[CENG 315 ALL Sections] Algorithms

Navigation

- Dashboard
  - Site home
  - Site pages
  - My courses
    - 571 - Computer Engineering
      - CENG 223 All Sections
      - CENG 223 Section 2
      - CENG 315 ALL Sections
        - Participants
        - Competencies
        - Grades
        - General
        - October 2 - October 8
        - October 9 - October 15
        - October 15 - October 22
        - October 23 - October 29
        - October 30 - November 5
        - November 6 - November 12
          - THE2
            - Description
            - Submission view
        - November 13 - November 19
        - November 20 - November 26
        - November 27 - December 3
        - December 3 - December 10
        - December 11 - December 17
        - December 18 - December 24
        - December 25 - December 31
        - January 1 - January 7
        - January 8 - January 14
      - CENG 315 Section 1

Description | Submission view

Grade

Reviewed on Wednesday, November 29, 2023, 11:52 AM by Automatic grade
Grade: 100.00 / 100.00

Assessment report [-]
[+]**Output of make**
[-]**For input 01:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 02:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 03:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 04:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 05:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 06:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 07:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 08:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 09:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 10:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 11:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 12:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 13:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 14:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 15:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 16:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 17:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 18:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 19:**
Sorting: Correct
# of Iterations: Correct
[-]**For input 20:**
Sorting: Correct
# of Iterations: Correct

Submitted on Saturday, November 11, 2023, 7:44 PM (Download)

Submitted on Saturday, November 11, 2023, 7:44 PM (Download)

the2.cpp

```cpp
#include "the2.h"

// do not add extra libraries here
/*
    arr       : array to be sorted, in order to get points this array should contain be in sorted state before returning
    ascending : true for ascending, false for descending
    size      : number of elements in the array
*/

// find the element that has maximum number of letter digits
// we will use it while iterating
int find_max_element(std::string* arr, int size, int &iter_count)
{
    int max_num_of_letters = 0;
    for (int i = 0; i < size; i++)
    {
        if(arr[i].size()>max_num_of_letters)
            max_num_of_letters=arr[i].size();
    }

    return max_num_of_letters;
}

void counting_sort(std::string* A, int size, int current_digit , bool ascending, int &iter_count){

    // there are 26 uppercase English letters
    int letter_count=26;
    std::string B[size+1] = {};
    int C[letter_count+1] = {0}; //+1 for shorter strings

    // subtract 64 to map uppercase letters from 65-90 (according to ASCII) to 1-26 interval
    // we will use 0th index for the shorter strings

    // find the frequency of letters
    for(int j = 0; j < size; j++)
    {
        if(A[j].size() < current_digit+1 ) // if the string is shorter
            C[0]++;
        else
            C[(A[j][current_digit]-64)]++;
        iter_count++;
    }

    // cumulative counting sort
    for(int i = 1; i < letter_count+1; i++)
    {
        C[i]=C[i]+C[i-1];
        iter_count++;
    }

    // sort the strings to B according to current_index
    for(int i = 0, j=size-1; j > -1; j--)
    {
        if(A[j].size() >= current_digit+1 )
        {
            B[C[A[j][current_digit]-64]-1] = A[j];
            C[A[j][current_digit]-64] = C[A[j][current_digit]-64]-1;
        }
        else // if the string is shorter
        {
            B[C[0]-1] = A[j];
            C[0]--;
        }
        iter_count++;

    }
    for(int j = 0, i = 0; i < size;i++) // copy the newly sorted array B into original array arr
    {
        A[i]=B[i];
        iter_count++;
    }

}

int radix_string_sort(std::string* arr, int size, bool ascending){

    int iter_count =0;
    int max_digits = find_max_element(arr,size,iter_count);

    // iterate based on the longest array
    for (int i = max_digits-1; i >-1 ; i-- )
        counting_sort(arr,size,i,ascending,iter_count);

    // simply reverse the array if the descending order is requested
    if (ascending == false)
    {
        for(int i = 0; i < size/2 ; i++)
        {
            std::string temp = arr[i];
            arr[i] = arr[size-1-i];
            arr[size-1-i] = temp;
        }
    }
    return iter_count;
}

```

test.cpp

```cpp
// this file is for you for testing purposes, it won't be included in evaluation.

#include <iostream>
#include <fstream>
#include "the2.h"

void file_input(std::string*& input_array, int& size, bool& ascending){
    std::string file_name = "inp02.txt"; // inp01-inp10 are available.
    std::ifstream infile (file_name);
    if(!infile.is_open()){
        std::cout << "Input file cannot be opened" << std::endl;
        std::cout << "File name: " << file_name << std::endl;
        return;
    }
    infile >> ascending;
    infile >> size;
    input_array = new std::string[size];
    for(int j=0; j<size; j++){
        infile >> input_array[j];
    }
    return;
}

void test(){
    int number_of_iteration, size;
    bool ascending;
    std::string* input_array;
    file_input(input_array, size, ascending);
    std::cout << "Size: " << size << std::endl <<
                 "Ascending: " << ascending << std::endl <<
                 "Array elements: {";
```

```
    for(int idx=0; idx < size - 1; idx++) std::cout << input_array[idx] << ", ";
    std::cout << input_array[size-1] << "}" << std::endl;
    number_of_iteration = radix_string_sort(input_array, size, ascending);
    std::cout << "Number of iterations: " << number_of_iteration << std::endl <<
                 "Sorted array: {";
    for(int idx=0; idx<size-1; idx++) std::cout << input_array[idx] << ", ";
    std::cout << input_array[size-1] << "}" << std::endl;
    return;
}

int main(){
    test();
    return 0;
}
```

VPL