	6	6	9	9	9	12

Counting Sort Example

element	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
count	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Array is sorted.

1	2	2	3	3	4	4	4	5	6	6	9	9	9	12

Counting Sort Implementation

```
void couting_sort(vector<int>& arr)
{
    int numbers_maximal_range = 1000;
    vector<int> count(numbers_maximal_range);
    for (int i = 0; i < arr.size(); ++i)
        ++count[arr[i]];
    int index = 0;
    for (int i = 0; i < count.size(); ++i)
        for (int j = 0; j < count[i]; ++j)
            arr[index++] = i;
}
```

Counting Sort Analysis

- Number of elements is n.
- Each item addition is performed in $O(1)$.
- In the last double-for loop we iterate over each element only once, but range can be greater then element number, we need to handle this in complexity finalizing.
- Complexity of this sort $O(n + range\_lenght)$.