Counting Sort 1

Comparison Sorting

Quicksort usually has a running time of $n \times log(n)$, but is there an algorithm that can sort even faster? In general, this is not possible. Most sorting algorithms are comparison sorts, i.e. they sort a list just by comparing the elements to one another. A comparison sort algorithm cannot beat $n \times log(n)$ (worst-case) running time, since $n \times log(n)$ represents the minimum number of comparisons needed to know where to place each element. For more details, you can see these notes (PDF).

Alternative Sorting

Another sorting method, the counting sort, does not require comparison. Instead, you create an integer array whose index range covers the entire range of values in your array to sort. Each time a value occurs in the original array, you increment the counter at that index. At the end, run through your counting array, printing the value of each non-zero valued index that number of times.

Example

$arr = [1, 1, 3, 2, 1]$

All of the values are in the range $[0 \ldots 3]$, so create an array of zeros, $result = [0, 0, 0, 0]$. The results of each iteration follow:

```
i       arr[i]  result
0       1       [0, 1, 0, 0]
1       1       [0, 2, 0, 0]
2       3       [0, 2, 0, 1]
3       2       [0, 2, 1, 1]
4       1       [0, 3, 1, 1]
```

The frequency array is $[0, 3, 1, 1]$. These values can be used to create the sorted array as well: $sorted = [1, 1, 1, 2, 3]$.

Note

For this exercise, always return a frequency array with 100 elements. The example above shows only the first 4 elements, the remainder being zeros.

Challenge

Given a list of integers, count and return the number of times each value appears as an array of integers.

Function Description

Complete the countingSort function in the editor below.

countingSort has the following parameter(s):

- arr[n]: an array of integers

Returns

- int[100]: a frequency array

Input Format

The first line contains an integer n, the number of items in arr.
Each of the next n lines contains an integer $arr[i]$ where $0 \leq i < n$.

Constraints

$100 \leq n \leq 10^6$
$0 \leq arr[i] < 100$

Sample Input

```
100
63 25 73 1 98 73 56 84 86 57 16 83 8 25 81 56 9 53 98 67 99 12 83 89 80 91 39 86 76 85 74 39
```

Sample Output

```
0 2 0 2 0 0 1 0 1 2 1 0 1 1 0 0 2 0 1 0 1 2 1 1 1 3 0 2 0 0 2 0 3 3 1 0 0 0 0 2 2 1 1 1 2 0 2
```

Author HackerRank
Difficulty Easy
Max Score 100
Submitted By 16131

NEED HELP?

View discussions
View editorial
View top submissions

RATE THIS CHALLENGE

MORE DETAILS

Download problem statement
Download sample test cases
Suggest Edits

Explanation

Each of the resulting values $result[i]$ represents the number of times i appeared in arr.

Change Theme Language C

```c
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'countingSort' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts INTEGER_ARRAY arr as parameter.
 */

/*
 * To return the integer array from the function, you should:
 *     - Store the size of the array to be returned in the result_count variable
 *     - Allocate the array statically or dynamically
 *
 * For example,
 * int* return_integer_array_using_static_allocation(int* result_count) {
 *     *result_count = 5;
 *
 *     static int a[5] = {1, 2, 3, 4, 5};
 *
 *     return a;
 * }
 *
 * int* return_integer_array_using_dynamic_allocation(int* result_count) {
 *     *result_count = 5;
 *
 *     int *a = malloc(5 * sizeof(int));
 *
 *     for (int i = 0; i < 5; i++) {
 *         *(a + i) = i + 1;
 *     }
 *
 *     return a;
 * }
```

Line: 29 Col: 41

Upload Code as File Test against custom input Run Code Submit Code