

Counting Sort

Assumptions:

- Input must be **discrete** value. e.g. the input can be grades but not gpa.
- There are n items in the **range** $[1 .. k]$ where $k \leq n$.
- There may be **repeating items**.

When there are a lot of repeating items, counting sort is the best sorting method.

Example:

There is a list of students' grades. BACBBAACCBBC
 We will convert the list into numbers. 2 1 3 2 2 1 1 3 3 2 2 3

Input Array:

	1	2	3	4	5	6	7	8	9	10	11	12
A	2	1	3	2	2	1	1	3	3	2	2	3

Counting Array:

	1	2	3	
	3	5	4	← Total number of repeating
C	3	8	12	← Up to the number of index

↑ $3+5=8$ ↙ $8+4=12$

Output Array:

5 of 2s and end at the 8 th box.

	1	2	3	4	5	6	7	8	9	10	11	12
B	1	1	1	2	2	2	2	2	3	3	3	3

```

void CountingSort( Item A[], Item B[])
{
    Item C[];                //counting array
    for(i=1; i<=k; i++)
        C[i]=0;              // k times
    for(j=1; j<=n; j++)
        C[A[j]]++;           // n times
    for(p=2; p<=k; p++)
        C[p]=C[p]+C[p-1];    // k times
    for(q=n; q>=1;q--)
    {
        B[C[A[q]]]=A[q];     // 2n times
        C[A[q]]--;
    }
}

```

The running time is $O(n + k) = O(n)$ since $k = O(n)$.
It is a **Linear Time**.

Counting Sort is **stable**:

Numbers with the **same value** appear in the **output** array in the same order as they do in the **input** array. When **satellite data** are carried around with the element being sorted, the property of stable is important.

(Thinking about students' records in a database, you may have same grades for several students but each one has its own key and other fields with it.)