**Counting Sort**

In Counting sort, the frequencies of distinct elements of the array to be sorted is counted and stored in an auxiliary array, by mapping its value as an index of the auxiliary array.

**Algorithm:**

Let's assume that, array A of size N needs to be sorted.

* Initialize the auxillary array Aux[] as 0.  
  **Note:** The size of this array should be ≥max(A[]).
* Traverse array A and store the count of occurrence of each element in the appropriate index of the Aux array, which means, execute Aux[A[i]]++ for each i, where i ranges from [0,N−1].
* Initialize the empty array sortedA[]
* Traverse array Aux and copy i into sortedA for Aux[i] number of times where 0≤i≤max(A[]).

**Note:** The array A can be sorted by using this algorithm only if the maximum value in array A is less than the maximum size of the array Aux. Usually, it is possible to allocate memory up to the order of a million (106). If the maximum value of A exceeds the maximum memory- allocation size, it is recommended that you do not use this algorithm. Use either the [quick sort](https://www.hackerearth.com/practice/algorithms/sorting/quick-sort/tutorial/) or [merge sort](https://www.hackerearth.com/practice/algorithms/sorting/merge-sort/tutorial/) algorithm.

**Implementation:**

Assume that the maximum element that can be in the array is K .  
Now take an Aux[] array of size K+1.  
A[] = Array to be sorted.  
sortedA[] = Sorted version of A[].

void counting\_sort(int A[], int Aux[], int sortedA[], int N) {

// First, find the maximum value in A[]

int K = 0;

for(int i=0; i<N; i++) {

K = max(K, A[i]);

}

// Initialize the elements of Aux[] with 0

for(int i=0 ; i<=K; i++) {

Aux[i] = 0;

}

// Store the frequencies of each distinct element of A[],

// by mapping its value as the index of Aux[] array

for(int i=0; i<N; i++) {

Aux[A[i]]++;

}

int j = 0;

for(int i=0; i<=K; i++) {

int tmp = Aux[i];

// Aux stores which element occurs how many times,

// Add i in sortedA[] according to the number of times i occured in A[]

while(tmp--) {

//cout << Aux[i] << endl;

sortedA[j] = i;

j++;

}

}

}

Example:  
Say A={5,2,9,5,2,3,5}.

Aux will be of the size 9+1 i.e. 10

Aux={0,0,2,1,0,3,0,0,0,2}.  
Notice that Aux[2]=2 which represents the number of occurrences of 2 in A[]. Similarly Aux[5]=3 which represents the number occurrences of 5 in A[].

After applying the counting sort algorithm, sortedA[] will be {2,2,3,5,5,5,9}

**Time Complexity:**  
The array A is traversed in O(N) time and the resulting sorted array is also computed in O(N) time. Aux[] is traversed in O(K) time. Therefore, the overall time complexity of counting sort algorithm is O(N+K).