> Assumes that each of the n input elements is an integer in the range 0 to k, for some integer k. (So it's useful when the elements are in a known **limited** range)

It runs in $\Theta(n+k)$ time so that when $k = O(n)$ counting sort runs in $\Theta(n)$ time.

How it works

Find the range

- Determine min and max number in the array #### Create a count array "C"
- This new array has a size equal to the range of elements: $k+1$ #### Edit C
- To have in `c[i]` the number of elements of $A \leq i$. #### Build the output array
- Initialize B, the output array
- Copy in B the elements of A, right to left, using C to know where.
- So, for each element:

1. Get the index from the count array C
2. Place the element in the output array in the position given by C
3. Decrement the count at that index to ensure that elements with the same value remain in the same original order. (**Stable**)

```
countingSort(A, n, k){
    B[1:n] and C[0, k] // Inizializzazione di C e B
    for (i=0 to k){
        C[i] = 0 // Inizializzazione di C con tutti 0
    }
    for (j = 1 to n){
        C[A[j]] = C[A[j]] +1 // riempimento di C con numero ricorrenze
           in A
    }
    // C[i] now contains the number of elements equal to i
    for (i=1 to k){
        C[i] = C[i] + C[i-1] // Sistemazione di C con le somme
    }
    // C[i] now contains the number of elements less than or equal to i

    // copy A to B, from right to left
    for (j=n downto 1){
        B[C[A[j]]] = A[j] // put A[i] in B in posizione C[A[j]]
        C[A[j]] = C[A[j]] - 1 // to handle duplucate values
    }
    return B
}
```

Complessità

Se viene rispettata l'ipotesi $k = O(n)$ allora è $\Theta(n)$. In generale è $\Theta(n + k)$ con k = dimensione intervallo di valori. Se k è tanto grande allora non diventa conveniente.